PADERBORN UNIVERSITY
The University for the Information Society

Dissertation

CCA-Security for Predicate Encryption Schemes

M. Sc. Gennadij Liske

Oktober 13, 2017

Submitted to the
Department of Computer Science
Paderborn University

for the degree of
Doktor der Naturwissenschaften
(doctor rerum naturalium)

Supervisor: Prof. Dr. rer. nat. Johannes Blömer

accepted on the recommendation of

Prof. Dr. Johannes Blömer
Paderborn University

Prof. Dr. Tibor Jager
Paderborn University

defended on
September 1, 2017

For My Parents

Acknowledgments

First of all, I am deeply grateful to my advisor, Prof. Dr. Johannes Blömer, for introducing me the field of cryptography, for scientific education, and for support through the past years. I thank Johannes for the opportunity to work on several interesting research projects and on the subject of this thesis. Furthermore, I am very grateful to Prof. Dr. Tibor Jager for the feedback on my work and on this thesis.

I thank my colleagues Jan Bobolz, Sascha Brauer, Kathrin Bujna, Fabian Eidens, Dr. Peter Günther, Jakob Juhnke, Dr. Saqib Kakvi, Dr. Volker Krummel, Nils Löken, and David Teusner for many small but fruitful discussions, feedback, new ideas, and proof-reading. I also thank Michelle Kloppenburg for proof-reading.

I am for ever grateful to Jim and Debbie Greensmith, Eric, Lucrezia, princess Layla Rose, big guy Connor James and baby Alex Carrillo for great experiences in San Diego. Jim and Debbie, thank you for giving me a real home far away from my family, for awesome adventures and movie evenings. God bless your families.

Ich bedanke mich ganz herzlich bei all meinen Freunden, die ich in Deutschland gefunden habe. Ein besonderer Dank an Martin Kramer, den ich an meinem ersten Tag in der Uni kennen gelernt habe. Du erinnerst dich bestimmt noch an die E-Mail, die ich dir daraufhin geschrieben habe. Danke, dass du und Julia mich so herzlich in euren Freundschaftskreis aufgenommen habt. Danke auch an Philipp Brandes, Kathrin Bujna, Andre Diekwisch, Dominik Leibenger, Jürgen und Irina Tessmann, Boris und Claudia Wolf, und an meine kleine Schwesterchen Claudia Schumacher. Ihr habt mein Studium zu einem besonderen Teil meines Lebens gemacht. Ich bezweifle, dass es eine bessere Uni-Clique gibt. Boris, ich danke dir für die Organisation der Fahrradtouren, die uns ermöglicht haben den Kontakt aufrecht zu erhalten und dabei wirklich etwas Besonderes jedes Jahr zu erleben. David, Fabian, Jakob und Sascha danke auch euch an dieser Stelle dafür, dass ihr meine Promotion in Freundschaft begleitet habt und mich beim Darts viel zu oft gewinnen lassen habt.

Ein besonderer Dank an die Familie Jahn. Amrei, Jette, Claudia und Christoph, ihr seid einfach nur die coolsten, lustigsten, freundlichsten und die herzlichsten überhaupt. Danke für alles, was ihr bewusst und unbewusst für mich in den letzten Jahren getan habt. Ich wünsche mir, dass es in der Welt mehr Menschen gebe, die so offen und gutherzig wären wie ihr, Familie Kramer und Familie Greensmith. Liebe Claudia, danke dir für ein offenes Ohr und für die immer offene Tür bei all unseren Anliegen und Späßen, sowie für die Organisation angefangen mit dem Anmelden der Urlaubstage bis hin zu AG-Reisen. Den Ostsee und das Nuscheln werde ich wohl trotz all deinen Mühen behalten.

Большое спасибо моим дорогим родителям Вилентине и Эрвину за то, что подарили мне жизнь, за воспитание, за данное вами образование и за поддержку в любых жизненных ситуациях. Вы самое дорогое, что есть в моей жизни и именно поэтому эта работа посвящена именно вам. Спасибо моей любимой сестрёнке Наде, которая принимала непосредственное участие в моём воспитании. Особое спасибо за то, что ты научила меня читать и за то, что ты для меня всегда была примером для подражания. Спасибо моей любимой Алёнке за поддержку и за терпение, которое было ой как необходимо во время оформления работы. С твоим появлением в моей жизни всё встало на свои места и работа начала продвигаться намного лучше. Спасибо также всей моей большой и дружной семье. Вы опора всей моей

жизни и без всех вас её даже представить себе невозможно. Особое спасибо моим бабушкам Вильме и Наде и дедушке Даниилу, которые год от года дарили нам самые счастливые летние каникулы. Вы всегда остаётесь в наших сердцах также как дядя Федя, совсем недавно покинувший нас.

Особое спасибо также моей первой Учительнице, Ночьвай Елене Георгиевне за то, что вы в раннем возрасте вдохновили меня на покорение научного пространства. Я до сих пор помню ваш самый первый урок, подаривший мне осознание бескрайности просторов нашей Родины. Спасибо и моей Учительнице математики в старших классах Рыжакиной Алле Дмитриевне за систематическое изучение математических основ, за то, что вы всегда находили возможность заинтересовать нас на уроках и за то, что все ваши ученики, несмотря на все наши различия всегда были заняты делом. Также спасибо Губарь Тамаре Борисовне, моей Учительнице химии, благодаря которой я сделал свои первые шаги в мир науки уже в 9 классе. Спасибо всем вам за вашу бескорыстную преданность людям, детям и вашим не всегда послушным ученикам.

Ну и конечно огромное спасибо моим лучшим друзьям, практически братьям, Андрею Орлу, Косте Майданюку, Олегу Ковалёву, Андрею Моору. Друзья детства, с которыми мы несмотря на расстояния и границы спустя 15 лет поддерживаем дружбу. Ну и особое спасибо Александру Лайкеру и солнышку Иришке за дружбу, которую я обрёл уже в Германии, за вашу поддержку в самые трудные для меня времена и за съеденные бутерброды, предназначенные вовсе не для меня.

Abstract

In this thesis we first take a critical look at established security definitions for predicate encryption with public index (PE) under chosen-plaintext attacks (CPA) and under chosen-ciphertext attacks (CCA) from the current state of research. In contrast to conventional public-key encryption (PKE), security definitions for PE have to deal with user collusion, which is modeled by an additional key generation oracle. We identify three different formalizations of user secret key handling in the literature implicitly assumed to lead to the same security notion. Contrary to this assumption, we prove that the corresponding models result in two different security notions under CPA and three different security notions under CCA. Similarly to the recent results for PKE and conventional key encapsulation mechanism (KEM) we also analyze subtleties in security definitions for PE and predicate key encapsulation mechanism (P-KEM) regarding the so-called *no-challenge-decryption* condition. While the results for PE and PKE are similar, the results for P-KEM significantly differ from the corresponding results for conventional KEM. As a conclusion of this investigation we suggest well-grounded security definitions for PE and P-KEM under different attack scenarios.

Our main contribution is the development of a general framework and techniques to construct efficient, fully secure predicate encryption schemes with public index withstanding adaptive chosen-ciphertext attacks. We follow the so-called direct chosen-ciphertext approach known from public key encryption [BMW05] and from identity-based encryption [KG09]. The application to sophisticated predicates, as well as the generic nature of our construction, are novel for the underlying techniques. We first develop our approach and proof techniques in composite-order groups based on the recently introduced pair encoding framework for constructing CPA-secure schemes [Att14a]. Schemes in composite-order groups benefit from simplicity and clarity and are well suitable for analysis and design of novel constructions and techniques. In particular, we introduce an elegant proof technique which can be applied when the dual system encryption methodology [Wat09a, LW10], used to construct almost all known fully CPA-secure PE schemes, is utilized in the context of CCA-security. On the one hand, due to this proof technique the reduction costs of our framework are comparable to the reduction costs of the underlying CPA-secure framework. This is important because low reduction costs significantly contribute to the efficiency and practicability of the resulting schemes. On the other hand, our technique remarkably simplifies the security proof of our resulting CCA-secure pair encoding framework.

Using the techniques developed in composite-order groups we finally introduce a framework for constructing fully CCA-secure predicate encryption schemes with public index in significantly more efficient groups of prime order. We extend the CPA-secure pair encoding framework of Attrapadung [Att16] without requiring any additional properties from pair encoding schemes. We use an additional collision-resistant hash function and apart from that, the public parameters and the ciphertexts are extended by a few group elements. The main additional effort is required in the decryption algorithm, which performs newly developed consistency checks before the actual decryption. The resulting framework leads to new, and for many sophisticated predicates first, fully CCA-secure PE schemes in prime-order groups. We achieve CCA-secure schemes for various key-policy attribute-based encryption (ABE), ciphertext-policy ABE, dual-policy ABE, and schemes for regular languages, to name just a few. Furthermore, new computationally secure pair encodings will directly lead to CCA-secure PE schemes for corresponding predicates from the presented framework. Finally, our construction bring new insights into the possibilities of consistency checks for schemes constructed using the dual system encryption methodology and the prime-order dual-system groups [CW13].

Zusammenfassung

In dieser Dissertationsschrift werfen wir zunächst einen kritischen Blick ausgehend vom aktuellen Stand der Forschung auf die Sicherheitsdefinitionen für die prädikatbasierten Verschlüsselungsverfahren mit öffentlichem Index (PE) unter Angriffen mit gewählten Klartexten (CPA) und unter Angriffen mit gewählten Chiffretexten (CCA). Im Gegensatz zu den herkömmlichen Public-Key-Verschlüsselungsverfahren (PKE) betrachten Sicherheitsdefinitionen für PE unter anderem betrügerische Benutzerabsprachen. Dies ist durch ein zusätzliches Orakel zur Generierung von Nutzerschlüsseln modelliert. Wir identifizieren drei unterschiedliche Formalisierungen für die Handhabung der Schlüssel in der Literatur. Implizit wurde dabei bisher angenommen, dass diese Formalisierungen zum selben Sicherheitsbegriff führen. Im Gegensatz zu dieser Annahme zeigen wir, dass die entsprechenden Sicherheitsmodelle in zwei unterschiedlichen Sicherheitsbegriffen unter CPA und in drei unterschiedlichen Sicherheitsbegriffen unter CCA resultieren. Weiterhin, ähnlich zu den kürzlich vorgestellten Ergebnissen für PKE und für die herkömmlichen Key Ensapculation Mechanismen (KEM) [BHK15], analysieren wir Feinheiten in den Sicherheitsdefinitionen für PE und Prädikat-KEM (P-KEM) in Bezug auf die Bedingung, die besagt, dass die Entschlüsselung der Herausforderung nicht angefragt werden darf. Während die Ergebnisse für PE und PKE sehr ähnlich sind, unterscheiden sich die Ergebnisse für P-KEM und KEM erheblich. Basierend auf dieser Analyse schlagen wir schließlich fundierte Sicherheitsdefinitionen für PE und P-KEM unter verschiedenen Angriffsszenarien vor.

Das Hauptresultat dieser Arbeit ist die Entwicklung eines Frameworks sowie von Techniken zur Instanziierung von effizienten, adaptiv sicheren prädikatbasierten Verschlüsselungsverfahren mit öffentlichem Index, die den Angriffen mit gewählten Chiffretexten standhalten. Dabei verfolgen wir den Ansatz einer direkten Konstruktion von CCA-sicheren Verfahren, welcher aus dem Kontext von PKE [BMW05] und aus dem Kontext von identitätsbasierten Verschlüsselungsverfahren [KG09] bekannt ist. Die Anwendung dieser Methode für komplexe Prädikate sowie die generische Natur des Frameworks sind neu für die zugrundeliegenden Techniken. Wir entwickeln unseren Ansatz in Gruppen zusammengesetzter Ordnung basierend auf dem vor kurzem vorgestellten Framework der sogenannten Paarkodierung [Att14a]. Verfahren in Gruppen zusammengesetzter Ordnung sind einfach und anschaulich. Somit eignen sich diese besonders für die Analyse und für das Design von neuen Konstruktionen und Techniken. Wir stellen insbesondere eine elegante Beweistechnik vor. Die Technik findet ihre Verwendung wenn die Methode der dualen Verschlüsselung [Wat09a, LW10], die bei der Konstruktion von fast allen adaptiv CPA-sicheren Verfahren verwendet wurde, im Kontext von CCA benutzt wird. Zum einen, vereinfacht unsere Technik den Sicherheitsbeweis für unser resultierendes CCA-sicheres Framework. Zum anderen, erreichen wir dank dieser Beweistechnik fast dieselben Reduktionskosten wie das zugrundeliegende CPA-sichere Framework. Das ist wichtig, da niedrige Reduktionskosten erheblich zur Effizienz der resultierenden Verfahren beitragen.

Unsere, in den Gruppen zusammengesetzter Ordnung entwickelten Techniken, benutzen wir schließlich um das Framework zur Instanziierung von adaptiv CCA-sicheren prädikatbasierten Verschlüsselungsverfahren mit öffentlichem Index in den erheblich effizienteren Gruppen von Primzahlordnung zu präsentieren. Wir erweitern dabei das CPA-sichere Paarkodierung Framework von Attrapadung [Att16] ohne jegliche Restriktionen an die Paarkodierung zu stellen. Wir nutzen eine zusätzliche kollisionsresistente Hashfunktion und erweitern die öffentlichen Parameter und die Chiffretexte mit wenigen Gruppenelementen. Der meiste zusätzliche Aufwand wird im Entschlüsselungsalgorithmus benötigt. Dieser Algorithmus führt neu entwickelte Konsistenzchecks für Chiffretexte vor der eigentlichen Entschlüsselung durch. Aus der re-

sultierenden Konstruktion ergeben sich neue, und für viele komplexe Prädikate erste, adaptiv CCA-sichere prädikatbasierte Verschlüsselungsverfahren in Gruppen von Primzahlordnung. Wir erreichen CCA-sichere Verfahren für verschiedenartige Key-Policy attributbasierte Verschlüsselungsverfahren (ABE), Chiffretext-Policy ABE, Dual-Policy ABE, Verfahren für reguläre Sprachen, und andere. Neu entwickelten Paarkodierungen werden weiterhin durch unsere Konstruktion direkt zu CCA-sicheren prädikatbasierten Verschlüsselungsverfahren führen. Schließlich, bringen unsere Techniken neue Erkenntnisse bezüglich der Möglichkeiten von Konsistenzchecks für PE Verfahren, die mit Hilfe von der Methode der dualen Verschlüsselung in den sogenannten dualen Gruppensystemen von Primzahlordnung [CW13] entwickelt wurden.

Contents

List of Figures

List of Tables

General Introduction

In the modern globally networked world the maintenance of confidentiality of personal and business data is increasingly becoming one of the most important and crucial challenges in computer sciences. Cryptography, and particularly encryption schemes, provide fundamental tools to develop solutions to this problem. In fact, encryption schemes themselves are hardly the target of adversarial attacks on networked systems since established primitives are well studied and standardized. Nevertheless, modern applications and complex computer systems often require encryption schemes with advanced functionality and even stronger security guarantees. Consequently, encryption schemes which support novel sophisticated systems build a forward-looking research field in modern cryptography. In our opinion, one of the most challenging parts in this research field is to bring the experience of decades of cryptographic research and the enormously fast evolving field of computer sciences together with the objective of developing a toolbox of strongly secure cryptographic primitives for current and future applications.

Predicate encryption (PE) is a relatively new but already established and well-studied cryptographic primitive which can be used to realize *fine-grained* access control to confidential data by cryptographic encryption. Intuitively, in predicate encryption schemes for predicate R, the data is encrypted under ciphertext indices cInd, which specify access requirements for this data. The users hold secret keys provided with key indices kInd which represent their access rights. A user with a secret key for key index kInd can reconstruct the message, encrypted under ciphertext index cInd, if and only if the predicate is satisfied by these indices – that is, if $\mathrm{R}(\mathrm{kInd}, \mathrm{cInd}) = 1$. PE schemes for various predicates are known. One of the simplest realized predicates is the equality predicate. However, there are also PE schemes for more sophisticated predicates, e.g. schemes for regular languages, where ciphertext indices are deterministic finite automatons and key indices are arbitrary bit strings. In PE schemes for this predicate the user with key index $\mathrm{kInd} = w$ is able to reconstruct the message encrypted under ciphertext index $\mathrm{cInd} = A$ if and only if the automaton A accepts the bit string w. Modeling of access rights and consequently the realization of systems for access control on data using PE schemes highly rely on the underlying predicates. Hence, a variety of PE schemes for suitable predicates is important for realization of different applications.

Research on predicate encryption began with the idea of identity-based encryption (IBE), a predicate encryption for equality predicate, introduced by Shamir in 1984 [Sha84]. The first fully functional IBE schemes were presented almost two decades later in the pioneering work of Boneh, Franklin [BF01, BF03] and Cocks [Coc01]. Furthermore, in [SW05] Sahai and Waters proposed a generalization of IBE, for the first time. These results initiated an extensive study of general predicate encryption which in turn resulted in schemes for various sophisticated predicates. An even more general notion of functional encryption was formalized by Boneh, Sahai, and Waters in [BSW11].

In [BSW11] the authors differ between two types of predicate encryption: predicate encryption with public index and index hiding predicate encryption. The latter are used in applications where the access rights required to reconstruct the encrypted data are confidential, because the ciphertexts of these schemes hide not only the actual message but also the ciphertext index. In turn, in predicate encryption schemes with public index (also called payload hiding PE) the ciphertext index remains public. The focus of this thesis is on predicate encryption with public index, which we simply call predicate encryption. Prominent representatives of PE with public index are hierarchical identity-based encryption (HIBE) [BF03, KG09, LW10], attribute-based encryption (ABE) [SW05, LOS$^+$10, LW12], and predicate-based encryption schemes for

regular languages [Wat12, Att14a, Att16], to name just a few. Hence, predicate encryption is a collective term for a variety of well-established and novel encryption schemes with sophisticated functionality which already serve as building blocks for modern applications or can be used to develop new systems with fine-grained access control.

In the cryptographic research community, semantic security under chosen-ciphertext attacks (CCA-security) is widely accepted to be the most desirable security notion for encryption schemes. The chosen-ciphertext attack scenario models active adversaries and nearly all practical applications require security against CCA. In contrast, security against chosen-plaintext attacks (CPA-security) covers only passive attacks. In the context of conventional public key encryption (PKE), as well as in the context of identity-based encryption, researchers have payed much attention to CCA-security. Surprisingly, this is not the case in the context of predicate encryption for more involved predicates. In this thesis we close this gap and present new and for many predicates the first efficient fully CCA-secure PE schemes.

1. Preliminaries

In this chapter we present general notational conventions and provide some cryptographic background required for this thesis. First, in Section 1.1 we define various families of cryptographic functions. Then, in Section 1.2 we provide formal definitions of predicate families and predicate-based schemes with public index. In this section we define important notational conventions that are extensively used in the whole thesis. Finally, in Section 1.3 we define pair encoding schemes, the central cryptographic primitive for our work, originally presented in [Att14a]. In the last section we consider properties of pair encodings which were partially used in [Att14a, AY15, Att16] and are essential for our constructions.

Notation

We denote by $\alpha := a$ the assignment of the value a to the variable α. The operator $:=$ is also used to define new objects. For $n \in \mathbb{N}$ we denote by $[n]$ the set $\{1, \ldots, n\}$ and by $[n]_0$ the set $\{0, \ldots, n\}$. Equations denoted by $\overset{\text{by def.}}{=}$ hold by corresponding definitions, which will be obvious from the context. Furthermore, equations denoted by $\overset{\text{by const.}}{=}$ hold by construction of the corresponding algorithms. The abbreviation *ppt* stands for probabilistic polynomial-time.

Let S be a finite set and X be a random variable on S. We denote by $[X]$ the support of X. This notation can be extended to ppt algorithms, since every ppt algorithm $\mathcal{A}$ on input x defines a finite output probability space, which we denote by $\mathcal{A}(x)$. That is, $[A(x)]$ denotes all possible outcomes of A on input x. In turn, we write $\alpha \leftarrow X$ to denote sampling of an element from S according to the distribution defined by X ($y \leftarrow A(x)$ for ppt algorithms). We also write $\alpha \leftarrow S$ when sampling an element from S according to the uniform distribution. The uniform distribution on $\{0,1\}^n$ is denoted by $\mathcal{U}_n$.

If $\mathcal{A}$ makes independent random choices, e.g. $r_1 \leftarrow X$, $r_2 \leftarrow Y$, we sometimes make some of these choices explicit and write $r_2 \leftarrow Y$, $\mathcal{A}(x; r_2)$. This means that r_2 is given as additional input to $\mathcal{A}$, whereas r_1 is still chosen by $\mathcal{A}$. Consequently, $\mathcal{A}(x; r_1, r_2)$ denotes deterministic computation. Furthermore, if some inputs or outputs of an algorithm are not relevant for the considered statement, we sometimes denote these by $\_$. For example, if $\mathcal{A}$ does not use the third input if the second input is equal one, we simply write $\mathcal{A}(x, 1, \_)$ to denote the corresponding computation.

1.1. Cryptographic Primitives

In this section we recall the formal definitions of well-known cryptographic primitives used in several parts of this thesis. We mostly define the primitives in accordance with [Gol04a, Gol04b], but we present more abstract definitions due to various scenarios where these primitives are used in this thesis (see also [KL15] for similar definitions).

1.1.1. Pseudorandom Functions

In this subsection we recall the formal definition of a pseudorandom function (PRF) with unbounded inputs (cf. [Gol04a, KL15]).

Definition 1.1. Let $l : \mathbb{N} \to \mathbb{N}$ be a polynomial. We say that

$$\mathcal{PRF} = \left\{ F_s : \{0,1\}^* \to \{0,1\}^{l(|s|)} \right\}_{s \in \{0,1\}^*}$$

is a **pseudorandom function** (PRF) with output length $l(\cdot)$ if the following conditions hold:

1. *Easy to compute* : There exists a polynomial time algorithm which on input $(s, x) \in \{0,1\}^* \times \{0,1\}^*$ outputs $F_s(x)$.
2. *Pseudorandom* : For every ppt oracle distinguisher $\mathcal{D}$ there exists a negligible function $\mathrm{negl}(\lambda)$ such that for sufficiently large λ it holds

$$\left| \Pr_{f \leftarrow \mathrm{Func}_\lambda} \left[\mathcal{D}^{f(\cdot)} \left(1^\lambda\right) = 1 \right] - \Pr_{s \leftarrow \{0,1\}^\lambda} \left[\mathcal{D}^{\mathcal{PRF}_s(\cdot)} \left(1^\lambda\right) = 1 \right] \right| \leq \mathrm{negl}(\lambda) \quad ,$$

   where $\mathrm{Func}_\lambda = \left\{ f : \{0,1\}^{\leq d(\lambda)} \to \{0,1\}^{l(\lambda)} \right\}$ is the set of all functions mapping strings with up to $d(\lambda)$-bits to $l(\lambda)$-bit-strings. In turn, $d(\lambda)$ is the upper bound for the length of the inputs submitted by $\mathcal{D}$ on input 1^λ to the oracle.

In this work we use PRFs with domains $D \neq \{0,1\}^*$. Given a PRF $\mathcal{F}$ with unbounded inputs the required PRFs can be easily constructed using an arbitrary, but fixed, efficiently computable canonical representation of the elements in D. That is, we require that every element of D has a unique binary representation. The domain D may be even further parametrized but this can also be covered by the encoding.

Namely, let P be a set of parameters, $\Delta = \{D_p\}_{p \in P}$ be a set of domains parametrized by $p \in P$ and

$$\mathcal{PRF} = \left\{ F_s : \{0,1\}^* \to \{0,1\}^{l(|s|)} \right\}_{s \in \{0,1\}^*}$$

be a PRF. Furthermore, let $\left\{ \mathrm{conv}_p : D_p \to \{0,1\}^* \right\}_{p \in P}$ be a family of efficiently computable canonical binary encoding functions. Then, for every $p \in P$ function family

$$\mathcal{PRF}'_p = \left\{ F'_s : D_p \to \{0,1\}^{l(|s|)} \right\}_{s \in \{0,1\}^*} \quad ,$$

where $F'_s : x \mapsto F_s\big(\mathrm{conv}_p(x)\big)$, is a PRF for domain D_p.

For simplicity we denote the random choice of s of length λ by $f \leftarrow \mathcal{PRF}\left(1^\lambda\right)$ and the evaluation of the corresponding function F_s on input x by $f(x)$. Furthermore, if f is given as input, this means that the corresponding key s is given. We always assume that efficiently computable canonical representations for the domains in Δ exist, and hence abstract from this technicality. We assume that f is as follows: $f : D_p \to \{0,1\}^{l(\lambda)}$, where p and consequently D_p will be fixed and explicitly specified by the overall scheme (as well as polynomial $l(\cdot)$).

1.1.2. One-Way-Functions

In this subsection we recall the formal definition of one-way functions (cf. [Gol04a, KL15]).

Definition 1.2. $\mathcal{F} = \left\{ f_s : \{0,1\}^* \to \{0,1\}^* \right\}_{s \in \{0,1\}^*}$ is called **one-way family of functions** if the following conditions hold:

1. *Easy to compute* : There exists a polynomial-time algorithm which on input $(s, x) \in \{0,1\}^* \times \{0,1\}^*$ returns $f_s(x)$.
2. *Hard to invert* : For all ppt algorithms $\mathcal{I}$ there exists a negligible function $\mathrm{negl}(\lambda)$ such that for sufficiently large λ it holds

$$\text{Adv-Invert}_{\mathcal{F},\mathcal{I}}(\lambda) := \Pr\left[\text{Invert}_{f,\mathcal{I}}(\lambda) = 1\right] \leq \mathrm{negl}(\lambda) \quad ,$$

where $\text{Invert}_{f,\mathcal{I}}(\lambda)$ is defined by: Compute $s, x \leftarrow \{0,1\}^\lambda$, $x' \leftarrow \mathcal{I}\left(1^\lambda, s, f_s(x)\right)$. The output of the experiment is one if and only if $f_s(x') = f_s(x)$.

For simplicity we denote the random choice of s of length λ by $f \leftarrow \mathcal{F}\left(1^\lambda\right)$ and the evaluation of the corresponding function f_s on input x by $f(x)$. If f is given as input, this means that s is given. Similar to the pseudorandom functions we sometimes require function families which have specific domains. We always assume that an efficiently computable canonical binary representation for the corresponding domain D exists and abstract from these technical details. The preimage x is then chosen uniformly at random from D and the inverter $\mathcal{I}$ has to output an element from this domain. Hence, experiment $\text{Invert}_{f,\mathcal{I}}(\lambda)$ in our notation is defined by $f \leftarrow \mathcal{F}\left(1^\lambda\right)$, $x \leftarrow D$, $x' \leftarrow \mathcal{I}\left(1^\lambda, f, f(x)\right)$, and the output of the experiment is equal to one if and only if $x' \in D$ and $f(x') = f(x)$.

1.1.3. Families of Collision-Resistant Hash Function

In this subsection we recall the formal definition of collision-resistant hash functions (cf. [Gol04b, KL15]).

Definition 1.3. Let $l : \mathbb{N} \to \mathbb{N}$ be a polynomial. A family of functions $\mathcal{H}$ of the form $\left\{h_s : \{0,1\}^* \to \{0,1\}^{l(|s|)}\right\}_{s \in \{0,1\}^*}$ is called **collision-resistant hash function family** if the following conditions hold:

1. *Easy to compute* : There exists a polynomial-time algorithm which on input $(s, x) \in \{0,1\}^* \times \{0,1\}^*$ returns $h_s(x)$.
2. *Hard-to-form collisions* : For all ppt algorithms $\mathcal{A}$ there exists a negligible function $\mathrm{negl}(\lambda)$ such that for sufficiently large λ it holds

$$\text{Adv-CR}_{\mathcal{H},\mathcal{A}}(\lambda) := \Pr\left[\text{CR}_{\mathcal{H},\mathcal{A}}(\lambda) = 1\right] \leq \mathrm{negl}(\lambda) \quad ,$$

where $\text{CR}_{\mathcal{H},\mathcal{A}}(\lambda)$ is defined by: Compute $s \leftarrow \{0,1\}^\lambda$, $(x_1, x_2) \leftarrow \mathcal{A}\left(1^\lambda, s\right)$. Output of the experiment is one if and only if $x_1 \neq x_2$ and $h_s(x_1) = h_s(x_2)$.

For simplicity we denote the random choice of s of length λ by $\mathrm{H} \leftarrow \mathcal{H}\left(1^\lambda\right)$ and the evaluation of the corresponding function h_s on input x by $\mathrm{H}(x)$. Furthermore, if H is given as input, this means that the key s is given. Similar to pseudorandom functions we sometimes require collision-resistant function families which have specific domain D and additionally with range $\mathbb{Z}_N$ instead of $\{0,1\}^{l(|s|)}$. We assume that there exists an efficiently computable canonical binary representation for the corresponding domain and abstract from these technicalities. The elements x_1, x_2 in the output of $\mathcal{A}$ must satisfy $x_1, x_2 \in D$. We formally treated collision-

resistant hash functions with range $\mathbb{Z}_N$ in [BL16a]. Also in [CS03] the authors use a similar formalization for so-called target collision-resistant hash functions.

1.2. Predicate-Based Schemes

In this section we recall formal definitions of predicate encryption and predicate key encapsulation mechanism. In the first part of the thesis we review security definitions for these primitives. Hence, in this section we discuss many common assumptions about the properties of these schemes. This assumptions slightly reduce the complexity of the corresponding security definitions and have to be carefully justified. Both definitions are presented with respect to the specific definition of predicate families.

1.2.1. Predicate Families

In general, a predicate family is just a set of relations. For our purposes, we require a more specific definition. We give a more fine-grained definition in comparison to [Att14a]. Namely, we split the relation indices into two parts which play different roles and have to be treated differently not only in the syntactical definitions, but also in the security definitions and the corresponding security proofs.

Definition 1.4. Let Ω, Σ be arbitrary sets. A **predicate family** $\mathcal{R}_{\Omega,\Sigma}$ is a set of binary relations

$$\mathcal{R}_{\Omega,\Sigma} = \left\{ \mathrm{R}_{\mathrm{des},\sigma} : \mathbb{X}_{\mathrm{des},\sigma} \times \mathbb{Y}_{\mathrm{des},\sigma} \to \{0,1\} \right\}_{\mathrm{des}\in\Omega,\sigma\in\Sigma} ,$$

where $\mathbb{X}_{\mathrm{des},\sigma}$ and $\mathbb{Y}_{\mathrm{des},\sigma}$ are sets called the **key index space** and the **ciphertext index space** of $\mathrm{R}_{\mathrm{des},\sigma}$, respectively. The following conditions must hold:

- *Efficient membership tests for key indices* : There exists a polynomial-time algorithm which on input $(\kappa, \mathrm{kInd}) \in (\Omega \times \Sigma) \times \{0,1\}^*$ returns one if and only if $\mathrm{kInd} \in \mathbb{X}_\kappa$.
- *Efficient membership tests for ciphertext indices* : There exists a polynomial-time algorithm which on input $(\kappa, \mathrm{cInd}) \in (\Omega \times \Sigma) \times \{0,1\}^*$ returns one if and only if $\mathrm{cInd} \in \mathbb{Y}_\kappa$.
- *Easy to evaluate* : There exists a polynomial-time algorithm which given the indices $(\kappa, \mathrm{kInd}, \mathrm{cInd}) \in (\Omega \times \Sigma) \times \mathbb{X}_\kappa \times \mathbb{Y}_\kappa$ returns $\mathrm{R}_\kappa(\mathrm{kInd}, \mathrm{cInd})$.

By definition, every predicate $\mathrm{R}_{\mathrm{des},\sigma} \in \mathcal{R}_{\Omega,\Sigma}$ is uniquely defined by two indices. The first index $\mathrm{des} \in \Omega$ specifies general description properties of the corresponding predicate (e.g. maximal number of attributes in a key). This index will be chosen by the system administrator depending on the requirements of the application. In turn, index $\sigma \in \Sigma$ specifies domain properties which depend on the security parameter λ (e.g. domain of computation $\mathbb{Z}_N$ with $|N| = \lambda$) and will be determined by the corresponding setup algorithm. We assume w.l.o.g. that $|\sigma| = \lambda$. We usually denote the relation index (des, σ) by κ and the corresponding relation by R_κ or simply by R if κ is obvious from the context. Furthermore, the elements of $\mathbb{X}_\kappa$ and the elements of $\mathbb{Y}_\kappa$ are called key indices and ciphertext indices, respectively. Finally, if for $\mathrm{kInd} \in \mathbb{X}_\kappa$ and $\mathrm{cInd} \in \mathbb{Y}_\kappa$ it holds $\mathrm{R}_\kappa(\mathrm{kInd}, \mathrm{cInd}) = 1$, then we say that indices kInd and cInd fit together (with respect to κ). We remark that for all predicate families considered in this thesis, we have $\Sigma \subset \mathbb{N}$.

We note that in many papers predicate encryption schemes are defined in such a form that the description parameters of the predicates are not given as an input of the setup algorithms, but are used in the description of the algorithms as constants. Furthermore, these constants are sometimes restricted by security assumption or rather the required security assumption

depends on the corresponding parameters (for example in [Wat11]). Hence, in our opinion, it is important to treat the description parameters formally and make them explicit.

In order to make the predicate families more accessible, let us consider the following example. We assume that the reader is familiar with monotone span programs and access structures (see e.g. [GPSW06] for formal definitions).

Example. Consider the following predicate family $\mathcal{R}^{\text{KP-ABE}}$:

- The set of description parameters is $\Omega := \mathbb{N}^2$ and the set of domain parameters is $\Sigma := \{p \in \mathbb{N} \mid p \text{ is a prime number}\}$.

Let $\kappa = \big((l_{max}, n_{max}), p\big) \in \Omega \times \Sigma$ be arbitrary, but fixed.

- $\mathbb{X}_\kappa := \big\{\big(\boldsymbol{M} \in \mathbb{Z}_p^{l \times d}, \rho : [l] \to \mathbb{Z}_p\big) \;\big|\; d \leq l \leq l_{max}\big\}$. That is, $\mathbb{X}_\kappa$ is the set of monotone span programs $\mathcal{M}_\mathbb{A} = (\boldsymbol{M}, \rho)$ over $\mathbb{Z}_p$ with labels from $\mathbb{Z}_p$ and with size restricted by l_{max}. $\mathbb{A}$ is the access structure defined by $\boldsymbol{M}$ and ρ – that is, $\mathbb{A}$ is a subset of the power set of $\mathbb{Z}_p$.
- $\mathbb{Y}_\kappa := \{\gamma \subset \mathbb{Z}_p \mid |\gamma| \leq n_{max}\}$.
- $\mathrm{R}_\kappa(\mathcal{M}_\mathbb{A}, \gamma) = 1$ if and only if γ is an authorized set in the access structure $\mathbb{A}$ ($\gamma \in \mathbb{A}$).

$\mathcal{R}^{\text{KP-ABE}}$ is a predicate family implicitly used in some key-policy attribute-based encryption (KP-ABE) schemes with attribute universe $\mathcal{U} = \mathbb{Z}_p$ (e.g. [GPSW06, ALdP11]). In our example, $\text{des} = (l_{max}, n_{max}) \in \Omega$ specifies the maximal size l_{max} of MSPs and the maximal number n_{max} of attributes in the ciphertext index. The index $\sigma = p \in \Sigma$ specifies the domain of computation $\mathbb{Z}_p$, while the size of p depends on the security parameter λ and by our convention $|p| = \lambda$. Obviously, for all $\kappa \in \Omega \times \Sigma$ there exist efficient membership tests for $\mathbb{X}_\kappa$ and $\mathbb{Y}_\kappa$. Furthermore, by the definition of monotone span programs, there is a ppt algorithm, which given κ, $\mathcal{M}_\mathbb{A}$ and γ accepts if and only if γ is an authorized set in $\mathbb{A}$ – that is, if and only if $\mathrm{R}_\kappa(\mathcal{M}_\mathbb{A}, \gamma) = 1$.

Our first framework to construct predicate encryption schemes is defined over composite-order groups, and hence we have to take care of zero-divisors in $\mathbb{Z}_N$ for composite $N \in \mathbb{N}$. The following definition is adapted from [Att14a] to our notation and specifies one of the properties of predicate families which is required for our framework in composite-order groups.

Definition 1.5. A predicate family $\mathcal{R}_{\Omega,\Sigma}$ is called **domain-transferable** if $\Sigma \subseteq \mathbb{N}$, and for every $\kappa = (\text{des}, N) \in \Omega \times \Sigma$ and every $p \in \mathbb{N}^{>1}$ with $p \mid N$ it holds $\kappa' = (\text{des}, p) \in \Omega \times \Sigma$, $\mathbb{X}_{\kappa'} \subseteq \mathbb{X}_\kappa$, and $\mathbb{Y}_{\kappa'} \subseteq \mathbb{Y}_\kappa$. Furthermore, there must exist a ppt algorithm Factor and projection maps $f_1 : \mathbb{X}_\kappa \mapsto \mathbb{X}_{\kappa'}$ and $f_2 : \mathbb{Y}_\kappa \mapsto \mathbb{Y}_{\kappa'}$ such that for all $\text{kInd} \in \mathbb{X}_\kappa$ and $\text{cInd} \in \mathbb{Y}_\kappa$ it holds:

Completeness: If $\mathrm{R}_\kappa(\text{kInd}, \text{cInd}) = 1$, then $\mathrm{R}_{\kappa'}\big(f_1(\text{kInd}), f_2(\text{cInd})\big) = 1$.

Soundness: If $\mathrm{R}_\kappa(\text{kInd}, \text{cInd}) = 0$ and $\mathrm{R}_{\kappa'}\big(f_1(\text{kInd}), f_2(\text{cInd})\big) = 1$, then a non-trivial factor F of N can be computed by $F := \text{Factor}(\kappa, \text{kInd}, \text{cInd})$.

Again, we consider an example of a natural domain-transferable predicate family.

Example. Consider the following predicate family $\mathcal{R}^{\text{HIBE}}$ implicitly used in some hierarchical identity-based encryption (HIBE) schemes (e.g. [Wat09a, LW10, LW11]):

- The set of description parameters is $\Omega := \mathbb{N}$ and the set of domain parameters is $\Sigma := \{N \in \mathbb{N} \mid N \text{ is a prime number or a product of 3 prime numbers}\}$.

Let $\kappa = (l, N) \in \Omega \times \Sigma$ be arbitrary, but fixed.

- $\mathbb{X}_\kappa = \mathbb{Y}_\kappa := \bigcup_{k=1}^{l} \mathbb{Z}_N^k$.

- $\mathrm{R}_\kappa(\mathrm{kInd}, \mathrm{cInd}) = 1$ if and only if $n = |\mathrm{kInd}| \leq |\mathrm{cInd}| = m$ and for all $i \in [n]$ it holds $d_i = c_i \pmod N$, where $\mathrm{kInd} = (d_1, \ldots, d_n)$ and $\mathrm{cInd} = (c_1, \ldots, c_m)$.

We claim that the predicate family $\mathcal{R}^{\mathrm{HIBE}}$ is domain-transferable. If kInd and cInd are obvious from the context, we use $n = |\mathrm{kInd}|$ and $(d_1, \ldots, d_n) = \mathrm{kInd}$ as well as $m = |\mathrm{cInd}|$ and $(c_1, \ldots, c_m) = \mathrm{cInd}$ without further explanation.

$\Sigma \subseteq \mathbb{N}$ by definition. Let $\kappa = (l, N) \in \Omega \times \Sigma$ and $p \in \mathbb{N}^{>1}$, $p \mid N$ be arbitrary but fixed. Then, it holds $\kappa' = (l, p) \in \Omega \times \Sigma$. Furthermore, $\mathbb{Z}_p \subseteq \mathbb{Z}_N$ and thus $\mathbb{X}_{\kappa'} \subseteq \mathbb{X}_\kappa$, $\mathbb{Y}_{\kappa'} \subseteq \mathbb{Y}_\kappa$. Define the projection maps $f_1 : \mathbb{X}_\kappa \to \mathbb{X}_{\kappa'}$ and $f_2 : \mathbb{Y}_\kappa \to \mathbb{Y}_{\kappa'}$ by $(d_1, \ldots, d_n) \mapsto \big(d_1 \pmod p, \ldots, d_n \pmod p\big)$ and by $(c_1, \ldots, c_m) \mapsto \big(c_1 \pmod p, \ldots, c_m \pmod p\big)$, respectively. Then, it holds:

- If $\mathrm{R}_N(\mathrm{kInd}, \mathrm{cInd}) = 1$, then $n \leq m$ and for all $i \in [n]$, it holds $d_i = c_i \pmod N$. Hence, for all $i \in [n]$, it holds $d_i = c_i \pmod p$ and thus $\mathrm{R}_p\big(f_1(\mathrm{kInd}), f_2(\mathrm{cInd})\big) = 1$.
- If $\mathrm{R}_N(\mathrm{kInd}, \mathrm{cInd}) = 0$, then $n > m$ or there exists $i \in [n]$ with $d_i \neq c_i \pmod N$ and we have three possible cases:
  - $n > m$: It holds $\mathrm{R}_p\big(f_1(\mathrm{kInd}), f_2(\mathrm{cInd})\big) = 0$.
  - $n \leq m$ and $\exists i \in [n] : d_i \neq c_i \pmod p$: It holds $\mathrm{R}_p\big(f_1(\mathrm{kInd}), f_2(\mathrm{cInd})\big) = 0$.
  - $n \leq m$ and $\forall i \in [n] : d_i = c_i \pmod p$: In this case we can efficiently compute a factor F of N by $F := \gcd(c_i - d_i, N)$, because $c_i = d_i + \tau \cdot p$ for some $\tau \in \mathbb{Z}$. Furthermore, $p \mid F$ and $F \mid N$.

We remark that the projection maps f_1 and f_2 do not have to be efficiently computable given κ and the corresponding indices. We conclude that domain-transferable predicate families are very natural predicate families in the context of composite-order groups. In prime-order groups domain-transferability is trivially satisfied as long as $\Sigma \subseteq \mathbb{N}$. This property was presented in [Att14a] in order to formally deal with technicalities regarding zero-divisors in $\mathbb{Z}_N$.

1.2.2. Predicate Encryption With Public Index

We now give a formal syntactic definition of predicate encryption schemes with public index in the terminology of [BSW11]. Afterwards, we specify some notational conventions and assumptions about the schemes which can be made w.l.o.g. and simplify security definitions and the corresponding security proofs. Most of our assumptions about the schemes are natural and have been previously used. We make these assumptions explicit and justify all of them.

Definition 1.6. A **predicate encryption with public index** Π for predicate family $\mathcal{R}_{\Omega,\Sigma}$ and message space $\mathcal{M}$ consists of four ppt algorithms:

Setup $\big(1^\lambda, \mathrm{des}\big) \to (\mathrm{msk}, \mathrm{pp}_\kappa)$: takes as input the unary encoded security parameter λ and description parameter des. It outputs a master secret key and public parameters. The latter specify a relation index κ.

KeyGen $\big(1^\lambda, \mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{kInd}\big) \to \mathrm{sk}$: takes as input public parameters, the master secret key msk, and a key index kInd. It generates a user secret key sk for kInd.

Enc $\big(1^\lambda, \mathrm{pp}_\kappa, \mathrm{cInd}, m\big) \to \mathrm{CT}$: takes as input public parameters, a ciphertext index cInd, and a message m. It outputs a ciphertext CT of m under cInd.

Dec $\big(1^\lambda, \mathrm{pp}_\kappa, \mathrm{sk}, \mathrm{CT}\big) \to m$: takes as input public parameters, a secret key sk and a ciphertext CT. It outputs a message $m \in \mathcal{M}$ or the error symbol $\perp \notin \mathcal{M}$.

Correctness: For every $\lambda \in \mathbb{N}$, every $\mathrm{des} \in \Omega$, every $(\mathrm{msk}, \mathrm{pp}_\kappa) \in \left[\mathrm{Setup}\left(1^\lambda, \mathrm{des}\right)\right]$, every $m \in \mathcal{M}$, every $\mathrm{kInd} \in \mathbb{X}_\kappa$ and $\mathrm{cInd} \in \mathbb{Y}_\kappa$ which satisfy $\mathrm{R}_\kappa\left(\mathrm{kInd}, \mathrm{cInd}\right) = 1$ it must hold

$$\Pr\left[\mathrm{Dec}\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{KeyGen}\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{kInd}\right), \mathrm{Enc}\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{cInd}, m\right)\right) = m\right] = 1 \ .$$

W.l.o.g. we assume that the setup algorithm on input 1^λ and $\mathrm{des} \in \Omega$ outputs public parameters pp_κ such that $\kappa = (\mathrm{des}, \sigma)$ for some $\sigma \in \Sigma$. Furthermore, we assume for convenience that public parameters pp_κ have a length of at least λ, and that λ and $\kappa \in \Omega \times \Sigma$ can be efficiently determined from pp_κ. In particular, we assume that $|\sigma| = \lambda$. Hence, we avoid writing 1^λ as an input of the algorithms (except for the setup algorithm). Furthermore, if the public parameters pp_κ are fixed and obvious from the context, we also avoid writing pp_κ as input of the algorithms.

Next, by Definition 1.4 of predicate families there are efficient membership tests for the key indices and for the ciphertext indices. Hence, w.l.o.g. we assume that the key generation algorithm and the encryption algorithm output the error symbol $\perp$ when given $\mathrm{kInd} \notin \mathbb{X}_\kappa$ or $\mathrm{cInd} \notin \mathbb{Y}_\kappa$. Furthermore, the message space $\mathcal{M}$ of predicate encryption schemes often depends on public parameters e.g. $\mathcal{M} = \mathbb{G}$ for some group $\mathbb{G}$ specified by pp_κ or is equal to $\{0,1\}^*$. We assume w.l.o.g. that the encryption algorithm outputs $\perp$ when given $m \notin \mathcal{M}$.

The characteristic property of predicate encryption schemes with public index is that the ciphertext index is not confidential, and hence this index does not need to be protected. Consequently, these schemes are often called *payload hiding* predicate encryption schemes, which differs from the terminology of [BSW11], that we use. In many predicate encryption schemes the ciphertext index is either implicitly or explicitly given by the ciphertext and is required to execute the decryption algorithm. That is, for every $\mathrm{CT} \in [\mathrm{Enc}\left(\mathrm{pp}_\kappa, \mathrm{cInd}, m\right)]$ the ciphertext index cInd can be efficiently reconstructed from CT. However, for some predicate encryption schemes this is indeed not the case. For example, in many identity-based encryption (IBE) schemes the identity cannot be easily computed from the ciphertext even though it is not confidential. In [BSW11] the authors suggest to extend the IBE schemes such that the ciphertexts explicitly include the corresponding identity (that is, the ciphertext index) in order to fit into their definitional framework. We do not explicitly distinguish between these cases and assume w.l.o.g. that the ciphertext index can be efficiently computed from every syntactically correct ciphertext. If this assumption does not hold, chosen-ciphertext security definitions and particularly the decryption oracle must be slightly adapted. We justify our assumption later when the security model is specified (see Remark 2.6 on page 35). In turn, for the user secret keys we can assume w.l.o.g. that the key indices are included in the secret keys. This assumption is not crucial, since the keys of the user are not under the control of adversary. Consequently, we always assume that the decryption algorithm outputs the error symbol $\perp$ when the key index kInd of the given user secret key and the ciphertext index cInd of the given ciphertext do not satisfy $\mathrm{R}_\kappa\left(\mathrm{kInd}, \mathrm{cInd}\right) = 1$.

Next, we consider the decryption algorithm and present some further notational conventions. It is evident that the decryption algorithm is crucial in the context of chosen-ciphertext attacks. Usually, certain consistency checks are performed on the ciphertext during the decryption. However, syntactical checks are often not explicitly mentioned, even if these checks are crucial for security. In order to prevent implementation errors and inaccuracy of security definitions we use the following approach: Let pp_κ be arbitrary but fixed public parameters. For every $\mathrm{cInd} \in \mathbb{Y}_\kappa$ we introduce the set $\mathbb{C}_{\mathrm{cInd}}$ of what we call syntactically correct ciphertexts under ciphertext index cInd with respect to pp_κ. Furthermore, due to our assumption that the ciphertext indices are public, we define the set $\mathbb{C}_{\mathrm{pp}_\kappa} := \biguplus_{\mathrm{cInd} \in \mathbb{Y}_\kappa} \mathbb{C}_{\mathrm{cInd}}$ of syntactically correct ciphertext with respect to pp_κ as disjoint conjunction. The convention is as follows: When syntactical checks are required, the sets $\mathbb{C}_{\mathrm{cInd}}$ should be explicitly defined and the decryption algorithm has to

check the correct form of the ciphertext with respect to $\mathbb{C}_{\mathrm{pp}_\kappa}$. We often use the requirement $\mathrm{CT} \in \mathbb{C}_{\mathrm{cInd}} \subseteq \mathbb{C}_{\mathrm{pp}_\kappa}$ in the definitions, which should be read as follows: The ciphertext index cInd of CT satisfies $\mathrm{cInd} \in \mathbb{Y}_\kappa$ and CT is a syntactically correct ciphertext under cInd with respect to pp_κ – that is, $\mathrm{CT} \in \mathbb{C}_{\mathrm{cInd}}$. According to our convention, the decryption algorithm outputs error symbol $\perp$ when given a ciphertext CT with index $\mathrm{cInd} \notin \mathbb{Y}_\kappa$ or if $\mathrm{cInd} \in \mathbb{Y}_\kappa$ but $\mathrm{CT} \notin \mathbb{C}_{\mathrm{cInd}}$. In the case that sets $\mathbb{C}_{\mathrm{cInd}}$ for $\mathrm{cInd} \in \mathbb{Y}_\kappa$ are not defined, the security proof and particularly the decryption oracle have to deal with default sets $\mathbb{C}_{\mathrm{cInd}} := \{\mathrm{cInd}\} \times \{0,1\}^*$.

Finally, in security definitions it is useful to be able to assume that keys satisfy a certain syntactical structure. For these purposes we additionally introduce for all $\mathrm{kInd} \in \mathbb{X}_\kappa$ the set $\mathbb{UK}_{\mathrm{kInd}}$ of syntactically correct user secret keys for key index kInd with respect to pp_κ. Furthermore, we define $\mathbb{UK}_{\mathrm{pp}_\kappa} := \biguplus_{\mathrm{kInd} \in \mathbb{X}_\kappa} \mathbb{UK}_{\mathrm{kInd}}$. We remark that, in the context of fault attacks, for example, consistency checks on the keys might be also important. In such contexts, our notational conventions for the ciphertext could also be used for the user secret keys. In this work we do not consider attack scenarios where the adversary can manipulate the keys of the honest users, and hence the sets $\mathbb{UK}_{\mathrm{kInd}}$ and $\mathbb{UK}_{\mathrm{pp}_\kappa}$ are used only to specify the key indices of the user secret keys. That is, the requirement $\mathrm{sk} \in \mathbb{UK}_{\mathrm{kInd}} \subseteq \mathbb{UK}_{\mathrm{pp}_\kappa}$ for the user secret keys should be read as follows: sk is a secret key for key index kInd which satisfies $\mathrm{kInd} \in \mathbb{X}_\kappa$.

1.2.3. Predicate Key Encapsulation Mechanism With Public Index

In practice, predicate encryption schemes are usually used to encrypt a symmetric secret key, which in turn is used to encrypt the actual (longer) message. Special PE schemes constructed extra for this application are also called a predicate key encapsulation mechanism. We use the same notational conventions as for PE, and hence simplify the description. Additionally, we use the so-called family of key spaces $\mathcal{K} = \{\mathbb{K}_\lambda\}_{\lambda \in \mathbb{N}}$, which represents the sets of symmetric keys for every security parameter λ. Typically, there are only few different sets in $\mathcal{K}$ representing key lengths supported by the corresponding symmetric encryption scheme.

Definition 1.7. A **predicate key encapsulation mechanism with public index** Π for predicate family $\mathcal{R}_{\Omega,\Sigma}$ and family of key spaces $\mathcal{K} = \{\mathbb{K}_\lambda\}_{\lambda \in \mathbb{N}}$ consists of four ppt algorithms:

Setup $\left(1^\lambda, \mathrm{des}\right) \to (\mathrm{msk}, \mathrm{pp}_\kappa)$: takes as input the unary encoded security parameter λ and description parameter des. It outputs a master secret key and public parameters.

KeyGen $\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{kInd}\right) \to \mathrm{sk}$: takes as input public parameters, a master secret key msk, and a key index kInd. It generates a user secret key sk for kInd.

Encaps $\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{cInd}\right) \to (\mathrm{K}, \mathrm{CT})$: takes as input public parameters and a ciphertext index cInd. It outputs a key $\mathrm{K} \in \mathbb{K}_\lambda$ and an encapsulation CT of this key.

Decaps $\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{sk}, \mathrm{CT}\right) \to \mathrm{K}$: takes as input public parameters, a secret key sk and an encapsulation CT. It outputs a key $\mathrm{K} \in \mathbb{K}_\lambda$ or the error symbol $\perp \notin \mathbb{K}_\lambda$.

Correctness: For every $\lambda \in \mathbb{N}$, every index $\mathrm{des} \in \Omega$, every $(\mathrm{msk}, \mathrm{pp}_\kappa) \in \left[\mathrm{Setup}\left(1^\lambda, \mathrm{des}\right)\right]$, every $\mathrm{kInd} \in \mathbb{X}_\kappa$ and $\mathrm{cInd} \in \mathbb{Y}_\kappa$ which satisfy $\mathrm{R}_\kappa(\mathrm{kInd}, \mathrm{cInd}) = 1$, and every $(\mathrm{K}, \mathrm{CT}) \in \left[\mathrm{Encaps}\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{cInd}\right)\right]$ it must hold

$$\Pr\left[\mathrm{Decaps}\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{KeyGen}\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{kInd}\right), \mathrm{CT}\right) = \mathrm{K}\right] = 1 \ .$$

We furthermore define the smoothness for P-KEM, which is similar to the definition of smoothness for conventional KEMs (cf. [BHK15]).

Definition 1.8. Let Π be a P-KEM with public index for predicate family $\mathcal{R}_{\Omega,\Sigma}$. Furthermore, let $\mathrm{des} \in \Omega$ and $\lambda \in \mathbb{N}$ be arbitrary. Define

$$\mathrm{Smth}_{\Pi}(\lambda, \mathrm{des}) := \mathbf{E}\left[\max_{\substack{\mathrm{cInd}\in\mathbb{Y}_\kappa,\\ \mathrm{CT}\in\{0,1\}^*}} \left(\Pr_{(\mathrm{K},\mathrm{CT}')\leftarrow\mathrm{Encaps}\left(1^\lambda,\mathrm{pp}_\kappa,\mathrm{cInd}\right)}\left[\mathrm{CT}' = \mathrm{CT}\right]\right)\right] ,$$

where the expected value is taken over $(\mathrm{msk}, \mathrm{pp}_\kappa) \leftarrow \mathrm{Setup}\left(1^\lambda, \mathrm{des}\right)$. Π is called **smooth** if for every $\mathrm{des} \in \Omega$ function $\mathrm{Smth}_{\Pi}(\cdot, \mathrm{des})$ is negligible.

1.3. Pair Encoding Schemes

In this subsection we first recall the formal definition of the *pair encoding* introduced in [Att14a] and slightly adapted to our notation. This novel cryptographic primitive is used to construct PE schemes with public index. Actually, pair encodings are multivariate polynomials which are evaluated during key generation and during encryption. We separate the notation of polynomial variables and the notation of corresponding elements in the schemes, which differs from [Att14a]. In our notation, every polynomial variable gets an index, which in turn is the name of the corresponding element in the schemes. For example, random element s in the scheme corresponds to the polynomial variable X_s in the pair encoding. This justifies the unusual names of variables. Recall that for domain-transferable predicate families it holds $\Sigma \subseteq \mathbb{N}$.

Definition 1.9. A **pair encoding scheme** $\mathrm{P} = (\mathrm{Param}, \mathrm{Enc1}, \mathrm{Enc2}, \mathrm{Pair})$ for a domain-transferable predicate family $\mathcal{R}_{\Omega,\Sigma}$ consists of four ppt algorithms:

Param $(\kappa) =: n$: takes as input $\kappa \in \Omega \times \Sigma$ and outputs $n \in \mathbb{N}$, which defines the number of so-called **public variables** denoted by $\{\mathrm{X}_{h_1}, \ldots, \mathrm{X}_{h_n}\} = \mathbf{X}_{\boldsymbol{h}}$.

Enc1 $(\kappa, \mathrm{kInd}) =: (\boldsymbol{k}, m_2)$: takes as input $\kappa = (\mathrm{des}, N) \in \Omega \times \Sigma$ and $\mathrm{kInd} \in \mathbb{X}_\kappa$. It outputs $m_2 \in \mathbb{N}$ and a vector $\boldsymbol{k} = (k_1, \ldots, k_{m_1})^\top$ of m_1 multivariate polynomials $k_1, \ldots, k_{m_1} \in \mathbb{Z}_N[\mathrm{X}_\alpha, \mathbf{X}_{\boldsymbol{r}}, \mathbf{X}_{\boldsymbol{h}}]$. The variable X_α is called the **master variable** and the variables $\{\mathrm{X}_{r_1}, \ldots, \mathrm{X}_{r_{m_2}}\} = \mathbf{X}_{\boldsymbol{r}}$ are called **key-specific variables**. The polynomials in $\boldsymbol{k}$ are restricted to linear combinations of monomials $\{\mathrm{X}_\alpha,\ \mathrm{X}_{r_i},\ \mathrm{X}_{h_j}\mathrm{X}_{r_i}\}_{i\in[m_2], j\in[n]}$.

Enc2 $(\kappa, \mathrm{cInd}) =: (\boldsymbol{c}, w_2)$: takes as input $\kappa = (\mathrm{des}, N) \in \Omega \times \Sigma$ and $\mathrm{cInd} \in \mathbb{Y}_\kappa$. It outputs $w_2 \in \mathbb{N}$ and a vector $\boldsymbol{c} = (c_1, \ldots, c_{w_1})^\top$ of w_1 multivariate polynomials $c_1, \ldots, c_{w_1} \in \mathbb{Z}_N[\mathrm{X}_{s_0}, \mathbf{X}_{\boldsymbol{s}}, \mathbf{X}_{\boldsymbol{h}}]$. The variable X_{s_0} and the variables $\{\mathrm{X}_{s_1}, \ldots, \mathrm{X}_{s_{w_2}}\} = \mathbf{X}_{\boldsymbol{s}}$ are called **ciphertext-specific variables**. The polynomials in $\boldsymbol{c}$ are restricted to linear combinations of monomials $\{\mathrm{X}_{s_i},\ \mathrm{X}_{h_j}\mathrm{X}_{s_i}\}_{i\in[w_2]_0, j\in[n]}$.

Pair $(\kappa, \mathrm{kInd}, \mathrm{cInd}) \to \boldsymbol{E}$: takes as input $\kappa = (\mathrm{des}, N) \in \Omega \times \Sigma$, $\mathrm{kInd} \in \mathbb{X}_\kappa$, and $\mathrm{cInd} \in \mathbb{Y}_\kappa$. It outputs a matrix $\boldsymbol{E} \in \mathbb{Z}_N^{m_1 \times w_1}$, where m_1 and w_1 are defined by $\mathrm{Enc1}(\kappa, \mathrm{kInd})$ and $\mathrm{Enc2}(\kappa, \mathrm{cInd})$, respectively.

Correctness: Let $\kappa = (\mathrm{des}, N) \in \Omega \times \Sigma$, $\mathrm{kInd} \in \mathbb{X}_\kappa$, $\mathrm{cInd} \in \mathbb{Y}_\kappa$ be arbitrary. Let $(\boldsymbol{k}, m_2) = \mathrm{Enc1}(\kappa, \mathrm{kInd})$, $m_1 = |\boldsymbol{k}|$, and $(\boldsymbol{c}, w_2) = \mathrm{Enc2}(\kappa, \mathrm{cInd})$, $w_1 = |\boldsymbol{c}|$. The following three properties must be fulfilled:

1. If $\mathrm{R}_\kappa(\mathrm{kInd}, \mathrm{cInd}) = 1$, then for every $\boldsymbol{E} \in [\mathrm{Pair}(\kappa, \mathrm{kInd}, \mathrm{cInd})]$ it holds
$$\boldsymbol{k}^\top \cdot \boldsymbol{E} \cdot \boldsymbol{c} = \sum_{\tau \in [m_1]} \sum_{\tau' \in [w_1]} e_{\tau,\tau'} \cdot k_\tau \cdot c_{\tau'} = \mathrm{X}_\alpha \cdot \mathrm{X}_{s_0} \ ,$$
where the computation is in $\mathbb{Z}_N[\mathrm{X}_\alpha, \mathrm{X}_{s_0}, \mathbf{X}_{\boldsymbol{s}}, \mathbf{X}_{\boldsymbol{r}}, \mathbf{X}_{\boldsymbol{h}}]$.
2. For every $\mathrm{kInd} \in \mathbb{X}_\kappa$ and every $p \in \mathbb{N}^{>1}$ with $p \mid N$ it holds
$$\boldsymbol{k}' = \boldsymbol{k} \pmod p \qquad \text{and} \qquad m_2 = m_2' \ ,$$
where $(\boldsymbol{k}', m_2') = \mathrm{Enc1}\big(\kappa, f_1(\mathrm{kInd})\big)$ and f_1 is the projection map from domain-transferability property of $\mathcal{R}_{\Omega,\Sigma}$.
3. For every $\mathrm{cInd} \in \mathbb{Y}_\kappa$ and every $p \in \mathbb{N}^{>1}$ with $p \mid N$ it holds
$$\boldsymbol{c}' = \boldsymbol{c} \pmod p \qquad \text{and} \qquad w_2 = w_2' \ ,$$
where $(\boldsymbol{c}', w_2') = \mathrm{Enc2}\big(\kappa, f_2(\mathrm{cInd})\big)$ and f_2 is the projection map from domain-transferability property $\mathcal{R}_{\Omega,\Sigma}$.

Notice that unlike [Att14a] we allow the algorithm Pair to be probabilistic. Many known pair encoding schemes have a probabilistic pair algorithm (e.g. attribute-based schemes) which cannot be neglected in the context of chosen-ciphertext security since Pair is used by the decryption algorithm and $\boldsymbol{E}$ specifies which parts of the ciphertext are used for decryption. This will be obvious when our frameworks are presented.

We denote the coefficients of polynomials $k_\tau \in \boldsymbol{k}$ by
$$k_\tau = a_\tau \cdot \mathrm{X}_\alpha + \sum_{i \in [m_2]} \left(a_{\tau,i} \cdot \mathrm{X}_{r_i} + \sum_{j \in [n]} a_{\tau,i,j} \cdot \mathrm{X}_{h_j} \cdot \mathrm{X}_{r_i} \right) \ .$$
The coefficients of polynomials $c_\tau \in \boldsymbol{c}$ are denoted by
$$c_\tau = \sum_{i \in [w_2]_0} \left(b_{\tau,i} \cdot \mathrm{X}_{s_i} + \sum_{j \in [n]} b_{\tau,i,j} \cdot \mathrm{X}_{h_j} \cdot \mathrm{X}_{s_i} \right) \ .$$
Hence, every polynomial k_τ is given by a list of $(n+1) \cdot m_2 + 1$ many coefficients in $\mathbb{Z}_N$ and every polynomial c_τ is given by the list of $(n+1) \cdot (w_2 + 1)$ many coefficients in $\mathbb{Z}_N$. In the known pair encoding schemes only few coefficients of the encodings are unequal zero. The same holds also for the reconstruction matrix $\boldsymbol{E}$. This should be taken into account when schemes constructed from pair encodings are implemented.

As a notational convention, whenever a particular relation index κ, a key index $\mathrm{kInd} \in \mathbb{X}_\kappa$, and a ciphertext index $\mathrm{cInd} \in \mathbb{Y}_\kappa$ are under consideration, the following values are also defined: $n = \mathrm{Param}(\kappa)$, $(\boldsymbol{k}, m_2) = \mathrm{Enc1}(\kappa, \mathrm{kInd})$, $m_1 = |\boldsymbol{k}|$, and $(\boldsymbol{c}, w_2) = \mathrm{Enc2}(\kappa, \mathrm{cInd})$, $w_1 = |\boldsymbol{c}|$. Nevertheless, we will often recall these definitions for the sake of readability.

1.3.1. Linearity Properties of Pair Encodings

Due to the restrictions on the polynomials in pair encoding schemes, the encoding polynomials satisfy certain linearity properties, which are formalized in the following lemma. These properties are ubiquitous in all constructions and security proofs.

Lemma 1.10 (Linearity properties of encodings)**.** *Let* $\Sigma \subseteq \mathbb{N}$ *and* P *be a pair encoding scheme for relation family* $\mathcal{R}_{\Omega,\Sigma}$*. Let* $\kappa = (\mathrm{des}, N) \in \Omega \times \Sigma$*,* $\mathrm{kInd} \in \mathbb{X}_\kappa$ *and* $\mathrm{cInd} \in \mathbb{Y}_\kappa$ *be arbitrary. Furthermore, let* $n = \mathrm{Param}(\kappa)$*,* $(\boldsymbol{k}, m_2) = \mathrm{Enc1}(\kappa, \mathrm{kInd})$*,* $m_1 = |\boldsymbol{k}|$*, and* $(\boldsymbol{c}, w_2) = \mathrm{Enc2}(\kappa, \mathrm{cInd})$*,* $w_1 = |\boldsymbol{c}|$*. Then, for every* $a, \alpha, \alpha', s_0 \in \mathbb{Z}_N$*, every* $\boldsymbol{h} \in \mathbb{Z}_N^n$*, every* $\boldsymbol{r}, \boldsymbol{r}' \in \mathbb{Z}_N^{w_2}$*, and every* $\boldsymbol{s} \in \mathbb{Z}_N^{m_2}$ *it holds*

$$a \cdot \boldsymbol{k}(\alpha, \boldsymbol{r}, \boldsymbol{h}) = \boldsymbol{k}(a \cdot \alpha, a \cdot \boldsymbol{r}, \boldsymbol{h}), \tag{1.1}$$
$$a \cdot \boldsymbol{c}(s_0, \boldsymbol{s}, \boldsymbol{h}) = \boldsymbol{c}(a \cdot s_0, a \cdot \boldsymbol{s}, \boldsymbol{h}), \tag{1.2}$$
$$\boldsymbol{k}(\alpha, \boldsymbol{r}, \boldsymbol{h}) + \boldsymbol{k}(\alpha', \boldsymbol{r}', \boldsymbol{h}) = \boldsymbol{k}(\alpha + \alpha', \boldsymbol{r} + \boldsymbol{r}', \boldsymbol{h}), \tag{1.3}$$
$$\boldsymbol{k}(\alpha, \boldsymbol{r}, \boldsymbol{h}) + \boldsymbol{k}(\alpha', \boldsymbol{0}, \boldsymbol{0}) = \boldsymbol{k}(\alpha + \alpha', \boldsymbol{r}, \boldsymbol{h}), \tag{1.4}$$
$$\boldsymbol{k}(\alpha, \boldsymbol{0}, \boldsymbol{h}) = \boldsymbol{k}(\alpha, \boldsymbol{0}, \boldsymbol{0}). \tag{1.5}$$

Proof. Due to the restrictions of polynomials it holds for every $\tau \in [m_1]$:

$$\begin{aligned}
a \cdot k_\tau(\alpha, \boldsymbol{r}, \boldsymbol{h}) &= a \cdot \left[a_\tau \cdot \alpha + \sum_{i \in [m_2]} \left(a_{\tau,i} \cdot r_i + \sum_{j \in [n]} a_{\tau,i,j} \cdot h_j \cdot r_i \right) \right] \\
&= a_\tau \cdot (a \cdot \alpha) + \sum_{i \in [m_2]} \left(a_{\tau,i} \cdot (a \cdot r_i) + \sum_{j \in [n]} a_{\tau,i,j} \cdot h_j \cdot (a \cdot r_i) \right) \\
&= k_\tau(a \cdot \alpha, a \cdot \boldsymbol{r}, \boldsymbol{h}) \quad , \qquad (1.1)
\end{aligned}$$

$$\begin{aligned}
k_\tau(\alpha, \boldsymbol{r}, \boldsymbol{h}) + k_\tau(\alpha', \boldsymbol{r}', \boldsymbol{h}) &= a_\tau \cdot \alpha + \sum_{i \in [m_2]} \left(a_{\tau,i} \cdot r_i + \sum_{j \in [n]} a_{\tau,i,j} \cdot h_j \cdot r_i \right) \\
&\quad + a_\tau \cdot \alpha' + \sum_{i \in [m_2]} \left(a_{\tau,i} \cdot r'_i + \sum_{j \in [n]} a_{\tau,i,j} \cdot h_j \cdot r'_i \right) \\
&= a_\tau \cdot (\alpha + \alpha') + \sum_{i \in [m_2]} \left(a_{\tau,i} \cdot (r_i + r'_i) + \sum_{j \in [n]} a_{\tau,i,j} \cdot h_j \cdot (r_i + r'_i) \right) \\
&= k_\tau(\alpha + \alpha', \boldsymbol{r} + \boldsymbol{r}', \boldsymbol{h}) \quad , \qquad (1.3)
\end{aligned}$$

$$\begin{aligned}
k_\tau(\alpha, \boldsymbol{r}, \boldsymbol{h}) + k_\tau(\alpha', \boldsymbol{0}, \boldsymbol{0}) &= a_\tau \cdot \alpha + \sum_{i \in [m_2]} \left(a_{\tau,i} \cdot r_i + \sum_{j \in [n]} a_{\tau,i,j} \cdot h_j \cdot r_i \right) + a_\tau \cdot \alpha' \\
&= a_\tau \cdot (\alpha + \alpha') + \sum_{i \in [m_2]} \left(a_{\tau,i} \cdot r_i + \sum_{j \in [n]} a_{\tau,i,j} \cdot h_j \cdot r_i \right) \\
&= k_\tau(\alpha + \alpha', \boldsymbol{r}, \boldsymbol{h}) \quad , \qquad (1.4)
\end{aligned}$$

$$\begin{aligned}
k_\tau(\alpha, \boldsymbol{0}, \boldsymbol{h}) &= a_\tau \cdot \alpha + \sum_{i \in [m_2]} \left(a_{\tau,i} \cdot 0 + \sum_{j \in [n]} a_{\tau,i,j} \cdot h_j \cdot 0 \right) \\
&= a_\tau \cdot \alpha \\
&= k_\tau(\alpha, \boldsymbol{0}, \boldsymbol{0}) \quad . \qquad (1.5)
\end{aligned}$$

Furthermore, for every $\tau \in [w_1]$ it holds

$$\begin{aligned} a \cdot c_\tau\left(s_0, \boldsymbol{s}, \boldsymbol{h}\right) &= a \cdot \sum_{i \in [w_2]_0} \left(b_{\tau,i} \cdot s_i + \sum_{j \in [n]} b_{\tau,i,j} \cdot h_j \cdot s_i \right) \\ &= \sum_{i \in [w_2]_0} \left(b_{\tau,i} \cdot (a \cdot s_i) + \sum_{j \in [n]} b_{\tau,i,j} \cdot h_j \cdot (a \cdot s_i) \right) \\ &= c_\tau\left(a \cdot s_0, a \cdot \boldsymbol{s}, \boldsymbol{h}\right) \quad . \end{aligned} \tag{1.2}$$

This finalizes the proof of the lemma. □

1.3.2. Normal and Regular Pair Encodings

In this subsection we define additional natural properties of pair encodings, that will be required, particularly in the framework on prime-order groups. These properties have been previously used in different works [AY15, Att16, AC17a]. Note that independently of these works we implicitly used the normality of ciphertext encodings in our semi-generic selectively CCA-secure framework for predicate-based schemes [BL14].

Definition 1.11. A pair encoding scheme $\mathrm{P} = (\mathrm{Param}, \mathrm{Enc1}, \mathrm{Enc2}, \mathrm{Pair})$ for domain-transferable predicate family $\mathcal{R}_{\Omega,\Sigma}$ (thus $\Sigma = \mathbb{N}$) is called **regular** if the following properties hold for every predicate index $\kappa \in \Omega \times \Sigma$, every $\mathrm{kInd} \in \mathbb{X}_\kappa$, and every $\mathrm{cInd} \in \mathbb{Y}_\kappa$:

1. (**Normal pair encodings**) There exist an index $\tau_0 \in [w_1]$ such that

   $$c_{\tau_0} = \mathrm{X}_{s_0} \quad .$$

   W.l.o.g. we assume that it holds $\tau_0 = 1$.

2. Suppose $\mathrm{R}_\kappa(\mathrm{kInd}, \mathrm{cInd}) = 1$ and let $\boldsymbol{E} \in [\mathrm{Pair}(\kappa, \mathrm{kInd}, \mathrm{cInd})]$ be arbitrary. Then, for every $\tau \in [m_1]$, $\tau' \in [w_1]$ it holds:

   $$\text{If} \quad \exists_{i \in [m_2]} \exists_{j \in [n]} : a_{\tau,i,j} \neq 0 \text{ and } \exists_{i' \in [w_2]_0} \exists_{j' \in [n]} : b_{\tau',i',j'} \neq 0 \quad \text{then} \quad e_{\tau,\tau'} = 0 \quad .$$

3. For every $i \in [m_2]$ it holds:

   $$\text{If} \quad \nexists_{\tau \in [m_1]} : k_\tau = \mathrm{X}_{r_i} \quad \text{then} \quad \forall_{\tau \in [m_1]} \forall_{j \in [n]} : a_{\tau,i,j} = 0 \quad .$$

4. For every $i \in [w_2]$ it holds:

   $$\text{If} \quad \nexists_{\tau \in [w_1]} : c_\tau = \mathrm{X}_{s_i} \quad \text{then} \quad \forall_{\tau \in [w_1]} \forall_{j \in [n]} : b_{\tau,i,j} = 0 \quad .$$

To the best of our knowledge, the first property is satisfied by all known pair encoding schemes. This is due to the special role of random element s_0 corresponding to variable X_{s_0} (recall that by correctness of pair encodings it holds $\boldsymbol{k}^\top \cdot \boldsymbol{E} \cdot \boldsymbol{c} = \mathrm{X}_\alpha \cdot \mathrm{X}_{s_0}$). The second property states that the reconstruction matrix $\boldsymbol{E}$ never combines k_τ and $c_{\tau'}$ (that is $e_{\tau,\tau'} = 0$) if both polynomials contain monomials with public variables (X_{h_i}'s). The third property states that if a key-specific variable (X_{r_i}) does not appear separately (by $k_\tau = \mathrm{X}_{r_i}$), it will never appear in a monomial with public variables ($a_{\tau,i,j} = 0$ for all τ and j). The last property is analogously defined for the

ciphertext encodings. That is, if a ciphertext-specific variable (X_{s_i}) does not appear separately (by $c_\tau = \mathrm{X}_{s_i}$), it will not appear in a monomial with public variables ($b_{\tau,i,j} = 0$ for all τ and j).

All restrictions of regular pair encodings are indeed natural and actually originate from the properties of bilinear maps, that are used to realize predicate encryption schemes. Namely, bilinear maps provide the possibility to combine two elements (one from the user secret key and one from the ciphertext), but only once. This strictly restricts the reconstruction function, which is formalized by matrix $\boldsymbol{E}$ in the pair encodings. Furthermore, for every $\boldsymbol{E}$ it must hold by correctness of pair encodings $\boldsymbol{k}^\top \cdot \boldsymbol{E} \cdot \boldsymbol{c} = \mathrm{X}_\alpha \cdot \mathrm{X}_{s_0}$. That is, all monomials except for $\mathrm{X}_\alpha \cdot \mathrm{X}_{s_0}$ have to be eliminated. To the best of our knowledge there are only few pair encodings which are not regular. All encodings from [Att14a] which are not regular came from the so-called dual transformations and were improved in [AY15], where the resulting schemes are regular.

1.3.3. Three Types of Ciphertext Polynomials

In this subsection we identify different types of variables and polynomials in the ciphertext encodings, that are useful when consistency checks in our CCA-secure constructions are considered. Namely, there are different consistency checks for different types of polynomials.

Definition 1.12. Let P be a pair encoding scheme for relation family $\mathcal{R}_{\Omega,\Sigma}$. Let $\kappa \in \Omega \times \Sigma$, $\mathrm{cInd} \in \mathbb{Y}_\kappa$ be arbitrary but fixed and $\left((c_1, \ldots, c_{w_1})^\top, w_2\right) = \mathrm{Enc2}\left(\kappa, \mathrm{cInd}\right)$. Define $I \subseteq [w_2]_0$ by

$$i \in I \Leftrightarrow \exists_{\tau \in [w_1]} : c_\tau = \mathrm{X}_{s_i} \ .$$

Furthermore, define $\overline{I} := [w_2]_0 \setminus I$. Ciphertext-specific variables in $\{\mathrm{X}_{s_i}\}_{i \in I}$ are called **direct ciphertext-specific variables**. Variables $\{\mathrm{X}_{s_i}\}_{i \in \overline{I}}$ are called **indirect ciphertext-specific variables**.

By the last property of regular pair encodings from Definition 1.11 it holds $\forall_{i \in \overline{I}} \forall_{j \in [n]} \forall_{\tau \in [w_1]} : b_{\tau,i,j} = 0$. Hence, we can write all polynomials $c_\tau \in \boldsymbol{c}$ of a regular pair encoding as follows:

$$c_\tau = \sum_{i \in \overline{I}} b_{\tau,i} \cdot \mathrm{X}_{s_i} + \sum_{i \in I} \left(b_{\tau,i} \cdot \mathrm{X}_{s_i} + \sum_{j \in [n]} b_{\tau,i,j} \cdot \mathrm{X}_{h_j} \cdot \mathrm{X}_{s_i} \right) \ .$$

Hence, in general, every polynomial c_τ of a regular pair encoding is given by the list of $\left|\overline{I}\right| + |I| + |I| \cdot n = w_2 + 1 + |I| \cdot n$ many coefficients.

As mentioned in a recent work [AC17b] on pair encodings, one can assume w.l.o.g. that for all $i \in I$ and for all $\tau \in [w_1]$ it holds $b_{\tau,i} = 0$ except of τ with $c_\tau = \mathrm{X}_{s_i}$. This is due to the fact that given $c_\tau = \mathrm{X}_{s_i}$ we can eliminate appearances of monomial X_{s_i} in all other polynomials. On the one hand this assumption further simplifies the form of encodings but on the other hand this might have negative impacts on the efficiency of the schemes. Our constructions do not need this simplification, but we notice that our verification checks can be slightly simplified if the pair encoding satisfies this more restrictive form.

Definition 1.13. Let P be a pair encoding scheme for relation family $\mathcal{R}_{\Omega,\Sigma}$. Let $\kappa \in \Omega \times \Sigma$, $\mathrm{cInd} \in \mathbb{Y}_\kappa$ be arbitrary but fixed and $\left((c_1, \ldots, c_{w_1})^\top, w_2\right) = \mathrm{Enc2}\left(\kappa, \mathrm{cInd}\right)$. We partition $[w_1]$ in three following sets depending of the form of the corresponding polynomials:

1. $\Gamma_0 \subseteq [w_1]$ is defined by $\tau \in \Gamma_0 \Leftrightarrow \exists_{i \in I} : c_\tau = \mathrm{X}_{s_i}$. Hence, $\tau \in \Gamma_0$ if and only if c_τ has the form:
$$c_\tau = \mathrm{X}_{s_i} \ .$$
The elements in Γ_0 corresponds to the set I and it holds $|\Gamma_0| = |I|$. We call the corresponding polynomials **fixing polynomials**.
2. $\Gamma \subseteq [w_1] \setminus \Gamma_0$ is defined by $\tau \in \Gamma \Leftrightarrow \forall_{i \in \overline{I}} \forall_{j \in [n]} : b_{\tau,i} = 0 \wedge b_{\tau,i,j} = 0$. Hence, $\tau \in \Gamma$ if and only if c_τ has the form:
$$c_\tau = \sum_{i \in I} \left(b_{\tau,i} \cdot \mathrm{X}_{s_i} + \sum_{j \in [n]} b_{\tau,i,j} \cdot \mathrm{X}_{h_j} \cdot \mathrm{X}_{s_i} \right) \ .$$
We call the corresponding polynomials **direct polynomials**.
3. $\overline{\Gamma} := [w_1] \setminus (\Gamma_0 \uplus \Gamma)$. We call the corresponding polynomials **indirect polynomials**.

The name of the fixing polynomials goes back to the fact that the corresponding elements in the ciphertext uniquely define the values of the direct ciphertext-specific variables. The name of polynomials in Γ is due to the fact the the form of the corresponding elements in the ciphertext can be directly checked using elements corresponding to the fixing polynomials. In turn, the elements of the ciphertext corresponding to $\overline{\Gamma}$ require more involved checks due to mutual dependencies.

The following lemma is used in the proofs. It states two properties of the defined partition on $[w_1]$ for regular pair encodings.

Lemma 1.14. *Let* P *be a* regular *pair encoding scheme for relation family* $\mathcal{R}_{\Omega,\Sigma}$. *Let* $\kappa \in \Omega \times \Sigma$, $\mathrm{cInd} \in \mathbb{Y}_\kappa$ *be arbitrary but fixed,* $\left((c_1, \ldots, c_{w_1})^\top, w_2 \right) = \mathrm{Enc2}\left(\kappa, \mathrm{cInd}\right)$ *and*

$$\forall_{\tau \in [w_1]} : c_\tau = \sum_{i \in [w_2]_0} \left(b_{\tau,i} \cdot \mathrm{X}_{s_i} + \sum_{j \in [n]} b_{\tau,i,j} \cdot \mathrm{X}_{h_j} \cdot \mathrm{X}_{s_i} \right) \ .$$

Then, it holds

$$\forall_{\tau \notin \overline{\Gamma}} \forall_{i \in \overline{I}} : b_{\tau,i} = 0$$

and

$$\forall_{\tau \in [w_1]} \forall_{i \in \overline{I}} \forall_{j \in [n]} : b_{\tau,i,j} = 0 \ .$$

Proof. The first statement holds by the definition of Γ_0 and Γ. The second statement holds by the last property of regular pair encodings. □

1.3.4. Coefficients Properties of Pair Encodings

Due to the correctness of pair encoding, the coefficients in the encodings are highly related if $\mathrm{R}\left(\mathrm{kInd}, \mathrm{cInd}\right) = 1$. The security analysis of our framework in prime-order groups and especially the guarantees of the consistency checks require a careful look at these coefficients and their properties. Partially, these properties have been considered in the correctness proof of [Att16] (cf. Claim 15 in [Att15]).

Lemma 1.15. *Let* $\mathrm{P} = (\mathrm{Param}, \mathrm{Enc1}, \mathrm{Enc2}, \mathrm{Pair})$ *be a pair encoding scheme for relation family* $\mathcal{R}_{\Omega,\Sigma}$. *Let* $\kappa \in \Omega \times \Sigma$, $\mathrm{kInd} \in \mathbb{X}_\kappa$, *and* $\mathrm{cInd} \in \mathbb{Y}_\kappa$ *be arbitrary such that* $\mathrm{R}(\mathrm{kInd}, \mathrm{cInd}) = 1$. *Suppose* $\left((k_1, \ldots, k_{m_1})^\top, m_2\right) = \mathrm{Enc1}(\kappa, \mathrm{kInd})$,

$$\forall_{\tau \in [m_1]} : k_\tau = a_\tau \cdot \mathrm{X}_\alpha + \sum_{i \in [m_2]} \left(a_{\tau,i} \cdot \mathrm{X}_{r_i} + \sum_{j \in [n]} a_{\tau,i,j} \cdot \mathrm{X}_{h_j} \cdot \mathrm{X}_{r_i} \right) \ ,$$

and $\left((c_1, \ldots, c_{w_1})^\top, w_2\right) = \mathrm{Enc2}(\kappa, \mathrm{cInd})$,

$$\forall_{\tau' \in [w_1]} : c_{\tau'} = \sum_{i' \in [w_2]_0} \left(b_{\tau',i'} \cdot \mathrm{X}_{s_{i'}} + \sum_{j' \in [n]} b_{\tau',i',j'} \cdot \mathrm{X}_{h_{j'}} \cdot \mathrm{X}_{s_{i'}} \right) \ .$$

Then, for every $\boldsymbol{E} = \left(e_{\tau,\tau'}\right)_{\tau \in [m_1], \tau' \in [w_1]} \in [\mathrm{Pair}(\kappa, \mathrm{kInd}, \mathrm{cInd})]$ *it holds:*

$$\sum_{\tau \in [m_1]} \sum_{\tau' \in [w_1]} e_{\tau,\tau'} \cdot a_\tau \cdot b_{\tau',0} = 1 \ , \tag{1.6}$$

$$\forall_{i' \in [w_2]} : \quad \sum_{\tau \in [m_1]} \sum_{\tau' \in [w_1]} e_{\tau,\tau'} \cdot a_\tau \cdot b_{\tau',i'} = 0 \ , \tag{1.7}$$

$$\forall_{i' \in [w_2]_0} \forall_{j' \in [n]} : \quad \sum_{\tau \in [m_1]} \sum_{\tau' \in [w_1]} e_{\tau,\tau'} \cdot a_\tau \cdot b_{\tau',i',j'} = 0 \ , \tag{1.8}$$

$$\forall_{i \in [m_2]} \forall_{i' \in [w_2]_0} : \quad \sum_{\tau \in [m_1]} \sum_{\tau' \in [w_1]} e_{\tau,\tau'} \cdot a_{\tau,i} \cdot b_{\tau',i'} = 0 \ , \tag{1.9}$$

$$\forall_{i \in [m_2]} \forall_{i' \in [w_2]_0} \forall_{j \in [n]} : \quad \sum_{\tau \in [m_1]} \sum_{\tau' \in [w_1]} e_{\tau,\tau'} \cdot \left(a_{\tau,i} \cdot b_{\tau',i',j} + a_{\tau,i,j} \cdot b_{\tau',i'} \right) = 0 \ , \tag{1.10}$$

$$\forall_{i \in [m_2]} \forall_{i' \in [w_2]_0} \forall_{j \in [n]} : \quad \sum_{\tau \in [m_1]} \sum_{\tau' \in [w_1]} e_{\tau,\tau'} \cdot a_{\tau,i,j} \cdot b_{\tau',i',j} = 0 \ , \tag{1.11}$$

$$\begin{matrix} \forall_{i \in [m_2]} \forall_{i' \in [w_2]_0} \\ \forall_{j \in [n-1]} \forall_{j' \in [n], j' > j} \end{matrix} : \quad \sum_{\tau \in [m_1]} \sum_{\tau' \in [w_1]} e_{\tau,\tau'} \cdot \left(a_{\tau,i,j} \cdot b_{\tau',i',j'} + a_{\tau,i,j'} \cdot b_{\tau',i',j} \right) = 0 \ . \tag{1.12}$$

Furthermore, if P *is regular it holds for every* $\boldsymbol{E} = \left(e_{\tau,\tau'}\right)_{\tau \in [m_1], \tau' \in [w_1]} \in [\mathrm{Pair}(\kappa, \mathrm{kInd}, \mathrm{cInd})]$*:*

$$\forall_{\tau \in [m_1]} \forall_{\tau' \in [w_1]} \forall_{i \in [m_2]} \forall_{i' \in [w_2]_0} \forall_{j \in [n]} \forall_{j' \in [n]} : \quad e_{\tau,\tau'} \cdot a_{\tau,i,j} \cdot b_{\tau',i',j'} = 0 \ . \tag{1.13}$$

That is, every single summand in (1.11) and in (1.12) is equal to zero.

Proof. Let $\kappa \in \Omega \times \Sigma$, $\mathrm{kInd} \in \mathbb{X}_\kappa$, and $\mathrm{cInd} \in \mathbb{Y}_\kappa$ be arbitrary but fixed. Let $\mathrm{Enc1}(\kappa, \mathrm{kInd}) = \left((k_1, \ldots, k_{m_1})^\top, m_2\right)$, $\mathrm{Enc2}(\kappa, \mathrm{cInd}) = \left((c_1, \ldots, c_{w_1})^\top, w_2\right)$. Furthermore, let reconstruction matrix $\boldsymbol{E} \in \mathrm{Pair}(\kappa, \mathrm{kInd}, \mathrm{cInd})$ be arbitrary but fixed.

By correctness of P it holds:

$$\sum_{\tau \in [m_1]} \sum_{\tau' \in [w_1]} e_{\tau,\tau'} \cdot k_\tau \cdot c_{\tau'} = \mathrm{X}_\alpha \cdot \mathrm{X}_{s_0} \ .$$

Hence, it holds

$$\begin{aligned}
\mathrm{X}_\alpha \cdot \mathrm{X}_{s_0} &= \sum_{\tau\in[m_1]} \sum_{\tau'\in[w_1]} e_{\tau,\tau'} \cdot k_\tau \cdot c_{\tau'} \\
&= \sum_{\tau\in[m_1]} \sum_{\tau'\in[w_1]} e_{\tau,\tau'} \cdot \left(a_\tau \cdot \mathrm{X}_\alpha + \sum_{i\in[m_2]} \left(a_{\tau,i} \cdot \mathrm{X}_{r_i} + \sum_{j\in[n]} a_{\tau,i,j} \cdot \mathrm{X}_{h_j} \cdot \mathrm{X}_{r_i} \right) \right) \\
&\quad \cdot \sum_{i'\in[w_2]_0} \left(b_{\tau',i'} \cdot \mathrm{X}_{s_{i'}} + \sum_{j'\in[n]} b_{\tau',i',j'} \cdot \mathrm{X}_{h_{j'}} \cdot \mathrm{X}_{s_{i'}} \right) \\
&= \mathrm{X}_\alpha \cdot \mathrm{X}_{s_0} \cdot \sum_{\tau'\in[w_1]} \sum_{\tau\in[m_1]} e_{\tau,\tau'} \cdot a_\tau \cdot b_{\tau',0} \\
&\quad + \sum_{i'\in[w_2]} \mathrm{X}_\alpha \cdot \mathrm{X}_{s_{i'}} \cdot \sum_{\tau'\in[w_1]} \sum_{\tau\in[m_1]} e_{\tau,\tau'} \cdot a_\tau \cdot b_{\tau',i'} \\
&\quad + \sum_{i'\in[w_2]_0} \sum_{j'\in[n]} \mathrm{X}_\alpha \cdot \mathrm{X}_{h_{j'}} \cdot \mathrm{X}_{s_{i'}} \cdot \sum_{\tau\in[m_1]} \sum_{\tau'\in[w_1]} e_{\tau,\tau'} \cdot a_\tau \cdot b_{\tau',i',j'} \\
&\quad + \sum_{i\in[m_2]} \sum_{i'\in[w_2]_0} \mathrm{X}_{r_i} \cdot \mathrm{X}_{s_{i'}} \cdot \sum_{\tau\in[m_1]} \sum_{\tau'\in[w_1]} e_{\tau,\tau'} \cdot a_{\tau,i} \cdot b_{\tau',i'} \\
&\quad + \sum_{i\in[m_2]} \sum_{i'\in[w_2]_0} \sum_{j'\in[n]} \mathrm{X}_{r_i} \cdot \mathrm{X}_{h_{j'}} \cdot \mathrm{X}_{s_{i'}} \cdot \sum_{\tau\in[m_1]} \sum_{\tau'\in[w_1]} e_{\tau,\tau'} \cdot a_{\tau,i} \cdot b_{\tau',i',j'} \\
&\quad + \sum_{i\in[m_2]} \sum_{j\in[n]} \sum_{i'\in[w_2]_0} \mathrm{X}_{h_j} \cdot \mathrm{X}_{r_i} \cdot \mathrm{X}_{s_{i'}} \cdot \sum_{\tau\in[m_1]} \sum_{\tau'\in[w_1]} e_{\tau,\tau'} \cdot a_{\tau,i,j} \cdot b_{\tau',i'} \\
&\quad + \sum_{i\in[m_2]} \sum_{i'\in[w_2]_0} \sum_{j\in[n]} \sum_{j'\in[n]} \mathrm{X}_{r_i} \cdot \mathrm{X}_{s_{i'}} \cdot \mathrm{X}_{h_j} \cdot \mathrm{X}_{h_{j'}} \cdot \sum_{\tau\in[m_1]} \sum_{\tau'\in[w_1]} e_{\tau,\tau'} \cdot a_{\tau,i,j} \cdot b_{\tau',i',j'} \ .
\end{aligned}$$

The last three terms contain some monomials twice. We can transform these terms as follows in order to get the coefficients for all monomials:

$$\begin{aligned}
&\sum_{i\in[m_2]} \sum_{i'\in[w_2]_0} \sum_{j'\in[n]} \mathrm{X}_{r_i} \cdot \mathrm{X}_{h_{j'}} \cdot \mathrm{X}_{s_{i'}} \cdot \sum_{\tau\in[m_1]} \sum_{\tau'\in[w_1]} e_{\tau,\tau'} \cdot a_{\tau,i} \cdot b_{\tau',i',j'} \\
&\quad + \sum_{i\in[m_2]} \sum_{j\in[n]} \sum_{i'\in[w_2]_0} \mathrm{X}_{h_j} \cdot \mathrm{X}_{r_i} \cdot \mathrm{X}_{s_{i'}} \cdot \sum_{\tau\in[m_1]} \sum_{\tau'\in[w_1]} e_{\tau,\tau'} \cdot a_{\tau,i,j} \cdot b_{\tau',i'} \\
&= \sum_{i\in[m_2]} \sum_{i'\in[w_2]_0} \sum_{j\in[n]} \mathrm{X}_{r_i} \cdot \mathrm{X}_{h_j} \cdot \mathrm{X}_{s_{i'}} \cdot \sum_{\tau\in[m_1]} \sum_{\tau'\in[w_1]} e_{\tau,\tau'} \cdot \left(a_{\tau,i} \cdot b_{\tau',i',j} + a_{\tau,i,j} \cdot b_{\tau',i'} \right) \ ,
\end{aligned}$$

and

$$\begin{aligned}
&\sum_{i\in[m_2]} \sum_{i'\in[w_2]_0} \sum_{j\in[n]} \sum_{j'\in[n]} \mathrm{X}_{r_i} \cdot \mathrm{X}_{s_{i'}} \cdot \mathrm{X}_{h_j} \cdot \mathrm{X}_{h_{j'}} \cdot \sum_{\tau\in[m_1]} \sum_{\tau'\in[w_1]} e_{\tau,\tau'} \cdot a_{\tau,i,j} \cdot b_{\tau',i',j'} \\
&= \sum_{i\in[m_2]} \sum_{i'\in[w_2]_0} \sum_{j\in[n]} \mathrm{X}_{r_i} \cdot \mathrm{X}_{s_{i'}} \cdot \mathrm{X}_{h_j}^2 \cdot \sum_{\tau\in[m_1]} \sum_{\tau'\in[w_1]} e_{\tau,\tau'} \cdot a_{\tau,i,j} \cdot b_{\tau',i',j} \\
&\quad + \sum_{i\in[m_2]} \sum_{i'\in[w_2]_0} \sum_{j\in[n-1]} \sum_{j'\in[n],j'>j} \mathrm{X}_{r_i} \cdot \mathrm{X}_{s_{i'}} \cdot \mathrm{X}_{h_j} \cdot \mathrm{X}_{h_{j'}} \\
&\quad \cdot \sum_{\tau\in[m_1]} \sum_{\tau'\in[w_1]} e_{\tau,\tau'} \cdot \left(a_{\tau,i,j} \cdot b_{\tau',i',j'} + a_{\tau,i,j'} \cdot b_{\tau',i',j} \right) \ .
\end{aligned}$$

Correctness of pair encoding schemes guaranties that all monomials except $\mathrm{X}_\alpha \cdot \mathrm{X}_{s_0}$ disappear. We deduce that it holds:

$$\mathrm{X}_\alpha \cdot \mathrm{X}_{s_0} : \quad \sum_{\tau'\in[w_1]} \sum_{\tau\in[m_1]} e_{\tau,\tau'} \cdot a_\tau \cdot b_{\tau',0} = 1 \ , \tag{1.6}$$

$$\mathrm{X}_{\alpha} \cdot \mathrm{X}_{s_{i'}} : \quad \forall_{i' \in [w_2]} : \sum_{\tau' \in [w_1]} \sum_{\tau \in [m_1]} e_{\tau,\tau'} \cdot a_{\tau} \cdot b_{\tau',i'} = 0 \ , \tag{1.7}$$

$$\mathrm{X}_{\alpha} \cdot \mathrm{X}_{h_{j'}} \cdot \mathrm{X}_{s_{i'}} : \quad \forall_{i' \in [w_2]_0} \forall_{j' \in [n]} : \sum_{\tau \in [m_1]} \sum_{\tau' \in [w_1]} e_{\tau,\tau'} \cdot a_{\tau} \cdot b_{\tau',i',j'} = 0 \ , \tag{1.8}$$

$$\mathrm{X}_{r_i} \cdot \mathrm{X}_{s_{i'}} : \quad \forall_{i \in [m_2]} \forall_{i' \in [w_2]_0} : \sum_{\tau \in [m_1]} \sum_{\tau' \in [w_1]} e_{\tau,\tau'} \cdot a_{\tau,i} \cdot b_{\tau',i'} = 0 \ , \tag{1.9}$$

$$\mathrm{X}_{r_i} \cdot \mathrm{X}_{h_j} \cdot \mathrm{X}_{s_{i'}} : \quad \forall_{i \in [m_2]} \forall_{i' \in [w_2]_0} \forall_{j \in [n]} : \sum_{\tau \in [m_1]} \sum_{\tau' \in [w_1]} e_{\tau,\tau'} \cdot \left(a_{\tau,i} \cdot b_{\tau',i',j} + a_{\tau,i,j} \cdot b_{\tau',i'} \right) = 0 \ , \tag{1.10}$$

$$\mathrm{X}_{r_i} \cdot \mathrm{X}_{s_{i'}} \cdot \mathrm{X}_{h_j}^2 : \quad \forall_{i \in [m_2]} \forall_{i' \in [w_2]_0} \forall_{j \in [n]} : \sum_{\tau \in [m_1]} \sum_{\tau' \in [w_1]} e_{\tau,\tau'} \cdot a_{\tau,i,j} \cdot b_{\tau',i',j} = 0 \ , \tag{1.11}$$

$$\mathrm{X}_{r_i} \cdot \mathrm{X}_{s_{i'}} \cdot \mathrm{X}_{h_j} \cdot \mathrm{X}_{h_{j'}} : \quad \forall_{i \in [m_2]} \forall_{i' \in [w_2]_0} \forall_{j \in [n-1]} \forall_{j' \in [n], j' > j} : \sum_{\tau \in [m_1]} \sum_{\tau' \in [w_1]} e_{\tau,\tau'} \cdot \left(a_{\tau,i,j} \cdot b_{\tau',i',j'} + a_{\tau,i,j'} \cdot b_{\tau',i',j} \right) = 0 \ . \tag{1.12}$$

Thus, the equation from the first part of the lemma are satisfied.

Now consider the case of regular pair encodings. The second property of regular pair encodings states that for every $\tau \in [m_1]$, $\tau' \in [w_1]$, if there exists i, j, i', j' such that $a_{\tau,i,j} \neq 0$ and $b_{\tau',i',j'} \neq 0$ then $e_{\tau,\tau'}$ must be equal to zero. Hence, it holds that

$$\forall_{i \in [m_2]} \forall_{i' \in [w_2]_0} \forall_{j \in [n]} \forall_{j' \in [n]} \forall_{\tau \in [m_1]} \forall_{\tau' \in [w_1]} : e_{\tau,\tau'} \cdot a_{\tau,i,j} \cdot b_{\tau',i',j'} = 0 \ , \tag{1.13}$$

since $a_{\tau,i,j} \cdot b_{\tau',i',j'} \neq 0$ implies $e_{\tau,\tau'} = 0$. This proves the stated property for coefficient of regular pair encoding. □

1.3.5. Security Notions for Pair Encoding Schemes

We prove the security of our frameworks based on the computational security notions of pair encoding schemes presented in [Att14a]. These security properties are called selectively master-key hiding (SMH) and co-selectively master-key hiding (CMH). We claim that similar to the underlying CPA-secure frameworks [Att14a, Att16] our proofs can be modified in order to deal with information-theoretical security notions. Notice that there are only few pair encodings which satisfy this stronger security notion.

Computational Security Notions of Pair Encodings in Composite-Order Groups

In this subsection we recall the computational security notions of pair encoding schemes in composite-order groups presented in [Att14a]. We formally consider the composite-order groups and the corresponding group generators in Subsection 4.1.1. For now, let $\mathcal{G}$ be a ppt algorithm, which on input 1^{λ} outputs description of bilinear groups

$$\mathbb{GD} = \left(p_1, p_2, p_3, (g, \mathbb{G}), \mathbb{G}_{\mathrm{T}}, \mathrm{e} \right) \ ,$$

where p_1, p_2, p_3 are distinct primes of length λ, $\mathbb{G}$ and $\mathbb{G}_{\mathrm{T}}$ are cyclic groups of order $N = p_1 p_2 p_3$, g is a generator of $\mathbb{G}$, and function $\mathrm{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_{\mathrm{T}}$ is a non-degenerate bilinear map: i.e., $\mathrm{e}(g, g)$ is a generator of $\mathbb{G}_{\mathrm{T}}$ and $\mathrm{e}\left(g^a, g^b\right) = \mathrm{e}(g, g)^{ab}$ for all $a, b \in \mathbb{Z}_N$. For all $i \in \{1, 2, 3\}$ we denote by $\mathbb{G}_{p_i}$ the unique subgroup of $\mathbb{G}$ of order p_i. We denote by $\mathbb{GD}_N$ a restricted description of bilinear groups corresponding to $\mathbb{GD}$, where the prime numbers are replaced by their product N.

First, define the following generic probability experiment which is parametrized with adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, Type $\in \{\mathrm{SMH}, \mathrm{CMH}\}$, and $\nu \in \{0, 1\}$.

$\mathbf{Exp}^{\mathrm{Type}}_{\mathrm{P},\mathcal{G},\nu,\mathcal{A}}(\lambda,\mathrm{des})$:

$\mathbb{GD} = \left(p_1,p_2,p_3,(g,\mathbb{G}),\mathbb{G}_{\mathrm{T}},\mathrm{e}\right) \leftarrow \mathcal{G}\left(1^{\lambda}\right)$;

$N := p_1p_2p_3, g_1 \leftarrow \mathbb{G}_{p_1}, g_2 \leftarrow \mathbb{G}_{p_2}, g_3 \leftarrow \mathbb{G}_{p_3}$;

$\kappa := (\mathrm{des}, N), n := \mathrm{Param}(\kappa), \hat{\alpha} \leftarrow \mathbb{Z}_N, \hat{\boldsymbol{h}} \leftarrow \mathbb{Z}_N^n$;

$St \leftarrow \mathcal{A}_1^{\mathcal{O}^1_{\mathrm{Type},\nu,\hat{\alpha},\hat{\boldsymbol{h}}}(\cdot)}(\mathrm{des},\mathbb{GD}_N,g_1,g_2,g_3)$;

$\nu' \leftarrow \mathcal{A}_2^{\mathcal{O}^2_{\mathrm{Type},\nu,\hat{\alpha},\hat{\boldsymbol{h}}}(\cdot)}(St)$;

Output $\nu' = \nu$;

Figure 1.1.: Generic experiment in composite-order groups.

On the basis of this generic experiment we define two security experiments in Fig. 1.2 and in Fig. 1.3 and the corresponding security notions of pair encoding schemes.

The advantage of $\mathcal{A}$ in the SMH-experiment is defined as:

$$\mathrm{Adv}^{\mathrm{SMH}}_{\mathrm{P},\mathcal{A}}(\lambda,\mathrm{des}) := \left|\Pr\left[\mathrm{Exp}^{\mathrm{SMH}}_{\mathrm{P},\mathcal{G},0,\mathcal{A}}(\lambda,\mathrm{des})=1\right] - \Pr\left[\mathrm{Exp}^{\mathrm{SMH}}_{\mathrm{P},\mathcal{G},1,\mathcal{A}}(\lambda,\mathrm{des})=1\right]\right| .$$

Definition 1.16. Pair encoding P is called **selectively master key hiding** with respect to $\mathcal{G}$ if for all $\mathrm{des} \in \Omega$ and all ppt adversaries $\mathcal{A}$, the function $\mathrm{Adv}^{\mathrm{SMH}}_{\mathrm{P},\mathcal{A}}(\lambda,\mathrm{des})$ is negligible.

$\mathbf{Exp}^{\mathrm{SMH}}_{\mathrm{P},\mathcal{G},\nu,\mathcal{A}}(\lambda,\mathrm{des})$ is an instantiation of $\mathrm{Exp}^{\mathrm{Type}}_{\mathrm{P},\mathcal{G},\nu,\mathcal{A}}(\lambda,\mathrm{des})$ with:

$\mathcal{O}^1_{\mathrm{SMH},\nu,\hat{\alpha},\hat{\boldsymbol{h}}}(\mathrm{cInd}^*)$ for $\mathrm{cInd}^* \in \mathbb{Y}_\kappa$: Can be queried only once. Compute encoding $(\boldsymbol{c}, w_2) := \mathrm{Enc2}(\kappa,\mathrm{cInd}^*)$, pick $\hat{s} \leftarrow \mathbb{Z}_N$, $\hat{\boldsymbol{s}} \leftarrow \mathbb{Z}_N^{w_2}$ and return $\widehat{\boldsymbol{C}} := g_2^{\boldsymbol{c}\left(\hat{s},\hat{\boldsymbol{s}},\hat{\boldsymbol{h}}\right)}$.

$\mathcal{O}^2_{\mathrm{SMH},\nu,\hat{\alpha},\hat{\boldsymbol{h}}}(\mathrm{kInd})$ for $\mathrm{kInd} \in \mathbb{X}_\kappa$: Can be queried polynomially many times. Return $\perp$ if $\mathrm{R}_{\mathrm{des},p_2}(f_1(\mathrm{kInd}), f_2(\mathrm{cInd}^*)) = 1$. Otherwise, compute $(\boldsymbol{k}, m_2) := \mathrm{Enc1}(\kappa,\mathrm{kInd})$, pick $\hat{\boldsymbol{r}} \leftarrow \mathbb{Z}_N^{m_2}$ and return $\widehat{\boldsymbol{K}} := g_2^{\boldsymbol{k}\left(0,\hat{\boldsymbol{r}},\hat{\boldsymbol{h}}\right)}$ if $\nu = 0$ and $\widehat{\boldsymbol{K}} := g_2^{\boldsymbol{k}\left(\hat{\alpha},\hat{\boldsymbol{r}},\hat{\boldsymbol{h}}\right)}$ if $\nu = 1$.

Figure 1.2.: SMH experiment in composite-order groups.

The advantage of $\mathcal{A}$ in CMH-experiment is defined as:

$$\mathrm{Adv}^{\mathrm{CMH}}_{\mathrm{P},\mathcal{A}}(\lambda,\mathrm{des}) := \left|\Pr\left[\mathrm{Exp}^{\mathrm{CMH}}_{\mathrm{P},\mathcal{G},0,\mathcal{A}}(\lambda,\mathrm{des})=1\right] - \Pr\left[\mathrm{Exp}^{\mathrm{CMH}}_{\mathrm{P},\mathcal{G},1,\mathcal{A}}(\lambda,\mathrm{des})=1\right]\right| .$$

Definition 1.17. Pair encoding P is called **co-selectively master key hiding** with respect to $\mathcal{G}$ if for all $\mathrm{des} \in \Omega$, and all ppt adversaries $\mathcal{A}$, the function $\mathrm{Adv}^{\mathrm{CMH}}_{\mathrm{P},\mathcal{A}}(\lambda,\mathrm{des})$ is negligible.

$\mathbf{Exp}^{\mathrm{CMH}}_{\mathrm{P},\mathcal{G},\nu,\mathcal{A}}(\lambda, \mathrm{des})$ is an instantiation of $\mathrm{Exp}^{\mathrm{Type}}_{\mathrm{P},\mathcal{G},\nu,\mathcal{A}}(\lambda, \mathrm{des})$ with:

$\mathcal{O}^1_{\mathrm{CMH},\nu,\hat{\alpha},\hat{\boldsymbol{h}}}(\mathrm{kInd})$ for $\mathrm{kInd} \in \mathbb{X}_\kappa$: Can be queried only once. $\mathcal{C}$ computes $(\boldsymbol{k}, m_2) := \mathrm{Enc1}(\kappa, \mathrm{kInd})$, $\hat{\boldsymbol{r}} \leftarrow \mathbb{Z}^{m_2}_{p_2}$ and returns $\widehat{\boldsymbol{K}} := g_2^{\boldsymbol{k}(0,\hat{\boldsymbol{r}},\hat{\boldsymbol{h}})}$ if $\nu = 0$ and $\widehat{\boldsymbol{K}} := g_2^{\boldsymbol{k}(\hat{\alpha},\hat{\boldsymbol{r}},\hat{\boldsymbol{h}})}$ if $\nu = 1$.

$\mathcal{O}^2_{\mathrm{CMH},\nu,\hat{\alpha},\hat{\boldsymbol{h}}}(\mathrm{cInd}^*)$ for $\mathrm{cInd}^* \in \mathbb{Y}_\kappa$: The oracle can be queried only once. Challenger $\mathcal{C}$ returns $\perp$ if $\mathrm{R}_{\mathrm{des},p_2}(f_1(\mathrm{kInd}), f_2(\mathrm{cInd}^*)) = 1$. Otherwise, $\mathcal{C}$ computes $(\boldsymbol{c}, w_2) := \mathrm{Enc2}(\kappa, \mathrm{cInd}^*)$, picks $\hat{s} \leftarrow \mathbb{Z}_N$ and $\hat{\boldsymbol{s}} \leftarrow \mathbb{Z}^{w_2}_N$ and returns $\widehat{\boldsymbol{C}} := g_2^{\boldsymbol{c}(\hat{s},\hat{\boldsymbol{s}},\hat{\boldsymbol{h}})}$.

Figure 1.3.: CMH experiment in composite-order groups.

Computational Security Notions of Pair Encodings in Prime-Order Groups

In this subsection we present the computational security notion for pair encodings in prime-order groups. Actually, the modifications are pure syntactical as mentioned in [Att16]. Namely, if we look at security definitions in the previous paragraph, we recognize that all computations are performed in the $\mathbb{G}_{p_2}$ subgroup – that is, in a prime-order group. Furthermore, one does not require that the factorization of N is hard to compute. The generators of all subgroups are known and as mentioned in [Att16] one could also give $\mathcal{A}$ the description $\mathbb{GD}$ – that is, the factorization of N. For the sake of completeness we present the definitions also for prime-order groups (see also [Att15]). First, we define a generic probability experiment in Fig. 1.4. which is parametrized with adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, $\mathrm{Type} \in \{\mathrm{SMH}, \mathrm{CMH}\}$, and $\nu \in \{0, 1\}$. The bilinear map e in this context will be asymmetric, that is $\mathrm{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_\mathrm{T}$ with $\mathbb{G}_1 \neq \mathbb{G}_2$.

$\mathbf{Exp}^{\mathrm{Type}}_{\mathrm{P},\mathcal{G},\nu,\mathcal{A}}(\lambda, \mathrm{des})$:

$\mathbb{GD} = (p, (g_1, \mathbb{G}_1), (g_2, \mathbb{G}_2), \mathbb{G}_\mathrm{T}, \mathrm{e}) \leftarrow \mathcal{G}\left(1^\lambda\right)$;

$\kappa := (\mathrm{des}, p), n := \mathrm{Param}(\kappa); \hat{\alpha} \leftarrow \mathbb{Z}_p, \hat{\boldsymbol{h}} \leftarrow \mathbb{Z}^n_p$;

$St \leftarrow \mathcal{A}_1^{\mathcal{O}^1_{\mathrm{Type},\nu,\hat{\alpha},\hat{\boldsymbol{h}}}(\cdot)}(\mathrm{des}, \mathbb{GD})$;

$\nu' \leftarrow \mathcal{A}_2^{\mathcal{O}^2_{\mathrm{Type},\nu,\hat{\alpha},\hat{\boldsymbol{h}}}(\cdot)}(St)$;

Output $\nu' = \nu$;

Figure 1.4.: Generic experiment in prime-order groups.

The SMH and the CMH experiments are presented in Fig. 1.5. The security definitions based on these experiments are as in the previous paragraph.

$\mathbf{Exp}^{\mathrm{SMH}}_{\mathrm{P},\mathcal{G},\nu,\mathcal{A}}(\lambda, \mathrm{des})$ is an instantiation of $\mathbf{Exp}^{\mathrm{Type}}_{\mathrm{P},\mathcal{G},\nu,\mathcal{A}}(\lambda, \mathrm{des})$ with the following oracles:

- $\mathcal{O}^1_{\mathrm{SMH},\nu,\hat{\alpha},\hat{\boldsymbol{h}}}(\mathrm{cInd}^*)$ for $\mathrm{cInd}^* \in \mathbb{Y}_\kappa$: Can be queried only once. Compute $(\boldsymbol{c}, w_2) := \mathrm{Enc2}(\kappa, \mathrm{cInd}^*)$, pick $\hat{s} \leftarrow \mathbb{Z}_p$, $\hat{\boldsymbol{s}} \leftarrow \mathbb{Z}_p^{w_2}$ and return $\widehat{\boldsymbol{C}} := g_1^{\boldsymbol{c}(\hat{s},\hat{\boldsymbol{s}},\hat{\boldsymbol{h}})}$.
- $\mathcal{O}^2_{\mathrm{SMH},\nu,\hat{\alpha},\hat{\boldsymbol{h}}}(\mathrm{kInd})$ for $\mathrm{kInd} \in \mathbb{X}_\kappa$: Can be queried polynomially many times. Return $\perp$ if $\mathrm{R}_\kappa(\mathrm{kInd}, \mathrm{cInd}^*) = 1$. Otherwise, compute $(\boldsymbol{k}, m_2) := \mathrm{Enc1}(\kappa, \mathrm{kInd})$, pick $\hat{\boldsymbol{r}} \leftarrow \mathbb{Z}_p^{m_2}$ and return $\widehat{\boldsymbol{K}} := g_2^{\boldsymbol{k}(0,\hat{\boldsymbol{r}},\hat{\boldsymbol{h}})}$ if $\nu = 0$ and $\widehat{\boldsymbol{K}} := g_2^{\boldsymbol{k}(\hat{\alpha},\hat{\boldsymbol{r}},\hat{\boldsymbol{h}})}$ if $\nu = 1$.

$\mathbf{Exp}^{\mathrm{CMH}}_{\mathrm{P},\mathcal{G},\nu,\mathcal{A}}(\lambda, \mathrm{des})$ is an instantiation of $\mathbf{Exp}^{\mathrm{Type}}_{\mathrm{P},\mathcal{G},\nu,\mathcal{A}}(\lambda, \mathrm{des})$ with the following oracles:

- $\mathcal{O}^1_{\mathrm{CMH},\nu,\hat{\alpha},\hat{\boldsymbol{h}}}(\mathrm{kInd})$ for $\mathrm{kInd} \in \mathbb{X}_\kappa$: Can be queried only once. Compute $(\boldsymbol{k}, m_2) := \mathrm{Enc1}(\kappa, \mathrm{kInd})$, pick $\hat{\boldsymbol{r}} \leftarrow \mathbb{Z}_p^{m_2}$ and return $\widehat{\boldsymbol{K}} := g_2^{\boldsymbol{k}(0,\hat{\boldsymbol{r}},\hat{\boldsymbol{h}})}$ if $\nu = 0$ and $\widehat{\boldsymbol{K}} := g_2^{\boldsymbol{k}(\hat{\alpha},\hat{\boldsymbol{r}},\hat{\boldsymbol{h}})}$ if $\nu = 1$.
- $\mathcal{O}^2_{\mathrm{CMH},\nu,\hat{\alpha},\hat{\boldsymbol{h}}}(\mathrm{cInd}^*)$ for $\mathrm{cInd}^* \in \mathbb{Y}_\kappa$: The oracle can be queried only once. Return $\perp$ if $\mathrm{R}_\kappa(\mathrm{kInd}, \mathrm{cInd}^*) = 1$. Otherwise, compute $(\boldsymbol{c}, w_2) := \mathrm{Enc2}(\kappa, \mathrm{cInd}^*)$, pick $\hat{s} \leftarrow \mathbb{Z}_p$ and $\hat{\boldsymbol{s}} \leftarrow \mathbb{Z}_p^{w_2}$ and returns $\widehat{\boldsymbol{C}} := g_1^{\boldsymbol{c}(\hat{s},\hat{\boldsymbol{s}},\hat{\boldsymbol{h}})}$.

Figure 1.5.: SMH and CMH experiments in prime-order groups.

Part I.

Security Definitions for Predicate-Based Schemes

Introduction, Related Work, and Main Contribution

Cryptographic primitives and schemes considered in modern cryptography have become increasingly more complex over the last decades. Clearly, one has to abstract from many details when novel constructions or techniques are presented. Therefore, it is all the more important that security models are rigorously studied and made accessible to the cryptographic community through comprehensive and perspicuous explanation, especially when these models are translated into novel contexts. In a recent work [KM13] the authors showed that well-known security definitions for digital signatures and for symmetric-key encryption do not provide a comprehensive model of adversarial behavior and fail to take into account many attack scenarios. Furthermore, in [BHK15] the authors analyzed several security definitions for conventional public-key encryption (PKE) under chosen-ciphertext attacks, which were assumed to lead to the same security notion. They proved that not all these definitions are equivalent. These two articles are just examples which show that security requirements need formal treatment and careful modeling. If even these well-established security definitions suffer from weaknesses, what about newly and more involved cryptographic schemes and their security models which are often justified by their origin in *well-known* definitions?

In the first part of this thesis we look at security definitions for predicate encryption (PE) with public index and predicate key encapsulation mechanism (P-KEM) with public index. On the one hand, these cryptographic primitives have been extensively studied due to their usefulness in various cryptographic applications [BF03, SW05, GPSW06, OSW07, Wat09a, LOS$^+$10, BSW11, Wat11, Wat12], to name just a few fundamental articles in this context. On the other hand, established security definitions for PE and P-KEM lean, to a large extent, on the corresponding security definitions in the context of PKE and conventional KEM, even though the functionality of predicate-based schemes is more complex than the functionality of conventional public-key schemes. We remark that many known predicate encryption schemes are proved to be selectively secure rather then fully (or adaptively) secure, e.g. [SW05, GPSW06, OSW07, Wat11, RW13]. The selective security model is a weak theoretical security model in which the adversary commits to the target of the attack before the public parameters are generated. This model was often used when predicate encryption schemes for novel predicates were developed. We remark that the selectively secure schemes can also be used to achieve further cryptographic primitives with strong security properties, e.g. [BCHK07]. In this thesis we only consider the adaptive security model, since our aim is to achieve fully secure predicate encryption schemes. We analyze the security definitions for PE and P-KEM, formalize the assumed adversarial behavior and identify several important subtleties in the previously considered definitions. As a result, we propose well-grounded security definitions for these cryptographic primitives under different attack scenarios.

We recall that the study of PE actually started when Shamir introduced the idea of identity-based encryption (IBE) [Sha84], a predicate encryption for the equality predicate. The first fully-functional IBE schemes were presented in [BF03, Coc01], whereas the study of PE for more sophisticated predicates started with [SW05]. Furthermore, the more general concept of functional encryption (FE) was introduced in [BSW11]. In contrast to conventional PKE, in PE schemes all user secret keys are generated from a single master secret key by a trusted authority. The authority is responsible for key generation; hence, it manages access to all encrypted data. Nevertheless, if we look at the security models previously used in the context of PE, we recognize

that they originate from the security models for IBE (cf. [SW05]), which in turn go back to the security models for PKE (cf. [BF03]). An intuitive informal security requirement for encryption schemes is that the ciphertext should not reveal any information about the encrypted message to anyone who is not allowed to get access to the message. Semantic security (SS) is an intuitive simulation-based formalization of this requirement, whereas indistinguishability (IND) of encryptions is an alternative definition. One evidence of a *good* security definition is its robustness under slightly adapted variations and under reasonable attack scenarios. In fact, in the case of PKE, a lot of different extensions and variants of semantic security definitions have been proved to be equivalent [Gol04b]. In particular, the involved definition of so-called adaptive multiple-challenge semantic security was shown to be equivalent to the simple single-challenge semantic security definition and to the indistinguishability of encryptions. These equivalences hold under (passive) chosen-plaintext attacks (CPA) and (active) chosen-ciphertext attacks (CCA). Among other results, this gave strong evidence that the SS-definition is the right formalization of the required security properties for PKE. In turn, the equivalent IND-definition is easier to handle, and hence it is widely used in security proofs for PKE schemes.

Starting with [BF03] the indistinguishability definition for PKE was adapted and used for IBE [Wat05, KG09, KV08, Wat09a, LW10] and for more involved PE [SW05, AI09, OT10a, YAHK11, YAS+12, LW12, Att14a, BL16b]. Consequently, in [ACG+06] security models for IBE were studied under different attack scenarios. One of the results of this work is an equivalence proof of the SS-definition and the IND-definition for IBE. Similarly to PKE, this result holds under CPA and under CCA even in the adaptive multiple-challenge scenario. On the contrary, for the more general context of functional encryption, the IND-definition was proved to be unsuitable [BSW11, O'N10]. In [BSW11] the authors also proved that semantic security cannot even be achieved for IBE. This result seemingly contradicts previous work since IND-definition and SS-definition are equivalent for IBE [ACG+06] and many IND-secure IBE schemes are known e.g. [Wat05, KG09]. The impossibility result of [BSW11] was identified in [BO13, BF13] as a consequence of the so-called key-revealing selective-opening attacks (SOA-Ks) which were implicitly covered by the SS-definitions of [O'N10, BSW11], but were not considered in [ACG+06]. We additionally remark that the analysis of [O'N10, BSW11, BO13, BF13] was restricted to the CPA attack scenario.

At first, it looks like the semantic security and the indistinguishability of encryptions for PE are well studied and well understood. Indeed, the results for FE under CPA can be applied to PE and the results for IBE cover CCA-security and can be extended to PE. However, in [BHK15] the authors already mentioned that the identified issues in the CCA-security definitions for PKE regarding the so-called no-challenge-decryption condition also appear in the definitions for IBE. Furthermore, in comparison to the previously used definitions, in [BL13, BL14, BL16b] we utilized a more sophisticated indistinguishability definition under chosen-ciphertext attacks and justified this by specific properties of PE schemes. These intricacies motivated us to reconsider security definitions in the context of predicate-based schemes.

Main Contribution

How to handle user secret keys? In the context of conventional PKE there is only a single secret key in question, whereas in PE schemes there are many user secret keys generated from the master secret key by a key generation algorithm. Actually, several users may hold different keys for the same key index, which has to be treated by the corresponding security model. Security definitions for IBE in [BF03] already explicitly prevent user collusion and formalize this by an additional key generation oracle. However, the IBE schemes in [BF03] have a very special property. Namely, the key generation algorithm is deterministic, and hence there is a unique user secret key for every identity. Hence, the existence of different user secret keys for the same key index was not considered in [BF03], neither under CPA nor under CCA. Furthermore, we remark that in the context of IBE key indices specify user's identities and

therefore, the uniqueness of user secret keys can be assumed w.l.o.g. for most applications even if the actual key generation algorithm is not deterministic. In PE schemes for sophisticated predicates, key indices are more complex and specify the access rights of the users. Hence, different user secret keys for the same key index exist and the assumption from above cannot be made for PE schemes in general. We identify three different formalizations regarding handling of user secret keys in security definitions for PE schemes and name these as follows:

`One-Key model` (OK-model)

`One-Use model` (OU-model)

`Covered-Key model` (CK-model)

In the first two models, the adversary (denoted by $\mathcal{A}$) has access to oracles $\mathbf{KGen}(\cdot)$ and $\mathbf{Dec}(\cdot,\cdot)$ under chosen-ciphertext attacks. For the first oracle, $\mathcal{A}$ specifies a key index and receives a secret key for this key index. For the second oracle, $\mathcal{A}$ specifies a ciphertext as well as a key index, and the ciphertext is decrypted using a secret key for the specified key index. The OK-model and the OU-model differ in how they handle the user secret keys in these oracles. In the OK-model, a unique secret key for kInd is generated and stored if this index is submitted by $\mathcal{A}$ for the first time. This user secret key is used to answer all oracle queries related to kInd. In particular, oracle query $\mathbf{KGen}(\text{kInd})$ always results in the same key. In turn, in the OU-model the challenger generates a new secret key for every key generation query and for every decryption query. Hence, every generated user secret key is used only once by the oracles. Under CCA the OK-model has previously been used, e.g. in [BF03, KG09, YAHK11, YAS$^+$12], and the OU-model has previously been used, e.g. in [BH08, KV08, OT10a]. We have already used the CK-model in early works on selectively CCA-secure PE schemes in [BL13, BL14] as well as in [BL16b]. In this model, the user secret keys are generated and numbered in the so-called covered key generation oracle $\mathbf{CKGen}(\cdot)$. The adversary can ask to reveal the generated keys (using an additional $\mathbf{Open}(\cdot)$ oracle), which realizes the functionality of the original key generation oracle, but now the adversary can also make more specific decryption queries. For such a query, $\mathcal{A}$ specifies the number of the secret key which has to be used for decryption. This models the fact that different keys, even for the same key index, are held and used by different users and an adversary might be able to realize a chosen-ciphertext attack on these users and their secret keys.

The three identified models have previously been used and most researchers seem to think they refer to the same security notion. In [BCHK07] the authors shortly discuss the OK-model and the OU-model for the case that the key generation algorithm in (H)IBE is probabilistic and state that "*The resulting security definitions [...] seem incomparable, and there does not appear to be a reason to prefer one over the other.*" [BCHK07, page 1309] As a consequence, the authors assume that the key generation algorithm is deterministic, which is a plausible assumption in the context of IBE but is not necessarily appropriate in the more general setting of predicate encryption schemes, as already discussed. We prove that, in general, under CPA the OK-model is weaker than the other two models, whereas the CK-model and the OU-model are equivalent. Under CCA we show that all three resulting security notions are different and that the security notion achieved from the CK-model is the strongest one. Both results hold for PE as well as for P-KEM. Hence, we examine the CK-security of known predicate-based schemes which were proved to satisfy the weaker security notions under CCA. Obviously, the CCA-secure scheme with unique secret keys from [BF03] is CK-secure. For other schemes this is not obvious and we have to look at their security proofs in detail. Interestingly, for many known schemes [BF03, BH08, KG09, KV08, YAHK11, YAS$^+$12] we can argue that they also achieve CK-security, but the arguments differ for every single scheme. For some schemes the question regarding CK-security remains open [OT10a, YAHK11].

When and how should challenge decryption queries be disallowed? Independently of the actual security definition, the goal of adversaries in the context of encryption schemes is to learn information about the encrypted message given a so-called challenge ciphertext. Hence, in order to achieve a meaningful security notion under chosen-ciphertext attacks, one has to require that the adversaries do not ask for decryption of the challenge ciphertext. This is the so-called no-challenge-decryption condition mentioned in the introduction. Different formalizations of this condition have previously been used in the context of PKE and KEM. Hence, we consider meaningful security models regarding this condition in the context of PE and P-KEM, as well. While it is not surprising that the results for PE are similar to the previous results for PKE [BHK15] (the authors already mentioned this for IBE), the situation is different when key encapsulation mechanisms are considered. Namely, in the context of conventional KEM six different security notions were identified and proved to be equivalent in [BHK15]. We consider two additional security notions (due to the additional key generation oracle) and prove that four of the eight resulting security definitions for P-KEM are too weak in general. The other four notions are in fact equivalent, but some reductions between these notions are not tight. Based on the mentioned results, we conclude that in the context of P-KEM the no-challenge-decryption condition should not be used in the first query phase. In contrast to this result, the first query phase can be completely dropped for conventional KEM [BHK15].

Reasonable restrictions of adversaries in SS-definitions and IND-definitions. We present an appropriate semantic security definition for PE and appropriate indistinguishability definitions for PE and P-KEM. On the one hand, the analysis of our previously mentioned results is based on these definitions. On the other hand, we justify the additional restrictions of adversaries previously used in all security definitions for PE. Namely, for meaningful security notion we have to require that adversaries do not query a user secret key neither in the first query phase nor in the second query phase if this key can be used to decrypt the challenge ciphertext. To the best of our knowledge, in all previously used security definitions for predicate-based schemes this restriction was modeled in exclusion-style – that is, adversaries which violate this restriction are not even considered. However, in security definitions for PE and P-KEM the challenge ciphertext index is not specified in the first query phase and one cannot decide if a key index, submitted in this phase by an adversary, matches this challenge ciphertext index. As observed in [BHK15] for the no-challenge-decryption condition, exclusion-style adversarial restrictions regarding the first query phase are counterintuitive and may have unpredictable consequences. We show that with slightly modified security definitions these issues do not arise with restrictions on key generation queries in the first query phase.

We prove in addition that our SS-definition and IND-definition for PE are equivalent under different attack scenarios including chosen-ciphertext attacks. As already mentioned above, this was not explicitly covered by previous results for IBE and FE. Finally, on the basis of our SS-definition for PE we shortly discuss the impossibility results known in the context of functional encryption.

Organization In Chapter 2 we first present definitional templates for the semantic security definitions and for the indistinguishability definitions. In these templates we abstract from concrete attack scenarios and oracle modeling. Based on these templates we prove the equivalence of SS-definition and IND-definition for PE under chosen-ciphertext attacks in Section 2.2. In Chapter 3 we consider the mentioned subtleties based on the indistinguishability definitions.

We remark that the results presented in this part will be published by the 7th International Conference on Mathematical Aspects of Computer and Information Sciences (MACIS 2017) and so far are published in Cryptology ePrint Archive [BL17].

2. Formalization of Security Properties

Section 2.1 contains the SS-definition and IND-definition for PE as well as the indistinguishability security definition for P-KEM. We analyze the relation between the different security notions and formally treat adversarial restrictions. In this chapter we assume w.l.o.g. that the message space of the encryption schemes is $\mathcal{M} = \{0,1\}^*$. Similarly, for key encapsulation mechanisms with families of key spaces $\mathcal{K} = \{\mathbb{K}_\lambda\}_{\lambda\in\mathbb{N}}$ we always assume that for every $\lambda \in \mathbb{N}$, it holds $\mathbb{K}_\lambda = \{0,1\}^{l(\lambda)}$ for some polynomial $l(\cdot)$.

2.1. Semantic Security and Indistinguishability for Predicate-Based Schemes

In this section we first define two formal security notions or rather definitional templates for predicate encryption: a semantic security (SS) template and an indistinguishability (IND) template. We explicitly notice that the definitions are not novel but, as mentioned in the introduction, there are important subtleties in these definitions in the context of PE which must be considered. The templates prescind from some details and prepare the formal treatment of adversarial behavior and some further definitional aspects considered in the following sections. Based on our templates we compare the SS-security notion and the IND-security notion in the context of PE independently of the concrete attack scenario. In Subsection 2.1.2 we also present an appropriate indistinguishability template for predicate key encapsulation mechanisms. Our formalization originates from the corresponding definitions for PKE in [Gol04b]. However, formal definitions of adversaries, explicit treatment of adversarial restrictions, and abstraction from attack scenarios are new and motivated by [BHK15]. To the best of our knowledge our semantic security definition is the first definition of this art which uses less restrictive penalty-style formalization of adversarial restrictions (in terms of [BHK15]). That is, adversaries which violate the restrictions are penalized at the end of the experiment, whereas in the usual exclusion-style definitions adversaries violating the restrictions are not considered at all. Due to this novelty we explain the used formalizations and conventions in detail.

2.1.1. Semantic Security Template for PE

We first define a basic version of semantic security for PE. We begin with a formal definition of adversaries with only few pure syntactical restrictions. Let Π be a PE scheme with public index for a predicate family $\mathcal{R}_{\Omega,\Sigma}$. Formally, a semantic security adversary $\mathcal{A}$ against Π is a pair $(\mathcal{A}_1, \mathcal{A}_2)$ of algorithms with oracle access. $\mathcal{A}_1$, given among other inputs correctly generated public parameters pp_κ, outputs a triple $(\mathrm{cInd}^*, \tau, St)$. The first element is referred to as the **challenge ciphertext index** and has to satisfy $\mathrm{cInd}^* \in \mathbb{Y}_\kappa$, where $\mathbb{Y}_\kappa$ is the ciphertext index space of R_κ; τ is referred to as the **challenge template** and contains a triple of circuits $\tau = (\hat{\mathcal{M}}, h, f)$ such that the number of input bits of h and f is equal to the number of output bits of $\hat{\mathcal{M}}$; the last element St is referred to as the **state information** and there are no demands on it. For technical reasons, explained below, we also allow $\mathcal{A}_1$ to output an error symbol $\perp$. The set of semantic security adversaries against Π is denoted by $\mathbf{A}^{\mathrm{SS}}_{\Pi}$ or just $\mathbf{A}^{\mathrm{SS}}$ if Π is obvious from the context. We notice that the restriction $\mathrm{cInd}^* \in \mathbb{Y}_\kappa$ regarding the challenge ciphertext index can be done, since by Definition 1.4 for every $\kappa \in \Omega \times \Sigma$ there is an efficient membership test for $\mathbb{Y}_\kappa$.

$$
\begin{array}{l}
\textbf{SS-PE}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}\left(\lambda,\mathrm{des}\right)\ : \\
\left(\mathrm{msk},\mathrm{pp}_{\kappa}\right)\leftarrow \mathrm{Setup}\left(1^{\lambda},\mathrm{des}\right); \\
\left(\mathrm{cInd}^{*},\left(\hat{\mathcal{M}},h,f\right),St\right)\leftarrow \mathcal{A}_{1}^{\mathbf{O}_{1}\left(\mathrm{pp}_{\kappa},\mathrm{msk},\cdot\right)}\left(1^{\lambda},\mathrm{pp}_{\kappa}\right); \\
\textbf{Output}\quad 0\quad \text{if } \mathcal{A}_{1} \text{ outputs } \perp; \\
\hat{m}\leftarrow \hat{\mathcal{M}}\left(\mathcal{U}_{\mathrm{poly}(\lambda)}\right); \mathrm{CT}^{*}\leftarrow \mathrm{Enc}\left(\mathrm{cInd}^{*},\hat{m}\right); \\
\nu\leftarrow \mathcal{A}_{2}^{\mathbf{O}_{2}\left(\mathrm{pp}_{\kappa},\mathrm{msk},\cdot\right)}\left(\mathrm{CT}^{*},h\left(\hat{m}\right),St\right); \\
\textbf{Output}\qquad \nu=f\left(\hat{m}\right)\quad\wedge\quad\overline{\mathrm{BadQuery}};
\end{array}
$$

Figure 2.1.: Real semantic security experiment for PEs.

The definition of semantic security is a simulation-based definition, and hence we require two probability experiments presented in Fig. 2.1 and in Fig. 2.2. The real experiment in Fig. 2.1 is parametrized by attack scenario ATK and by adversary $\mathcal{A}\in\mathbf{A}^{\mathrm{SS}}$, whereas in the simulation experiment (Fig. 2.2) algorithm $\mathcal{A}'=(\mathcal{A}'_1,\mathcal{A}'_2)$ is (for a moment) arbitrary. In the real experiment, we use oracles $\mathbf{O}_1\left(\mathrm{pp}_{\kappa},\mathrm{msk},\cdot\right)$ and $\mathbf{O}_2\left(\mathrm{pp}_{\kappa},\mathrm{msk},\cdot\right)$ in order to model additional power of $\mathcal{A}$ in the first and in the second query phase. Concrete specifications of these oracles depend on the attack scenario ATK and are considered later in detail. We require that both oracles can be efficiently realized given public parameters and the master secret key. In order to make this requirement explicit we set pp_{κ} and msk as fixed inputs of the oracles. The event BadQuery will be used to formally define restrictions on the oracle queries of $\mathcal{A}$. The convention is that a BadQuery event occurs if adversary violates these restrictions. Consider for a moment the usual restriction in the context of PE which states that $\mathcal{A}$ is not allowed to query a user secret key if this key can be used to decrypt ciphertext CT^{*}. In this section we only notice that in all considered attack scenarios the event BadQuery can be efficiently recognized *at the end* of the experiment and the oracles can be efficiently realized using the public parameters and the master secret key as mentioned before.

By the definitions of the experiments, the challenge message $\hat{m}$ is chosen according to the probability distribution implicitly specified by $\hat{\mathcal{M}}$. The notation $\hat{\mathcal{M}}\left(\mathcal{U}_{\mathrm{poly}(\lambda)}\right)$ means that polynomially many input bits of $\hat{\mathcal{M}}$ are chosen uniformly at random. Algorithms $\mathcal{A}_2$ and $\mathcal{A}'_2$ are given the value $h\left(\hat{m}\right)$. The real adversary $\mathcal{A}_2$ is also given an encryption CT^{*} of $\hat{m}$, which is called the **challenge ciphertext**. The output of the real experiment is defined to be zero if $\mathcal{A}$ caused the BadQuery event. Apart from that, the output of the experiments is defined to be one if $\mathcal{A}_2$ respectively $\mathcal{A}'_2$ correctly predicts $f(\hat{m})$.

$$
\begin{array}{l}
\textbf{SS-PE-Sim}_{\Pi,\mathcal{A}'}\left(\lambda,\mathrm{des}\right)\ : \\
\left(\left(\hat{\mathcal{M}},h,f\right),St\right)\leftarrow \mathcal{A}'_{1}\left(1^{\lambda},\mathrm{des}\right); \\
\textbf{Output}\quad 0 \text{ if } \mathcal{A}'_{1} \text{ outputs } \perp; \\
\hat{m}\leftarrow \hat{\mathcal{M}}\left(\mathcal{U}_{\mathrm{poly}(\lambda)}\right); \nu\leftarrow \mathcal{A}'_{2}\left(h\left(\hat{m}\right),St\right); \\
\textbf{Output}\qquad \nu=f\left(\hat{m}\right);
\end{array}
$$

Figure 2.2.: Simulation semantic security experiment for PEs.

We call an algorithm $\mathcal{A}'$ a simulator for $\mathcal{A}$ with respect to $\mathrm{des}\in\Omega$ if for every $\lambda\in\mathbb{N}$ and for

every possible challenge template $\hat{\tau}$ outputted by $\mathcal{A}_1$ (incl. $\perp$) it holds

$$\Pr\left[\mathcal{A}_1' \text{ outputs } \hat{\tau}\right] \geq \Pr\left[\mathcal{A}_1 \text{ outputs } \hat{\tau} \wedge \overline{\text{BadQuery}}\right] \quad , \tag{2.1}$$

where the probability distribution on the left is defined by $\text{SS-PE-Sim}_{\Pi,\mathcal{A}'}(\lambda, \text{des})$ and the probability distribution on the right is defined by $\text{SS-PE}^{\text{ATK}}_{\Pi,\mathcal{A}}(\lambda, \text{des})$. The advantage of $\mathcal{A}$ w.r.t. $\mathcal{A}'$ under attack scenario ATK is defined by

$$\text{Adv-SS-PE}^{\text{ATK}}_{\Pi,\mathcal{A},\mathcal{A}'}(\lambda, \text{des}) := \Pr\left[\text{SS-PE}^{\text{ATK}}_{\Pi,\mathcal{A}}(\lambda, \text{des}) = 1\right] - \Pr\left[\text{SS-PE-Sim}_{\Pi,\mathcal{A}'}(\lambda, \text{des}) = 1\right] \quad .$$

We say that the advantage of an adversary $\mathcal{A} \in \mathbf{A}^{\text{SS}}_{\Pi}$ under attack scenario ATK is negligible if for all $\text{des} \in \Omega$ there exists a ppt simulator $\mathcal{A}'$ of $\mathcal{A}$ with respect to parameter des such that the advantage $\text{Adv-SS-PE}^{\text{ATK}}_{\Pi,\mathcal{A},\mathcal{A}'}(\lambda, \text{des})$ of $\mathcal{A}$ w.r.t. $\mathcal{A}'$ under attack scenario ATK is negligible in λ. Finally, semantic security is defined as follows.

Definition 2.1. A predicate encryption scheme Π with public index is called **semantically secure** under attack scenario ATK (or SS-ATK-secure) if every ppt adversary $\mathcal{A} \in \mathbf{A}^{\text{SS}}_{\Pi}$ has negligible advantage under attack scenario ATK.

Intuitively, semantic security states that a ppt adversary $\mathcal{A}$ cannot learn anything about the message $\hat{m}$ from its encryption CT^* except for negligible probability. Formally, this is proved by providing a simulator $\mathcal{A}'$, which can perform as well as $\mathcal{A}$ but is not given the challenge ciphertext CT^*. In comparison to the real adversary, simulator $\mathcal{A}_1'$ gets $\text{des} \in \Omega$ instead of pp_κ as input. This is due to the fact that all inputs of $\mathcal{A}_2'$ are independent of the concrete public parameters. We explicitly mention that in the more general context of functional encryption the authors of [BF13] revealed issues regarding the possibility of the simulators to generate the public parameters for themselves. However, they also proved that *all or nothing* schemes (including PE), where the user secret keys only allow to reconstruct the original message rather than some function on this message, are not affected.

Our definition differs from the SS-definitions for PKEs [Gol04b], IBE [ACG+06], and functional encryption [O'N10, BSW11, BO13, BF13] in two ways. First of all, we make the restrictions on the adversaries *explicit* by presenting the BadQuery event and a penalty for adversaries if they violate the restrictions of the experiment. This kind of formalization goes back to [BHK15], where the authors showed that assumptions about adversarial behavior must be formally justified in order to prevent weaknesses in security definitions. Due to the explicit penalty through the BadQuery event, we slightly modified the definition of the simulators. We notice that a simulator of $\mathcal{A}$ is allowed to output an arbitrary challenge template in the case that $\mathcal{A}$ violates the restrictions of the experiment. In turn, the challenge templates must be identically distributed in the experiments $\text{SS-PE}^{\text{ATK}}_{\Pi,\mathcal{A}}(\lambda, \text{des})$ and $\text{SS-PE-Sim}_{\Pi,\mathcal{A}'}(\lambda, \text{des})$ if $\mathcal{A}$ never causes the BadQuery event (that is, $\mathcal{A}$ does not violate the restrictions of the experiment). This is the usual condition in SS-definitions in order to relate the computation of simulator $\mathcal{A}'$ to the computation of adversary $\mathcal{A}$.

The second difference is due to the presented possibility of adversary to abort after the first query phase, which is already the result of the formal treatment of adversarial restrictions. In security definitions for predicate-based schemes, the adversary $\mathcal{A}$ is allowed to query secret keys, which models collusion attacks. We call a key index kInd a corrupted key index if $\mathcal{A}$ queries a key for kInd. $\mathcal{A}$ is prohibited to corrupt kInd if $\text{R}(\text{kInd}, \text{cInd}^*) = 1$, where cInd^* is the challenge ciphertext index. To the best of our knowledge in all previous security definitions for predicate-based schemes this restriction is always modeled in exclusion-style (in terms of [BHK15]). That is, adversaries which violate this restriction are not considered at all. In order to formally justify

this assumption for the first query phase, one has to show that given a polynomially large set of (corrupted) key indices chosen by $\mathcal{A}$, a ciphertext index cInd* which does not match all these key indices can be efficiently found. For sophisticated predicates this can be a hard problem. For example, in key-policy attribute-based schemes, in order to find a set of attributes, that can be used as the challenge ciphertext index, one has to find an assignment which does not satisfy the given Boolean formulas corresponding to the corrupted key indices. In order to deal with this problem we allow the adversary to output an error symbol $\perp$ after the first query phase which is treated as an early guess and is not penalized. This modeling enables us to keep the restriction of adversary regarding the corruption of key indices without loss of generality.

We remark that by our definition, the introduced possibility to abort does not influence the security guaranties, since adversaries have to abort before they get the challenge and in the case that adversaries violate the restrictions they are penalized even if they output $\perp$. This last circumstance is realized by the condition defined in (2.1), since simulator $\mathcal{A}'$ for $\mathcal{A}$ does not have to output $\perp$ if $\mathcal{A}$ causes the BadQuery event. This ensures that the possibility to output $\perp$ does not contribute to the advantage of $\mathcal{A}$. On the one hand, our extension simplifies the description of our constructed adversaries in the cases where $\mathcal{A}_1$ has to abort. On the other hand, we have to take care of this output during the simulation of adversaries and in the formal security analysis.

2.1.2. Indistinguishability Templates for PE and P-KEM

Next, we define indistinguishability adversaries and an indistinguishability template in the usual single-challenge form for PE as well as for P-KEM. Here we also abstract from concrete attack scenarios and make the restrictions of adversaries explicit through a BadQuery event. The experiments in this subsection are similar to the corresponding experiments for PKE and KEM from [BHK15].

Let Π be a PE scheme for predicate family $\mathcal{R}_{\Omega,\Sigma}$. An indistinguishability adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ against Π is a pair of algorithms with oracle access. $\mathcal{A}_1$, given correctly generated public parameters pp_κ, outputs the error symbol $\perp$ or a tuple $(\mathrm{cInd}^*, m_0, m_1, St)$ satisfying $\mathrm{cInd}^* \in \mathbb{Y}_\kappa$, $m_0, m_1 \in \mathcal{M}$ and $|m_0| = |m_1|$, whereas $\mathcal{A}_2$ always outputs a bit. The set of IND-adversaries against Π is denoted by $\mathbf{A}^{\mathrm{IND}}_{\Pi}$ or just by $\mathbf{A}^{\mathrm{IND}}$ if Π is obvious from the context. The set $\mathbf{A}_{\Pi,\text{P-KEM}}$ (or simply $\mathbf{A}_{\text{P-KEM}}$) of adversaries against P-KEM scheme Π is defined similarly except for the output of $\mathcal{A}_1$, which does not contain messages. The indistinguishability

$\mathbf{IND\text{-}PE}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})$:

$b \leftarrow \{0,1\}$; $(\mathrm{msk}, \mathrm{pp}_\kappa) \leftarrow \mathrm{Setup}\left(1^\lambda, \mathrm{des}\right)$;

$(\mathrm{cInd}^*, m_0, m_1, St) \leftarrow \mathcal{A}_1^{\mathbf{O}_1(\mathrm{pp}_\kappa, \mathrm{msk}, \cdot)}\left(1^\lambda, \mathrm{pp}_\kappa\right)$;

Output $b \wedge \overline{\mathrm{BadQuery}}$ if $\mathcal{A}_1$ outputs $\perp$;

$\mathrm{CT}^* \leftarrow \mathrm{Enc}\left(\mathrm{pp}_\kappa, \mathrm{cInd}^*, m_b\right)$;

$b' \leftarrow \mathcal{A}_2^{\mathbf{O}_2(\mathrm{pp}_\kappa, \mathrm{msk}, \cdot)}\left(\mathrm{CT}^*, St\right)$;

Output $b = b' \quad \wedge \quad \overline{\mathrm{BadQuery}}$;

Figure 2.3.: Indistinguishability experiments for PE.

experiments for PE and P-KEM are presented in Fig. 2.3 and in Fig. 2.4, respectively. In the case of P-KEM the family of key spaces of Π is denoted by $\mathcal{K} = \{\mathbb{K}_\lambda\}_{\lambda \in \mathbb{N}}$.

Definition 2.2. A predicate encryption scheme Π with public index for predicate family $\mathcal{R}_{\Omega,\Sigma}$ has **indistinguishable encryptions** under attack ATK (or IND-ATK-secure) if for every $\mathrm{des} \in \Omega$ and every ppt adversary $\mathcal{A} \in \mathbf{A}^{\mathrm{IND}}$ the advantage

$$\text{Adv-IND-PE}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) := 2 \cdot \Pr\left[\text{IND-PE}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) = 1\right] - 1$$

is negligible.

The security definition for P-KEM is similar and is given below. We notice that by our definition $\mathcal{A}$ cannot increase its advantage using the error symbol $\perp$ but this behavior is also not penalized. Indeed, as long as $\mathcal{A}$ does not cause the event BadQuery, the output of the experiment will be one with probability $\frac{1}{2}$ in the case that $\mathcal{A}$ outputs $\perp$. Using this modeling approach we deal with the fact that $\mathcal{A}$ must not find a valid challenge ciphertext index in order to avoid the BadQuery event. A side effect of our definition is that $\mathcal{A}_1$ can output *a guess*, in the form of $\perp$, which often simplifies the description of adversaries.

$\mathbf{P\text{-}KEM}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})$:
$b \leftarrow \{0,1\}\,; \mathrm{K}_1 \leftarrow \mathbb{K}_\lambda;$
$(\mathrm{msk}, \mathrm{pp}_\kappa) \leftarrow \mathrm{Setup}\left(1^\lambda, \mathrm{des}\right);$
$(\mathrm{cInd}^*, St) \leftarrow \mathcal{A}_1^{\mathbf{O}_1(\mathrm{pp}_\kappa, \mathrm{msk}, \cdot)}\left(1^\lambda, \mathrm{pp}_\kappa\right);$
Output $b \wedge \overline{\mathrm{BadQuery}}$ if $\mathcal{A}_1$ outputs $\perp$;
$(\mathrm{K}_0, \mathrm{CT}^*) \leftarrow \mathrm{Encaps}\left(\mathrm{pp}_\kappa, \mathrm{cInd}^*\right); \mathrm{K}^* := \mathrm{K}_b;$
$b' \leftarrow \mathcal{A}_2^{\mathbf{O}_2(\mathrm{pp}_\kappa, \mathrm{msk}, \cdot)}\left(\mathrm{K}^*, \mathrm{CT}^*, St\right);$
Output $b = b' \quad \wedge \quad \overline{\mathrm{BadQuery}}.$

Figure 2.4.: Indistinguishability experiments for P-KEM.

Definition 2.3. A predicate key encapsulation mechanism Π with public index for predicate family $\mathcal{R}_{\Omega,\Sigma}$ has **indistinguishable encapsulations** under attack ATK (or ATK-secure) if for every $\mathrm{des} \in \Omega$ and every ppt adversary $\mathcal{A} \in \mathbf{A}_{\Pi,\text{P-KEM}}$ the advantage

$$\text{Adv-P-KEM}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) := 2 \cdot \Pr\left[\text{P-KEM}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) = 1\right] - 1$$

is negligible.

2.1.3. Attack Scenarios and Additional Restrictions of Adversaries

Next, we first of all specify *to a certain extent* the BadQuery event and the oracles for attack scenarios $\mathrm{ATK} \in \{\mathrm{CPA}, \mathrm{CCA1}, \mathrm{CCA2}\}$. We use the syntax of P-KEM, since during the further analysis we use P-KEM rather than PE. All results in this subsection can be translated to PE for both SS and IND. We formally show which assumptions about the behavior of adversaries can be made without loss of generality. Even though most of these assumptions are folklore, the

work of [BHK15] showed that even in the context of PKE the real effects of certain assumptions about adversarial behavior have sometimes been underestimated or even misunderstood.

In comparison to the PKE setting, in the case of predicate-based schemes one has to prevent so-called collusion attacks. Intuitively, this means that a set of users should not be able to combine their secret keys in order to decrypt a ciphertext, that none of the users can decrypt on its own. In the security experiments for P-KEM (respectively PE) this is modeled by the key generation oracle **KGen**. This oracle takes a key index $\mathrm{kInd} \in \mathbb{X}_\kappa$ as input and returns a user secret key for kInd. We call $\mathrm{kInd} \in \mathbb{X}_\kappa$ a **corrupted key index** if $\mathcal{A}$ queried **KGen** (kInd). The key generation oracle is the only one available if chosen-plaintext attacks (CPAs) are considered.

Under the so-called adaptive chosen-ciphertext attack (CCA2) the adversary against a PE scheme additionally gets access to the decryption oracle **Dec** in both query phases, whereas under the a priori chosen-ciphertext attacks (CCA1) this oracle is available only for $\mathcal{A}_1$. The decryption oracle takes as input a ciphertext CT and a key index $\mathrm{kInd} \in \mathbb{X}_\kappa$. It returns the decryption of CT under a secret key for kInd. In the context of P-KEM this oracle is called the decapsulation oracle and is denoted by **Decaps**. Notice that in this subsection we do not specify which concrete user secret key is used in decryption respectively decapsulation oracles. This will be considered in detail in the following section, whereas in this section it is sufficient to notice that the oracles can be realized given pp_κ and msk as required in the previous subsections.

Through the BadQuery event we specify two additional restrictions on $\mathcal{A}$. The first restriction is folklore and states that the adversary is not allowed to corrupt a key index kInd if this index and the challenge ciphertext index fit together – that is, if $\mathrm{R}_\kappa(\mathrm{kInd}, \mathrm{cInd}^*) = 1$. It is important to notice that in the first query phase cInd^* is not specified, but at the end of the second phase the BadQuery event can be recognized as required. The second restriction that we consider is for CCA2 attack scenario. It disallows decryption/decapsulation query on CT^* in the *second* query phase.

The following lemma summarizes assumptions about the behavior of $\mathcal{A}$ that can be made w.l.o.g. in the context of P-KEM. The formal proof is presented in Subsection 2.2.2. Notice that for the sake of completeness we explicitly include syntactical restrictions on adversary which we defined through the set $\mathbf{A}_{\Pi,\text{P-KEM}}$.

Lemma 2.4. *Let* $\mathrm{ATK} \in \{\mathrm{CPA}, \mathrm{CCA1}, \mathrm{CCA2}\}$ *and* Π *be a* P-KEM *scheme with public index for predicate family* $\mathcal{R}_{\Omega,\Sigma}$. Π *is* ATK*-secure if and only if every ppt adversary* $\mathcal{A} \in \mathbf{A}_{\Pi,\text{P-KEM}}$ *which satisfy the following conditions has negligible advantage under attack scenario* ATK. *Let* pp_κ *be the public parameter generated during the experiment. The conditions are as follows:*

- *The output of* $\mathcal{A}_1$ *is the error symbol* $\perp$ *or a tuple* $(\mathrm{cInd}^*, \mathrm{St})$ *such that* $\mathrm{cInd}^* \in \mathbb{Y}_\kappa$. *The output of* $\mathcal{A}_2$ *is a bit.*
- *All key indices* kInd *submitted by* $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *satisfy* $\mathrm{kInd} \in \mathbb{X}_\kappa$.
- *For every* CT *submitted to the decapsulation oracle by* $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *it holds* $\mathrm{CT} \in \mathbb{C}_{\mathrm{cInd}} \subseteq \mathbb{C}_{\mathrm{pp}_\kappa}$. *Furthermore, the key index* kInd *specified for the decapsulation query on* CT *satisfies* $\mathrm{R}_\kappa(\mathrm{kInd}, \mathrm{cInd}) = 1$.
- *For every corrupted key index* kInd *(in both query phases) it holds* $\mathrm{R}_\kappa(\mathrm{kInd}, \mathrm{cInd}^*) = 0$.
- *(Only for* CCA2*)* $\mathcal{A}_2$ *never submits* CT^* *to the decapsulation oracle.*

In the proof, given an arbitrary $\mathcal{A} \in \mathbf{A}_{\text{P-KEM}}$ we construct $\mathcal{A}' \in \mathbf{A}_{\text{P-KEM}}$, which achieves the advantage of $\mathcal{A}$ and satisfies the conditions in the lemma. In particular, $\mathcal{A}'$ never causes the BadQuery event as defined above. The main non-trivial step in the proof is to ensure that the

condition regarding corrupted key indices is satisfied in the first query phase. Similar lemmata can also be proved in the context of predicate encryption schemes for our SS-security definition and for our IND-security definition when the first restriction on the form of the output of $\mathcal{A}_1$ and the output of $\mathcal{A}_2$ are adapted according to the definition of $\mathbf{A}^{\mathrm{SS}}$ and $\mathbf{A}^{\mathrm{IND}}$, respectively.

Different authors use different formalizations of security experiments, not only due to personal preferences but also due to the concrete context and considered questions. By Lemma 2.4 many of the usual assumptions about adversarial behavior are already covered, but the list is far from being complete. The additional syntactical variations are covered by the following remark.

Remark 2.5. The restrictions on adversarial queries can be extended w.l.o.g. as long as the adversaries can verify the corresponding conditions prior to the query by themselves.

This remark can be formally proved similarly to Lemma 2.4. Furthermore, the last restriction in this lemma is a good example for syntactical restrictions mentioned in the remark, whereas a similar restriction for $\mathcal{A}_1$ is not covered by Remark 2.5, since the challenge ciphertext is not defined in the first query phase. The possible restrictions on the decryption queries in the first query phase and their consequences will be considered in detail in Section 3.2.

Remark 2.6. Let us reconsider our assumption about the ciphertext index presented in the introduction of Subsection 1.2.2. Recall that we assume that the ciphertext index can be efficiently computed from a syntactically correct ciphertext. If this is not the case, than two possible cases have to be considered. Either the decryption algorithm gets as additional input the ciphertext index or the decryption algorithm does not need the ciphertext index. In the first case also the decryption oracle additionally gets the ciphertext index, and hence this index can be used instead of the ciphertext index computed from the ciphertext and we do not have to change anything else. In the second case decryption oracle can be realized without the ciphertext index. In this case one has to permit decryption queries on CT^* as long as the key indices kInd, specified for these queries, satisfy $\mathrm{R}_\kappa(\mathrm{kInd}, \mathrm{cInd}^*) = 0$. We do not consider this case separately, since in our modeling such a decryption query on CT^* corresponds to a decryption query on CT' which is obtain from CT^* by substitution of cInd^* by $\mathrm{cInd} \neq \mathrm{cInd}^*$. Such a query is allowed in our security models. Hence, we indeed can assume w.l.o.g. that the ciphertext index can be efficiently computed from a syntactically correct ciphertext.

2.2. Relations Between SS-Security and IND-Security for PE

For identity-based encryption (IBE), a special case of PE, different security notions and attack scenarios were considered in [ACG+06]. In turn, for more general functional encryption (FE) SS-definitions and IND-definitions were previously analyzed under CPA in [O'N10, BSW11, BO13, BF13]. In this subsection, based on the presented templates we discuss the relations between SS-security and IND-security in the context of PE. We can prove the following theorem similarly to the case of PKE and IBE. The formal proof is presented in Subsection 2.2.1.

Theorem 2.7. *Suppose* $\mathrm{ATK} \in \{\mathrm{CPA}, \mathrm{CCA1}, \mathrm{CCA2}\}$ *and* Π *is a predicate encryption scheme with public index. Then,* Π *is* SS-ATK*-secure if and only if it is* IND-ATK*-secure.*

With this theorem one might think that the relation between indistinguishability and semantic security are evident. However, as already mentioned, our SS-definition is really basic. Indeed, there are several reasonable and syntactically stronger definitions of semantic security. Let us first consider the probability experiments presented in Fig. 2.5 and compare these to our original experiments in Fig. 2.1 and in Fig. 2.2.

$\mathbf{aSS\text{-}PE}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda,\mathrm{des})$: $(\mathrm{msk},\mathrm{pp}_\kappa)\leftarrow\mathrm{Setup}\left(1^\lambda,\mathrm{des}\right)$;	$\mathbf{aSS\text{-}PE\text{-}Sim}_{\Pi,\mathcal{A}'}(\lambda,\mathrm{des})$:
$\left(\mathrm{cInd}^*,\left(\hat{\mathcal{M}},h\right),St\right)\leftarrow\mathcal{A}_1^{\mathbf{O}_1(\mathrm{pp}_\kappa,\mathrm{msk},\cdot)}\left(1^\lambda,\mathrm{pp}_\kappa\right)$;	$\left(\left(\hat{\mathcal{M}},h\right),St\right)\leftarrow\mathcal{A}'_1\left(1^\lambda,\mathrm{des}\right)$;
Output 0 if $\mathcal{A}_1$ outputs $\perp$;	**Output** 0 if $\mathcal{A}'_1$ outputs $\perp$;
$\hat{m}\leftarrow\hat{\mathcal{M}}\left(\mathcal{U}_{\mathrm{poly}(\lambda)}\right)$; $\mathrm{CT}^*\leftarrow\mathrm{Enc}\left(\mathrm{cInd}^*,\hat{m}\right)$;	$\hat{m}\leftarrow\hat{\mathcal{M}}\left(\mathcal{U}_{\mathrm{poly}(\lambda)}\right)$;
$(f,\nu)\leftarrow\mathcal{A}_2^{\mathbf{O}_2(\mathrm{pp}_\kappa,\mathrm{msk},\cdot)}\left(\mathrm{CT}^*,h\left(\hat{m}\right),St\right)$;	$(f,\nu)\leftarrow\mathcal{A}'_2\left(h\left(\hat{m}\right),St\right)$;
Output $\quad\nu=f\left(\hat{m}\right)\quad\wedge\quad\overline{\mathrm{BadQuery}}$;	**Output** $\quad\nu=f\left(\hat{m}\right)$;

Figure 2.5.: Adaptive semantic security experiments.

In the experiments in Fig. 2.5 the target function f is specified by adversary only at the end of the experiment. According to this syntactical modification we can adapt the definition of $\mathbf{A}^{\mathrm{SS}}_{\Pi}$ and denote the resulting set of adaptive semantic security adversaries by $\mathbf{A}^{\mathrm{aSS}}_{\Pi}$. Notice that in the adaptive SS-experiments function f might also depend on CT^* as well as on the computations and on the queries of $\mathcal{A}_2$. It seems like $\mathcal{A}$ has much more power in this experiment. We indeed have to weaken the restriction on the simulator and allow the distributions of the triples $(\hat{\mathcal{M}},h,f)$ to be indistinguishable in both experiments. We claim that w.l.o.g. we can assume that $\mathcal{A}$ does not cause the BadQuery event as defined in the previous subsection. (Formally we have to prove a statement of Lemma 2.4 adapted to the new experiments. The proof is very similar.) The following definition is additionally simplified due to this assumption.

Definition 2.8. Let $\mathrm{ATK}\in\{\mathrm{CPA},\mathrm{CCA1},\mathrm{CCA2}\}$. A PE scheme Π for predicate family $\mathcal{R}_{\Omega,\Sigma}$ is called **adaptively semantically secure** under attack scenario ATK (aSS-ATK-secure) if for every $\mathrm{des}\in\Omega$ and for every ppt algorithm $\mathcal{A}\in\mathbf{A}^{\mathrm{aSS}}$ there exists a ppt algorithm $\mathcal{A}'=(\mathcal{A}'_1,\mathcal{A}'_2)$ such that the following conditions are satisfied:

1. $\Pr\left[\mathcal{A}'_1\text{ outputs }\perp\right]=\Pr\left[\mathcal{A}_1\text{ outputs }\perp\right]$ and the distributions of the triples $\left(\hat{\mathcal{M}},h,f\right)$ in $\text{aSS-PE}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda,\mathrm{des})$ and in $\text{aSS-PE-Sim}_{\Pi,\mathcal{A}'}(\lambda,\mathrm{des})$ are computationally indistinguishable.
2. The advantage of $\mathcal{A}$ defined by

$$\text{Adv-aSS-PE}^{\mathrm{ATK}}_{\Pi,\mathcal{A},\mathcal{A}'}(\lambda,\mathrm{des}):=\Pr\left[\text{aSS-PE}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda,\mathrm{des})=1\right]-\Pr\left[\text{aSS-PE-Sim}_{\Pi,\mathcal{A}'}(\lambda,\mathrm{des})=1\right]$$

is a negligible (in λ) function.

Notice that in comparison to Definition 2.1 we defined the requirements on simulators in the definition itself. That is, as before, we call an algorithm $\mathcal{A}'$ which satisfies the first requirement of Definition 2.8 for $\mathrm{des}\in\Omega$ a simulator of $\mathcal{A}$ with respect to des. We can prove the following theorem (see the proof in Subsection 2.2.2).

Theorem 2.9. *Suppose* $\mathrm{ATK} \in \{\mathrm{CPA}, \mathrm{CCA1}, \mathrm{CCA2}\}$ *and* Π *is a predicate encryption scheme with public index. Then,* Π *is semantically secure under attack scenario* ATK *if and only if it is adaptive semantically secure under attack scenario* ATK*.*

The adaptive semantic security definition is only a few (non-trivial) steps shy of so-called SS2-security definition for functional encryption (FE) presented in [BO13]. This definition is proved to be equivalent to the indistinguishability definition even in the more general context of FE. The required extensions toward the SS2-security are similar to the extensions for PKE and IBE and we refer to [Gol04b, ACG+06] for extensive study of these extensions.

The SS2 security experiments differ from the experiments in Fig. 2.5 mainly in the ability of adversary $\mathcal{A}$ to query several challenges. The goal of $\mathcal{A}$ is then to predict $f(\hat{m}_1, \ldots, \hat{m}_l)$, where $\hat{m}_i$'s are the chosen challenge messages. We notice that for the corresponding security notion in the context of PKE one can show that revealing some of the challenge messages does not help the adversary to learn something about the remaining messages (cf. [Gol04b]). Formally, in the corresponding simulation experiment, also the simulator gets access to the revealed message through an oracle. At this point there is an important difference between PE (as well as IBE, FE) and PKE. Namely, in PE we could also reveal $\hat{m}_1$, encrypted under cInd_1^*, when a key index $\widehat{\mathrm{kInd}}$ that fits together with cInd_1^* is corrupted. The other challenge messages might still be hidden if encrypted under ciphertext indices that do not fit together with $\widehat{\mathrm{kInd}}$. An essential part of the SS2 security definition is that it explicitly excludes these so-called key-revealing selective-opening attacks (SOA-Ks). That is, adversaries are not allowed to corrupt a key index kInd *after* they are given a challenge for ciphertext index cInd^* if $\mathrm{R}_\kappa(\mathrm{kInd}, \mathrm{cInd}^*) = 1$.

Finally, we notice that the SS1 security for FE (in the terminology of [BO13]) captures security in the presence of SOA-Ks, which not only destroys the equivalence to the IND-security but also leads to the impossibility results analyzed in [O'N10, BSW11, BO13, BF13]. In [ACG+06] the authors did not consider SOA-Ks, which explains their equivalence results. For more details and an explanation of these impossibility results we refer to [BO13, BF13].

Based on our results from this section and the results in [O'N10, BSW11, BO13, BF13] for functional encryption under CPA, we deduce that, as long as key-revealing selective-opening attacks (SOA-Ks) are not considered, the indistinguishability definition remains the most suitable security definition in the context of PE. In the context of key-revealing selective-opening attacks, the adversary is allowed to query user secret keys that can be used to decrypt the challenge ciphertext in the second query phase, which leads to very strong security notions. Furthermore, for general functional encryption schemes the possibility to corrupt user secret keys that can be used to decrypt the challenge ciphertext leads to impossibility results even in the non-adaptive model and in the random oracle model [AGVW13, AKW16].

2.2.1. Equivalence of SS and IND for PE

In this subsection we prove Theorem 2.7. In order to prevent additional work regarding the ability of the adversary to abort in the first query phase let us consider here how it influences the analysis of the advantage of adversaries. These results are used in the following proofs.

Let us consider an arbitrary, but fixed adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2) \in \mathbf{A}^{\mathrm{SS}}$ that satisfies the conditions of Lemma 2.4 adapted to PE and SS-definition. Let $\mathrm{ATK} \in \{\mathrm{CPA}, \mathrm{CCA1}, \mathrm{CCA2}\}$ and $\mathrm{des} \in \Omega$ be arbitrary but fixed. We denote by E the event that $\mathcal{A}_1$ outputs $\perp$ in the experiment $\text{SS-PE}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})$. Let $p_{\mathrm{des}}(\lambda)$ be the probability for the event $\overline{\mathrm{E}}$ in this experiment – that is, $p_{\mathrm{des}}(\lambda) = \Pr\left[\overline{\mathrm{E}}\right] = 1 - \Pr[\mathrm{E}]$. $\mathcal{A}$ never causes the event BadQuery, and hence for every simulator $\mathcal{A}' = (\mathcal{A}'_1, \mathcal{A}'_2)$ for $\mathcal{A}$ with respect to des the challenge templates must be identically distributed in $\text{SS-PE}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})$ and in $\text{SS-PE-Sim}_{\Pi,\mathcal{A}'}(\lambda, \mathrm{des})$. In particular, the

probability that $\mathcal{A}_1'$ outputs $\perp$ (event E') is equal to $1 - p_{\mathrm{des}}(\lambda)$. Since $\mathcal{A}$ as well as $\mathcal{A}'$ lose in their experiments if the event E respectively E' occur, it holds

$$\begin{aligned}
& \text{Adv-SS-PE}^{\mathrm{ATK}}_{\Pi,\mathcal{A},\mathcal{A}'}(\lambda, \mathrm{des}) \\
\overset{\text{by def.}}{=}\ & \Pr\left[\text{SS-PE}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) = 1\right] - \Pr\left[\text{SS-PE-Sim}_{\Pi,\mathcal{A}'}(\lambda, \mathrm{des}) = 1\right] \\
=\ & p_{\mathrm{des}}(\lambda) \cdot \Big(\Pr\left[\text{SS-PE}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) = 1 \mid \overline{\mathrm{E}}\right] \\
& \qquad - \Pr\left[\text{SS-PE-Sim}_{\Pi,\mathcal{A}'}(\lambda, \mathrm{des}) = 1 \mid \overline{\mathrm{E}'}\right]\Big). \qquad (2.2)
\end{aligned}$$

By the analysis of adversarial advantage, if $p_{\mathrm{des}}(\lambda)$ is negligible, the second factor does not matter, or rather such an adversary outputs $\perp$ with all except negligible probability, cannot achieve non-negligible advantage, and hence can be ignored. On the contrary, if $p_{\mathrm{des}}(\lambda)$ is not negligible, it is sufficient to consider the success probability of $\mathcal{A}$, conditioned on the fact that $\mathcal{A}_1$ did not output $\perp$.

Let us also consider the advantage of an arbitrary adversary $\mathcal{A} = (A_1, A_2) \in \mathbf{A}^{\mathrm{IND}}_{\Pi}$ that satisfies the conditions of Lemma 2.4 adapted to PE and IND-definition. By E we denote the event that $\mathcal{A}_1$ outputs $\perp$ in the experiment $\text{IND-PE}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})$. Furthermore, let $p_{\mathrm{des}}(\lambda)$ be the probability for the event $\overline{\mathrm{E}}$ in this experiment – that is, $p_{\mathrm{des}}(\lambda) = \Pr\left[\overline{\mathrm{E}}\right] = 1 - \Pr[\mathrm{E}]$. Then, it holds by the definition of the experiment:

$$\begin{aligned}
& \text{Adv-IND-PE}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) \\
\overset{\text{by def.}}{=}\ & 2 \cdot \Pr\left[\text{IND-PE}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) = 1\right] - 1 \\
=\ & 2 \cdot \left(\frac{1}{2} \cdot \Pr[\mathrm{E}] + \Pr\left[\text{IND-PE}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) = 1 \wedge \overline{\mathrm{E}}\right]\right) - 1 \\
=\ & 2 \cdot \Pr\left[\text{IND-PE}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) = 1 \mid \overline{\mathrm{E}}\right] \cdot \Pr\left[\overline{\mathrm{E}}\right] - (1 - \Pr[\mathrm{E}]) \\
=\ & p_{\mathrm{des}}(\lambda) \cdot \left(2 \cdot \Pr\left[\text{IND-PE}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) = 1 \mid \overline{\mathrm{E}}\right] - 1\right) \\
=\ & p_{\mathrm{des}}(\lambda) \cdot \Big(\Pr\left[\text{IND-PE}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) = 1 \mid \overline{\mathrm{E}} \wedge b = 0\right] \\
& \qquad + \Pr\left[\text{IND-PE}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) = 1 \mid \overline{\mathrm{E}} \wedge b = 1\right] - 1\Big) \\
=\ & p_{\mathrm{des}}(\lambda) \cdot \Big(\Pr\left[b' = 0 \mid \overline{\mathrm{E}} \wedge b = 0\right] \\
& \qquad + \Pr\left[b' = 1 \mid \overline{\mathrm{E}} \wedge b = 1\right] - 1\Big)\ , \qquad (2.3)
\end{aligned}$$

where in the penultimate equation we used the fact the the choice of b and the event E are independent. Notice that all except the last equation hold also if $\mathcal{A}$ causes the event BadQuery. In particular, the second equation holds due to the fact that the event BadQuery cannot be caused if $\mathcal{A}_1$ outputs $\perp$.

Remark 2.10. Similarly to the semantic security, by the analysis of adversarial advantage, if $p_{\mathrm{des}}(\lambda)$ is negligible, the second factor in (2.3) does not matter or rather such an adversary outputs $\perp$ with all except negligible probability, and hence can be ignored. Contrary, if $p_{\mathrm{des}}(\lambda)$ is non-negligible, it is sufficient to analyze the success probability of $\mathcal{A}$ conditioned on the fact that $\mathcal{A}_1$ did not output $\perp$. Hence, even though formally we have to give $\mathcal{A}_1$ the possibility to abort, it is still sufficient to analyze the success probability of $\mathcal{A}$ assuming that $\mathcal{A}_1$ never outputs $\perp$. Hence, for simplicity we suggest to formalize the security models under the assumption that $\mathcal{A}_1$ never outputs error symbol when concrete schemes are considered.

Indistinguishability Implies Semantic Security

The following lemma shows that it is sufficient to prove the indistinguishability of encryptions in order to prove semantic security of the scheme.

Lemma 2.11. *Let* $\mathrm{ATK} \in \{\mathrm{CPA}, \mathrm{CCA1}, \mathrm{CCA2}\}$ *and* Π *be a predicate encryption scheme for predicate family* $\mathcal{R}_{\Omega,\Sigma}$ *which is* IND-ATK*-secure. Then,* Π *is* SS-ATK*-secure. In particular, for every ppt adversary* $\mathcal{A} \in \mathbf{A}^{\mathrm{SS}}$ *there exists a ppt simulator* $\mathcal{A}'$ *for* $\mathcal{A}$ *and a ppt adversary* $\mathcal{B} \in \mathbf{A}^{\mathrm{IND}}$ *such that for every* $\mathrm{des} \in \Omega$, $\lambda \in \mathbb{N}$ *it holds*

$$\text{Adv-IND-PE}^{\mathrm{ATK}}_{\Pi,\mathcal{B}}(\lambda, \mathrm{des}) \quad \geq \quad \text{Adv-SS-PE}^{\mathrm{ATK}}_{\Pi,\mathcal{A},\mathcal{A}'}(\lambda, \mathrm{des}) \quad .$$

Proof. Note that the statement in the lemma is even stronger than required by the definition of semantic security, since $\mathcal{A}'$ will be a simulator for $\mathcal{A}$ with respect to every $\mathrm{des} \in \Omega$.

We define a modification $\text{SS-PE-Mod}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})$ of the experiment $\text{SS-PE}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})$, where CT^* is computed by $\mathrm{Enc}\left(\mathrm{pp}_\kappa, \mathrm{cInd}^*, 1^{|\hat{m}|}\right)$ instead of $\mathrm{Enc}\left(\mathrm{pp}_\kappa, \mathrm{cInd}^*, \hat{m}\right)$. It is important to notice that the experiments are identical until the generation of the challenge ciphertext.

Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2) \in \mathbf{A}^{\mathrm{SS}}$ be an arbitrary but fixed adversary against Π which does not cause the BadQuery. In order to prove the statement of the lemma it is sufficient to show that the advantage of $\mathcal{A}$ is negligible (due to Lemma 2.4 adapted to PE and SS-definition). Note that even though $\mathcal{A}$ never causes the event BadQuery in $\text{SS-PE}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})$, the probability $\Pr[\mathrm{BadQuery}]$ in the modified experiment $\text{SS-PE-Mod}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})$ is not necessarily zero. The same holds also for the other conditions defined in Lemma 2.4. Nevertheless, due to the special modification of the experiment $\text{SS-PE-Mod}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})$ all these properties hold for $\mathcal{A}_1$. This will be explicitly used in our construction of $\mathcal{B} \in \mathbf{A}^{\mathrm{IND}}$ below.

Let additionally $\mathrm{des} \in \Omega$, $\lambda \in \mathbb{N}$ be arbitrary, but fixed. First, we construct a simulation algorithm $\mathcal{A}' = (\mathcal{A}'_1, \mathcal{A}'_2)$ which uses $\mathcal{A}$ as a subroutine. Note that simulator $\mathcal{A}'_2$ does not receive the encryption of $\hat{m}$, and hence cannot simulate $\mathcal{A}_2$ as in $\text{SS-PE}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})$, but $\mathcal{A}'_2$ can properly simulate the view of $\mathcal{A}$ in $\text{SS-PE-Mod}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})$. Intuitively, due to the indistinguishability of ciphertexts $\mathcal{A}$ cannot notice the difference. This will be formally proved in the second part of the proof.

$\mathcal{A}'_1\left(1^\lambda, \mathrm{des}\right)$:

- Generate $(\mathrm{msk}, \mathrm{pp}_\kappa) \leftarrow \mathrm{Setup}\left(1^\lambda, \mathrm{des}\right)$.
- Compute $\left(\mathrm{cInd}^*, \left(\hat{\mathcal{M}}, h, f\right), St\right) \leftarrow \mathcal{A}_1^{\mathbf{O}_1(\mathrm{pp}_\kappa, \mathrm{msk}, \cdot)}\left(1^\lambda, \mathrm{pp}_\kappa\right)$ using $\mathrm{msk}, \mathrm{pp}_\kappa$.
- **Output** $\perp$ if $\mathcal{A}_1$ outputs $\perp$.
- Set $St' := (St, \mathrm{msk}, \mathrm{pp}_\kappa, \mathrm{cInd}^*, |\hat{m}|)$ and **output** $\left(\mathrm{cInd}^*, \left(\hat{\mathcal{M}}, h, f\right), St'\right)$.

$\mathcal{A}'_2\left(h\left(\hat{m}\right), St'\right)$ with $St' = (St, \mathrm{msk}, \mathrm{pp}_\kappa, \mathrm{cInd}^*, |\hat{m}|)$:

- Generate $\mathrm{CT}' \leftarrow \mathrm{Enc}\left(\mathrm{pp}_\kappa, \mathrm{cInd}^*, 1^{|\hat{m}|}\right)$.
- Compute $\nu \leftarrow \mathcal{A}_2^{\mathbf{O}_2(\mathrm{pp}_\kappa, \mathrm{msk}, \cdot)}\left(\mathrm{CT}', h\left(\hat{m}\right), St\right)$ using $\mathrm{msk}, \mathrm{pp}_\kappa$, and **output** ν.

The distribution of the challenge template generated by $\mathcal{A}'_1$ is as required in Definition 2.1, since $\mathcal{A}'_1$ uses the correctly generated $(\mathrm{msk}, \mathrm{pp}_\kappa)$ and $\mathcal{A}_1$ in order to generate $\left(\hat{\mathcal{M}}, h, f\right)$. Hence,

$\mathcal{A}'$ is a simulator for $\mathcal{A}$ with respect to every $\mathrm{des} \in \Omega$. Furthermore, by construction of $\mathcal{A}'$, the view of $\mathcal{A}$ in this experiment is the same as in the experiment $\text{SS-PE-Mod}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})$. Hence, if $\mathcal{A}$ wins in $\text{SS-PE-Mod}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})$, then $\mathcal{A}'$ wins in $\text{SS-PE-Sim}_{\Pi,\mathcal{A}'}(\lambda, \mathrm{des})$. The opposite direction does not necessarily hold, since $\mathcal{A}$ might get a penalty (caused by $\mathcal{A}_2$) whereas $\mathcal{A}'$ could still win in this case. We deduce that it holds

$$\Pr\left[\text{SS-PE-Sim}_{\Pi,\mathcal{A}'}(\lambda, \mathrm{des}) = 1\right] \quad \geq \quad \Pr\left[\text{SS-PE-Mod}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) = 1\right] \ . \tag{2.4}$$

Now we are ready to construct an adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2) \in \mathbf{A}^{\mathrm{IND}}$ for the indistinguishability experiment $\text{IND-PE}^{\mathrm{ATK}}_{\Pi,\mathcal{B}}(\lambda, \mathrm{des})$ and to relate the success probability of this algorithm to advantage $\text{Adv-SS-PE}^{\mathrm{ATK}}_{\Pi,\mathcal{A},\mathcal{A}'}(\lambda, \mathrm{des})$. By construction it obviously holds $\mathcal{B} \in \mathbf{A}^{\mathrm{IND}}$. $\mathcal{B}$ (in particular $\mathcal{B}_1$) redirects all queries of $\mathcal{A}$ and does not make additional queries. Furthermore, in the first query phase $\mathcal{B}_1$ perfectly simulates the view of $\mathcal{A}_1$ in the experiment $\text{SS-PE}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})$ (respectively in the experiment $\text{SS-PE-Mod}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})$). Hence, $\mathcal{B}_1$ never causes the event BadQuery. In turn, $\mathcal{B}_2$ explicitly prevents the event BadQuery. We deduce that $\mathcal{B}$ will never cause a penalty and wins if $b' = b$, where b is its challenge bit.

$\mathcal{B}_1^{\mathbf{O}_1(\mathrm{pp}_\kappa, \mathrm{msk}, \cdot)}\left(1^\lambda, \mathrm{pp}_\kappa\right)$:

- Compute $\left(\mathrm{cInd}^*, \left(\hat{\mathcal{M}}, h, f\right), St\right) \leftarrow \mathcal{A}_1^{\mathbf{O}_1(\mathrm{pp}_\kappa, \mathrm{msk}, \cdot)}\left(1^\lambda, \mathrm{pp}_\kappa\right)$ using own oracles in order to answer the queries of $\mathcal{A}_1$. **Output** $\perp$ if the output of $\mathcal{A}_1$ is $\perp$.
- Choose $\hat{m} \leftarrow \hat{\mathcal{M}}\left(\mathcal{U}_{\mathrm{poly}(\lambda)}\right)$, set $m_0 := 1^{|\hat{m}|}$ and $m_1 := \hat{m}$. Set $St' := (St, h(\hat{m}), f(\hat{m}))$.
- **Output** $(\mathrm{cInd}^*, m_0, m_1, St')$.

$B_2^{\mathbf{O}_2(\mathrm{pp}_\kappa, \mathrm{msk}, \cdot)}(\mathrm{CT}^*, St')$ with $St' = \left(St, \hat{h}, \hat{f}\right)$:

- Simulate $\nu \leftarrow \mathcal{A}_2^{\mathbf{O}_2(\mathrm{pp}_\kappa, \mathrm{msk}, \cdot)}\left(\mathrm{CT}^*, \hat{h}, St\right)$ using own oracles in order to answer the queries of $\mathcal{A}_2$. Thereby abort the simulation of $\mathcal{A}_2$ and **output** $b' = 0$ if $\mathcal{A}_2$ causes the event BadQuery.
- **Output** $b' = 1$ if $\nu = \hat{f}$. Otherwise **output** 0.

Let E be the event that an adversary outputs $\perp$ after the first query phase of the corresponding experiment (we will make the adversary explicit in the subindex). Furthermore, let $p_{\mathrm{des}}(\lambda) = \Pr\left[\overline{\mathrm{E}_{\mathcal{B}}}\right] = \Pr\left[\overline{\mathrm{E}_{\mathcal{A}}}\right]$, where the second equation holds by construction. Let b be the challenge bit of $\mathcal{B}$. We will analyze the view of $\mathcal{A}_2$ and the success probability of $\mathcal{B}$ for both values of b conditioned on the event $\overline{\mathrm{E}_{\mathcal{B}}}$.

- $\overline{\mathrm{E}_{\mathcal{B}}} \wedge b = 0$: It holds $\mathrm{CT}^* = \mathrm{Enc}\left(\mathrm{pp}_\kappa, \mathrm{cInd}^*, 1^{|\hat{m}|}\right)$, where $\hat{m} \leftarrow \hat{\mathcal{M}}\left(\mathcal{U}_{\mathrm{poly}(\lambda)}\right)$. Hence, the view of $\mathcal{A}_2$ is as in the experiment $\text{SS-PE-Mod}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})$. In particular, $\mathcal{A}_2$ might still cause the event BadQuery. In this case $\mathcal{B}_2$ outputs $b' = 0$ without causing the penalty and wins. We deduce

$$\begin{aligned}
\Pr\left[b' = 0 \mid \overline{\mathrm{E}_{\mathcal{B}}} \wedge b = 0\right] &\overset{(*)}{=} \Pr\left[\mathrm{BadQuery} \mid \overline{\mathrm{E}_{\mathcal{A}}}\right] \\
&\quad + \Pr\left[\nu \neq f(\hat{m}) \wedge \overline{\mathrm{BadQuery}} \mid \overline{\mathrm{E}_{\mathcal{A}}}\right] \\
&= 1 - \Pr\left[\nu = f(\hat{m}) \wedge \overline{\mathrm{BadQuery}} \mid \overline{\mathrm{E}_{\mathcal{A}}}\right] \\
&= 1 - \frac{1}{p_{\mathrm{des}}(\lambda)} \cdot \Pr\left[\nu = f(\hat{m}) \wedge \overline{\mathrm{BadQuery}} \wedge \overline{\mathrm{E}_{\mathcal{A}}}\right] \\
&\overset{\text{by def.}}{=} 1 - \frac{1}{p_{\mathrm{des}}(\lambda)} \cdot \Pr\left[\text{SS-PE-Mod}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) = 1\right]
\end{aligned}$$

$$\overset{(2.4)}{\geq} \quad 1 - \frac{1}{p_{\mathrm{des}}(\lambda)} \cdot \Pr\left[\text{SS-PE-Sim}_{\Pi,\mathcal{A}'}(\lambda, \mathrm{des}) = 1\right] \quad ,$$

where in $(*)$ we switch from probability distribution defined by $\text{IND-PE}^{\mathrm{ATK}}_{\Pi,\mathcal{B}}(\lambda, \mathrm{des})$ to the probability distribution defined by $\text{SS-PE-Mod}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})$.

- $\overline{\mathrm{E}_{\mathcal{B}}} \wedge b = 1$: It holds $\mathrm{CT}^* = \mathrm{Enc}(\mathrm{pp}_\kappa, \mathrm{cInd}^*, \hat{m})$, where $\hat{m} \leftarrow \hat{\mathcal{M}}\left(\mathcal{U}_{\mathrm{poly}(\lambda)}\right)$. Hence, the view of $\mathcal{A}_2$ is as in the experiment $\text{SS-PE}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})$. In particular, $\mathcal{A}_2$ never causes the event BadQuery. We deduce

$$\begin{aligned}
\Pr\left[b' = 1 \;\middle|\; \overline{\mathrm{E}_{\mathcal{B}}} \wedge b = 1\right] &\overset{(*)}{=} \Pr\left[\nu = f(\hat{m}) \;\middle|\; \overline{\mathrm{E}_{\mathcal{A}}}\right] \\
&= \frac{1}{p_{\mathrm{des}}(\lambda)} \cdot \Pr\left[\nu = f(\hat{m}) \wedge \overline{\mathrm{E}_{\mathcal{A}}}\right] \\
&\overset{\text{by def.}}{=} \frac{1}{p_{\mathrm{des}}(\lambda)} \cdot \Pr\left[\text{SS-PE}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) = 1\right] \quad ,
\end{aligned}$$

where in $(*)$ we switch from probability distribution defined by $\text{IND-PE}^{\mathrm{ATK}}_{\Pi,\mathcal{B}}(\lambda, \mathrm{des})$ to the probability definition defined by $\text{SS-PE}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})$.

$\mathcal{B}$ does not cause the event BadQuery, and hence the advantage of $\mathcal{B}$ is as follows

$$\begin{aligned}
& \text{Adv-IND-PE}^{\mathrm{ATK}}_{\Pi,\mathcal{B}}(\lambda, \mathrm{des}) \\
\overset{(2.3)}{=} \quad & p_{\mathrm{des}}(\lambda) \cdot \left(\Pr\left[b' = 0 \;\middle|\; \overline{\mathrm{E}} \wedge b = 0\right] + \Pr\left[b' = 1 \;\middle|\; \overline{\mathrm{E}} \wedge b = 1\right] - 1\right) \\
\geq \quad & \Pr\left[\text{SS-PE}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) = 1\right] - \Pr\left[\text{SS-PE-Sim}_{\Pi,\mathcal{A}'}(\lambda, \mathrm{des}) = 1\right] \\
\overset{\text{by def.}}{=} \quad & \text{Adv-SS-PE}^{\mathrm{ATK}}_{\Pi,\mathcal{A},\mathcal{A}'}(\lambda, \mathrm{des}) \quad .
\end{aligned}$$

However, for every $\mathcal{B} \in \mathbf{A}^{\mathrm{IND}}$ and every $\mathrm{des} \in \Omega$ the advantage $\text{Adv-IND-PE}^{\mathrm{ATK}}_{\Pi,\mathcal{B}}(\lambda, \mathrm{des})$ is negligible due to the IND-ATK-security of Π. This concludes the proof. □

Due to this lemma, in order to prove that a predicate encryption scheme is semantically secure it is sufficient to prove that this scheme has indistinguishable encryptions.

Semantic Security Implies Indistinguishability of Encryptions

The following lemma shows that the notion of indistinguishable encryptions is not too strong.

Lemma 2.12. *Let* $\mathrm{ATK} \in \{\mathrm{CPA}, \mathrm{CCA1}, \mathrm{CCA2}\}$ *and* Π *be a predicate encryption scheme for predicate family* $\mathcal{R}_{\Omega,\Sigma}$ *which is* SS-ATK*-secure. Then,* Π *is* IND-ATK*-secure. In particular, for every ppt adversary* $\mathcal{A} \in \mathbf{A}^{\mathrm{IND}}$ *there exists a ppt adversary* $\mathcal{B} \in \mathbf{A}^{\mathrm{SS}}$ *such that for every* $\mathrm{des} \in \Omega$ *and every ppt simulator* $\mathcal{B}'$ *of* $\mathcal{B}$ *with respect to* des *it holds*

$$2 \cdot \text{Adv-SS-PE}^{\mathrm{ATK}}_{\Pi,\mathcal{B},\mathcal{B}'}(\lambda, \mathrm{des}) \quad \geq \quad \text{Adv-IND-PE}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) \quad .$$

Proof. Let $\lambda \in \mathbb{N}$, $\mathrm{des} \in \Omega$, $\mathrm{ATK} \in \{\mathrm{CPA}, \mathrm{CCA1}, \mathrm{CCA2}\}$, and $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2) \in \mathbf{A}^{\mathrm{IND}}$ be arbitrary but fixed such that $\mathcal{A}$ satisfies the restrictions defined in Lemma 2.4 (adapted to PE). In particular, $\mathcal{A}$ never causes the event BadQuery.

We construct $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2) \in \mathbf{A}^{\mathrm{SS}}$ which exploits the advantage of $\mathcal{A}$. $\mathcal{B} \in \mathbf{A}^{\mathrm{SS}}$ by construction. Furthermore, the view of $\mathcal{A}$ is as defined in $\text{IND-PE}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})$. In particular, $\mathcal{A}_1$ receives the correctly generated public parameters and all queries of $\mathcal{A}_1$ are correctly answered.

$\mathcal{B}_1^{\mathbf{O}_1(\mathrm{pp}_\kappa,\mathrm{msk},\cdot)}\left(1^\lambda,\mathrm{pp}_\kappa\right)$:

- Simulate $(\mathrm{cInd}^*,m_0,m_1,St)\leftarrow\mathcal{A}_1^{\mathbf{O}_1(\mathrm{pp}_\kappa,\mathrm{msk},\cdot)}\left(1^\lambda,\mathrm{pp}_\kappa\right)$ using the own oracles in order to answer the queries of $\mathcal{A}_1$.
- **Output** $\perp$ if the output of $\mathcal{A}_1$ is $\perp$.
- Set $\hat{\mathcal{M}}$ to a circuit corresponding to the uniform distribution on $\{m_0,m_1\}$, set h to an arbitrary function such that $h(m_0)=h(m_1)$, and set f to an arbitrary function such that $f(m_0)=0$ and $f(m_1)=1$.[a]
- **Output** $\left(\mathrm{cInd}^*,\left(\hat{\mathcal{M}},h,f\right),St\right)$.

$\mathcal{B}_2^{\mathbf{O}_2(\mathrm{pp}_\kappa,\mathrm{msk},\cdot)}\left(\mathrm{CT}^*,h(\hat{m}),St\right)$:

- Simulate $b'\leftarrow\mathcal{A}_2^{\mathbf{O}_2(\mathrm{pp}_\kappa,\mathrm{msk},\cdot)}\left(\mathrm{CT}^*,St\right)$ using the own oracles in order to answer the queries of $\mathcal{A}_1$.
- **Output** $\nu:=b'$.

[a] For simplicity we assumed w.l.o.g. that $m_0\neq m_1$. If $m_0=m_1$, $\mathcal{A}$ cannot have any advantage and $\mathcal{B}_1$ can output $\perp$.

In turn, $\mathcal{A}_2$ receives the encryption of m_0 or the encryption of m_1, both with probability $\frac{1}{2}$ and all queries of $\mathcal{A}_2$ are also correctly answered. Hence, $\mathcal{B}$ never causes the event BadQuery since $\mathcal{A}$ never causes the corresponding event. Consider an arbitrary simulator $\mathcal{B}'=(\mathcal{B}'_1,\mathcal{B}'_2)$ of $\mathcal{B}$ with respect to an arbitrary but fixed $\mathrm{des}\in\Omega$. Due to the requirement on $\mathcal{B}'$ the distribution of $\left(\hat{\mathcal{M}},h,f\right)$ generated by $\mathcal{B}'_1$ must be the same as above in the experiment with $\mathcal{B}$. By construction, $\mathcal{B}_1$ outputs $\perp$ if and only if $\mathcal{A}_1$ outputs $\perp$ and $\mathcal{B}'_1$ outputs $\perp$ with the same probability by definition. Let E be the event that an adversary (or a simulator) outputs $\perp$ after the first query phase (we will make the adversary explicit in the subindex). Furthermore, let $p_{\mathrm{des}}(\lambda)=\Pr\left[\overline{\mathrm{E}_{\mathcal{B}}}\right]=\Pr\left[\overline{\mathrm{E}_{\mathcal{B}'}}\right]=\Pr\left[\overline{\mathrm{E}_{\mathcal{A}}}\right]$. Due to $h(m_0)=h(m_1)$ and $|m_0|=|m_1|$ the input of $\mathcal{B}'_2$ is independent of $\hat{m}$ and it holds

$$\Pr\left[\text{SS-PE-Sim}_{\Pi,\mathcal{B}'}(\lambda,\mathrm{des})=1\;\middle|\;\overline{\mathrm{E}_{\mathcal{B}'}}\right]\leq\frac{1}{2}\ ,$$

since $\hat{\mathcal{M}}$ is the uniform distribution on $\{m_0,m_1\}$, $f(m_0)=0$, and $f(m_1)=1$.

Consider the sucess probability of $\mathcal{B}$. By construction it holds

$$\begin{aligned}
&\Pr\left[\text{SS-PE}^{\mathrm{ATK}}_{\Pi,\mathcal{B}}(\lambda,\mathrm{des})=1\;\middle|\;\overline{\mathrm{E}_{\mathcal{B}}}\right]\\
=\;&\Pr\left[\hat{m}=m_0\wedge b'=f(\hat{m})\;\middle|\;\overline{\mathrm{E}_{\mathcal{B}}}\right]+\Pr\left[\hat{m}=m_1\wedge b'=f(\hat{m})\;\middle|\;\overline{\mathrm{E}_{\mathcal{B}}}\right]\\
=\;&\Pr\left[b'=f(\hat{m})\;\middle|\;\overline{\mathrm{E}_{\mathcal{B}}}\wedge\hat{m}=m_0\right]\cdot\Pr\left[\hat{m}=m_0\;\middle|\;\overline{\mathrm{E}_{\mathcal{B}}}\right]\\
&+\Pr\left[b'=f(\hat{m})\;\middle|\;\overline{\mathrm{E}_{\mathcal{B}}}\wedge\hat{m}=m_1\right]\cdot\Pr\left[\hat{m}=m_1\;\middle|\;\overline{\mathrm{E}_{\mathcal{B}}}\right]\\
\overset{f}{=}\;&\frac{1}{2}\cdot\left(\Pr\left[b'=0\;\middle|\;\overline{\mathrm{E}_{\mathcal{A}}}\wedge b=0\right]+\Pr\left[b'=1\;\middle|\;\overline{\mathrm{E}_{\mathcal{A}}}\wedge b=1\right]\right)\\
\overset{(2.3)}{=}\;&\frac{1}{2\cdot p_{\mathrm{des}}(\lambda)}\cdot\text{Adv-IND-PE}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda,\mathrm{des})+\frac{1}{2}\ .
\end{aligned}$$

We deduce that for every simulator $\mathcal{B}'$ of $\mathcal{B}$ with respect to des it holds (we use the fact that $\Pr[\mathrm{BadQuery}]=0$ in $\text{SS-PE}^{\mathrm{ATK}}_{\Pi,\mathcal{B}}(\lambda,\mathrm{des})$):

$$2\cdot\text{Adv-SS-PE}^{\mathrm{ATK}}_{\Pi,\mathcal{B},\mathcal{B}'}(\lambda,\mathrm{des})$$

$$\begin{aligned} &\overset{(2.2)}{=} && 2 \cdot p_{\mathrm{des}}(\lambda) \cdot \Big(\Pr\left[\text{SS-PE}^{\mathrm{ATK}}_{\Pi,\mathcal{B}}(\lambda, \mathrm{des}) = 1 \mid \overline{\mathrm{E}_{\mathcal{B}}}\right] \\ &&& \qquad - \Pr\left[\text{SS-PE-Sim}_{\Pi,\mathcal{B}'}(\lambda, \mathrm{des}) = 1 \mid \overline{\mathrm{E}_{\mathcal{B}'}}\right] \Big) \\ &\geq && \text{Adv-IND-PE}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) \quad . \end{aligned}$$

Using this result we argue by contradiction. Assume that Π is not IND-ATK-secure. That is, there exists $\widehat{\mathrm{des}} \in \Omega$ and a ppt adversary $\hat{\mathcal{A}} = \left(\hat{\mathcal{A}}_1, \hat{\mathcal{A}}_2\right) \in \mathbf{A}^{\mathrm{IND}}$ which satisfies the restrictions defined in Lemma 2.4 such that the advantage $\text{Adv-IND-PE}^{\mathrm{ATK}}_{\Pi,\hat{\mathcal{A}}}\left(\lambda, \widehat{\mathrm{des}}\right)$ is non-negligible. Then, $\hat{\mathcal{B}} \in \mathbf{A}^{\mathrm{SS}}$ constructed as above from $\hat{\mathcal{A}}$ has non-negligible advantage under attack scenario ATK. In particular, for every simulator $\hat{\mathcal{B}}'$ of $\hat{\mathcal{B}}$ with respect to $\widehat{\mathrm{des}}$ it holds

$$\text{Adv-SS-PE}^{\mathrm{ATK}}_{\Pi,\hat{\mathcal{B}},\hat{\mathcal{B}}'}\left(\lambda, \widehat{\mathrm{des}}\right) \quad \geq \quad \frac{1}{2} \cdot \text{Adv-IND-PE}^{\mathrm{ATK}}_{\Pi,\hat{\mathcal{A}}}\left(\lambda, \widehat{\mathrm{des}}\right) \quad .$$

Hence, $\text{Adv-SS-PE}^{\mathrm{ATK}}_{\Pi,\hat{\mathcal{B}},\hat{\mathcal{B}}'}\left(\lambda, \widehat{\mathrm{des}}\right)$ is non-negligible and this finally proves the lemma. □

2.2.2. Further Proofs

In this subsection we present formal proofs skipped in the previous presentation.

Proof. (Proof of Lemma 2.4) Let Π and ATK be as in the lemma. Furthermore, let $\mathcal{B} \in \mathbf{A}_{\text{P-KEM}}$ be arbitrary. We construct an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2) \in \mathbf{A}_{\text{P-KEM}}$ which simulates $\mathcal{B}$, has the same advantage and satisfies the conditions defined in the lemma – that is, it never causes the BadQuery event as defined in Subsection 2.1.3.

By construction it holds $\mathcal{A} \in \mathbf{A}_{\text{P-KEM}}$, all queries of $\mathcal{A}$ are syntactically correct, and for every $\mathrm{des} \in \Omega$ it holds $\Pr[\text{BadQuery}] = 0$ in $\text{P-KEM}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})$. Note that $\mathcal{A}$ correctly answers all queries of $\mathcal{B}$, since it directly returns $\perp$ as answer to the queries if and only if the inputs are syntactically incorrect and the oracle query would result in the same output. Next we will analyze the advantage of $\mathcal{A}$.

Let BD be the event in the experiment $\text{P-KEM}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})$ that $\mathcal{A}_1$ outputs $\perp$ due to the illegal challenge index or $\mathcal{A}_2$ aborts the simulation of $\mathcal{B}_2$ and outputs a guess. By construction, event BD occurs if and only if $\mathcal{B}$ causes the event BadQuery in the corresponding experiment $\text{P-KEM}^{\mathrm{ATK}}_{\Pi,\mathcal{B}}(\lambda, \mathrm{des})$. Hence, we deduce that it holds for every $\mathrm{des} \in \Omega$:

$$\begin{aligned} \text{Adv-P-KEM}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) &\overset{\text{by def.}}{=} 2 \cdot \Pr\left[\text{P-KEM}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) = 1\right] - 1 \\ &= 2 \cdot \Big(\Pr\left[\text{P-KEM}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) = 1 \wedge \overline{\mathrm{BD}}\right] \\ &\quad + \Pr\left[\text{P-KEM}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) = 1 \wedge \mathrm{BD}\right] \Big) - 1 \\ &\overset{(*)}{\geq} 2 \cdot \left(\Pr\left[\text{P-KEM}^{\mathrm{ATK}}_{\Pi,\mathcal{B}}(\lambda, \mathrm{des}) = 1 \wedge \overline{\mathrm{BadQuery}}\right] + 0 \right) - 1 \\ &= 2 \cdot \Pr\left[\text{P-KEM}^{\mathrm{ATK}}_{\Pi,\mathcal{B}}(\lambda, \mathrm{des}) = 1\right] - 1 \\ &\overset{\text{by def.}}{=} \text{Adv-P-KEM}^{\mathrm{ATK}}_{\Pi,\mathcal{B}}(\lambda, \mathrm{des}) \quad , \end{aligned}$$

where in the inequality $(*)$ we ignore the success probability in the case of BD. Furthermore, the event BD occurs if and only if the event BadQuery occurs and for all $\mathrm{des} \in \Omega$ the view of $\mathcal{B}$ is as defined in the experiment $\text{P-KEM}^{\mathrm{ATK}}_{\Pi,\mathcal{B}}(\lambda, \mathrm{des})$ if $\mathcal{A}$ does not abort the simulation. The penultimate equation holds since $\text{P-KEM}^{\mathrm{ATK}}_{\Pi,\mathcal{B}}(\lambda, \mathrm{des}) = 1$ implies that the event BadQuery does not occur. □

$\mathcal{A}_1^{\mathbf{O}_1(\mathrm{pp}_\kappa,\mathrm{msk},\cdot)}\left(1^\lambda,\mathrm{pp}_\kappa\right)$:
Simulate $(\mathrm{cInd}^*,St)\leftarrow\mathcal{B}_1^{\mathbf{O}_1(\mathrm{pp}_\kappa,\mathrm{msk},\cdot)}\left(1^\lambda,\mathrm{pp}_\kappa\right)$ using own oracles in order to answer the queries of $\mathcal{B}_1$. Store all corrupted key indices in the set S_{ck}. Furthermore, answer the queries with respect to the following rules:

- Return $\perp$ without querying the own oracle if the input of the queries in not syntactically correct:
  - kInd submitted to the key generation oracle must satisfy $\mathrm{kInd}\in\mathbb{X}_\kappa$.
  - CT submitted to the decapsulation oracle must satisfy $\mathrm{CT}\in\mathbb{C}_{\mathrm{cInd}}\subseteq\mathbb{C}_{\mathrm{pp}_\kappa}$. That is, there must be $\mathrm{cInd}\in\mathbb{Y}_\kappa$ such that $\mathrm{CT}\in\mathbb{C}_{\mathrm{cInd}}$.
  - If $\mathrm{CT}\in\mathbb{C}_{\mathrm{cInd}}$ is submitted to the decapsulation oracle, the key index kInd related to this query must satisfy $\mathrm{R}_\kappa(\mathrm{kInd},\mathrm{cInd})=1$.

Output $\perp$ if the output of $\mathcal{B}_1$ is $\perp$.
Output $\perp$ if there is $\mathrm{kInd}\in S_{ck}$ such that $\mathrm{R}_\kappa(\mathrm{kInd},\mathrm{cInd}^*)=1$. Otherwise output (cInd^*,St).

$\mathcal{A}_2^{\mathbf{O}_2(\mathrm{pp}_\kappa,\mathrm{msk},\cdot)}(\mathrm{K}^*,\mathrm{CT}^*,St)$
Simulate $b'\leftarrow\mathcal{B}_2^{\mathbf{O}_2(\mathrm{pp}_\kappa,\mathrm{msk},\cdot)}(\mathrm{K}^*,\mathrm{CT}^*,St)$ using the own oracles in order to answer the queries of $\mathcal{B}_2$ with respect to the following rules:

- If the input of the queries in not syntactically correct perform as $\mathcal{A}_1$.
- If $\mathcal{B}_2$ queries the key generation oracle for $\mathrm{kInd}\in\mathbb{X}_\kappa$ such that $\mathrm{R}_\kappa(\mathrm{kInd},\mathrm{cInd}^*)=1$ abort the simulation and **output** a guess $b'\leftarrow\{0,1\}$.
- (Only for ATK = CCA2) If $\mathcal{B}_2$ queries the decryption of CT^* (and the key index kInd specified for this query satisfies $\mathrm{R}_\kappa(\mathrm{kInd},\mathrm{cInd}^*)=1$ [a]) abort the simulation and **output** a guess $b'\leftarrow\{0,1\}$.

Output b'.

[a] Already ensured by syntactical checks.

Proof. (Proof of Theorem 2.9) Adaptive semantic security trivially implies semantic security, since every adversary in the adaptive security experiment can easily be adapted to a valid adversary for the semantic security experiment and would have the same success probability. The target function f can be just passed from $\mathcal{A}_1$ to $\mathcal{A}_2$ using St. The other direction is formally proved next.

Let $\mathrm{ATK}\in\{\mathrm{CPA},\mathrm{CCA1},\mathrm{CCA2}\}$ be arbitrary. Assume that there exists a predicate encryption scheme Π which is SS-ATK-secure, but is not adaptively SS-ATK-secure. From the latter we deduce that there exists an adversary $\hat{\mathcal{A}}\in\mathbf{A}^{\mathrm{aSS}}$ with non-negligible advantage. Hence, there exists $\widehat{\mathrm{des}}\in\Omega$ such that for every simulator $\hat{\mathcal{A}}'$ for $\hat{\mathcal{A}}$ with respect to $\widehat{\mathrm{des}}$ the advantage $\mathrm{Adv\text{-}aSS\text{-}PE}^{\mathrm{ATK}}_{\Pi,\hat{\mathcal{A}},\hat{\mathcal{A}}'}\left(\lambda,\widehat{\mathrm{des}}\right)$ is not-negligible. Given such an $\hat{\mathcal{A}}$, we construct an adversary $\mathcal{A}\in\mathbf{A}^{\mathrm{SS}}$ with non-negligible advantage. In particular, $\mathcal{A}$ will be such that for every simulator $\mathcal{A}'$ for $\mathcal{A}$ with respect to $\widehat{\mathrm{des}}$ the advantage $\mathrm{Adv\text{-}SS\text{-}PE}^{\mathrm{ATK}}_{\Pi,\mathcal{A},\mathcal{A}'}\left(\lambda,\widehat{\mathrm{des}}\right)$ is non-negligible.

Let $\hat{\mathcal{A}}$ and $\widehat{\mathrm{des}}$ be as described above. Let E be the event that $\hat{\mathcal{A}}_1$ outputs $\perp$ and $p_{\widehat{\mathrm{des}}}(\lambda)=\Pr\left[\overline{\mathrm{E}}\right]$ be the probability of $\overline{\mathrm{E}}$ in $\mathrm{aSS\text{-}PE}^{\mathrm{ATK}}_{\Pi,\hat{\mathcal{A}}}\left(\lambda,\widehat{\mathrm{des}}\right)$. We deduce that $p_{\widehat{\mathrm{des}}}(\lambda)$ is non-negligible, since otherwise the advantage of $\hat{\mathcal{A}}$ would be negligible. First of all, consider the presented

simulator $\hat{\mathcal{A}}'$ for $\hat{\mathcal{A}}$ with respect to $\widehat{\text{des}}$. We have to show that $\hat{\mathcal{A}}'$ is indeed a simulator of $\hat{\mathcal{A}}$

$\hat{\mathcal{A}}'_1\left(1^\lambda, \widehat{\text{des}}\right)$:

- Compute $(\text{msk}, \text{pp}_\kappa) \leftarrow \text{Setup}\left(1^\lambda, \widehat{\text{des}}\right)$.
- Simulate $\left(\text{cInd}^*, \left(\hat{\mathcal{M}}, h\right), St\right) \leftarrow \hat{\mathcal{A}}_1^{\mathbf{O}_1(\text{pp}_\kappa, \text{msk}, \cdot)}\left(1^\lambda, \text{pp}_\kappa\right)$ using $(\text{msk}, \text{pp}_\kappa)$.
- **Output** $\perp$ if $\mathcal{A}_1$ outputs $\perp$.
- Set $St' := (St, \text{msk}, \text{pp}_\kappa, \text{cInd}^*, \mu)$, where μ is the number of output bits of $\hat{\mathcal{M}}$.
- **Output** $\left(\text{cInd}^*, \left(\hat{\mathcal{M}}, h\right), St'\right)$.

$\hat{\mathcal{A}}'_2\left(h\left(\hat{m}\right), St'\right)$ with $St' = (St, \text{msk}, \text{pp}_\kappa, \text{cInd}^*, \mu)$:

- Compute $\text{CT}' := \text{Enc}\left(\text{pp}_\kappa, \text{cInd}^*, 1^\mu\right)$.
- Simulate $(f, \nu) \leftarrow \hat{\mathcal{A}}_2^{\mathbf{O}_2(\text{pp}_\kappa, \text{msk}, \cdot)}\left(\text{CT}', h\left(\hat{m}\right), St\right)$ using $(\text{msk}, \text{pp}_\kappa)$.
- **Output** the output of $\hat{\mathcal{A}}_2$.

with respect to $\widehat{\text{des}}$. By construction $\hat{\mathcal{A}}'_1$ outputs $\perp$ if and only if $\hat{\mathcal{A}}'$ outputs $\perp$. It remains to show that $\left(\hat{\mathcal{M}}, h, f\right)$ in the real experiment with $\hat{\mathcal{A}}$ is indistinguishable from $\left(\hat{\mathcal{M}}, h, f\right)$ in the simulation experiment with $\hat{\mathcal{A}}'$. It is important to notice that the distribution of the first two elements is identical in both experiments, since $\hat{\mathcal{A}}'$ simulates $\hat{\mathcal{A}}$ perfectly using correctly generated public parameter and the master secret key. Furthermore, $\hat{\mathcal{A}}'_2$ simulates $\hat{\mathcal{A}}$ using CT' generated by $\text{Enc}\left(\text{pp}_\kappa, \text{cInd}^*, 1^{|\hat{m}|}\right)$ instead of $\text{CT}^* = \text{Enc}\left(\text{pp}_\kappa, \text{cInd}^*, \hat{m}\right)$, since $\mu = |\hat{m}|$.

Assume that there is a ppt distinguisher $\mathcal{D}$ for the triples $\left(\hat{\mathcal{M}}, h, f\right)$ generated in the real experiment $\text{aSS-PE}^{\text{ATK}}_{\Pi, \hat{\mathcal{A}}}\left(\lambda, \widehat{\text{des}}\right)$ and in the simulation experiment $\text{aSS-PE-Sim}_{\Pi, \hat{\mathcal{A}}'}\left(\lambda, \widehat{\text{des}}\right)$. More specifically, let $\varepsilon\left(\lambda\right)$ be the non-negligible advantage of $\mathcal{D}$: [1]

$$\begin{aligned}\varepsilon\left(\lambda\right) \;:=\; & \Pr\left[\mathcal{D}\left(\hat{\mathcal{M}}, h, f\right) = 1 \;\middle|\; \overline{\text{E}_{\hat{\mathcal{A}}}} \wedge \text{CT}^* = \text{Enc}\left(\text{pp}_\kappa, \text{cInd}^*, 1^{|\hat{m}|}\right)\right] \\ & - \Pr\left[\mathcal{D}\left(\hat{\mathcal{M}}, h, f\right) = 1 \;\middle|\; \overline{\text{E}_{\hat{\mathcal{A}}}} \wedge \text{CT}^* = \text{Enc}\left(\text{pp}_\kappa, \text{cInd}^*, \hat{m}\right)\right] \;.\end{aligned}$$

We show that this contradicts the semantic security property of Π. Namely, we construct an adversary $\mathcal{B} \in \mathbf{A}^{\text{SS}}$ with non-negligible advantage under the assumption that $\varepsilon\left(\lambda\right)$ is non-negligible. $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2) \in \mathbf{A}^{\text{SS}}$ is as presented in the box. By construction of $\mathcal{B}$ and particularly by construction of h' the view of any simulator $\mathcal{B}' = (\mathcal{B}'_1, \mathcal{B}'_2)$ of $\mathcal{B}$ with respect to $\widehat{\text{des}}$ is independent of the challenge message and by construction of f' it holds

$$\Pr\left[\text{SS-PE-Sim}_{\Pi, \mathcal{B}'}\left(\lambda, \widehat{\text{des}}\right) = 1 \;\middle|\; \overline{\text{E}_{\mathcal{B}'}}\right] \leq \frac{1}{2} \;.$$

[1] If $-\varepsilon(\lambda)$ is non-negligible, just invert the output of $\mathcal{D}$ in the following contraction of $\mathcal{B}$.

$\mathcal{B}_1^{\mathbf{O}_1(\mathrm{pp}_\kappa,\mathrm{msk},\cdot)}\left(1^\lambda,\mathrm{pp}_\kappa\right)$:

- Simulate $\left(\mathrm{cInd}^*,\left(\hat{\mathcal{M}},h\right),St\right)\leftarrow\hat{\mathcal{A}}_1^{\mathbf{O}_1(\mathrm{pp}_\kappa,\mathrm{msk},\cdot)}\left(1^\lambda,\mathrm{pp}_\kappa\right)$ using own oracles.
- **Output** $\perp$ if the output of $\hat{\mathcal{A}}_1$ is $\perp$.
- Choose $\hat{m}\leftarrow\hat{\mathcal{M}}$, set $m_0:=\hat{m}$ and $m_1:=1^{|\hat{m}|}$. Construct a circuit $\hat{\mathcal{M}}'$ which corresponds to the uniform distribution on $\{m_0,m_1\}$. Compute $\hat{h}:=h\left(\hat{m}\right)$. Choose an arbitrary h' such that $h'\left(m_0\right)=h'\left(m_1\right)$. Choose an arbitrary f' such that $f'\left(m_0\right)=0$ and $f'\left(m_1\right)=1$.[a] Set $St':=\left(St,\hat{\mathcal{M}},h,\hat{h}\right)$ and **output** $\left(\mathrm{cInd}^*,\left(\hat{\mathcal{M}}',f',h'\right),St'\right)$.

$\mathcal{B}_2^{\mathbf{O}_2(\mathrm{pp}_\kappa,\mathrm{msk},\cdot)}\left(\mathrm{CT}^*,h\left(\hat{m}'\right),St'\right)$ with $St'=\left(St,\hat{\mathcal{M}},h,\hat{h}\right)$.

- Simulate $(f,\nu)\leftarrow\hat{\mathcal{A}}_2^{\mathbf{O}_2(\mathrm{pp}_\kappa,\mathrm{msk},\cdot)}\left(\mathrm{CT}^*,\hat{h},St\right)$ using the own oracles.[b]
- Simulate $\mathcal{D}$ on input $\left(\hat{\mathcal{M}},h,f\right)$ and **output** the output of $\mathcal{D}$.

[a] We assume w.l.o.g. that $m_0\neq m_1$, which implies $\hat{m}\neq 1^{|\hat{m}|}$. In the case $\hat{m}=1^{|\hat{m}|}$ we could set $f':=h'$ and output the correct value. No simulator can do better in this case. At the same time both distributions in question are the same in this case.

[b] W.l.o.g. we assume that $\hat{\mathcal{A}}_2$ does not cause the BadQuery event, since this can happen only if m_1 is encrypted and $\mathcal{B}$ can output the correct bit. In the proof of Lemma 2.11 we already considered similar case formally.

Now, let us consider the success probability of $\mathcal{B}$:

$$\begin{aligned}
\Pr\left[\text{SS-PE}^{\mathrm{ATK}}_{\Pi,\mathcal{B}}\left(\lambda,\widehat{\mathrm{des}}\right)=1\;\middle|\;\overline{\mathrm{E}_{\mathcal{B}}}\right] &= \Pr\left[\hat{m}'=m_1\wedge\mathcal{D}\left(\hat{\mathcal{M}},h,f\right)=f'\left(\hat{m}'\right)\;\middle|\;\overline{\mathrm{E}_{\mathcal{B}}}\right]\\
&\quad+\Pr\left[\hat{m}'=m_0\wedge\mathcal{D}\left(\hat{\mathcal{M}},h,f\right)=f'\left(\hat{m}'\right)\;\middle|\;\overline{\mathrm{E}_{\mathcal{B}}}\right]\\
&\overset{\hat{\mathcal{M}}',f'}{=}\frac{1}{2}\cdot\left(\Pr\left[\mathcal{D}\left(\hat{\mathcal{M}},h,f\right)=1\;\middle|\;\overline{\mathrm{E}'}\wedge\hat{m}'=1^{|\hat{m}|}\right]\right.\\
&\quad\left.+\Pr\left[\mathcal{D}\left(\hat{\mathcal{M}},h,f\right)=0\;\middle|\;\overline{\mathrm{E}'}\wedge\hat{m}'=\hat{m}\right]\right)\\
&=\frac{1}{2}\cdot\left(\Pr\left[\mathcal{D}\left(\hat{\mathcal{M}},h,f\right)=1\;\middle|\;\overline{\mathrm{E}'}\wedge\hat{m}'=1^{|\hat{m}|}\right]\right.\\
&\quad\left.1-\Pr\left[\mathcal{D}\left(\hat{\mathcal{M}},h,f\right)=1\;\middle|\;\overline{\mathrm{E}'}\wedge\hat{m}'=\hat{m}\right]\right)\\
&=\frac{1}{2}+\frac{1}{2}\cdot\varepsilon\left(\lambda\right)\quad.
\end{aligned}$$

By construction it holds $\Pr\left[\overline{\mathrm{E}_{\mathcal{B}}}\right]=\Pr\left[\overline{\mathrm{E}_{\mathcal{B}}}\right]=\Pr\left[\overline{\mathrm{E}_{\hat{\mathcal{A}}}}\right]=p_{\widehat{\mathrm{des}}}\left(\lambda\right)$. Hence, all together we get

$$\begin{aligned}
\text{Adv-SS-PE}^{\mathrm{ATK}}_{\Pi,\mathcal{B},\mathcal{B}'}\left(\lambda,\mathrm{des}\right) &\overset{(2.2)}{=} p_{\widehat{\mathrm{des}}}\left(\lambda\right)\cdot\left(\Pr\left[\text{SS-PE}^{\mathrm{ATK}}_{\Pi,\mathcal{B}}\left(\lambda,\mathrm{des}\right)=1\;\middle|\;\overline{\mathrm{E}_{\mathcal{B}}}\right]\right.\\
&\quad\left.-\Pr\left[\text{SS-PE-Sim}_{\Pi,\mathcal{B}'}\left(\lambda,\mathrm{des}\right)=1\;\middle|\;\overline{\mathrm{E}_{\mathcal{B}'}}\right]\right)\\
&\geq\frac{1}{2}\cdot p_{\widehat{\mathrm{des}}}\left(\lambda\right)\cdot\varepsilon\left(\lambda\right)\quad,
\end{aligned}$$

where $p_{\widehat{\mathrm{des}}}\left(\lambda\right)$ and $\varepsilon\left(\lambda\right)$ are non-negligible, which contradicts the SS-ATK-security of Π. We deduce that $\hat{\mathcal{A}}'$ is a simulator of $\hat{\mathcal{A}}$ with respect to $\widehat{\mathrm{des}}$, which finalizes the first part of the proof.

By our assumption about $\hat{\mathcal{A}}$ the advantage $\text{Adv-aSS-PE}^{\text{ATK}}_{\Pi,\hat{\mathcal{A}},\hat{\mathcal{A}}'}\left(\lambda,\widehat{\text{des}}\right)$ is non-negligible. Next, we construct $\mathcal{A}=(\mathcal{A}_1,\mathcal{A}_2)\in\mathbf{A}^{\text{SS}}$ such that for every simulator $\mathcal{A}'$ of $\mathcal{A}$ with respect to $\widehat{\text{des}}$ the advantage $\text{Adv-SS-PE}^{\text{ATK}}_{\Pi,\mathcal{A},\mathcal{A}'}\left(\lambda,\widehat{\text{des}}\right)$ is non-negligible. It is easy to verify that $\mathcal{A}\in\mathbf{A}^{\text{SS}}$. By

$\mathcal{A}_1^{\mathbf{O}_1(\text{pp}_\kappa,\text{msk},\cdot)}\left(1^\lambda,\text{pp}_\kappa\right)$:

- Simulate $\left(\text{cInd}^*,\left(\hat{\mathcal{M}},h\right),St\right)\leftarrow\hat{\mathcal{A}}_1^{\mathbf{O}_1(\text{pp}_\kappa,\text{msk},\cdot)}\left(1^\lambda,\text{pp}_\kappa\right)$ using own oracles.
- **Output** $\perp$ if the output of $\hat{\mathcal{A}}_1$ is $\perp$.
- Choose $\hat{m}\leftarrow\hat{\mathcal{M}}$, set $m_0:=\hat{m}$ and $m_1:=1^{|\hat{m}|}$. Construct a circuit $\hat{\mathcal{M}}'$ which corresponds to the uniform distribution on $\{m_0,m_1\}$. Choose an arbitrary h' such that $h'(m_0)=h'(m_1)$. Choose an arbitrary f' such that $f'(m_0)=0$ and $f'(m_1)=1$.[a] Compute $\hat{h}:=h(\hat{m})$, set $St':=\left(St,\hat{h},\hat{m}\right)$ and **output** $\left(\text{cInd}^*,\left(\hat{\mathcal{M}}',f',h'\right),St'\right)$.

$\mathcal{A}_2^{\mathbf{O}_2(\text{pp}_\kappa,\text{msk},\cdot)}\left(\text{CT}^*,h(\hat{m}'),St'\right)$ with $St'=\left(St,\hat{h},\hat{m}\right)$

- Simulate $(f,\nu)\leftarrow\hat{\mathcal{A}}_2^{\mathbf{O}_2(\text{pp}_\kappa,\text{msk},\cdot)}\left(\text{CT}^*,\hat{h},St\right)$ using own oracles.
- **Output** $b':=0$ if $f(\hat{m})=\nu$ and $b':=1$ otherwise.

[a] For simplicity we ignore the case $m_0=m_1$, where $\hat{\mathcal{A}}$ cannot have any advantage. In this case $\mathcal{A}_1$ just sets $f(m_0)=f(m_1)=0$ and $\mathcal{A}_2$ outputs 0. Hence, no simulator of $\mathcal{A}$ can do better in this case.

construction, also for every simulator $\mathcal{A}'$ of $\mathcal{A}$ with respect to $\widehat{\text{des}}$ it holds

$$\Pr\left[\text{SS-PE-Sim}_{\Pi,\mathcal{A}'}\left(\lambda,\widehat{\text{des}}\right)=1\;\middle|\;\overline{\text{E}_{\mathcal{A}'}}\right]\leq\frac{1}{2}\ .$$

Consider the success probability of $\mathcal{A}$. Let $\text{E}_{\mathcal{A}}$ be the event that $\mathcal{A}_1$ outputs $\perp$, which implies $\Pr\left[\overline{\text{E}_{\mathcal{A}}}\right]=p_{\widehat{\text{des}}}(\lambda)$. Analogously to the previous analyzes it holds

$$\begin{aligned}
&\Pr\left[\text{SS-PE}^{\text{ATK}}_{\Pi,\mathcal{A}}\left(\lambda,\widehat{\text{des}}\right)=1\;\middle|\;\overline{\text{E}_{\mathcal{A}}}\right]\\
=\;&\Pr\left[\hat{m}'=m_0\wedge b'=f'\left(\hat{m}'\right)\;\middle|\;\overline{\text{E}_{\mathcal{A}}}\right]+\Pr\left[\hat{m}'=m_1\wedge b'=f'\left(\hat{m}'\right)\;\middle|\;\overline{\text{E}_{\mathcal{A}}}\right]\\
\overset{f,\hat{M}}{=}\;&\frac{1}{2}\cdot\left(\Pr\left[b'=0\;\middle|\;\hat{m}'=m_0\wedge\overline{\text{E}_{\mathcal{A}}}\right]+\Pr\left[b'=1\;\middle|\;\hat{m}'=m_1\wedge\overline{\text{E}_{\mathcal{A}}}\right]\right)\\
=\;&\frac{1}{2}\cdot\Big(\Pr\left[f(\hat{m})=\nu\;\middle|\;\text{CT}^*=\text{Enc}\left(\text{pp}_\kappa,\text{cInd}^*,\hat{m}\right)\wedge\overline{\text{E}_{\mathcal{A}}}\right]\\
&\qquad+1-\Pr\left[f(\hat{m})=\nu\;\middle|\;\text{CT}^*=\text{Enc}\left(\text{pp}_\kappa,\text{cInd}^*,1^{|\hat{m}|}\right)\wedge\overline{\text{E}_{\mathcal{A}}}\right]\Big)\\
\overset{(*)}{=}\;&\frac{1}{2}+\frac{1}{2}\cdot\left(\Pr\left[\text{aSS-PE}^{\text{ATK}}_{\Pi,\hat{\mathcal{A}}}\left(\lambda,\widehat{\text{des}}\right)=1\;\middle|\;\overline{\text{E}_{\hat{\mathcal{A}}}}\right]\right.\\
&\qquad\left.-\Pr\left[\text{aSS-PE-Sim}_{\Pi,\hat{\mathcal{A}}'}\left(\lambda,\widehat{\text{des}}\right)=1\;\middle|\;\overline{\text{E}_{\hat{\mathcal{A}}'}}\right]\right)\\
=\;&\frac{1}{2}+\frac{1}{2\cdot p_{\widehat{\text{des}}}(\lambda)}\cdot\text{Adv-aSS-PE}^{\text{ATK}}_{\Pi,\hat{\mathcal{A}},\hat{\mathcal{A}}'}\left(\lambda,\widehat{\text{des}}\right)\ ,
\end{aligned}$$

where in equation $(*)$ we switch from the probability distribution defined by $\text{SS-PE}^{\text{ATK}}_{\Pi,\mathcal{A}}\left(\lambda,\widehat{\text{des}}\right)$ to the probability distribution defined by $\text{aSS-PE}^{\text{ATK}}_{\Pi,\hat{\mathcal{A}}}\left(\lambda,\widehat{\text{des}}\right)$ conditioned on different computations of the challenge ciphertext CT^*. The last equation can be proved similarly to (2.2).

All together, for every simulator $\mathcal{A}'$ of $\mathcal{A}$ with respect to $\widehat{\text{des}}$ it holds:

$$\begin{aligned}
& \text{Adv-SS-PE}^{\text{ATK}}_{\Pi,\mathcal{A},\mathcal{A}'}\left(\lambda, \widehat{\text{des}}\right) \\
\overset{(2.2)}{=} \; & p_{\widehat{\text{des}}}(\lambda) \cdot \Big(\Pr\left[\text{SS-PE}^{\text{ATK}}_{\Pi,\mathcal{A}}\left(\lambda, \widehat{\text{des}}\right) = 1 \;\middle|\; \overline{\text{E}_{\mathcal{A}}}\right] \\
& \qquad - \Pr\left[\text{SS-PE-Sim}_{\Pi,\mathcal{A}'}\left(\lambda, \widehat{\text{des}}\right) = 1 \;\middle|\; \overline{\text{E}_{\mathcal{A}'}}\right]\Big) \\
\geq \; & \frac{1}{2} \cdot \text{Adv-aSS-PE}^{\text{ATK}}_{\Pi,\hat{\mathcal{A}},\hat{\mathcal{A}}'}\left(\lambda, \widehat{\text{des}}\right)
\end{aligned}$$

This contradicts our precondition that the advantage of $\mathcal{A}$ is negligible. Hence, the statement of the theorem holds. □

3. Subtleties in Security Definitions for PE and P-KEM

In this chapter, based on the security templates from Chapter 2, in Section 3.1 we look at different formalizations regarding the handling of user secret keys. Furthermore, in Section 3.2 we consider security notions originating in the restrictions of adversaries to query the decryption of the challenge ciphertext.

3.1. Handling of User Secret Keys

Whereas in the context of conventional PKE, there is only a single secret key in question, in predicate encryption (PE) schemes there are many user secret keys generated from the same master secret key. Actually, several users may hold (different) keys for the same key index. In the templates from the previous sections we prescinded from details regarding these circumstances. The goal of this section is to consider the different possibilities to handle user secret keys in the security experiments. Indeed, we identify three different formalizations regarding the user secret keys in the literature and name these as follows: one-key model (OK-model), one-use model (OU-model), and covered key model (CK-model). The oracles for these models under CCA2 attacks are presented in Table 3.1. The oracles for CPA and CCA1 are the same with the usual restrictions.

OK	OU	CK

KGen (kInd):
If (kInd, sk) $\in S_k$ return sk;
sk $\leftarrow$ KeyGen (msk, kInd);
S_k.add$\big((\text{kInd}, \text{sk})\big)$;
Return sk;
Decaps (CT, kInd):
sk := **KGen** (kInd);
Return Decaps (sk, CT);
	KGen (kInd):
	sk $\leftarrow$ KeyGen (msk, kInd);
	Return sk;
	Decaps (CT, kInd):
	sk := **KGen** (kInd)
	Return Decaps (sk, CT);
		CKGen (kInd):
		sk $\leftarrow$ KeyGen (msk, kInd);
		i++; S_k.add$\big((i, \text{sk})\big)$;
		Open (i):
		Return sk from $(i, \text{sk}) \in S_k$;
		Decaps (CT, i):
		Return Decaps (sk, CT),
		where $(i, \text{sk}) \in S_k$;

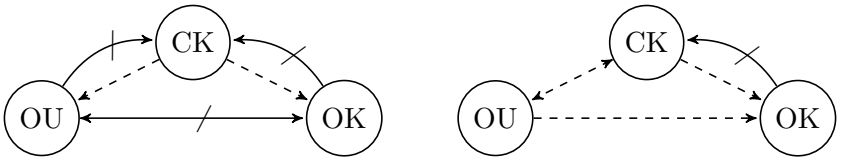
S_k contains all keys which have been generated by **KGen** / **CKGen**.
The oracles for CPA and CCA1 are the same with the usual restrictions.

Table 3.1.: Oracle specification for different models under CCA2 attacks.

In the OK-model the challenger generates and stores a unique secret key for kInd if this index is submitted by $\mathcal{A}$ for the first time. This user secret key is used to answer all oracle queries related to kInd. In particular, oracle query **KGen**(kInd) always results in the same key. The OK-model was previously used in [BF03, KG09, YAHK11, YAS$^+$12]. In the OU-model the challenger generates a new secret key for every query and the generated key will be used only once. This model was previously used in [BH08, KV08]. In the CK-model the adversary specifies not only the key indices, but also the keys which have to be used to answer the decryption queries, which is formalized using the additional covered key generation oracle **CKGen**. The CK-model intuitively reflects the fact that users hold specific secret keys and use their keys for decryption. Hence, adversaries realizing chosen-ciphertext attacks might not only know the access rights of the users (that is, the key indices of their keys), but could also exploit

the fact that the same secret key is used several times. In our constructions in the following parts we explicitly use the CK-model.

In this section we prove that under chosen-ciphertext attacks the OK-model and the OU-model are weaker than the CK-model (cf. Fig. 3.1). We notice that by using the **CKGen** oracle and the **Open** oracle we can simulate the behavior of every adversary in the other two models. Hence, CK-security obviously implies OK-security and OU-security. Furthermore, under CPA the CK-model and the OU-model are equivalent due to the absence of the decryption oracle. All mentioned results hold for PE as well as for P-KEM. We show the separation results for P-KEM, since the constructions in the proofs are a bit more involved in this context.



A barred arrow is a separation. A dashed arrow denotes trivial implication.

Figure 3.1.: Relation between different security models for PE and P-KEM under CCA1 and CCA2 attacks on the left and under CPA attacks on the right.

For convenience, we define sets $\mathbf{A}^{\mathrm{OK}}_{\text{P-KEM}}$, $\mathbf{A}^{\mathrm{OU}}_{\text{P-KEM}}$, $\mathbf{A}^{\mathrm{CK}}_{\text{P-KEM}}$ of adversaries which are as $\mathbf{A}_{\text{P-KEM}}$, but the adversaries use the oracles as defined in the corresponding models. Obviously, the oracles in all three models can be realized using the master secret key and the public parameters as required in Section 2.1. Experiment $\text{P-KEM}^{\mathrm{ATK,mod}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})$ is the same as experiment $\text{P-KEM}^{\mathrm{ATK}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})$ presented in Fig. 2.4 on page 33, except for more specific oracles which are as defined in Table 3.1 for $\mathrm{mod} \in \{\mathrm{OK}, \mathrm{OU}, \mathrm{CK}\}$. Furthermore, at the beginning of the experiment we additionally initialize $S_k := \emptyset$ and $i := 0$.

Definition 3.1. Let $\mathrm{mod} \in \{\mathrm{OK}, \mathrm{OU}, \mathrm{CK}\}$ and $\mathrm{ATK} \in \{\mathrm{CPA}, \mathrm{CCA1}, \mathrm{CCA2}\}$. A P-KEM Π with public index for predicate family $\mathcal{R}_{\Omega,\Sigma}$ and a family of key spaces $\mathcal{K}$ is called secure in model mod under attack ATK (mod-ATK-secure) if for every $\mathrm{des} \in \Omega$ and every ppt adversary $\mathcal{A} \in \mathbf{A}^{\mathrm{mod}}_{\Pi,\text{P-KEM}}$, the advantage with respect to des defined by

$$\text{Adv-P-KEM}^{\mathrm{ATK,mod}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) := 2 \cdot \Pr\left[\text{P-KEM}^{\mathrm{ATK,mod}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) = 1\right] - 1$$

is negligible.

In the following subsections we usually write mod-secure, instead of mod-ATK-secure, when the attack scenario is obvious from the context.

3.1.1. OK-Security Does Not Imply OU-Security and CK-Security

In this subsection we construct an OK-secure scheme that is neither OU-secure nor CK-secure. The attack scenario is not relevant for this construction. We start from an OK-secure scheme and assume existence of pseudorandom functions.

Let $\mathcal{R}_{\Omega,\Sigma}$ be an arbitrary predicate family, Π be a P-KEM for $\mathcal{R}_{\Omega,\Sigma}$. Furthermore, let $\mathcal{PRF}$ be a family of pseudorandom functions as defined in Subsection 1.1.1.

Theorem 3.2. *Let* ATK $\in \{\mathrm{CPA}, \mathrm{CCA1}, \mathrm{CCA2}\}$ *be an attack scenario and* $\mathcal{PRF}$ *be a family of* PRF*s. Suppose* Π *is an* OK-ATK-*secure* P-KEM *for predicate family* $\mathcal{R}_{\Omega,\Sigma}$ *and family of key spaces* $\mathcal{K}$*. Then, there exists an* OK-ATK-*secure* P-KEM *scheme* Π' *for* $\mathcal{R}_{\Omega,\Sigma}$ *and* $\mathcal{K}$ *which is neither* OU-ATK-*secure nor* CK-ATK-*secure.*

Proof. Let $\Pi = (\mathrm{Setup}, \mathrm{KeyGen}, \mathrm{Encaps}, \mathrm{Decaps})$ and $\mathcal{PRF}$ be as defined in the theorem. Furthermore, let $\langle\cdot\rangle$ be any canonical binary representation of the master secret keys. W.l.o.g. we assume that for every $\mathrm{des} \in \Omega$ and every $(\mathrm{msk}, \_) \in \left[\mathrm{Setup}\left(1^\lambda, \mathrm{des}\right)\right]$ it holds $|\langle\mathrm{msk}\rangle| = \mathrm{MKLen}(\lambda)$ where $\mathrm{MKLen}(\cdot)$ is a polynomial representing the output length of $\mathcal{PRF}$. The pseudorandom function will be applied to key indices and we additionally require arbitrary canonical binary representation of these indices. Recall that for simplicity the random choice of a pseudorandom function with key length λ is denoted by $f \leftarrow \mathcal{PRF}\left(1^\lambda\right)$.

P-KEM $\Pi' = (\mathrm{Setup}', \mathrm{KeyGen}', \mathrm{Encaps}', \mathrm{Decaps}')$ is defined as follows:

- $\mathrm{Setup}'\left(1^\lambda, \mathrm{des}\right)$: generate $(\mathrm{msk}, \mathrm{pp}_\kappa) \leftarrow \mathrm{Setup}\left(1^\lambda, \mathrm{des}\right)$. Choose a pseudorandom function $f \leftarrow \mathcal{PRF}\left(1^\lambda\right)$ and output $\mathrm{msk}' := (\mathrm{msk}, f)$ and pp_κ.
- $\mathrm{KeyGen}'\left(\mathrm{msk}', \mathrm{kInd}\right)$ for $\mathrm{kInd} \in \mathbb{X}_\kappa$ and $\mathrm{msk}' = (\mathrm{msk}, f)$: generate a user secret key $\mathrm{sk} \leftarrow \mathrm{KeyGen}(\mathrm{msk}, \mathrm{kInd})$, choose a bit $b \leftarrow \{0,1\}$, set

$$\mathrm{rand} := \begin{cases} f(\mathrm{kInd}) & \text{if } b = 0 \\ f(\mathrm{kInd}) \oplus \langle\mathrm{msk}\rangle & \text{if } b = 1 \end{cases},$$

  and output $\mathrm{sk}' := (\mathrm{sk}, \mathrm{rand})$.
- $\mathrm{Encaps}'(\mathrm{cInd}) = \mathrm{Encaps}(\mathrm{cInd})$.
- $\mathrm{Decaps}'\left(\mathrm{sk}', \mathrm{CT}\right)$ for $\mathrm{sk}' = (\mathrm{sk}, \mathrm{rand})$ returns $\mathrm{Decaps}(\mathrm{sk}, \mathrm{CT})$.

Scheme Π' is trivially broken in the OU-model and in the CK-model, where the adversary may get several keys for the same key index, and hence learns the master secret key. At the same time Π' remains OK-secure, since the adversary receives for every kInd either $f(\mathrm{kInd})$ or $f(\mathrm{kInd}) \oplus \langle\mathrm{msk}\rangle$, and hence due to the property of the pseudorandom function these values are useless for $\mathcal{A}$. To prove this formally it is sufficient to consider an imaginary scheme where f is replaced by a truly random function. For such a scheme it is clear that the additional values in the user secret keys are independent of the master secret key, and hence these values are useless for adversary. Consequently, the scheme is OK-secure due to the security property of Π. Furthermore, no ppt adversary can distinguish between this imaginary scheme and the scheme Π', since otherwise there would be a ppt distinguisher for $\mathcal{PRF}$. □

The construction in the proof reveals the weakness of the OK-model even though the scheme is artificial. Namely, the OK-model does not cover the fact that several keys for the same key index exist. At least potentially, user secret keys for the same key index can leak more information about the master secret key than user secret keys for different key indices. Hence, already under CPA the OK-model makes unwarranted restrictions of adversarial abilities which might cause security issues.

3.1.2. OU-Security Does Not Imply OK-Security and CK-Security Under CCA

In this subsection we consider only chosen-ciphertext attack and construct an OU-secure scheme that is neither OK-secure nor CK-secure.

Theorem 3.3. *Let* $\mathrm{ATK} \in \{\mathrm{CCA1}, \mathrm{CCA2}\}$ *be an attack scenario. Suppose* Π *is an* OU-ATK*-secure* P-KEM *for predicate family* $\mathcal{R}_{\Omega,\Sigma}$ *and family of key spaces* $\mathcal{K} = \{\mathbb{K}_\lambda\}$*. Then, there exists an* OU-ATK*-secure* P-KEM *scheme* Π' *for* $\mathcal{R}_{\Omega,\Sigma}$ *and* $\mathcal{K}$ *which is neither* OK-ATK*-secure nor* CK-ATK*-secure.*

Proof. Let $\mathbb{K}_\lambda = \{0,1\}^{\mathrm{KLen}(\lambda)}$ for polynomial $\mathrm{KLen}(\lambda)$. In the following construction of Π' we first assume that for all $\lambda \in \mathbb{N}$, all $(\mathrm{msk}, \mathrm{pp}_\kappa) \in \left[\mathrm{Setup}\left(1^\lambda, \mathrm{des}\right)\right]$, all $\mathrm{kInd} \in \mathbb{X}_\kappa$ and all $\mathrm{sk} \in [\mathrm{KeyGen}(\mathrm{msk}, \mathrm{kInd})]$ it holds $|\langle \mathrm{sk} \rangle| = \mathrm{KLen}(\lambda)$, where $\langle \cdot \rangle$ is any canonical representation of the user secret keys. This enhances the perspicuity of the presented construction. Below, we explain how to drop this assumption.

P-KEM $\Pi' = (\mathrm{Setup}', \mathrm{KeyGen}', \mathrm{Encaps}', \mathrm{Decaps}')$ is defined as follows:

- $\mathrm{Setup}'\left(1^\lambda, \mathrm{des}\right) = \mathrm{Setup}\left(1^\lambda, \mathrm{des}\right)$.
- $\mathrm{KeyGen}'(\mathrm{msk}, \mathrm{kInd})$: generate a user secret key $\mathrm{sk} \leftarrow \mathrm{KeyGen}(\mathrm{msk}, \mathrm{kInd})$, choose a bit string $r \leftarrow \{0,1\}^{|\langle \mathrm{sk} \rangle|}$, output $\mathrm{sk}' := (\mathrm{sk}, r)$.
- $\mathrm{Encaps}'(\mathrm{cInd})$: generate $(\mathrm{CT}, \mathrm{K}) \leftarrow \mathrm{Encaps}(\mathrm{cInd})$ set $\mathrm{CT}' = 00 \| \mathrm{CT}$ and output $(\mathrm{CT}', \mathrm{K})$.
- $\mathrm{Decaps}'\left(\mathrm{sk}', \mathrm{CT}'\right)$: parse $\mathrm{CT}' = b_1 b_2 \| \mathrm{CT}$, where $b_1, b_2 \in \{0,1\}$ and $\mathrm{sk}' = (\mathrm{sk}, r)$. Output

$$\mathrm{K} = \begin{cases} \mathrm{Decaps}(\mathrm{sk}, \mathrm{CT}) & \text{if } b_1 = b_2 = 0 \\ r & \text{if } b_1 = 1 \wedge b_2 = 0 \\ r \oplus \langle \mathrm{sk} \rangle & \text{if } b_1 = b_2 = 1 \\ \perp & \text{otherwise} \end{cases}.$$

Π' is OU-secure since Π is OU-secure and using the decapsulation oracle with $b_1 \neq 0 \vee b_2 \neq 0$ the adversary will be able to learn either r or $r \oplus \langle \mathrm{sk} \rangle$ which on their own are useless, since uniformly and independently distributed by the choice of r. This is due to the fact that every key is used only once in OU-model. In the other two security models the adversary can get every user secret key using only two decapsulation queries and can trivially break the scheme.

Now we explain how to drop our assumption from the beginning of the proof. Indeed, it is sufficient to have a single key index kInd such that for all $\mathrm{sk} \in [\mathrm{KeyGen}(\mathrm{msk}, \mathrm{kInd})]$ it holds $|\langle \mathrm{sk} \rangle| \leq l(\lambda)$ for some polynomial $l(\lambda)$, which can be assumed w.l.o.g. If $l > \mathrm{KLen}(\lambda)$, we extend the encapsulation by $\left\lceil \log\left(\frac{l(\lambda)}{\mathrm{KLen}(\lambda)}\right)\right\rceil + 2$ bits such that the first bit encodes if the encapsulation is correct ($b_1 = 0$) or not ($b_1 = 1$), the second bit encodes if r or $r \oplus \langle \mathrm{sk} \rangle$ should be used, and the following bits encode the number of the block which should be returned. That is, if $b_1 = 1$, $b_2 = 0$ and $b_3 b_4 \ldots b_l = t$, the output will be the t's block of $\langle r \rangle$, where every block is of size $\mathrm{KLen}(\lambda)$. Then, using $2 \cdot (l-1)$ many queries one can get the key for kInd and break the scheme. □

In the case of PE schemes with message space $\mathcal{M} = \{0,1\}^*$ we must not take care about the length of $\langle \mathrm{sk} \rangle$ and can use the construction in the proof with two additional bits in the ciphertexts.

Even though the scheme Π' from the proof is artificial, it shows the main weakness of the OU-model. Namely, the model does not ensure that the decryption oracle does not leak partial information about the used user secret key if queried with an malformed ciphertext. This kind of partial information is difficult to exploit but might cause security issues.

3.1.3. Discussion

In this section we discuss the security of known schemes. As mentioned in the introduction to this part of the thesis, most known PE schemes are proved to be secure in the OK-security model or in the OU-security model. Hence, we examine the CK-CCA2-security of the above mentioned schemes and summarize the results in Table 3.2.

Construction	Type	Model used	CK-secure
[BF03]	IBE	OK	YES
[BCHK07]	(H)IBE	OK	YES
[BH08]	(H)IBE	OU	YES
[KG09] (explicit check)	IB-KEM	OK	YES
[KG09] (implicit check)	IB-KEM	OK	YES
[KV08] (scheme I)	IBE	OU	YES
[KV08] (scheme II)	IBE	OU	YES
[OT10a]	PE	OU	?
[YAHK11] (from verifyability)	ABE	OK	YES
[YAHK11] (from delegation)	ABE	OK	?
[YAS$^+$12]	PE	OK	YES

Table 3.2.: CK-CCA2-security for PE schemes proved to be OU-CCA2-secure or OK-CCA2-secure.

In the case of IBE, the key index is the identity of the user, and hence it is often (implicitly) assumed that for every key index there is a unique secret key. Furthermore, in the (H)IBE schemes from [BF03, BCHK07] the keys are unique by construction, and hence all three security notions are identical for these schemes. Usually, though, IBE schemes do not have unique user secret keys (cf. [Wat05, BH08, KG09, KV08]). As already mentioned in the introduction to this part, in [BCHK07] the authors assume that the key generation algorithm is deterministic. The CPA-to-CCA transformation from [BCHK07] was proposed in [BH08] in order to achieve CCA-security for their (selectively) CPA-secure scheme. Even though the key generation algorithm in [BH08] is not deterministic, the user secret keys are randomized for every decryption and the generated key is distributed as a freshly generated key. This ensures that all three models do not differ for the resulting CCA-secure scheme. The public verifiability of the first scheme in [KG09] ensures that the output of the decryption algorithm executed with different secret keys is the same, since malformed ciphertexts are explicitly rejected. The second scheme from [KG09] uses fresh randomness during the decryption and rejects malformed ciphertexts with overwhelming probability independently of the used key. The schemes in [KV08] ensure that the malformed ciphertexts are rejected with overwhelming probability due to the authenticated symmetric encryption. Furthermore, the generic transformations from CPA to CCA2 secure schemes for attribute-based encryption (ABE) [YAHK11] and for predicate encryption [YAS$^+$12] require the existence of verification algorithms that ensure that the output of the decryption algorithm is independent of the used secret key. For the remaining schemes from [OT10a, YAHK11] it is at least not trivial to argue from the original proofs if the CK-security notion is satisfied or not. We leave this as an open question.

We remark that for most known PE schemes the correct form of the ciphertexts cannot be efficiently checked. This is due to the dual system encryption methodology [Wat09a, LW10, LW12] used to construct most known adaptively secure PE schemes. In the schemes from this technique there exist incorrectly formed ciphertexts which are indistinguishable from correctly generated ciphertexts. Hence, one cannot simply reject malformed ciphertext as in [KG09] in order to achieve CK-security.

Due to the results of this section regarding CCA2 security we encourage to use the CK-model in order to specify the security guarantees of the schemes precisely. If the CPA attacks are considered we recommend to use the simpler OU-model.

3.2. When and How to Restrict Challenge Decryption

In this section we consider possible restrictions of adversarial abilities regarding the decryption of the challenge ciphertext under adaptive chosen-ciphertext attacks (CCA2). We formally analyze the corresponding security notions in order to prevent mistakes and misunderstanding as previously made in the literature in the context of PKE, as described in [BHK15].

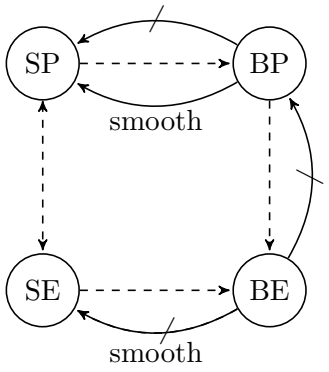
3.2.1. Valid Adversaries and Security Notions

Due to the results in Section 3.1, in this section we only consider the covered key model (CK-model). Furthermore, we mainly consider P-KEM and first of all redefine the set $\mathbf{A}_{\text{P-KEM}}$ of valid adversaries according to the syntax of CK-CCA2-model and using the results from Subsection 2.1.3.

$\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2) \in \mathbf{A}_{\text{P-KEM}}$ if and only if the following conditions are satisfied:

- Given public parameters pp_κ the algorithm $\mathcal{A}_1$ outputs the error symbol $\perp$ or a tuple $(\text{cInd}^*, \text{St})$ such that $\text{cInd}^* \in \mathbb{Y}_\kappa$. The output of $\mathcal{A}_2$ is a bit.
- $\mathcal{A}_1$ and $\mathcal{A}_2$ query the oracle **CKGen** only on $\text{kInd} \in \mathbb{X}_\kappa$.
- $\mathcal{A}_1$ and $\mathcal{A}_2$ submit index i to oracles **Open** and **Decaps** only after the i'th query to the **CKGen** oracle.
- For every (CT, i) submitted to the decapsulation oracle by $\mathcal{A}_1$ and $\mathcal{A}_2$ it holds $\text{CT} \in \mathbb{C}_{\text{cInd}} \subseteq \mathbb{C}_{\text{pp}_\kappa}$ and $\text{R}_\kappa(\text{kInd}, \text{cInd}) = 1$, where kInd is the key index submitted during the i'th covered key generation query.
- For every corrupted key index kInd it holds $\text{R}_\kappa(\text{kInd}, \text{cInd}^*) = 0$.

We dropped the last restriction of Lemma 2.4. The restriction regarding the decapsulation query on the challenge encapsulation will be considered separately in this section.



The continuous arrows denote implications and the barred arrows denote separations. The dashed arrows denote trivial implications.

Figure 3.2.: Relation between different security models for PE.

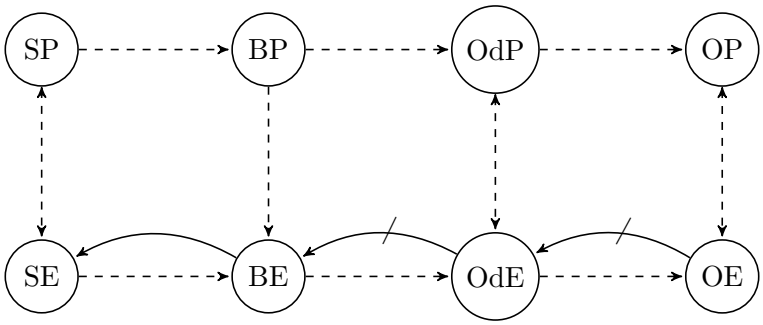
Let us shortly recall four different security notions for PKE identified and formalized in the article [BHK15]. According to this work there are two dimensions in the definition of

CCA2-security regarding the restrictions of adversaries to query the decryption of the challenge ciphertext. The first dimension specifies the style of the restrictions. The authors differentiate between the penalty style definitions (denoted by P) and the exclusion style definition (denoted by E). So far we have used the former style in this work. That is, an adversary which violates restrictions of the security experiment is penalized at the end of the experiment. In the exclusion style definitions, the set of adversaries is restricted from the beginning such that for every considered adversary the probability that the restrictions are violated is equal zero. Furthermore, we can disallow the adversary to query the decryption of the challenge ciphertext only in the second query phase (denoted by S) or in both query phases (denoted by B). That is, either $\mathcal{A}_2$ is not allowed to query the decryption of CT^*, or $\mathcal{A}_1$ and $\mathcal{A}_2$ are not allowed to make such a query. As result, we have four different security notions denoted by SP, SE, BP, and BE.

For PE we can prove the same relations between mentioned notions as for conventional PKE (cf. Fig. 3.2), which is not surprising and was stated without a proof in [BHK15] for IBE schemes. The proof ideas are the same as for PKE even though some non-trivial extensions are required.

In the context of conventional KEM, two additional notions have been considered in [BHK15]. Namely, the first query phase can be completely dropped mainly due to the fact that the adversary cannot influence the generated challenge. This results in two additional security notions denoted by OP and OE, where "O" stands for "One phase". Indeed, for conventional KEM all six security notions are equivalent [BHK15].

P-KEM substantially differs from conventional KEM, since the adversary might be able to influence the choice of the ciphertext index. We indeed prove that the corresponding security notions are not all equal due to this property. First of all, because of the key generation oracle we consider two additional security notions, where the adversary has access to this oracle in both phases, but the decapsulation oracle is available only in the second query phase. The corresponding penalty style security notion is denoted by OdP and the exclusion style notion is denoted by OdE. Here "Od" stands for a "One decapsulation" phase.



The continuous arrows denote implications and the barred arrows denote separations. The dashed arrows denote trivial implications.

Figure 3.3.: Relation between different security models for P-KEM.

We prove that the OdE and the OdP security notions are weaker than the BE security notion (cf. Fig. 3.3). The one-phase security notions OE and OP are even weaker. We also prove that the other four security notions (SP, SE, BP, and BE) are equivalent for P-KEM. Nevertheless, the reductions are not all tight and we advice against both BP and BE security models. The main difference between KEM and P-KEM is in the fact that whereas security of conventional KEM implies smoothness [BHK15], this is not the case for P-KEM. Rather, BE-security implies that every ppt algorithm has only a negligible advantage in finding a ciphertext index with only few possible encapsulations, which we call a **weak ciphertext** index.

In order to formalize the exclusion style definitions we define restricted sets of adversaries. Furthermore, we keep a uniform security experiment and formalize the required restrictions in the first query phase by appropriate restrictions of the adversaries. Let S_1 and S_2 be the sets of encapsulations submitted to the decapsulation oracle by $\mathcal{A}_1$ and by $\mathcal{A}_2$, respectively. Furthermore, let SK_1 be the set of key indices corrupted by $\mathcal{A}_1$. For every security model $\text{mod} \in \{\text{SP}, \text{BP}, \text{OdP}, \text{OP}, \text{SE}, \text{BE}, \text{OdE}, \text{OE}\}$ we define a set of adversaries denoted by $\mathbf{A}^{\text{mod}}_{\text{P-KEM}}$. Namely, for every adversary $\mathcal{A} \in \mathbf{A}_{\text{P-KEM}}$ we define:

$$\mathcal{A} \in \mathbf{A}^{\text{SE}}_{\text{P-KEM}} \text{ iff } \Pr\left[\text{CT}^* \notin S_2\right] = 1 \ , \quad \mathcal{A} \in \mathbf{A}^{\text{BE}}_{\text{P-KEM}} \text{ iff } \Pr\left[\text{CT}^* \notin S_1 \cup S_2\right] = 1 \ ,$$

$$\mathcal{A} \in \mathbf{A}^{\text{OdP}}_{\text{P-KEM}} \text{ iff } \Pr\left[S_1 = \emptyset\right] = 1 \ , \quad \mathcal{A} \in \mathbf{A}^{\text{OdE}}_{\text{P-KEM}} \text{ iff } \Pr\left[S_1 = \emptyset \wedge \text{CT}^* \notin S_2\right] = 1 \ ,$$

$$\mathcal{A} \in \mathbf{A}^{\text{OP}}_{\text{P-KEM}} \text{ iff } \Pr\left[SK_1 = \emptyset \wedge S_1 = \emptyset\right] = 1 \ , \quad \text{and}$$

$$\mathcal{A} \in \mathbf{A}^{\text{OE}}_{\text{P-KEM}} \text{ iff } \Pr\left[SK_1 = \emptyset \wedge S_1 = \emptyset \wedge \text{CT}^* \notin S_2\right] = 1 \ .$$

Furthermore, we define $\mathbf{A}^{\text{SP}}_{\text{P-KEM}} = \mathbf{A}^{\text{BP}}_{\text{P-KEM}} = \mathbf{A}_{\text{P-KEM}}$.

Notice that the exclusion/penalty style of the definitions only refer to the restriction regarding the decapsulation queries on the challenge encapsulation. The additional restrictions regarding the first query phase are all in exclusion style. This is only due to our goal of uniform templates. Alternatively, we could define extra probability experiments with restricted first query phase or without the first query phase at all.

3.2.2. CCA2-Security Template for P-KEM

Let $\text{mod} \in \{\text{SP}, \text{BP}, \text{OdP}, \text{OP}, \text{SE}, \text{BE}, \text{OdE}, \text{OE}\}$, $\mathcal{R}_{\Omega,\Sigma}$ be a predicate family, $\mathcal{K} = \{\mathbb{K}_\lambda\}_{\lambda\in\mathbb{N}}$ be a family of key spaces, and Π be a P-KEM for $\mathcal{R}_{\Omega,\Sigma}$ and $\mathcal{K}$. CCA2-security experiment $\text{P-KEM}^{\text{CCA2,mod}}_{\Pi,\mathcal{A}}(\lambda, \text{des})$ is defined in Fig. 3.4.

$\mathbf{P\text{-}KEM}^{\text{CCA2,mod}}_{\Pi,\mathcal{A}}(\lambda, \text{des})$:
$b \leftarrow \{0,1\}\,; S_1, S_2, S_k \leftarrow \emptyset; i := 0;$
$(\text{msk}, \text{pp}_\kappa) \leftarrow \text{Setup}\left(1^\lambda, \text{des}\right);$
$(\text{cInd}^*, St) \leftarrow \mathcal{A}_1^{\mathbf{CKGen}(\cdot),\mathbf{Open}(\cdot),\mathbf{Decaps}_1(\cdot,\cdot)}\left(1^\lambda, \text{pp}_\kappa\right);$
Output b if the output of $\mathcal{A}_1$ is $\perp$;
$(\text{K}_0, \text{CT}^*) \leftarrow \text{Enc}\left(\text{pp}_\kappa, \text{cInd}^*\right); \text{K}_1 \leftarrow \mathbb{K}_\lambda; \text{K}^* := \text{K}_b;$
$b' \leftarrow \mathcal{A}_2^{\mathbf{CKGen}(\cdot),\mathbf{Open}(\cdot),\mathbf{Decaps}_2(\cdot,\cdot)}\left(\text{K}^*, \text{CT}^*, St\right);$
Output :
for SE, BE, OdE, OE : $b' = b$;
for SP, OdP, OP : $b' = b \wedge (\text{CT}^* \notin S_2)\,;$
for BP : $b' = b \wedge (\text{CT}^* \notin S_1 \cup S_2)\,;$

Figure 3.4.: CCA2-security experiment for different security notions of P-KEM.

The oracles are as defined in Table 3.1 except for decapsulation oracle which additionally stores all queried ciphertexts in S_1 or in S_2 in the first phase or in the second phase, respectively.

Definition 3.4. Let Π be a P-KEM with public index for predicate family $\mathcal{R}_{\Omega,\Sigma}$ and mod $\in \{\mathrm{SP}, \mathrm{BP}, \mathrm{OdP}, \mathrm{OP}, \mathrm{SE}, \mathrm{BE}, \mathrm{OdE}, \mathrm{OE}\}$ be a security model. Π is called **secure under adaptive chosen-ciphertext attacks in model** mod (mod-secure) if for every des $\in \Omega$ the advantage of $\mathcal{A} \in \mathbf{A}^{\mathrm{mod}}_{\text{P-KEM}}$ with respect to des, defined by

$$\text{Adv-P-KEM}^{\mathrm{CCA2,mod}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) := 2 \cdot \Pr\left[\text{P-KEM}^{\mathrm{CCA2,mod}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) = 1\right] - 1$$

is negligible.

3.2.3. Separation Results and Implication Results

In this subsection we prove the implication results and the separation results mentioned in Fig. 3.3. It is easy to verify that the four top down and the six rightwards relations hold mainly by the definition of the experiments and by the definitions of the corresponding sets of adversaries. Furthermore, three implications between the exclusion and the corresponding penalty notions are covered by Remark 2.5 from Subsection 2.1.3. The relation BE $\rightarrow$ BP is not covered, since in the first query phase the challenge ciphertext CT^* is not defined, and hence one cannot verify if encapsulation CT submitted by adversary to the decapsulation oracle violates the restriction $\mathrm{CT} \neq \mathrm{CT}^*$.

Notice that for both separation results we present counterexamples based on concrete predicate families. Alternatively, one could also define properties for predicate families which are sufficient to apply constructions in the corresponding proofs. We do not present these properties, since they do not provide significant additional benefit. We remark that for the first construction an exponentially large (in security parameter λ) ciphertext index space is sufficient, whereas for the second construction certain structure on the ciphertext index space is required.

OE-Security Does Not Imply OdE-Security

In this subsection we show that OE-security notion is weaker then the OdE-security notion. The difference between the OE-security notion and the OdE-security notion is in the first query phase. Whereas in the OE-model this phase does not exist at all, in the OdE-model the adversary is allowed to corrupt user secret keys. To prove the stated separation, we present a counterexample based on a natural predicate family, which points out the weakness of the one phase notions for P-KEMs.

Consider a predicate family $\mathcal{R}$ for equality predicate family with $\mathbb{X}_\kappa = \mathbb{Y}_\kappa = \{0,1\}^n$, where $n = \lambda$ and λ is the security parameter which determines $\sigma \in \Sigma$, $\sigma \in \kappa$. We recall that by our convention it holds $|\sigma| = \lambda$. The predicate is defined by: $\mathrm{R}_\kappa(\mathrm{kInd}, \mathrm{cInd}) = 1$ if and only if kInd = cInd and corresponds to identity-based schemes. Furthermore, let $\mathcal{F}$ be a family of injective one-way functions for domain $D = \{0,1\}^{|s|}$, where s is the key for the function. Let $\text{Adv-Invert}_{\mathcal{F},\mathcal{I}}(\lambda)$ be the advantage of algorithm $\mathcal{I}$ in experiment $\mathrm{Invert}_{f,\mathcal{I}}(\lambda)$ as defined in Subsection 1.1.2. That is, $\mathcal{I}$ has to compute a preimage of y in $\{0,1\}^\lambda$ defined by $f \leftarrow \mathcal{F}\left(1^\lambda\right)$, $x \leftarrow \{0,1\}^\lambda$, $y := f(x)$.

Theorem 3.5. *Suppose $\mathcal{F}$ is a family of injective one-way functions and Π is an* OE*-secure* P-KEM *for predicate family $\mathcal{R}$ and a family of key spaces $\mathcal{K}$. Then, there exists a* P-KEM *Π' for $\mathcal{R}$ and $\mathcal{K}$ which is not* OdE*-secure but is* OE*-secure. In particular, for every* des $\in \Omega$ *and*

every ppt $\mathcal{A} \in \mathbf{A}^{\mathrm{OE}}_{\Pi',\text{P-KEM}}$ *there is a ppt* $\mathcal{B} \in \mathbf{A}^{\mathrm{OE}}_{\Pi,\text{P-KEM}}$ *and a ppt inverter* $\mathcal{I}$ *such that it holds*

$$\text{Adv-P-KEM}^{\mathrm{CCA2,OE}}_{\Pi',\mathcal{A}}(\lambda, \mathrm{des}) \leq \text{Adv-P-KEM}^{\mathrm{CCA2,OE}}_{\Pi,\mathcal{B}}(\lambda, \mathrm{des}) + \text{Adv-Invert}_{\mathcal{F},\mathcal{I}}(\lambda) \quad .$$

Proof. Let $\Pi = (\mathrm{Setup}, \mathrm{KeyGen}, \mathrm{Encaps}, \mathrm{Decaps})$ be as defined in the theorem, $\mathcal{K} = \{\mathbb{K}_\lambda\}_{\lambda\in\mathbb{N}}$.

$\Pi' = (\mathrm{Setup}', \mathrm{KeyGen}', \mathrm{Encaps}', \mathrm{Decaps}')$ as follows:

Setup$'\left(1^\lambda, \mathrm{des}\right)$: Generate $(\mathrm{pp}_\kappa, \mathrm{msk}) \leftarrow \mathrm{Setup}\left(1^\lambda, \mathrm{des}\right)$. Choose function $f \leftarrow \mathcal{F}\left(1^\lambda\right)$, index $\mathrm{cInd}_w \leftarrow \{0,1\}^\lambda$, and compute $Y := f(\mathrm{cInd}_w)$. Choose $\mathrm{K}_w \leftarrow \mathbb{K}_\lambda$, set $\mathrm{pp}'_\kappa := (\mathrm{pp}_\kappa, f, \mathrm{K}_w, Y)$, and $\mathrm{msk}' = (\mathrm{msk}, \mathrm{cInd}_w)$. Output $\left(\mathrm{pp}'_\kappa, \mathrm{msk}'\right)$.

KeyGen$'\left(\mathrm{pp}'_\kappa, \mathrm{msk}', \mathrm{kInd}\right)$: Parse $\mathrm{pp}'_\kappa = (\mathrm{pp}_\kappa, f, \mathrm{K}_w, Y)$ and $\mathrm{msk}' = (\mathrm{msk}, \mathrm{cInd}_w)$. Generate a secret key $\mathrm{sk} \leftarrow \mathrm{KeyGen}(\mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{kInd})$ and output $\mathrm{sk}' := (\mathrm{sk}, \mathrm{cInd}_w)$.

Encaps$'(\mathrm{pp}'_\kappa, \mathrm{cInd})$: Parse $\mathrm{pp}'_\kappa := (\mathrm{pp}_\kappa, f, \mathrm{K}_w, Y)$. If $f(\mathrm{cInd}) = Y$ output key K_w and encapsulation $\mathrm{CT}' = 0\|\left(\mathrm{cInd}, 1^\lambda\right)$. Otherwise compute $(\mathrm{K}, \mathrm{CT}) \leftarrow \mathrm{Encaps}(\mathrm{pp}_\kappa, \mathrm{cInd})$, set $\mathrm{CT}' := 1\|\mathrm{CT}$ and output $(\mathrm{K}, \mathrm{CT}')$.

Decaps$'\left(\mathrm{pp}'_\kappa, \mathrm{CT}', \mathrm{sk}'\right)$: Parse $\mathrm{pp}'_\kappa := (\mathrm{pp}_\kappa, f, \mathrm{K}_w, Y)$, $\mathrm{CT}' = c\|\mathrm{CT}$, and $\mathrm{sk}' = (\mathrm{sk}, \mathrm{cInd}_w)$. Output K_w if $\mathrm{CT}' = 0\|\left(\mathrm{cInd}, 1^\lambda\right)$ and $f(\mathrm{cInd}) = Y$. If $c = 1$, output $\mathrm{Decaps}(\mathrm{pp}_\kappa, \mathrm{sk}, \mathrm{CT})$. Otherwise output $\perp$.

Obviously, Π' is not OdE-secure since every secret key reveals cInd_w and the challenge for this ciphertext index can be easily solved.

Next we argue that Π' is OE-secure. Intuitively, the adversary in this model cannot exploit the modification of the scheme, since it has to commit to the challenge ciphertext index without querying the key generation oracle. Formally, from every $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2) \in \mathbf{A}^{\mathrm{OE}}_{\Pi',\text{P-KEM}}$ we construct $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2) \in \mathbf{A}^{\mathrm{OE}}_{\Pi,\text{P-KEM}}$ which simulates $\mathcal{A}$ and exploits its success probability except for the case that $\mathcal{A}_1$ outputs cInd^* such that $f(\mathrm{cInd}^*) = Y$. In turn, we can construct an inverter $\mathcal{I}$ for $\mathcal{F}$, which given the challenge $\left(f, f(x)\right)$ (where $f \leftarrow \mathcal{F}\left(1^\lambda\right)$, $x \leftarrow \{0,1\}^\lambda$) simulates $\mathcal{A}$ using f and $Y := f(x)$, and exploits the event that cInd^* satisfies $f(\mathrm{cInd}^*) = Y$. □

The main lesson from the relation between the OE and the OdE notions is that the ability of an adversary to collect secret keys should not be neglected without convincing arguments. If such arguments cannot be presented, one should avoid corresponding simplifications in the security experiment.

OdE-Security Does Not Imply BE-Security

By definition of adversaries it holds $\mathbf{A}^{\mathrm{OdE}}_{\text{P-KEM}} \subset \mathbf{A}^{\mathrm{BE}}_{\text{P-KEM}}$. The adversaries in $\mathbf{A}^{\mathrm{OdE}}_{\text{P-KEM}}$ have the additional restriction that they do not make decapsulation queries in the first query phase. Consider a predicate family $\mathcal{R}$ of prefix predicates with $\mathbb{X}_\kappa = \mathbb{Y}_\kappa = \{0,1\}^{\leq n}$, where $n = \lambda$ and λ is the security parameter which determines $\sigma \in \Sigma$, $\sigma \in \kappa$. The predicate is defined by: $\mathrm{R}_\kappa(\mathrm{kInd}, \mathrm{cInd}) = 1$ if and only if kInd is a prefix of cInd. Note that CCA2-secure P-KEM for $\mathcal{R}$ can be realized from hierarchical IBE [LW10]. Furthermore, let $\mathcal{F}$ be a family of injective one-way functions for domain $D = \{0,1\}^{2\cdot|s|}$, where s is the key for the function. Let $\text{Adv-Invert}_{\mathcal{F},\mathcal{I}}(\lambda)$ be the advantage of algorithm $\mathcal{I}$ in experiment $\mathrm{Invert}_{f,\mathcal{I}}(\lambda)$ as defined in Subsection 1.1.2. That is, $\mathcal{I}$ has to compute a preimage of y in $\{0,1\}^{2\cdot\lambda}$ defined by $f \leftarrow \mathcal{F}\left(1^\lambda\right)$, $x \leftarrow \{0,1\}^{2\cdot\lambda}$, $y := f(x)$.

Theorem 3.6. *Suppose $\mathcal{F}$ is a family of injective one-way functions and Π is an* OdE*-secure* P-KEM *for predicate family $\mathcal{R}$ and a family of key spaces $\mathcal{K}$. Then, there exists a* P-KEM Π' *for $\mathcal{R}$ and $\mathcal{K}$ which is not* BE*-secure but is* OdE*-secure. In particular, for every* $\mathrm{des} \in \Omega$ *and every ppt* $\mathcal{A} \in \mathbf{A}^{\mathrm{OdE}}_{\Pi',\text{P-KEM}}$ *there exist ppt adversaries* $\mathcal{B}, \mathcal{B}' \in \mathbf{A}^{\mathrm{OdE}}_{\Pi,\text{P-KEM}}$ *and a ppt inverter $\mathcal{I}$ such that*

$$\begin{aligned}\text{Adv-P-KEM}^{\mathrm{CCA2,OdE}}_{\Pi',\mathcal{A}}(\lambda,\mathrm{des}) \leq\ & \text{Adv-P-KEM}^{\mathrm{CCA2,OdE}}_{\Pi,\mathcal{B}}(\lambda,\mathrm{des}) + \text{Adv-Invert}_{\mathcal{F},\mathcal{I}}(\lambda) \\ & + l(\lambda)\cdot \text{Adv-P-KEM}^{\mathrm{CCA2,OdE}}_{\Pi,\mathcal{B}'}(\lambda,\mathrm{des}) \quad ,\end{aligned}$$

where $l(\cdot)$ is a polynomial.

Proof. Let $\Pi = (\mathrm{Setup}, \mathrm{KeyGen}, \mathrm{Encaps}, \mathrm{Decaps})$ be as defined in the theorem, $\mathcal{K} = \{\mathbb{K}_\lambda\}_{\lambda\in\mathbb{N}}$. Assume w.l.o.g. that $\mathcal{K} = \{\mathbb{K}_\lambda\}_{\lambda\in\mathbb{N}}$, $\mathbb{K}_\lambda = \{0,1\}^\lambda$ and that the encapsulations under cInd are of the form (cInd, ct). Π' constructed from Π as follows:

$\Pi' = (\mathrm{Setup}', \mathrm{KeyGen}', \mathrm{Encaps}', \mathrm{Decaps}')$ as follows:

Setup' $\left(1^\lambda, \mathrm{des}\right)$: Choose $(\mathrm{pp}_\kappa, \mathrm{msk}) \leftarrow \mathrm{Setup}\left(1^\lambda, \mathrm{des}\right)$, $r = (r_1, \ldots, r_\lambda) \leftarrow \{0,1\}^\lambda$. Define $\mathrm{cInd}_w := 1^\lambda \| r$ and for every $i \in [\lambda]$: $\mathrm{kInd}_i = \mathrm{cInd}_i := 1^i$. For every $i \in [\lambda]$ generate $(\mathrm{CT}_i, \mathrm{K}_i) \leftarrow \mathrm{Encaps}(\mathrm{pp}_\kappa, \mathrm{cInd}_i)$ until the i'th bit of K_i is equal r_i. Choose $\mathrm{K}_w \leftarrow \mathbb{K}_\lambda$, $f \leftarrow \mathcal{F}$, and compute $Y := f(r)$. Output the public parameters $\mathrm{pp}'_\kappa := (\mathrm{pp}_\kappa, f, \mathrm{K}_w, Y, \mathrm{CT}_1, \ldots, \mathrm{CT}_\lambda)$ and msk.

KeyGen' $(\mathrm{pp}'_\kappa, \mathrm{msk}, \mathrm{kInd})$: Parse $\mathrm{pp}'_\kappa = (\mathrm{pp}_\kappa, f, \mathrm{K}_w, Y, \mathrm{CT}_1, \ldots, \mathrm{CT}_\lambda)$ and output the secret key $\mathrm{sk} \leftarrow \mathrm{KeyGen}(\mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{kInd})$.

Encaps' $(\mathrm{pp}'_\kappa, \mathrm{cInd})$: Parse $\mathrm{pp}'_\kappa = (\mathrm{pp}_\kappa, f, \mathrm{K}_w, Y, \mathrm{CT}_1, \ldots, \mathrm{CT}_\lambda)$. If $\mathrm{cInd} = 1^\lambda \| r'$ and $f(r') = Y$ output K_w and $\mathrm{CT}' = 1 \| \left(\mathrm{cInd}, 1^\lambda\right)$. Otherwise compute $(\mathrm{K}, \mathrm{CT}) \leftarrow \mathrm{Encaps}(\mathrm{pp}_\kappa, \mathrm{cInd})$ and output K and $\mathrm{CT}' := 0 \| \mathrm{CT}$.

Decaps' $(\mathrm{pp}'_\kappa, \mathrm{CT}', \mathrm{sk})$ for $\mathrm{CT}' = b \| (\mathrm{cInd}, ct)$: Output K_w if $b = 1$, $ct = 1^\lambda$, $\mathrm{cInd} = 1^\lambda \| r'$, and $f(r') = Y$. If $b = 0$ output $\mathrm{Decaps}(\mathrm{pp}_\kappa, (\mathrm{cInd}, ct), \mathrm{sk})$. Otherwise output $\perp$.

Obviously Π' is not BE-secure, since cInd_w can be revealed using decapsulation oracle on CT_i's. The idea of the presented construction is as follows. In order to decapsulate just one $\mathrm{CT}_i \in \mathrm{pp}'_\kappa$ in the first query phase, an OdE-adversary needs a key for any prefix of cInd_λ. If $\mathcal{A}$ queries the corresponding key, it cannot use the weak ciphertext index $\mathrm{cInd}_w = 1^\lambda \| r$ for the challenge anymore.

Next we give a sketch of the proof that Π' is OdE-secure. Formally, from every ppt $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2) \in \mathbf{A}^{\mathrm{OdE}}_{\Pi',\text{P-KEM}}$ we construct $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2) \in \mathbf{A}^{\mathrm{OdE}}_{\Pi,\text{P-KEM}}$ which extends the public parameters, simulates $\mathcal{A}$ and exploits its success probability except for the case that $\mathcal{A}_1$ outputs $\mathrm{cInd}^* = \mathrm{cInd}_w$. Then, we prove that the probability for $\mathrm{cInd}^* = \mathrm{cInd}_w$ is negligible. Namely, we construct an inverter $\mathcal{I}$ for $\mathcal{F}$, which given the challenge (f, y) generates the public parameters and thereby computes all CT_i by $\mathrm{Encaps}\left(1^\lambda, \mathrm{cInd}_i\right)$. $\mathcal{I}$ wins if $\mathcal{A}$ outputs $\mathrm{cInd}^* = \mathrm{cInd}_w = 1^\lambda \| r'$, $f(r') = y$. The last step is to prove that the probability for $\mathrm{cInd}^* = \mathrm{cInd}_w$ in the real experiment and in the experiment with public parameters as generated by $\mathcal{I}$ is the same except for negligible probability. For this last step it is sufficient to consider hybrid distributions, one for the modification of a single CT_i which results in the reduction algorithm $\mathcal{B}'$ and security loss of $l(\lambda) = 4 \cdot \lambda$. □

From the presented theorem we deduce that decapsulation queries in the first query phase could be used to find a *weak* ciphertext index for the challenge. This is the main difference to

the conventional KEMs, where the challenge is generated independently of any computation of adversary, and hence the first query phase can be completely dropped.

From the OdE-security of the scheme Π' used in the proof we also deduce that OdE-security of a P-KEM scheme does not necessarily imply the smoothness of the scheme (cf. Definition 1.8). This result is in contrast to the secure conventional KEMs, which are smooth [BHK15].

BE-Security Implies SE-Security

Next we prove that BE-security implies SE-security for P-KEMs. Even though we are able to prove this result, the corresponding reduction is not tight even for smooth schemes. Namely, the security guaranties linearly decrease in the number of decapsulation queries of adversary in the first query phase. In contrast, the corresponding reduction for conventional KEMs from [BHK15] is tight for smooth schemes.

The difference between SE-security and BE-security is in the definition of $\mathbf{A}^{\mathrm{SE}}_{\text{P-KEM}}$ and $\mathbf{A}^{\mathrm{BE}}_{\text{P-KEM}}$, since the corresponding security experiments are equal. By definition, for every $\mathcal{A} \in \mathbf{A}_{\text{P-KEM}}$ it holds $\mathcal{A} \in \mathbf{A}^{\mathrm{SE}}_{\text{P-KEM}}$ if and only if $\Pr[\mathrm{CT}^* \in S_2] = 0$, and $\mathcal{A} \in \mathbf{A}^{\mathrm{BE}}_{\text{P-KEM}}$ if and only if $\Pr[\mathrm{CT}^* \in S_1 \cup S_2] = 0$. Every adversary against the scheme in SE-model with non-negligible advantage achieves a non-negligible advantage also in BE model as long as $p = \Pr[\mathrm{CT}^* \in S_1] = 0$. The main difficulty in the proof is to deal with adversaries with $p > 0$. In order to ensure $\Pr[\mathrm{CT}^* \in S_1] = 0$ we have to avoid any decapsulation queries on $\mathrm{CT} \in \mathbb{C}_{\mathrm{cInd}^*}$ in the first query phase. The idea is to guess the number of the decapsulation query, where the challenge index cInd^* (or rather an encapsulation $\mathrm{CT} \in \mathbb{C}_{\mathrm{cInd}^*}$) is used for the first time. Already in this query we ask for the challenge by the own challenger. If the guess is correct, we ensure $\Pr[\mathrm{CT}^* \in S_1] = 0$ and if not, we abort without querying the challenge and also ensure $\Pr[\mathrm{CT}^* \in S_1] = 0$.

Theorem 3.7. *Suppose* Π *is a* BE*-secure* P-KEM *for predicate family* $\mathcal{R}_{\Omega,\Sigma}$*. Then,* Π *is* SE*-secure. In particular, for every* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2) \in \mathbf{A}^{\mathrm{SE}}_{\text{P-KEM}}$ *and every* $\mathrm{des} \in \Omega$ *there exist* $\mathcal{A}', \mathcal{A}'' \in \mathbf{A}^{\mathrm{BE}}_{\text{P-KEM}}$ *such that*

$$\begin{aligned}\text{Adv-P-KEM}^{\mathrm{CCA2,SE}}_{\Pi,\mathcal{A}}(\lambda,\mathrm{des}) \leq (l+1)\cdot \text{Adv-P-KEM}^{\mathrm{CCA2,BE}}_{\Pi,\mathcal{A}'}(\lambda,\mathrm{des}) \\ + l\cdot\sqrt{2\cdot(l+1)\cdot \text{Adv-P-KEM}^{\mathrm{CCA2,BE}}_{\Pi,\mathcal{A}''}(\lambda,\mathrm{des})}\ ,\end{aligned}$$

where $l = l(\lambda,\mathrm{des})$ *is the upper bound for the maximum number of decapsulation queries of* $\mathcal{A}_1$ *in* $\text{P-KEM}^{\mathrm{CCA2,SE}}_{\Pi,\mathcal{A}}(\lambda,\mathrm{des})$*. For smooth* Π *it holds*

$$\begin{aligned}\text{Adv-P-KEM}^{\mathrm{CCA2,SE}}_{\Pi,\mathcal{A}}(\lambda,\mathrm{des}) \leq (l+1)\cdot \text{Adv-P-KEM}^{\mathrm{CCA2,BE}}_{\Pi,\mathcal{A}'}(\lambda,\mathrm{des}) \\ + l\cdot \mathrm{Smth}_{\Pi}(\lambda,\mathrm{des})\ .\end{aligned}$$

Proof. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2) \in \mathbf{A}^{\mathrm{SE}}_{\text{P-KEM}}$ and $\mathrm{des} \in \Omega$ are arbitrary, but fixed. Let $l = l(\lambda,\mathrm{des})$ be the upper bound for the number of decapsulation queries of $\mathcal{A}_1$ in $\text{P-KEM}^{\mathrm{CCA2,SE}}_{\Pi,\mathcal{A}}(\lambda,\mathrm{des})$. W.l.o.g. we can assume that $\mathcal{A}_1$ makes exactly l decapsulation queries in this experiment.[1] Let us first estimate the advantage of $\mathcal{A}$ according to the event $\mathrm{CT}^* \in S_1$, which we denote by BD:

$$\begin{aligned}&\text{Adv-P-KEM}^{\mathrm{CCA2,SE}}_{\Pi,\mathcal{A}}(\lambda,\mathrm{des}) \\ \overset{\text{by def.}}{=}\ &2\cdot\Pr\left[\text{P-KEM}^{\mathrm{CCA2,SE}}_{\Pi,\mathcal{A}}(\lambda,\mathrm{des}) = 1\right] - 1\end{aligned}$$

[1] This cannot be assumed if $\mathcal{A}_1$ outputs $\perp$ without querying the decapsulation oracle, but in this case $\Pr[\mathrm{CT}^* \in S_1] = 0$ and we do not have to change the behavior of the adversary in this case.

$$
\begin{aligned}
= \quad & 2 \cdot \Pr\left[\text{P-KEM}_{\Pi,\mathcal{A}}^{\text{CCA2,SE}}(\lambda, \text{des}) = 1 \wedge \overline{\text{BD}}\right] \\
& + 2 \cdot \Pr\left[\text{P-KEM}_{\Pi,\mathcal{A}}^{\text{CCA2,SE}}(\lambda, \text{des}) = 1 \wedge \text{BD}\right] - \left(\Pr\left[\overline{\text{BD}}\right] + \Pr\left[\text{BD}\right]\right) \\
\leq \quad & \left(2 \cdot \Pr\left[\text{P-KEM}_{\Pi,\mathcal{A}}^{\text{CCA2,SE}}(\lambda, \text{des}) = 1 \wedge \overline{\text{BD}}\right] - \Pr\left[\overline{\text{BD}}\right]\right) + \Pr\left[\text{BD}\right] \quad . \qquad (3.1)
\end{aligned}
$$

For smooth schemes we can immediately estimate the probability for the event BD by union bound, since it holds $\Pr[\text{BD}] \leq l \cdot \text{Smth}_{\Pi}(\lambda, \text{des})$. However, whereas secure conventional KEMs are smooth as proved in [BHK15], this is not the case for P-KEMs, as shown in the previous paragraph. What we prove is that for every $\mathcal{A} \in \mathbf{A}_{\text{P-KEM}}^{\text{SE}}$ the probability $\Pr[\text{BD}]$ is negligible due to the BE-security of Π.

Let us first prove that the first summand in (3.1) is negligible. We construct an adversary $\mathcal{A}' = (\mathcal{A}'_1, \mathcal{A}'_2) \in \mathbf{A}_{\text{P-KEM}}^{\text{BE}}$ which exploits the success probability of $\mathcal{A} \in \mathbf{A}_{\text{P-KEM}}^{\text{SE}}$ in the case that the event BD does not occur in $\text{P-KEM}_{\Pi,\mathcal{A}}^{\text{CCA2,SE}}(\lambda, \text{des})$. We have to ensure that the probability that $\mathcal{A}'$ (and especially $\mathcal{A}'_1$) submits CT^* to the decapsulation oracle is equal zero. Whereas in the second query phase this is trivial, the main difficulty is to prevent such a query in the first query phase, where the challenge ciphertext index cInd^* is not known. The main observation is that the probability for the event $\text{CT}^* \in S_1$ is equal zero if and only if $\mathcal{A}_1$ does not query the decapsulation oracle on $\text{CT} \in [\text{Encaps}(\text{cInd}^*)]$. $\mathcal{A}'_1$ avoids to submit $\text{CT} \in \mathbb{C}_{\text{cInd}^*}$ and this way we ensure the required property, since $[\text{Encaps}(\text{cInd}^*)] \subseteq \mathbb{C}_{\text{cInd}^*}$.

In order to prevent the event $\text{CT}^* \in S_1$ in the first query phase, $\mathcal{A}'$ guesses by $j \leftarrow [l+1]$ the step where $\mathcal{A}_1$ will use the challenge index cInd^* *for the first time*. This can be a decapsulation query ($j \in [l]$) or the challenge ciphertext index itself ($j = l+1$). Let us assume at this point that j was correctly guessed and $j \in [l]$. That is, the ciphertext index cInd_j of the submitted ciphertext CT_j is equal to the challenge ciphertext index and has not been used before. When the j'th decapsulation query is asked by $\mathcal{A}$, then $\mathcal{A}'$ directly asks for the challenge on cInd_j and after that proceeds the simulation of $\mathcal{A}_1$. If the guess was correct, $\mathcal{A}'$ receives CT^* already in the j'th query and can avoid decapsulation queries on $\text{CT} = \text{CT}^* \in \mathbb{C}_{\text{cInd}^*}$ in the first query phase. Furthermore, if j was correctly guessed, $\mathcal{A}$ can be perfectly simulated as long as it does not cause BD event. In turn, if j was guessed incorrectly, $\mathcal{A}'$ outputs $\perp$, and hence also avoid the BD event. Hence, the advantage of $\mathcal{A}'$ compared to the advantage of $\mathcal{A}$ decreases only due to the BD event $\text{CT}^* \in S_1$ in $\text{P-KEM}_{\Pi,\mathcal{A}}^{\text{CCA2,SE}}(\lambda, \text{des})$ and by factor $\frac{1}{l+1}$ due to the guess. We can prove that it holds

$$
\text{Adv-P-KEM}_{\Pi,\mathcal{A}'}^{\text{CCA2,BE}}(\lambda, \text{des}) \geq \frac{1}{l+1} \cdot \left(\text{Adv-P-KEM}_{\Pi,\mathcal{A}}^{\text{CCA2,SE}}(\lambda, \text{des}) - \Pr\left[\text{BD}\right]\right).
$$

For smooth schemes (cf. Definition 1.8) we immediately get the statement in the theorem. For the general case we construct $\mathcal{A}'' = (\mathcal{A}''_1, \mathcal{A}''_2) \in \mathbf{A}_{\text{P-KEM}}^{\text{BE}}$ which exploits the event BD in $\text{P-KEM}_{\Pi,\mathcal{A}}^{\text{CCA2,SE}}(\lambda, \text{des})$. Again we use similar ideas in order to ensure that $\mathcal{A}''$ never queries the decapsulation of CT^* in the first query phase. However, there is an additional challenge. Namely, $\mathcal{A}''$ cannot directly exploit the BD event $\text{CT}^* \in S_1$ in order to break the challenge, since the corresponding encapsulated key is not given in the query. Rather, the main observation is that if this event occurs with non-negligible probability, then for the given ciphertext index cInd^* the probability $\Pr_{(\text{K},\text{CT}) \leftarrow \text{Encaps}(\text{cInd}^*)}[\text{CT} = \text{CT}^*]$ is non-negligible. Hence, $\mathcal{A}''$ just generates an additional encapsulation (K', CT') and then solves the challenge given the correct encapsulated key K' for CT^* if $\text{CT}' = \text{CT}^*$. Note that $\mathcal{A}''$ will not even use $\mathcal{A}_2$. We can prove that it holds

$$
\text{Adv-P-KEM}_{\Pi,\mathcal{A}''}^{\text{CCA2,BE}}(\lambda, \text{des}) \quad \geq \quad \frac{1}{2 \cdot (l+1) \cdot l^2} \cdot \left(\Pr\left[\text{BD}\right]\right)^2 \quad .
$$

From this we finally deduce the statement of the theorem. □

We deduce that the security notions SP, SE, BP, and BE are equivalent for P-KEMs. However, the reductions are not all tight, due to the security guarantees in Theorem 3.7. In

particular, the reductions for BE $\Rightarrow$ SE and BE $\Rightarrow$ BP are not tight even for smooth schemes. For the implication BP $\Rightarrow$ SP a tight reduction can be presented for smooth schemes, mainly due to the fact that we do not have to avoid the BD event as in the proof of Theorem 3.7. To the best of our knowledge all practical predicate encryption schemes are smooth, and hence we could also use the BP-model for these schemes. However, since the probability for the event $\mathrm{CT}^* \in S_1$ can always be estimated by $l \cdot \mathrm{Smth}_{\Pi}(\lambda, \mathrm{des})$, we do not really gain any advantage from this model.

Part II.

Fully CCA-Secure Framework in Composite-Order Groups

Introduction, Related Work, and Main Contribution

Except for identity-based encryption (IBE), constructions of fully secure predicate encryption (PE) schemes have been missing for a long time. The dual system encryption methodology, introduced and extended by Waters and Lewko [Wat09a, LW10], provides fundamental techniques to achieve fully secure PE schemes withstanding chosen-plaintext attacks (CPA). Based on this technique, schemes for various predicates such as (hierarchical) identity-based encryption [Wat09a, LW10], attribute-based encryption [LW11, LOS+10], inner-product encryption [OT10a, AL12], spatial encryption [Ham11, AL12], and predicate encryption schemes for regular languages [Att14a], to name just a few, have been constructed.

Although many PE schemes have been presented, constructions for new predicates have each been built from the ground up until the following results were published. Attrapadung [Att14a] and Wee [Wee14] independently introduced generic frameworks for the design and the analysis of PE schemes with public index from composite-order bilinear groups. These frameworks exploit newly defined, related cryptographic primitives called pair encoding and predicate encoding, respectively. The authors show that fully CPA-secure PE schemes can be constructed from encoding schemes in a generic fashion, based on the dual system encryption methodology. In doing so, they achieved many previously known and new PE schemes by development of the corresponding encoding schemes. Both frameworks consider information-theoretical security notions for encoding schemes, but the pair encoding framework of Attrapadung also captures computationally secure pair encodings that lead to adaptively secure schemes for sophisticated predicates. The approach of [Att14a, Wee14] simplifies the development of new schemes, since the complexity of security proofs is reduced. Furthermore, the properties required to achieve CPA-secure constructions are better understood, structured, and defined in terms of security properties of encodings. Both frameworks were adapted to prime-order groups in [Att16, AC16] and in [CGW15], respectively. Overall, the research on encodings resulted in new and efficient CPA-secure schemes for various predicates. Furthermore, we remark that recently a new pair encoding framework was proposed in [AC17b]. The main goal of this work was to simplify the security proofs for the pair encoding schemes themselves.

CCA-security for PKE and IBE. Even though there exist many adaptively CPA-secure PE schemes for various predicates, only few works consider the realization of fully secure PE schemes withstanding chosen-ciphertext attacks (CCA), the most desirable security notion in practice. Furthermore, comparing this situation with conventional public key encryption (PKE) and IBE, we recognize that some techniques, exploited in these contexts to achieve CCA-secure schemes, have not yet been considered for PE. Indeed, many different approaches and techniques have been used to construct *efficient* CCA-secure PKE and IBE schemes *without the idealized Random Oracle model*. However, one can divide the known constructions into two classes depending on ciphertext properties which are checked during the decryption. We consider these two classes next and then look at known CCA-secure PE schemes.

The first approach goes back to the first efficient CCA-secure (without Random Oracle) PKE schemes introduced in [CS98, Sho00, CS03] and to their generalization from [CS02]. Schemes following this approach achieve CCA-security using a kind of *consistency checks* for ciphertexts before the actual decryption. The overall idea behind consistency checks is to prevent the adversary from being able to construct a ciphertext, which pass the consistency checks, except

using the proper encryption algorithm. In doing so, the decryption oracle becomes useless for adversaries, since they either know the plaintext anyway or the ciphertext is rejected by consistency checks. Interestingly, in [BMW05] the authors presented a CCA-secure PKE scheme based on an IBE scheme. Their techniques are also called *direct chosen-ciphertext techniques*, because in contrast to the generic transformations from [BCHK07] which we discuss below, the CCA-secure PKE scheme is achieved from a particular IBE scheme in a non-generic fashion. The direct chosen-ciphertext approach was applied to achieve efficient CCA-secure IBE schemes from particular CPA-secure IBE schemes in [KG06, KV08, KG09]. In the resulting PKE and IBE schemes from [BMW05, KG06, KG09] the form of the ciphertexts is checked before the actual decryption as well. A further CCA-secure IBE scheme, which utilizes consistency checks, was presented in [Gen06].

CCA-secure schemes in the second mentioned above class of PKE and IBE constructions do not explicitly check the form of the ciphertexts. Schemes in this class can further be divided into two groups according to the quite different techniques used to construct the corresponding schemes. Generic transformations of CPA-secure schemes into CCA-secure schemes presented in [CHK04, BK05, BCHK07] utilize message authentication codes (MAC) combined with a kind of commitment schemes or one-time signature (OTS) schemes. The so-called CHK-transformation from [CHK04] use OTS schemes in a specific manner. Namely, a pair $(\mathrm{vk}, \mathrm{sk})$ of a verification key and a signing key is generated for every encryption; verification key vk is integrated into the original CPA-secure ciphertext and this ciphertext is signed by sk. The main idea behind this construction is that adversaries either have to forge the signature or have to change the verification key in a given ciphertext in order to be able to exploit the decryption oracle. The impossibility of the second adversarial approach is ensured by the CPA-security of the underlying schemes and by an appropriate integration of vk into the ciphertext. The transformation from [BK05] is similar and improves the efficiency of the CHK-transformation by utilizing a MAC and a commitment scheme instead of an one-time signature scheme. PKE and IBE schemes achieved from these techniques verify either the signature of the OTS scheme or the tag of the MAC before the actual decryption, but do not check the further structure of the ciphertext. One of the main drawbacks of these so-called CHK-like transformations is that either the verification key of the signature scheme or the commitment of the commitment scheme has to be integrated into the original structure of the CPA-secure scheme. Namely, in the IBE-to-PKE transformations these elements are used as the identity of IBE scheme while in the HIBE-to-IBE transformations these components are integrated into the additional hierarchy level. This is the main reason why schemes achieved using the non-generic direct chosen-ciphertext techniques [BMW05, KG06, KG09], discussed above, are more efficient than schemes from CHK-like transformations [CHK04, BK05, BCHK07].

The second group of schemes which do not perform explicit consistency checks of the ciphertext origin from the CCA-secure PKE schemes from [KD04, DGKS10]. The authors construct hybrid encryption schemes. That is, the PKE scheme is used to encrypt a random key which is then used to encrypt the actual payload by a symmetric encryption scheme. The corresponding approach was applied in the context of IBE in [KV08]. These works improve the efficiency of the previously mentioned schemes from [Sho00, CS03] and from [Gen06] using implicit rejection of malformed ciphertexts by the utilized authenticated symmetric encryption. That is, the form of the ciphertexts is not explicitly checked, but the malformed ciphertexts are rejected with overwhelming probability. This last approach leads to the most efficient PKE and IBE schemes which are proved to achieve CCA-security without Random Oracle model. We remark that in [KV08] the authors also presented a CCA-secure variant of the CPA-secure IBE scheme from [Wat05]. Furthermore, the explicit consistency checks in the basic scheme from [KG09] can also be replaced by implicit checks. However, the resulting IBE scheme has a probabilistic decryption algorithm and is less efficient than the schemes from [KV08]. The fact that explicit checks can be replaced by implicit rejection is indeed not obvious. The mentioned constructions

exploit the simple structure of the ciphertext in the context of PKE and IBE in order to argue about the result of the decryption of malformed ciphertexts. This is essential in order to analyze the probability for the rejection in the authenticated symmetric encryption.

We conclude that CCA-secure constructions of PKE and IBE have been studied extensively. In general, the CHK-like transformations trade off efficiency for generality. We remark, that the CHK-transformation [CHK04] was also applied to achieve CCA-secure PE schemes constructed from lattices [ABV+12]. In turn, direct chosen-ciphertext techniques exploit specific properties of the underlying CPA-secure schemes and lead to more efficient CCA-secure constructions but only for particular schemes. Finally, the efficiency of the schemes with explicit consistency checks was further improved using implicit rejection techniques. Surprisingly, before our work only the CHK-like transformations were known for PE schemes with sophisticated predicates, as we explain in the following.

CCA-security for PE. The CHK-transformation [CHK04] was often proposed to be applied in order to achieve CCA-secure attribute-based encryption (ABE) schemes (e.g. [BSW07, Wat11]). In [GPSW06] the authors present the corresponding construction for their selectively secure key-policy ABE based on delegatability of user secret keys. This method was generalized in [YAHK11] to various types of ABE and in [NP15] to predicate encryption. Even under the variety of selectively secure ABE only few schemes satisfy the demands of these transformations (cf. [YAHK11, NP15]). Several adaptively secure PE schemes in composite-order groups satisfy the delegatability demands, as well. Under the fully secure PE schemes in prime-order groups considered in [YAHK11, NP15] only the ABE schemes from [OT10a] and the inner-product encryption schemes from [OT11] satisfy the required delegatability property. Therefore, it seems that the applicability of the transformation from delegatability is limited despite the generic nature of the underlying techniques. The main challenge in the application of this transformation to PE schemes consists in the secure integration of the verification key of the one-time signature scheme into the ciphertext. In fact, not only the ciphertexts are modified but also the public parameters of the scheme have to support this kind of modifications and must be extended. Furthermore, during the decryption the user secret keys are modified before the actual decryption. Consequently, schemes achieved from delegatability suffer from strong requirements and complex extensions. Finally, the required modifications of the user secret keys make the resulting schemes improper for many applications when the user secret keys are stored in a cryptographic storage and the decryption has to be performed in a secured environment.

In [YAHK11] the authors furthermore introduced a generic transformation for ABE based on the newly defined verifiability property. This construction is also inspired by the CHK-transformation [CHK04] and was generalized to PE schemes in [YAS+12] as well. In order to apply this transformation, the underlying CPA-secure scheme has to satisfy the following verifiability property: Given a (possibly malformed) ciphertext CT and two key indices, one must be able to check if the decryption of CT using *every* secret key for one of the given key indices results in the same output. The corresponding verification algorithm is used before the actual decryption additionally to the verification of the one-time signature. Hence, on the one hand, this transformation uses extensions of the CHK-transformation but on the other hand it also uses explicit consistency checks. Consequently the resulting schemes suffer from both, complex CHK-like extensions of the CPA-secure schemes and costly explicit consistency checks due to the required verifiability property. However, the user secret keys do not have to be modified during the decryption which is an advantage in comparison to the transformation from the delegatability property.

If we compare the required verifiability property of [YAHK11, YAS+12] with the guarantees of the explicit consistency checks used in the context of PKE and IBE we recognize that these are related but not equivalent. Indeed, for identity-based schemes the stronger notion of public verifiability can be achieved easily. Public verifiability means that the corresponding verification

algorithm (alias consistency check) accepts only ciphertexts that have the correct form with respect to the encryption algorithm. In contrast, most known fully CPA-secure PE schemes for involved predicates are constructed using the dual system encryption methodology [Wat09a, LW10]. An unavoidable side effect of this technique is that there exist malformed ciphertexts which are indistinguishable from the correctly generated ciphertexts. Hence, public verifiability is unattainable for schemes constructed from this technique. However, for schemes in composite-order groups, consistency checks satisfying the demands of the verifiability property of [YAHK11, YAS+12] are known. Already in [YAHK11] appropriate checks were proposed for a slightly modified variant of the ciphertext-policy ABE from [LOS+10]. One of our results in this part of the thesis is a general method for constructing appropriate consistency checks for PE schemes in composite-order groups, which covers not only the key-policy ABE of [LOS+10] but also many other schemes. However, only two of the PE schemes considered in [YAS+12] and satisfying the required verifiability property are constructed in prime-order groups. In fact, the authors state that for the broadcast encryption scheme from [Wat09b] and for the inner-product encryption scheme from [AL10] the techniques applied to construct verification algorithms in composite-order groups do not seem to be enough. For these schemes the authors suggest to provide every user with two secret keys for the same key index. Then, the user checks whether the decryption using both keys results in the same output or not, which "*indicates that the ciphertext is invalid*" [YAS+12, page 258]. For sophisticated predicates this strategy to construct appropriate verification algorithms does not seem to be sufficient. For example, this approach does not work for key-policy ABE where, depending on the key index, different parts of the ciphertext are used in the decryption algorithm. Namely, a successful consistency check as above with respect to one key index does not necessarily imply correct decryption using a secret key for a different key index, which is required by the verifiability property. We refer to the last part of the thesis for discussion about additional challenges by construction of consistency checks in prime-order groups.

We conclude that despite the generic nature of the CHK-transformation, the application of this technique in the context of PE schemes is limited for both transformation types presented in [YAHK11], especially for the most desirable schemes in prime-order groups. The resulting schemes suffer from complex extensions and inappropriate properties. Furthermore, only few PE schemes are proved to be fully CCA-secure without applying the generic transformations from [YAHK11, YAS+12, NP15]. To the best of our knowledge, these are the broadcast-encryption scheme from [PPSS13] and the (index hiding) encryption for relations that are specified by non-monotone access structures combined with inner product relations [OT10a]. The techniques from [PPSS13] are closely related to the techniques used for adaptively secure IBE schemes and do not seem to be applicable to more complex predicates. The schemes from [OT10a] achieve CCA-security using one-time signature schemes and their techniques are closely related to [YAHK11].

We finally remark that the so-called Noar-Yung paradigm [NY90, Sah99, DDN03] to construct CCA-secure PKE schemes exploits non-interactive zero-knowledge proofs and can be seen as the feasibility result for the existence of CCA-secure PKE schemes based on general assumptions. In turn, in the idealized Random Oracle model it is often suggested to apply the Fujisaki-Okamoto transformation [FO13] presented for PKE to PE as well. These approaches are not considered in this work.

Main Contribution

Framework for construction of CCA-secure PE schemes. In this part of the thesis we take a significant step to close the gap between PKE/IBE and PE with respect to non-generic CCA-secure constructions with explicit consistency checks from direct chosen-ciphertext techniques [BMW05, KG09]. We extend the pair-encoding framework of Attrapadung [Att14a] to achieve fully CCA-secure PE schemes. We chose this framework because of its powerful

computational (rather than information-theoretic) security notion which allows us to capture sophisticated predicates. In this part of the thesis we mainly focus on development of techniques for constructing CCA-secure predicate encryption schemes, and hence extend the pair-encoding framework in composite-order rather than the framework in prime-order groups as explained below. Furthermore, we go beyond the dual system encryption methodology [Wat09a, LW10] and extend the corresponding proof techniques to the context of chosen-ciphertext attacks. Our techniques result in reduction costs that are comparable to the reduction costs of the underlying CPA-secure pair-encoding framework [Att14a]. The framework in this part can be seen as a blueprint when the dual system encryption methodology and direct chosen-ciphertext techniques [BMW05, KG09] are combined in order to achieve fully CCA-secure schemes. Our framework in the third part of the thesis shows how these techniques can be adapted to more efficient prime-order groups and results in a variety of new CCA-secure PE schemes for sophisticated predicates that are not covered by the previously known generic CHK-like transformations.

The main advantage of composite-order groups is the simplicity and the clearness of the resulting constructions, which is mainly due to the special properties of composite-order groups. Therefore, as many other works, we exploit this advantage in order to facilitate the fundamental understanding of underlying problems which have to be solved concerning CCA-security in the context of predicate encryption schemes for involved predicates. Our second framework on prime-order groups is more efficient and benefits from our techniques developed in this part of the thesis. We note that there are different techniques to transform schemes constructed in composite-order groups to schemes in prime-order groups [Fre10, Lew12, HHH$^+$14], but there is no generic transformation even for CPA-secure predicate encryption schemes. As pointed out by [Lew12] this is due to the parameter hiding property required in dual system proof techniques. Also, in our case the composite-order framework leads to a guideline for the framework in prime-order groups, but there is no generic transformation from schemes resulted from the framework in composite-order groups to schemes in prime-order groups. Indeed, our prime-order framework and especially the consistency checks used in this framework significantly differ from their counterparts in composite-order groups presented in this part of the thesis.

Given any pair encoding scheme (with natural restrictions), secure in terms of [Att14a], we construct a fully CCA-secure key encapsulation mechanism for the corresponding predicate based on newly developed consistency checks. Surprisingly, due to the pair encoding abstraction, we achieve a semi-generic transformation and still exploit structural properties of the underlying CPA-secure schemes. Combined with an appropriate symmetric encryption, our framework leads to various new fully CCA-secure PE schemes through the usual hybrid construction. In fact, for efficiency reasons, hybrid schemes are preferred to plain encryption schemes in practice. Nevertheless, our framework can be easily adapted in order to achieve a fully functional predicate encryption scheme directly. Furthermore, in prime-order group we present a fully functional predicate encryption, which in turn can be adapted to a key encapsulation mechanism.

Although our extensions of CPA-secure schemes are similar to those used in conventional PKE schemes [CS03] and in IBE schemes [KG09], the application to complex predicates as well as the generic nature of our construction are novel for the underlying techniques. When considering schemes in composite-order groups, we achieve simpler and usually more efficient constructions than those obtained from CPA-secure schemes and the generic transformations based on one-time signatures [YAHK11, YAS$^+$12, NP15]. Furthermore, we keep the advantage of tight reductions from the original framework of Attrapadung [Att14a], and the reduction costs of our CCA-secure construction are comparable to the reduction costs of the underlying CPA-secure construction. This is indeed surprising and is due to our extension of the dual system encryption proof techniques which we describe below. The only additional cryptographic primitive required by our construction is a collision-resistant hash function, which is used to add a single redundant group element to the ciphertext. Apart from that, we add two group

elements to the public parameters of the underlying CPA-secure scheme independently of the predicate. The security of our framework is based on the same security assumptions as the security of the original CPA-secure framework except for the additional assumption about the collision-resistance of the hash function.

Moving beyond the dual system encryption methodology. Security proofs in cryptography often consist of a sequence of probability experiments (or games) with small differences. The first experiment is the target security experiment (CCA-security experiment in our case), whereas the last experiment is constructed in such a way that the adversaries cannot achieve any advantage. The task of the proof is to show that consecutive experiments are computationally indistinguishable. This proof structure is also used in the dual system encryption methodology [Wat09a, LW10], but the modifications between the experiments are quite special. The main idea of this technique is to define the so-called semi-functional keys and semi-functional ciphertexts that are indistinguishable from their normal counterparts. In the proof sequence, the challenge ciphertext and all generated keys are transformed from normal to semi-functional one by one. In the last experiment, when all elements are modified, the challenge can be changed to the ciphertext of a randomly chosen message and no ppt algorithm can detect this modification.

The obvious way to apply dual system encryption methodology in the context of CCA-security is to treat keys used to answer decryption queries in the same way as keys queried by the adversary. This strategy was followed in [OT10a] (see discussion of this work below), but our proof strategy diverges from it. We deal with decryption queries in a novel and surprisingly simple manner. As an additional advantage, the reductions of the original CPA-security proof require only a few simple modifications. The main idea is to answer decryption queries in *all* games using separately generated *normal* keys. Our consistency checks ensure that this modification cannot be noticed. Mainly because of our handling of user secret keys for decryption queries we can keep the basic structure of the original CPA-security proof of [Att14a] and achieve the previously mentioned security guaranties. We only have to add four additional experiments: three at the beginning and one before the last game. In our last game we show that by using the redundant element added to the ciphertext we can answer all decryption queries without the user secret keys. The indistinguishability for this experiment is again based on our consistency checks. Moreover, we ensure that normal and semi-functional ciphertexts both pass our checks. Hence, even though we cannot verify whether or not a ciphertext is a correctly formed normal ciphertext, which would contradict the property of indistinguishable semi-functional ciphertexts, we design consistency checks that are sufficient to achieve CCA-security. As already mentioned above, our method to construct these checks leads to verification algorithms that satisfy the required properties of generic transformations from [YAHK11, YAS+12].

The main advantage of our construction and our proof strategy becomes obvious if compared to the techniques in [OT10a], where *all keys* are changed and the security guarantees decrease linearly in the number of decryption queries and in the number of corrupted keys (cf. Theorem 4 in [OT10b]). In our approach, the number of decryption queries influences the security guarantees only negligibly. In realistic scenarios, the number of decryption queries is usually assumed to be much larger than the number of corrupted keys. Hence, our approach results in smaller security parameters, which additionally increases efficiency.

Organization. In Chapter 4 we present our fully CCA-secure framework in composite-order groups and explain the proof techniques. The formal security proof is presented in Chapter 5.

We remark that the results presented in this part are published in [BL16b].

4. Background, Construction and Intuition

This chapter is organized as follows. First of all, in Section 4.1 we state the required preliminaries, define the security assumptions and specify the security definition based on the analysis from the previous part. Then, in Section 4.2 we introduce the additional requirements on pair encodings needed to apply our framework presented in Subsection 4.2.1. Furthermore, in the subsequent Subsection 4.2.3 we give a high level intuition behind the consistency checks and explain the required properties. In this subsection we also define the semi-functional algorithms which are essential for the proof.

4.1. Preliminaries

4.1.1. Composite-Order Bilinear Groups

In this subsection we briefly recall the main properties of composite-order bilinear groups (cf. [LW10]). We define the security assumptions with respect to a ppt algorithm $\mathcal{G}$, called composite-order group generator, which takes as input a security parameter 1^λ and outputs a description $\mathbb{GD}$ of bilinear groups. Consequently, the security of our framework is proved with respect to $\mathcal{G}$. We require that group generator $\mathcal{G}$ on input 1^λ outputs group description

$$\mathbb{GD} = \left(p_1, p_2, p_3, (g, \mathbb{G}), \mathbb{G}_\mathrm{T}, \mathrm{e}\right) \ ,$$

where p_1, p_2, p_3 are distinct primes of length λ, $\mathbb{G}$ and $\mathbb{G}_\mathrm{T}$ are cyclic groups of order $N = p_1 p_2 p_3$, g is a generator of $\mathbb{G}$, and function $\mathrm{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_\mathrm{T}$ is a non-degenerate bilinear map: i.e., $\mathrm{e}(g, g)$ is a generator of $\mathbb{G}_\mathrm{T}$ and $\mathrm{e}\left(g^a, g^b\right) = \mathrm{e}(g, g)^{ab}$ for all $a, b \in \mathbb{Z}_N$. The prime numbers are crucial for the security of the schemes and are not made public. Hence, we denote by $\mathbb{GD}_N$ a restricted description of bilinear groups corresponding to $\mathbb{GD}$, where the prime numbers are replaced by N. We require that the group operations as well as the bilinear map e can be evaluated in polynomial time with respect to λ given the restricted description $\mathbb{GD}_N$.

Group $\mathbb{G}$ can be decomposed as $\mathbb{G}_{p_1} \times \mathbb{G}_{p_2} \times \mathbb{G}_{p_3}$, where for every prime $p_i \mid N$ we denote by $\mathbb{G}_{p_i}$ the unique subgroup of $\mathbb{G}$ of order p_i. Let g_i be an arbitrary but fixed generator of $\mathbb{G}_{p_i}$. Every $h \in \mathbb{G}$ can be expressed as $g_1^{a_1} g_2^{a_2} g_3^{a_3}$, where all a_i's are uniquely defined modulo corresponding p_i's. Hence, we will call $g_i^{a_i}$ the $\mathbb{G}_{p_i}$ component of h. Notice that, e.g., $g^{p_1 p_2}$ generates $\mathbb{G}_{p_3}$, and hence given the factorization of N we can pick random elements from every subgroup of $\mathbb{G}$. A further important property of composite-order bilinear groups is that for $p_i \neq p_j$ and $h_i \in \mathbb{G}_{p_i}$, $h_j \in \mathbb{G}_{p_j}$ it holds $\mathrm{e}(h_i, h_j) = 1_{\mathbb{G}_\mathrm{T}}$. Namely, let p_k be the third prime, then h_i can be written as $g^{a \cdot p_j \cdot p_k}$ and analogously $h_j = g^{b \cdot p_i \cdot p_k}$. Hence, $\mathrm{e}(h_i, h_j) = \left(\mathrm{e}(g, g)^N\right)^{a \cdot b \cdot p_k} = 1_{\mathbb{G}_\mathrm{T}}$.

We will also use the following common notation for vectors of group elements. In this part of the thesis we do not distinguish between row and column vectors. It will be always obvious from the context which computations we mean. Let $g, h, r \in \mathbb{G}$, $\boldsymbol{v}, \boldsymbol{w}, \boldsymbol{u} \in \mathbb{Z}_N^k$, and $\boldsymbol{E} \in \mathbb{Z}_N^{k \times d}$ for $k, d \in \mathbb{N}$. We denote by $g^{\boldsymbol{v}}$ the vector $(g^{v_1}, g^{v_2}, \ldots, g^{v_k}) \in \mathbb{G}^k$. Furthermore, we define $g^{\boldsymbol{v}} \cdot g^{\boldsymbol{w}} := g^{\boldsymbol{v}+\boldsymbol{w}}$, $(g^{\boldsymbol{v}})^{\boldsymbol{E}} := g^{\boldsymbol{v} \cdot \boldsymbol{E}}$, and

$$\mathrm{e}(g^{\boldsymbol{v}}, h^{\boldsymbol{w}}) := \prod_{i=1}^{k} \mathrm{e}(g^{v_i}, h^{w_i}) \ .$$

Hence, it also holds

$$\mathrm{e}\left(g^{\boldsymbol{v}}, h^{\boldsymbol{w}} \cdot r^{\boldsymbol{u}}\right) = \prod_{i=1}^{k} \mathrm{e}\left(g^{v_i}, h^{w_i} \cdot r^{u_i}\right) = \mathrm{e}\left(g^{\boldsymbol{v}}, h^{\boldsymbol{w}}\right) \cdot \mathrm{e}\left(g^{\boldsymbol{v}}, r^{\boldsymbol{u}}\right) \quad .$$

Furthermore, given $g^{\boldsymbol{v}}$ and $\boldsymbol{E}$ one can efficiently compute each component of $\left(g^{\boldsymbol{v}}\right)^{\boldsymbol{E}} \in \mathbb{G}^d$. Namely, the j'th component of $\boldsymbol{v} \cdot \boldsymbol{E}$ is $\sum_{i=1}^{k} v_i \cdot e_{i,j}$, and hence the j'th component of $\left(g^{\boldsymbol{v}}\right)^{\boldsymbol{E}}$ can be computed by

$$\prod_{i=1}^{k} \left(g^{v_i}\right)^{e_{i,j}} = g^{\sum_{i=1}^{k} v_i \cdot e_{i,j}} \quad .$$

We furthermore define the following notational convention. Let $\mathbb{G}$ be a group of composite order N and $\mathbb{H}$ be a subgroup of $\mathbb{G}$, then we denote by $h \leftarrow \mathbb{H}$ a random choice of a *generator* of $\mathbb{H}$. As explained above, a random element of $\mathbb{H}$ can be efficiently chosen if the factorization of N is known.

4.1.2. Security Assumptions

In this subsection we define three so-called Subgroup Decision Assumptions used to prove the security of our construction. We use exactly the same assumptions as the original CPA-secure framework [Att14a]. See also [LW10] for validity of these assumptions in the generic group model. Let $\mathcal{G}$ be a composite-order group generator. The first step of the probability experiments presented in Fig. 4.1 is always $\mathbb{GD} = \left(p_1, p_2, p_3, (g, \mathbb{G}), \mathbb{G}_{\mathrm{T}}, \mathrm{e}\right) \leftarrow \mathcal{G}\left(1^{\lambda}\right)$.

$$\begin{array}{ll}
\textbf{SD1}\,(\lambda): & g_1 \leftarrow \mathbb{G}_{p_1}, \quad g_3 \leftarrow \mathbb{G}_{p_3} \ , \\
& D := \left(\mathbb{GD}_N, g_1, g_3\right), \quad Z_0 \leftarrow \mathbb{G}_{p_1}, \quad Z_1 \leftarrow \mathbb{G}_{p_1 p_2} \ . \\
\textbf{SD2}\,(\lambda): & g_1, X_1 \leftarrow \mathbb{G}_{p_1}, \quad X_2, Y_2 \leftarrow \mathbb{G}_{p_2}, \quad g_3, Y_3 \leftarrow \mathbb{G}_{p_3} \ , \\
& D := \left(\mathbb{GD}_N, g_1, X_1 X_2, Y_2 Y_3, g_3\right), \quad Z_0 \leftarrow \mathbb{G}_{p_1 p_3}, \quad Z_1 \leftarrow \mathbb{G} \ . \\
\textbf{SD3}\,(\lambda): & g_1 \leftarrow \mathbb{G}_{p_1}, \quad g_2, X_2, Y_2 \leftarrow \mathbb{G}_{p_2}, \quad g_3 \leftarrow \mathbb{G}_{p_3}, \quad \alpha, s \leftarrow \mathbb{Z}_N \\
& D := \left(\mathbb{GD}_N, g_1, g_1^{\alpha} X_2, g_1^{s} Y_2, g_2, g_3\right), \ Z_0 \leftarrow \mathbb{G}_{\mathrm{T}}, \ Z_1 := \mathrm{e}\left(g_1, g_1\right)^{\alpha \cdot s} \quad .
\end{array}$$

Figure 4.1.: Subgroup Decision Assumptions.

Note that due to our notational conventions, all group elements chosen in the experiments are generators of the corresponding groups. For example, in experiment SD1 it holds $Z_1 = \tilde{g}_1^{a}, \tilde{g}_2^{b}$, where $\tilde{g}_1$ and $\tilde{g}_2$ are arbitrary generators of $\mathbb{G}_{p_1}$ and $\mathbb{G}_{p_2}$, and $a \neq 0 \pmod{p_1}$, $b \neq 0 \pmod{p_2}$. Formally, this differs from the usual form of subgroup decision assumptions used in the previous works, where chosen group elements are, strictly speaking, not necessarily generators of the corresponding groups. But indeed, this is only a technical convention, since one can easily prove that the distributions are statistically close. We introduce our convention in order to avoid formally incorrect statements.

The advantage of $\mathcal{A}$ in breaking experiment $\mathrm{SD}i\,(\lambda)$ is defined as

$$\mathrm{Adv}_{\mathcal{A}}^{\mathrm{SD}i}(\lambda) := \left|\Pr\left[\mathcal{A}\left(D, Z_0\right) = 1\right] - \Pr\left[\mathcal{A}\left(D, Z_1\right) = 1\right]\right| \ ,$$

where D, Z_0 and Z_1 are as defined in $\mathrm{SD}i\,(\lambda)$.

Assumption 4.1. We say that $\mathcal{G}$ satisfies Assumption SDi if for every ppt algorithm $\mathcal{A}$ function $\mathrm{Adv}_{\mathcal{A}}^{\mathrm{SD}i}(\lambda)$ is negligible.

The following lemma was implicitly proved in [LW10] (cf. Lemma 5 in [LW09]). This lemma states that under Assumption SD2, it is computationally infeasible to compute a non-trivial factor of N.

Lemma 4.1. *There exists a ppt algorithm $\mathcal{A}$ with*

$$\mathrm{Adv}^{\mathrm{SD2}}_{\mathcal{A}}(\lambda) = \Big|\Pr\big[\mathcal{A}(D, Z_0, F) = 1\big] - \Pr\big[\mathcal{A}(D, Z_1, F) = 1\big]\Big| = 1 \ ,$$

where D, Z_0, Z_1 are distributed as defined in Experiment SD2*, and F is a non-trivial factor of N (N is defined by $\mathbb{GD}_N \in D$).*

Proof. See the proof in Subsection 5.5.1. □

4.1.3. CCA Security Definition for P-KEMs

In this subsection we present the security definition which is used in the formal security proof for our framework. This definition is slightly adapted from the definition in Chapter 2. Namely, as explained in Remark 2.10 we can assume that $\mathcal{A}$ never aborts in the first query phase. In the proof we consider slightly modified distributions. Those parts of the experiment which will be changed later are framed and numbered.

Let Π be a P-KEM for predicate family $\mathcal{R}_{\Omega,\Sigma}$ and family $\mathcal{K} = \{\mathbb{K}_\lambda\}$ of key spaces. The full (adaptive) security experiment $\text{aP-KEM}^{\mathrm{CCA2}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})$ against adaptive chosen-ciphertext attacks between challenger $\mathcal{C}$ and adversary $\mathcal{A}$ is defined next. In this experiment, index i denotes the number of a covered key generation query and kInd_i denotes the key index used in the query with number i. W.l.o.g. we assume that $\mathcal{A}$ uses index i in the oracle queries only after the i'th query to the covered key generation oracle. Furthermore, due to the analysis from Chapter 2 we always assume that all inputs of oracles are syntactically correct and for every CT submitted to the decapsulation oracle it holds $\mathrm{CT} \in \mathbb{C}_{\mathrm{cInd}} \subseteq \mathbb{C}_{\mathrm{pp}_\kappa}$ and $\mathrm{R}_\kappa(\mathrm{kInd}_i, \mathrm{cInd}) = 1$, where i is the index specified for the decapsulation query. W.l.o.g. we can also assume that $\mathcal{A}$ never asks for the decapsulation of CT^* in the second query phase. That is, we use the SE-model as defined in Chapter 2.

The advantage of $\mathcal{A}$ in security experiment $\text{aP-KEM}^{\mathrm{CCA2}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})$ presented in Fig. 4.2 is defined as

$$\text{Adv-aP-KEM}^{\mathrm{CCA2}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) := \Pr\left[\text{aP-KEM}^{\mathrm{CCA2}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) = 1\right] - \frac{1}{2} \ .$$

Definition 4.2. A predicate key encapsulation mechanism Π for predicate family $\mathcal{R}_{\Omega,\Sigma}$ is called **fully (or adaptively) secure against adaptively chosen-ciphertext attacks (or CCA2-secure)** if for every $\mathrm{des} \in \Omega$ and every ppt adversary $\mathcal{A}$ the function $\text{Adv-aP-KEM}^{\mathrm{CCA2}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})$ is negligible in λ.

4.2. Fully CCA-Secure Framework

4.2.1. Additional Requirements of CCA-Secure Framework

In this subsection we present additional properties of pair encoding schemes, that are sufficient to achieve CCA-secure P-KEMs using our framework. First of all, as in the underlying CPA-secure framework [AY15], we require *normality* of pair encoding P, a very natural restriction

$\mathbf{aP\text{-}KEM}^{\mathrm{CCA2}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})$:

Setup : $\mathcal{C}$ generates $^{\langle 1\rangle}$ $\boxed{(\mathrm{msk}, \mathrm{pp}_\kappa) \leftarrow \mathrm{Setup}\left(1^\lambda, \mathrm{des}\right)}$ and starts $\mathcal{A}$ on input $\left(1^\lambda, \mathrm{pp}_\kappa\right)$.

Phase I : $\mathcal{A}$ has access to the following oracles:

- **CKGen** (kInd_i) for $\mathrm{kInd}_i \in \mathbb{X}_\kappa$: Challenger $\mathcal{C}$ generates a secret key for kInd_i $^{\langle 2\rangle}$ $\boxed{\mathrm{sk}_i \leftarrow \mathrm{KeyGen}(\mathrm{msk}, \mathrm{kInd}_i)}$, stores sk_i, and returns nothing.
- **Open** (i) for $i \in \mathbb{N}$: $\mathcal{C}$ returns $^{\langle 3\rangle}$ $\boxed{\mathrm{sk}_i}$. We call the corresponding key index kInd_i of sk_i a corrupted key index.
- **Decaps** (CT, i) with $\mathrm{CT} \in \mathbb{C}_{\mathrm{cInd}} \subseteq \mathbb{C}_{\mathrm{pp}_\kappa}$ and $i \in \mathbb{N}$: Challenger $\mathcal{C}$ returns the decapsulation $^{\langle 4\rangle}$ $\boxed{\mathrm{Decaps}(\mathrm{sk}_i, \mathrm{CT})}$.

Challenge : $\mathcal{A}$ submits a target ciphertext index $\mathrm{cInd}^* \in \mathbb{Y}_\kappa$ such that for every corrupted key index kInd it holds $\mathrm{R}_\kappa(\mathrm{kInd}, \mathrm{cInd}^*) = 0$. Challenger $\mathcal{C}$ computes $^{\langle 5\rangle}$ $\boxed{(\mathrm{K}_0, \mathrm{CT}^*) \leftarrow \mathrm{Encaps}(\mathrm{cInd}^*)}$, chooses $\mathrm{K}_1 \leftarrow \mathbb{K}_\lambda$, flips a bit $b \leftarrow \{0,1\}$, sets $^{\langle 6\rangle}$ $\boxed{\mathrm{K}^* := \mathrm{K}_b}$, and returns the challenge $(\mathrm{K}^*, \mathrm{CT}^*)$.

Phase II : $\mathcal{A}$ has access to the following oracles:

- **CKGen** (kInd_i) for $\mathrm{kInd}_i \in \mathbb{X}_\kappa$: As before, challenger $\mathcal{C}$ generates a secret key $^{\langle 7\rangle}$ $\boxed{\mathrm{sk}_i \leftarrow \mathrm{KeyGen}(\mathrm{msk}, \mathrm{kInd}_i)}$, stores sk_i, and returns nothing.
- **Open** (i) for $i \in \mathbb{N}$ such that $\mathrm{R}_\kappa(\mathrm{kInd}_i, \mathrm{cInd}^*) = 0$: $\mathcal{C}$ returns $^{\langle 8\rangle}$ $\boxed{\mathrm{sk}_i}$.
- **Decaps** (CT, i) with $\mathrm{CT} \in \mathbb{C}_{\mathrm{cInd}} \subseteq \mathbb{C}_{\mathrm{pp}_\kappa}$, $\mathrm{CT} \neq \mathrm{CT}^*$ and $i \in \mathbb{N}$: Challenger $\mathcal{C}$ returns the decapsulation $^{\langle 9\rangle}$ $\boxed{\mathrm{Decaps}(\mathrm{sk}_i, \mathrm{CT})}$.

Guess : $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$. $^{\langle 10\rangle}$ $\boxed{\text{The output of the experiment is 1 iff } b' = b}$.

Figure 4.2.: CCA-secure experiment for the framework in composite-order groups.

which is one of the restrictions of regular encodings from Definition 1.11. This property was also used in [AC16].

Next, we formally define the second required property, which we call the verifiability property. As mentioned before, we have to check the form of the encapsulation to a certain extent in order to achieve CCA-security. The verifiability property itself does not ensure the CCA-security and has to be considered in the context of our extended framework. Hence, for the intuition behind this property we refer to the discussion in Subsection 4.2.3. We notice that in Theorem 5.2 we will state that all regular pair encodings schemes are verifiable according to our definition.

Let $\mathcal{R}_{\Omega,\Sigma}$ be a domain-transferable predicate family, $\mathcal{G}$ be a composite-order group generator and $\lambda \in \mathbb{N}$ be a security parameter. Let $\mathbb{GD} = \left(p_1, p_2, p_3, (g, \mathbb{G}), \mathbb{G}_\mathrm{T}, \mathrm{e}\right) \in \left[\mathcal{G}\left(1^\lambda\right)\right]$ and $\mathbb{GD}_N$ be the corresponding restricted group description. W.l.o.g. we assume that $\mathcal{G}$ is compatible with $\mathcal{R}_{\Omega,\Sigma}$ – that is, $N \in \Sigma$ for every $\mathbb{GD} \in \left[\mathcal{G}\left(1^\lambda\right)\right]$.

Recall that whenever $\kappa \in \Omega \times \Sigma$, $\mathrm{kInd} \in \mathbb{X}_\kappa$, and $\mathrm{cInd} \in \mathbb{Y}_\kappa$ are obvious from the context, the following values are also defined $n = \mathrm{Param}(\kappa)$, $(\boldsymbol{k}, m_2) = \mathrm{Enc1}(\kappa, \mathrm{kInd})$, $m_1 = |\boldsymbol{k}|$, and $(\boldsymbol{c}, w_2) = \mathrm{Enc2}(\kappa, \mathrm{cInd})$, $w_1 = |\boldsymbol{c}|$.

Definition 4.3. *(Verifiability)* P is called **verifiable with respect to** $\mathcal{G}$ if it is normal and there exists a deterministic polynomial-time algorithm Vrfy that given des, $\mathbb{GD}_N$ (which define $\kappa =$

$(\mathrm{des}, N))$, a generator $g_1 \in \mathbb{G}_{p_1}$, $g_1^{\boldsymbol{h}} \in \mathbb{G}_{p_1}^n$, $\mathrm{kInd} \in \mathbb{X}_\kappa$ and $\mathrm{cInd} \in \mathbb{Y}_\kappa$ such that $\mathrm{R}_\kappa(\mathrm{kInd}, \mathrm{cInd}) = 1$, $\boldsymbol{E} \in [\mathrm{Pair}(\kappa, \mathrm{kInd}, \mathrm{cInd})]$, and $\boldsymbol{C} = (C_1, \ldots, C_{w_1}) \in \mathbb{G}^{w_1}$ outputs 0 or 1 such that:

Completeness: The output is 1 if there exist $s \in \mathbb{Z}_N$ and $\boldsymbol{s} \in \mathbb{Z}_N^{w_2}$ such that the $\mathbb{G}_{p_1}$ components of the elements in $\boldsymbol{C}$ are equal to $g_1^{\boldsymbol{c}(s,\boldsymbol{s},\boldsymbol{h})}$.

Soundness: If the output is 1, then for every $\alpha \in \mathbb{Z}_N$, $\boldsymbol{r} \in \mathbb{Z}_N^{m_2}$ it holds:

$$\mathrm{e}\left(g_1^{\boldsymbol{k}(\alpha,\boldsymbol{r},\boldsymbol{h})\cdot\boldsymbol{E}}, \boldsymbol{C}\right) = \mathrm{e}(g_1, C_1)^{\alpha} \quad . \tag{4.1}$$

The intuition behind our verifiability property will be clear when the framework is presented; hence, we refer to Subsection 4.2.3 for a detailed explanation. Nevertheless, let us state the following lemma.

Lemma 4.4. *Suppose pair encoding* P *is normal and there is a ppt algorithm* Vrfy *that outputs 1* if and only if *there exist* $s \in \mathbb{Z}_N$ *and* $\boldsymbol{s} \in \mathbb{Z}_N^{w_2}$ *such that the* $\mathbb{G}_{p_1}$ *components of* $\boldsymbol{C}$ *are equal to* $g_1^{\boldsymbol{c}(s,\boldsymbol{s},\boldsymbol{h})}$. *Then,* P *is verifiable.*

Proof. Due to normality of P the $\mathbb{G}_{p_1}$ component of C_1 is equal g_1^s. Hence, the soundness property is satisfied due to the correctness of the pair encoding scheme, which ensures that for every $\boldsymbol{E} \in [\mathrm{Pair}(\kappa, \mathrm{kInd}, \mathrm{cInd})]$ it holds $\boldsymbol{k}(\alpha, \boldsymbol{r}, \boldsymbol{h}) \cdot \boldsymbol{E} \cdot \boldsymbol{c}(s, \boldsymbol{s}, \boldsymbol{h}) = \alpha \cdot s$ and consequently

$$\begin{aligned}
\mathrm{e}\left(g_1^{\boldsymbol{k}(\alpha,\boldsymbol{r},\boldsymbol{h})\cdot\boldsymbol{E}}, \boldsymbol{C}\right) &= \mathrm{e}\left(g_1^{\boldsymbol{k}(\alpha,\boldsymbol{r},\boldsymbol{h})\cdot\boldsymbol{E}}, g_1^{\boldsymbol{c}(s,\boldsymbol{s},\boldsymbol{h})}\right) \\
&= \mathrm{e}(g_1, g_1)^{\boldsymbol{k}(\alpha,\boldsymbol{r},\boldsymbol{h})\cdot\boldsymbol{E}\cdot\boldsymbol{c}(s,\boldsymbol{s},\boldsymbol{h})} \\
&= \mathrm{e}(g_1, g_1)^{\alpha\cdot s} \\
&= \mathrm{e}(g_1, C_1)^{\alpha} \quad ,
\end{aligned}$$

where the first equation holds because the $\mathbb{G}_{p_2}$ and the $\mathbb{G}_{p_3}$ components of $\boldsymbol{C}$ disappear since the first input of the pairing does not contain the $\mathbb{G}_{p_2}$ and $\mathbb{G}_{p_3}$ components. □

A verification algorithm that satisfies the property in the lemma is indeed sufficient, but the property stated in the claim is not required. Nevertheless, this lemma gives the main idea of how to construct such an algorithm. See Section 5.1 for more details and for constructions of appropriate verification algorithms.

4.2.2. Fully CCA-Secure Framework

In this subsection we present our framework for constructing fully CCA-secure P-KEMs from pair encoding schemes. We note that it will be not necessarily obvious from the following definitions of the algorithms that the computations related to encodings can be performed in polynomial time. The form used in the definition simplifies the description of the algorithms and in the correctness proof we will show that all algorithms are indeed ppt algorithms.

Let P be a verifiable pair encoding scheme for domain-transferable predicate family $\mathcal{R}_{\Omega,\Sigma}$ and Vrfy be the algorithm from Definition 4.3. Let $\mathcal{G}$ be a composite-order group generator, and $\mathcal{H}$ be a family of collision-resistant hash functions as defined in Subsection 1.1.3. We require that a hash function from $\mathcal{H}$ chosen by $h \leftarrow \mathcal{H}\left(1^\lambda\right)$ works on domain $\mathbb{Y}_\kappa \times \mathbb{G}^{\mathrm{poly}(\lambda)}$ and has range $\mathbb{Z}_N$, where N is of length λ. Formally, we require a collision-resistant hash function with range

$\mathbb{Z}_N$ and a canonical binary representation of elements $\mathbb{Y}_\kappa \times \mathbb{G}^{\mathrm{poly}(\lambda)}$. We abstract from these technicalities.

A P-KEM Π for $\mathcal{R}_{\Omega,\Sigma}$ is defined as follows:

Setup $\left(1^\lambda, \mathrm{des}\right)$ for $\mathrm{des} \in \Omega$: generate $\mathbb{GD} = \left(p_1, p_2, p_3, (g, \mathbb{G}), \mathbb{G}_\mathrm{T}, \mathrm{e}\right) \leftarrow \mathcal{G}\left(1^\lambda\right)$, $g_1 \leftarrow \mathbb{G}_{p_1}$ and $g_3 \leftarrow \mathbb{G}_{p_3}$. Set $\kappa := (\mathrm{des}, N)$, where $N = p_1 p_2 p_3$. Compute $n := \mathrm{Param}(\kappa)$, pick $\boldsymbol{h} \leftarrow \mathbb{Z}_{p_1}^n$, and compute $g_1^{\boldsymbol{h}}$. Choose $\alpha, u, v \leftarrow \mathbb{Z}_{p_1}$ and set $Y := \mathrm{e}(g_1, g_1)^\alpha$, $U_1 := g_1^u$, and $V_1 := g_1^v$. Choose $\mathrm{H} \leftarrow \mathcal{H}\left(1^\lambda\right)$ and output $\mathrm{msk} := \alpha$ and $\mathrm{pp}_\kappa := \left(\mathrm{des}, \mathbb{GD}_N, g_1, g_1^{\boldsymbol{h}}, U_1, V_1, g_3, Y, \mathrm{H}\right)$.

KeyGen $(\mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{kInd})$: If $\mathrm{kInd} \notin \mathbb{X}_\kappa$, return $\perp$. Compute $(\boldsymbol{k}, m_2) := \mathrm{Enc1}(\kappa, \mathrm{kInd})$ (let $m_1 = |\boldsymbol{k}|$). Pick $\boldsymbol{r} \leftarrow \mathbb{Z}_N^{m_2}$, $\boldsymbol{R}_3 \leftarrow \mathbb{G}_{p_3}^{m_1}$, and compute $\boldsymbol{K} := g_1^{\boldsymbol{k}(\mathrm{msk}, \boldsymbol{r}, \boldsymbol{h})} \cdot \boldsymbol{R}_3$. Output $\mathrm{sk} := (\mathrm{kInd}, \boldsymbol{K})$.

Encaps $(\mathrm{pp}_\kappa, \mathrm{cInd})$ for $\mathrm{cInd} \in \mathbb{Y}_\kappa$: Compute $(\boldsymbol{c}, w_2) := \mathrm{Enc2}(\kappa, \mathrm{cInd})$ (let $w_1 = |\boldsymbol{c}|$). Pick $s \leftarrow \mathbb{Z}_N$, $\boldsymbol{s} \leftarrow \mathbb{Z}_N^{w_2}$, and compute $\boldsymbol{C} := g_1^{\boldsymbol{c}(s, \boldsymbol{s}, \boldsymbol{h})} = (C_1, \ldots, C_{w_1})$. Compute

$$t := \mathrm{H}\big(\mathrm{cInd}, \mathrm{e}(g_1, C_1), \ldots, \mathrm{e}(g_1, C_{w_1})\big) \in \mathbb{Z}_N \tag{4.2}$$

and $C'' := \left(U_1^t \cdot V_1\right)^s$. Set $\mathrm{CT} := (\mathrm{cInd}, \boldsymbol{C}, C'')$, $\mathrm{K} := Y^s$, and output $(\mathrm{K}, \mathrm{CT})$. The ciphertext space for $\mathrm{cInd} \in \mathbb{Y}_\kappa$ is $\mathbb{C}_\mathrm{cInd} := \{\mathrm{cInd}\} \times \mathbb{G}^{w_1+1}$.

Note that, given $\mathrm{CT} \in \mathbb{C}_\mathrm{cInd}$, the corresponding hash value can be computed efficiently. We denote by **HInput** (CT) the input of the hash function as defined in (4.2).

Decaps $(\mathrm{pp}_\kappa, \mathrm{sk}, \mathrm{CT})$: Parse $\mathrm{sk} = (\mathrm{kInd}, \boldsymbol{K})$. Output $\perp$ if $\mathrm{CT} \notin \mathbb{C}_\mathrm{cInd} \subseteq \mathbb{C}_\mathrm{cInd}$. Otherwise parse $\mathrm{CT} \in \mathbb{C}_\mathrm{cInd}$ by $\mathrm{CT} = \left(\mathrm{cInd}, C_0, \overrightarrow{\mathbf{C}}, \mathbf{C}''\right)$. Output $\perp$ if $\mathrm{R}_\kappa(\mathrm{kInd}, \mathrm{cInd}) \neq 1$. Compute $t := \mathrm{H}(\mathrm{HInput}(\mathrm{CT}))$ and $\boldsymbol{E} \leftarrow \mathrm{Pair}(\kappa, \mathrm{kInd}, \mathrm{cInd})$. Output $\perp$, if one of the following checks fails:

$$\mathrm{e}(C'', g_1) \stackrel{?}{=} \mathrm{e}\left(C_1, U_1^t \cdot V_1\right) \ , \tag{4.3}$$

$$\mathrm{e}(C'', g_3) \stackrel{?}{=} 1 \text{ and } \forall_{i \in [w_1]} : \mathrm{e}(C_i, g_3) \stackrel{?}{=} 1 \ , \tag{4.4}$$

$$\mathrm{Vrfy}\left(\mathrm{des}, \mathbb{GD}_N, g_1, g_1^{\boldsymbol{h}}, \mathrm{kInd}, \mathrm{cInd}, \boldsymbol{E}, \boldsymbol{C}\right) \stackrel{?}{=} 1 \ , \tag{4.5}$$

where $w_1 = |\boldsymbol{C}|$. Otherwise, output $\mathrm{K} := \mathrm{e}\left(\boldsymbol{K}^{\boldsymbol{E}}, \boldsymbol{C}\right)$.

In comparison to the original CPA-secure framework of [Att14a] we only add the hash function H and the group elements $U_1, V_1 \in \mathbb{G}$ to the public parameter. The user secret keys are not changed at all. The encapsulation is extended by a single group element $C'' \in \mathbb{G}$. The checks in (4.3), (4.4) and (4.5) are new in the decapsulation algorithm. We call these checks *consistency checks* or *verification checks* and explain them in more detail below.

Correctness is based mainly on the correctness of pair encoding and the completeness of the verification algorithm. Since it is important to understand the functionality of the scheme before we explain our extensions, we first consider the correctness of the scheme.

Correctness of the Framework

In this subsection we first show that the algorithms of our framework, especially the key generation algorithm and the encapsulation algorithm, are ppt algorithms with respect to the security parameter. Then we present the correctness proof for our construction.

Lemma 4.5. *(Key computability) For every security parameter $\lambda \in \mathbb{N}$, every composite-order group description $\mathbb{GD} = \left(p_1, p_2, p_3, (g, \mathbb{G}), \mathbb{G}_{\mathrm{T}}, \mathrm{e}\right) \in \left[\mathcal{G}\left(1^{\lambda}\right)\right]$ and every $\mathrm{des} \in \Omega$ one can efficiently compute*

$$g_1^{\boldsymbol{k}(\alpha,\boldsymbol{r},\boldsymbol{h})} \in \mathbb{G}_{p_1}^{m_1} \ ,$$

given $\mathbb{GD}_N$, $g_1 \in \mathbb{G}_{p_1}$, $g_1^{\boldsymbol{h}} \in \mathbb{G}_{p_1}^n$, $\alpha \in \mathbb{Z}_N$ (or g_1^{α}), and $\boldsymbol{r} \in \mathbb{Z}_N^{m_2}$. Here, $N = p_1p_2p_3$, $\mathbb{GD}_N$ is the restricted description of $\mathbb{GD}$, $\kappa = (\mathrm{des}, N)$, $n = \mathrm{Param}(\kappa)$, $\mathrm{kInd} \in \mathbb{X}_{\kappa}$ is arbitrary, $(\boldsymbol{k}, m_2) = \mathrm{Enc1}(\kappa, \mathrm{kInd})$, and $m_1 = |\boldsymbol{k}|$.

Proof. Due to the restrictions on polynomials, for every $\tau \in [m_1]$ the polynomials in $\boldsymbol{k}$ are of the form $k_\tau = a_\tau \cdot \mathrm{X}_\alpha + \sum_{i\in[m_2]} \left(a_{\tau,i} \cdot \mathrm{X}_{r_i} + \sum_{j\in[n]} a_{\tau,i,j} \cdot \mathrm{X}_{h_j} \cdot \mathrm{X}_{r_i}\right)$. Hence, given the coefficient of the polynomials (given by $\mathrm{Enc1}(\kappa, \mathrm{kInd})$), we can compute for every τ:

$$\begin{aligned} g_1^{k_\tau(\alpha,\boldsymbol{r},\boldsymbol{h})} &= (g_1^{\alpha})^{a_\tau} \cdot g_1^{\sum_{i\in[m_2]} a_{\tau,i}\cdot r_i} \cdot \prod_{j\in[n]} \left(g_1^{h_j}\right)^{\sum_{i\in[m_2]} a_{\tau,i,j}\cdot r_i} \\ &= g_1^{a_\tau\cdot\alpha + \sum_{i\in[m_2]}\left(a_{\tau,i}\cdot r_i + \sum_{j\in[n]} a_{\tau,i,j}\cdot h_j\cdot r_i\right)} \ . \end{aligned}$$

This proves the lemma. □

The following lemma and the proof are analogous for the encapsulation algorithm.

Lemma 4.6. *(Ciphertext computability) For every security parameter $\lambda \in \mathbb{N}$, every group description $\mathbb{GD} = \left(p_1, p_2, p_3, (g, \mathbb{G}), \mathbb{G}_{\mathrm{T}}, \mathrm{e}\right) \in \left[\mathcal{G}\left(1^{\lambda}\right)\right]$ and every $\mathrm{des} \in \Omega$ one can efficiently compute*

$$g_1^{\boldsymbol{c}(s_0,\boldsymbol{s},\boldsymbol{h})} \in \mathbb{G}_{p_1}^{w_1} \ ,$$

given $\mathbb{GD}_N$, $g_1 \in \mathbb{G}_{p_1}$, $g_1^{\boldsymbol{h}} \in \mathbb{G}_{p_1}^n$, $s_0 \in \mathbb{Z}_N$, and $\boldsymbol{s} \in \mathbb{Z}_N^{w_2}$. Here, $N = p_1p_2p_3$, $\mathbb{GD}_N$ is the restricted description of $\mathbb{GD}$, $\kappa = (\mathrm{des}, N)$, $n = \mathrm{Param}(\kappa)$, $\mathrm{cInd} \in \mathbb{Y}_{\kappa}$ is arbitrary, $(\boldsymbol{c}, w_2) = \mathrm{Enc2}(\kappa, \mathrm{cInd})$, and $w_1 = |\boldsymbol{c}|$.

Proof. Due to the restrictions on polynomials, for every $\tau \in [w_1]$ the polynomials in $\boldsymbol{c}$ are of the form $c_\tau = \sum_{i\in[w_2]_0} \left(b_{\tau,i} \cdot \mathrm{X}_{s_i} + \sum_{j\in[n]} b_{\tau,i,j} \cdot \mathrm{X}_{h_j} \cdot \mathrm{X}_{s_i}\right)$. Hence, given the coefficient of the polynomials we can compute:

$$\begin{aligned} g_1^{c_\tau(s_0,\boldsymbol{s},\boldsymbol{h})} &= g_1^{\sum_{i\in[w_2]_0} b_{\tau,i}\cdot s_i} \cdot \prod_{j\in[n]} \left(g_1^{h_j}\right)^{\sum_{i\in[w_2]_0} b_{\tau,i,j}\cdot s_i} \\ &= g_1^{\sum_{i\in[w_2]_0}\left(b_{\tau,i}\cdot s_i + \sum_{j\in[n]} b_{\tau,i,j}\cdot h_j\cdot s_i\right)} \ . \end{aligned}$$

This proves the lemma. □

Next we present the correctness proof for our framework.

Proof. (*Correctness of* P-KEM Π *achieved using the framework*) The elements from $g_1^{\boldsymbol{k}(\mathrm{msk},\boldsymbol{r},\boldsymbol{h})}$ can be efficiently computed from $g_1^{\boldsymbol{h}}$, msk, and $\boldsymbol{r}$ due to Lemma 4.5. Analogously, the elements from $g_1^{\boldsymbol{c}(s,\boldsymbol{s},\boldsymbol{h})}$ can be computed due to Lemma 4.6. Furthermore, since $\boldsymbol{K} \in \mathbb{G}^{m_1}$, the elements of $\boldsymbol{K}^{\boldsymbol{E}} \in \mathbb{G}^{w_1}$ can be efficiently computed one by one as described in Subsection 4.1.1.

Let $\lambda \in \mathbb{N}$, $\mathrm{des} \in \Omega$, and $\left(\mathrm{msk}, \left(\mathrm{des}, \mathbb{GD}_N, g_1, g_1^{\boldsymbol{h}}, U_1, V_1, g_3, Y, \mathrm{H}\right)\right) \in \left[\mathrm{Setup}\left(1^\lambda, \mathrm{des}\right)\right]$ be arbitrary but fixed and let $\kappa = (\mathrm{des}, N)$. Furthermore, let $\mathrm{kInd} \in \mathbb{X}_\kappa$ and $\mathrm{cInd} \in \mathbb{Y}_\kappa$ be arbitrary but fixed such that $\mathrm{R}_\kappa(\mathrm{kInd}, \mathrm{cInd}) = 1$. In turn, let $\mathrm{sk} = (\mathrm{kInd}, \boldsymbol{K}) \in [\mathrm{KeyGen}(\mathrm{msk}, \mathrm{kInd})]$ and $(\mathrm{K}, \mathrm{CT}) = (\mathrm{K}, (\mathrm{cInd}, \boldsymbol{C}, C'')) \in [\mathrm{Encaps}(\mathrm{cInd})]$ be arbitrary. Suppose $\mathrm{K} = Y^s$ for some $s \in \mathbb{Z}_N$. Then, by construction, there exist $\boldsymbol{r} \in \mathbb{Z}_N^{m_2}$, $\boldsymbol{s} \in \mathbb{Z}_N^{w_2}$, and $\boldsymbol{R}_3 \in \mathbb{G}_{p_3}^{m_1}$ such that $\boldsymbol{K} = g_1^{\boldsymbol{k}(\mathrm{msk}, \boldsymbol{r}, \boldsymbol{h})} \cdot \boldsymbol{R}_3$, and $\boldsymbol{C} = g_1^{\boldsymbol{c}(s, \boldsymbol{s}, \boldsymbol{h})}$, where $\boldsymbol{h} \pmod{p_1}$ is defined by $g_1^{\boldsymbol{h}} \in \mathrm{pp}_\kappa$. Furthermore, by construction $C'' = \left(U_1^t \cdot V_1\right)^s$, where t is the hash value for the encapsulation as defined in the algorithm Encaps. Finally, by the normality of pair encoding it holds $C_1 = g_1^s$.

The decapsulation algorithm Decaps computes $\boldsymbol{E} \leftarrow \mathrm{Pair}(\kappa, \mathrm{kInd}, \mathrm{cInd})$. The checks in (4.4) are satisfied since the elements in $\boldsymbol{C}$ and C'' do not contain $\mathbb{G}_{p_3}$ components. The check in (4.3) is satisfied due to the form of C_1 and C'':

$$\mathrm{e}\left(C'', g_1\right) = \mathrm{e}\left(g_1, g_1\right)^{(u \cdot t + v) \cdot s} = \mathrm{e}\left(C_1, U_1^t \cdot V_1\right) \quad .$$

The check in (4.5) is satisfied since Vrfy outputs 1 due to its completeness property.

All consistency checks are satisfied, and hence the decapsulation algorithm outputs

$$\begin{aligned} \mathrm{e}\left(\boldsymbol{K}^{\boldsymbol{E}}, \boldsymbol{C}\right) &= \mathrm{e}\left(\left(g_1^{\boldsymbol{k}(\mathrm{msk}, \boldsymbol{r}, \boldsymbol{h})} \cdot \boldsymbol{R}_3\right)^{\boldsymbol{E}}, g_1^{\boldsymbol{c}(s, \boldsymbol{s}, \boldsymbol{h})}\right) \\ &= \mathrm{e}\left(g_1, g_1\right)^{\boldsymbol{k}(\mathrm{msk}, \boldsymbol{r}, \boldsymbol{h}) \cdot \boldsymbol{E} \cdot \boldsymbol{c}(s, \boldsymbol{s}, \boldsymbol{h})} = \mathrm{e}\left(g_1, g_1\right)^{\mathrm{msk} \cdot s} = \mathrm{K} \quad , \end{aligned}$$

where in the second step $\boldsymbol{R}_3$ disappears since $\boldsymbol{C}$ does not contain $\mathbb{G}_{p_3}$ components, whereas the third equation holds due to the correctness of the pair encoding scheme. Hence, the predicate-based key encapsulation mechanism Π is correct. □

4.2.3. Intuition Behind the Consistency Checks

In this subsection we provide a high-level explanation of why the consistency checks render the decapsulation oracle useless to any ppt adversary. Our explanation in this section leaves out many important details of the formal proof.

Assume for a moment that $\mathcal{A}$ queries the decapsulation oracle on $\mathrm{CT} = (\mathrm{cInd}, \boldsymbol{C}, C'') \in \mathbb{C}_{\mathrm{cInd}}$ such that the group elements of CT contain *only the* $\mathbb{G}_{p_1}$ *components* and CT passes the consistency checks. Let $\mathrm{sk} = (\mathrm{kInd}, \boldsymbol{K}) = \left(\mathrm{kInd}, g_1^{\boldsymbol{k}(\mathrm{msk}, \boldsymbol{r}, \boldsymbol{h})} \cdot \boldsymbol{R}_3\right)$ be the user secret key generated for this query. Under the mentioned assumption and especially since CT passes the check in (4.5) it holds $\mathrm{e}\left(\boldsymbol{K}^{\boldsymbol{E}}, \boldsymbol{C}\right) = \mathrm{e}\left(g_1, C_1\right)^{\mathrm{msk}}$. Hence, the output of the decapsulation algorithm will be $\mathrm{e}\left(g_1, C_1\right)^{\mathrm{msk}} = Y^s$, where $Y \in \mathrm{pp}_\kappa$ and s is defined by the $\mathbb{G}_{p_1}$ component g_1^s of C_1. Next, our additional element C'' and the consistency check in (4.3) guarantee that s is known to $\mathcal{A}$, since otherwise $\mathcal{A}$ would not be able to compute C'', which passes the check in (4.3). That is, element C'' is a kind of proof of knowledge for the exponent s. But then, $\mathcal{A}$ knows Y and s and can compute Y^s, which makes the decapsulation oracle useless.

We still have to justify the assumption that the elements in CT contain only the $\mathbb{G}_{p_1}$ components. The remaining checks in (4.4) guarantee that the elements of CT contain no $\mathbb{G}_{p_3}$ components. Then, essentially the subgroup decision assumptions ensure that the group elements in CT does not contain $\mathbb{G}_{p_2}$ components. The main difficulty in the security proof will be to prove this statement formally.

The following lemmata are important for the security proof. Lemma 4.7 shows that due to the consistency checks in (4.4) and in (4.5), there is no difference which *normal key* is used in the decapsulation algorithm for reconstruction of the encapsulated key.

Lemma 4.7. *Let* $\lambda \in \mathbb{N}$, $\mathrm{des} \in \Omega$, $(\mathrm{msk}, \mathrm{pp}_\kappa) \in \left[\mathrm{Setup}\left(1^\lambda, \mathrm{des}\right)\right]$, $\mathrm{kInd} \in \mathbb{X}_\kappa$, $\mathrm{cInd} \in \mathbb{Y}_\kappa$ *be arbitrary such that it holds* $\mathrm{R}_\kappa(\mathrm{kInd}, \mathrm{cInd}) = 1$. *Furthermore, let* $\mathrm{sk} = (\mathrm{kInd}, \boldsymbol{K}) \in [\mathrm{KeyGen}(\mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{kInd})]$, $\boldsymbol{E} \in [\mathrm{Pair}(\kappa, \mathrm{kInd}, \mathrm{cInd})]$, *and* $\mathrm{CT} \in \mathbb{C}_{\mathrm{cInd}}$ *be arbitrary such that* CT *passes the consistency check in* (4.5) *with respect to* $\boldsymbol{E}$ *and also passes the consistency checks in* (4.4). *Then,*

$$\mathrm{e}\left(\boldsymbol{K}^{\boldsymbol{E}}, \boldsymbol{C}\right) = Y^s \enspace ,$$

where $Y \in \mathrm{pp}_\kappa$ *and* $s \pmod{p_1}$ *is defined by the* $\mathbb{G}_{p_1}$ *component* g_1^s *of* $C_1 \in \boldsymbol{C}$.

Proof. Let $(\boldsymbol{k}, m_2) = \mathrm{Enc1}(\kappa, \mathrm{kInd})$, $m_1 = |\boldsymbol{k}|$, and $(\boldsymbol{c}, w_2) = \mathrm{Enc2}(\kappa, \mathrm{cInd})$, $w_1 = |\boldsymbol{c}|$. It holds $\mathrm{CT} \in \mathbb{C}_{\mathrm{cInd}}$ and we can parse $\mathrm{CT} = (\mathrm{cInd}, \boldsymbol{C}, C'')$, where $\boldsymbol{C} \in \mathbb{G}^{w_1}$ and $C'' \in \mathbb{G}$. Furthermore, $\mathrm{sk} \in [\mathrm{KeyGen}(\mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{kInd})]$, and hence there exist $\boldsymbol{r} \in \mathbb{Z}_N^{m_2}$ and $\boldsymbol{R}_3 \in \mathbb{G}_{p_3}^{m_1}$ such that $\boldsymbol{K} = g_1^{\boldsymbol{k}(\mathrm{msk}, \boldsymbol{r}, \boldsymbol{h})} \cdot \boldsymbol{R}_3$, where $g_1^{\boldsymbol{h}} \in \mathrm{pp}_\kappa$. Due to the consistency checks in (4.4) the group elements in $\boldsymbol{C}$ do not contain $\mathbb{G}_{p_3}$ components. We deduce that it holds:

$$\begin{aligned}\mathrm{e}\left(\boldsymbol{K}^{\boldsymbol{E}}, \boldsymbol{C}\right) &= \mathrm{e}\left(\left(g_1^{\boldsymbol{k}(\mathrm{msk}, \boldsymbol{r}, \boldsymbol{h})} \cdot \boldsymbol{R}_3\right)^{\boldsymbol{E}}, \boldsymbol{C}\right) \\ &= \mathrm{e}\left(g_1^{\boldsymbol{k}(\mathrm{msk}, \boldsymbol{r}, \boldsymbol{h}) \cdot \boldsymbol{E}}, \boldsymbol{C}\right) \enspace .\end{aligned}$$

Furthermore, CT passes the consistency check in (4.5), and hence by the soundness property of Vrfy it holds:

$$\mathrm{e}\left(g_1^{\boldsymbol{k}(\mathrm{msk}, \boldsymbol{r}, \boldsymbol{h}) \cdot \boldsymbol{E}}, \boldsymbol{C}\right) = \mathrm{e}\left(g_1, C_1\right)^{\mathrm{msk}} \enspace ,$$

where C_1 is the first element of $\boldsymbol{C}$. By definition of Setup it holds $Y = \mathrm{e}(g_1, g_1)^{\mathrm{msk}} \in \mathrm{pp}_\kappa$ and consequently

$$\mathrm{e}\left(\boldsymbol{K}^{\boldsymbol{E}}, \boldsymbol{C}\right) = Y^s \enspace ,$$

where $s \pmod{p_1}$ is defined through the $\mathbb{G}_{p_1}$ component g_1^s of C_1. This finally proves the lemma. □

Lemma 4.7 is essential for the following lemma, which states that the output distribution of $\mathrm{Decaps}(\mathrm{pp}_\kappa, \mathrm{sk}_1, \mathrm{CT})$ and $\mathrm{Decaps}(\mathrm{pp}_\kappa, \mathrm{sk}_2, \mathrm{CT})$ are identical for every $\mathrm{CT} \in \{0,1\}^*$ as long as sk_1 and sk_2 are correctly generated normal secret keys for the same key index.

Lemma 4.8. *Let* $\lambda \in \mathbb{N}$, $\mathrm{des} \in \Omega$, $(\mathrm{msk}, \mathrm{pp}_\kappa) \in \left[\mathrm{Setup}\left(1^\lambda, \mathrm{des}\right)\right]$, $\mathrm{kInd} \in \mathbb{X}_\kappa$ *be arbitrary. Furthermore, let* $\mathrm{sk}_1, \mathrm{sk}_2 \in [\mathrm{KeyGen}(\mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{kInd})]$, *and* $\mathrm{CT} \in \{0,1\}^*$ *be arbitrary, too. Then, the output distributions of* $\mathrm{Decaps}(\mathrm{pp}_\kappa, \mathrm{sk}_1, \mathrm{CT})$ *and* $\mathrm{Decaps}(\mathrm{pp}_\kappa, \mathrm{sk}_1, \mathrm{CT})$ *are equal. In particular, for every* $k \in \mathbb{G}_\mathrm{T} \cup \{\perp\}$ *it holds*

$$\Pr\left[\mathrm{K} = k : \mathrm{K} \leftarrow \mathrm{Decaps}(\mathrm{pp}_\kappa, \mathrm{sk}_1, \mathrm{CT})\right] = \Pr\left[\mathrm{K} = k : \mathrm{K} \leftarrow \mathrm{Decaps}(\mathrm{pp}_\kappa, \mathrm{sk}_2, \mathrm{CT})\right] \enspace .$$

Proof. Let $\lambda \in \mathbb{N}$, $\mathrm{des} \in \Omega$, $(\mathrm{msk}, \mathrm{pp}_\kappa) \in \left[\mathrm{Setup}\left(1^\lambda, \mathrm{des}\right)\right]$, $\mathrm{kInd} \in \mathbb{X}_\kappa$, secret keys $\mathrm{sk}_1, \mathrm{sk}_2 \in [\mathrm{KeyGen}(\mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{kInd})]$, and $\mathrm{CT} \in \{0,1\}^*$ be arbitrary, but fixed. Let $\mathrm{sk}_1 = (\mathrm{kInd}, \boldsymbol{K}_1)$, $\mathrm{sk}_2 = (\mathrm{kInd}, \boldsymbol{K}_2)$. We consider only the case that there exists an index $\mathrm{cInd} \in \mathbb{Y}_\kappa$ such that $\mathrm{CT} \in \mathbb{C}_{\mathrm{cInd}}$ and $\mathrm{R}_\kappa(\mathrm{kInd}, \mathrm{cInd}) = 1$, since otherwise the decapsulation algorithm Decaps will output $\perp$ independently of the secret key. Hence, we can parse $\mathrm{CT} = (\mathrm{cInd}, \boldsymbol{C}, C'')$, where $\boldsymbol{C} \in \mathbb{G}^{w_1}$ and $C'' \in \mathbb{G}$.

Both probability distributions are over the random choice of $\boldsymbol{E} \leftarrow \mathrm{Pair}(\kappa, \mathrm{kInd}, \mathrm{cInd})$. The choice of $\boldsymbol{E} \in \mathbb{Z}_N^{m_1 \times w_1}$ depends on kInd, but is independent of the concrete secret key for kInd. Hence, every $\boldsymbol{E} \in [\mathrm{Pair}(\kappa, \mathrm{kInd}, \mathrm{cInd})]$ is chosen with the same probability in both cases. Let $\boldsymbol{E} \in [\mathrm{Pair}(\kappa, \mathrm{kInd}, \mathrm{cInd})]$ be arbitrary, but fixed. We claim that independently of the concrete secret key, the result of the decapsulation algorithm using $\boldsymbol{E}$ will be the same. This will immediately prove the lemma.

It is important to notice that for a fixed $\boldsymbol{E}$ the consistency checks are deterministic. In particular, Vrfy is a deterministic algorithm by definition. Furthermore, if one of the consistency checks fails, the output of the decapsulation algorithm will be $\perp$ independently of the concrete secret key. Hence, it remains to consider the case that $\mathrm{CT} = (\mathrm{cInd}, \boldsymbol{C}, C'')$ passes all consistency checks. In this case we can apply Lemma 4.7 and deduce from the definition of Decaps that it holds:

$$\mathrm{Decaps}(\mathrm{pp}_\kappa, \mathrm{sk}_1, \mathrm{CT}) = \mathrm{e}\left(\boldsymbol{K}_1^{\boldsymbol{E}}, \boldsymbol{C}\right) = Y^s = \mathrm{e}\left(\boldsymbol{K}_2^{\boldsymbol{E}}, \boldsymbol{C}\right) = \mathrm{Decaps}(\mathrm{pp}_\kappa, \mathrm{sk}_2, \mathrm{CT})$$

where $Y \in \mathrm{pp}_\kappa$ and $s \pmod{p_1}$ is defined by the $\mathbb{G}_{p_1}$ component g_1^s of $C_1 \in \boldsymbol{C}$. This finally proves the lemma, since every $\boldsymbol{E}$ is chosen with the same probability in both probability distributions, as explained above. □

In summary, we state that the consistency checks in (4.4) and in (4.5) ensure that the output distribution of the decapsulation algorithm is the same for every correctly generated normal user secret key. Furthermore, all consistency checks together ensure that the decapsulation oracle is useless for a ppt adversary, since it either knows the results of the decapsulation or will be not able to compute a ciphertext, that passes all consistency checks.

4.2.4. Semi-Functional Algorithms

The following semi-functional algorithms are basically from [Att14a] and are essential to prove the adaptive security of the original and our extended framework. The main idea is to extend the keys and the ciphertexts with components from the $\mathbb{G}_{p_2}$ subgroup. These modifications cannot be noticed by a ppt adversary, mainly due to the subgroup decision assumptions and since the public parameters do not contain a generator of $\mathbb{G}_{p_2}$. We extended the algorithms from [Att14a] by semi-functional components for our additional elements in the public parameters (U_1, V_1) and in the encapsulation (C'').

SFSetup $\left(1^\lambda, \mathrm{des}\right)$: Generate $\mathbb{GD} = \left(p_1, p_2, p_3, (g, \mathbb{G}), \mathbb{G}_\mathrm{T}, \mathrm{e}\right) \leftarrow \mathcal{G}\left(1^\lambda\right)$, and $(\mathrm{msk}, \mathrm{pp}_\kappa) \leftarrow \mathrm{Setup}\left(1^\lambda, \mathrm{des}; \mathbb{GD}\right)$, $g_2 \leftarrow \mathbb{G}_{p_2}$, $\hat{\boldsymbol{h}} \leftarrow \mathbb{Z}_{p_2}^n$ and $\hat{u}_2, \hat{v}_2 \leftarrow \mathbb{Z}_{p_2}$. Output the master secret and the public parameter as well as the semi-functional components $\left(\mathrm{msk}, \mathrm{pp}_\kappa, g_2, \hat{\boldsymbol{h}}, \hat{u}_2, \hat{v}_2\right)$.
We explicitly notice that g_2 is a generator of $\mathbb{G}_{p_2}$ by our convention.

SFKeyGen $\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{kInd}, \mathrm{type}, \hat{\alpha}, g_2, \hat{\boldsymbol{h}}\right)$ for $\hat{\alpha} \in \mathbb{Z}_N$, $\mathrm{type} \in \{1, 2, 3\}$: Generate a key by $(\mathrm{kInd}, \boldsymbol{K}_1) \leftarrow \mathrm{KeyGen}(\mathrm{msk}, \mathrm{kInd})$, pick $\hat{\boldsymbol{r}} = (\hat{r}_1, \ldots, \hat{r}_{m_2}) \leftarrow \mathbb{Z}_N^{m_2}$, and compute

$$\widehat{\boldsymbol{K}} := \begin{cases} g_2^{\boldsymbol{k}(0, \hat{\boldsymbol{r}}, \hat{\boldsymbol{h}})} & \text{if type} = 1 \\ g_2^{\boldsymbol{k}(\hat{\alpha}, \hat{\boldsymbol{r}}, \hat{\boldsymbol{h}})} & \text{if type} = 2 \\ g_2^{\boldsymbol{k}(\hat{\alpha}, 0, \boldsymbol{0})} & \text{if type} = 3 \end{cases}.$$

Set $\boldsymbol{K} := \boldsymbol{K}_1 \cdot \widehat{\boldsymbol{K}}$ and output a semi-functional key $\mathrm{sk} := (\mathrm{kInd}, \boldsymbol{K})$.

SFEncaps $\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{cInd}, g_2, \hat{\boldsymbol{h}}, \hat{u}_2, \hat{v}_2\right)$: Choose $s \leftarrow \mathbb{Z}_N$ and generate $(\mathrm{K}, (\mathrm{cInd}, \boldsymbol{C}_1, \_)) \leftarrow \mathrm{Encaps}(\mathrm{cInd}; s)$. Choose $\hat{s} \leftarrow \mathbb{Z}_N$, $\hat{\boldsymbol{s}} \leftarrow \mathbb{Z}_N^{w_2}$ and compute $\widehat{\boldsymbol{C}} := g_2^{\boldsymbol{c}(\hat{s}, \hat{\boldsymbol{s}}, \hat{\boldsymbol{h}})}$. Next, set $\boldsymbol{C} :=$

$\boldsymbol{C}_1 \cdot \widehat{\boldsymbol{C}}$, compute the hash value $t = \mathrm{H}(\mathrm{HInput}(\mathrm{cInd}, \boldsymbol{C}, \_))$ and $C'' := \left(U_1^t \cdot V_1\right)^s \cdot \left(g_2^{\hat{u}_2 \cdot t} \cdot g_2^{\hat{v}_2}\right)^{\hat{s}}$. Output key K and $\mathrm{CT} = (\mathrm{cInd}, \boldsymbol{C}, C'') \in \mathbb{C}_{\mathrm{cInd}}$.

Notice that all semi-functional components of user secret keys and of the ciphertexts can be computed in the same way as the corresponding normal components using g_2 and $g_2^{\hat{\boldsymbol{h}}}$ instead of g_1 and $g_1^{\boldsymbol{h}}$.

Remark 4.9. Note that differently from [Att14a], we define the semi-functional common elements $\hat{h}_1, \ldots, \hat{h}_n$, $\hat{u}_2$, and $\hat{v}_2$ as uniformly distributed elements in $\mathbb{Z}_{p_2}$ instead of $\mathbb{Z}_N$. All these elements are used only in the exponents of $g_2 \in \mathbb{G}_{p_2}$, and hence by Chinese Remainder Theorem, we did not change the distributions of the user secret keys and the distributions of encapsulations. However, our variant simplifies the argumentation in the proofs.

5. Security of the Framework and the Extended Proof Technique

In this chapter we present the security proof for our framework from the previous chapter. Therefore, we especially go into detail of our proof technique, which adapts and extends the dual system encryption methodology [Wat09a, LW10] to the case of CCA-security.

5.1. Main Theorem and an Overview of the Proof

In this section we first state our main theorem and then explain the proof technique.

Theorem 5.1. *Let* Π *be the* P-KEM *from Section 4.2.2. Suppose that the subgroup decision assumptions from Section 4.1.2 are correct, the underlying pair encoding scheme* P *is selectively and co-selectively master key hiding, and* $\mathcal{H}$ *is a family of collision-resistant hash functions. Then,* Π *is fully* CCA2*-secure with respect to Definition 4.2. In particular, for every* $\mathrm{des} \in \Omega$ *and every ppt algorithm* $\mathcal{A}$*, there exists a negligible function* negl *and there exist ppt algorithms* $\mathcal{B}_1, \ldots, \mathcal{B}_6$ *with essentially the same running time as* $\mathcal{A}$ *such that for sufficiently large* λ *it holds*

$$\begin{aligned}\mathrm{Adv\text{-}aP\text{-}KEM}^{\mathrm{CCA2}}_{\Pi,\mathcal{A}}(\lambda,\mathrm{des}) \leq\ & \mathrm{Adv}^{\mathrm{CR}}_{\mathcal{H},\mathcal{B}_1}(\lambda) + \mathrm{Adv}^{\mathrm{SD1}}_{\mathcal{B}_2}(\lambda) + \mathrm{Adv}^{\mathrm{SD3}}_{\mathcal{B}_4}(\lambda) \\ & + (2q_1+4)\cdot \mathrm{Adv}^{\mathrm{SD2}}_{\mathcal{B}_3}(\lambda) + \mathrm{Adv}^{\mathrm{SMH}}_{\mathrm{P},\mathcal{B}_6}(\lambda,\mathrm{des}) \\ & + q_1 \cdot \mathrm{Adv}^{\mathrm{CMH}}_{\mathrm{P},\mathcal{B}_5}(\lambda,\mathrm{des}) + \frac{q_{\mathrm{dec1}}}{2^\lambda} + \mathrm{negl}(\lambda) \quad ,\end{aligned}$$

where q_1 *is the number of keys that are corrupted in* **Phase I** *and* q_{dec1} *is the number of decapsulation queries in* **Phase I** *of experiment* $\mathrm{aP\text{-}KEM}^{\mathrm{CCA2}}_{\Pi,\mathcal{A}}(\lambda,\mathrm{des})$.

For simplicity, we collected some negligible terms such as $\frac{1}{p_1} \leq \frac{2}{2^\lambda}$ in $\mathrm{negl}(\lambda)$. It is important to notice that the number of decapsulation queries from **Phase I** only appears in the term $\frac{q_{\mathrm{dec1}}}{2^\lambda}$ and decreases the security guarantees only negligibly. Furthermore, in comparison to the CPA-secure framework of [Att14a] we only lose the additional terms $\mathrm{Adv}^{\mathrm{CR}}_{\mathcal{H},\mathcal{B}_1}(\lambda)$ and $\mathrm{Adv}^{\mathrm{SD2}}_{\mathcal{B}_3}(\lambda)$.

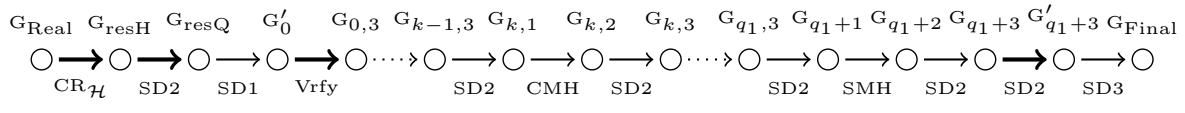


Figure 5.1.: Proof structure for the framework in prime-order groups.

The structure for the proof of Theorem 5.1 is presented in Fig. 5.1. The nodes represent different probability experiments. In Table 5.1 the modifications between the probability experiments are defined. These modifications are explained in detail in the corresponding proofs. The first experiment G_{Real} is the target experiment $\mathrm{aP\text{-}KEM}^{\mathrm{CCA2}}_{\Pi,\mathcal{A}}(\lambda,\mathrm{des})$ from Fig. 4.2 and the last experiment is constructed in such a way that the advantage of every adversary is zero. The edges represent reduction steps and their labels the underlying security assumptions, except for

the edge labeled with Vrfy. The corresponding proof is based on the verifiability property of the pair encoding scheme. In the proof we show that no ppt algorithm can distinguish between any pair of consecutive experiments. The formal proof of Theorem 5.1 is given in Section 5.2. Here, we explain the main steps of the proof and the proof technique.

$\mathrm{G}_{\mathrm{resH}}$:	Modify $\langle 10 \rangle$	Output is 0 if there is a collision for H
$\mathrm{G}_{\mathrm{resQ}}$:	Modify $\langle 10 \rangle$	Output 0 if $\mathcal{A}$ implicitly found a factor of N.
G'_0:	Modify $\langle 1 \rangle$ Modify $\langle 5 \rangle$	$\left(\mathrm{msk}, \mathrm{pp}, g_2, \hat{\boldsymbol{h}}, \hat{u}_2, \hat{v}_2\right) \leftarrow \mathrm{SFSetup}\left(1^\lambda, \mathrm{des}\right)$ $(\mathrm{K}_0, \mathrm{CT}^*) \leftarrow \mathrm{SFEncaps}\left(\mathrm{cInd}^*, g_2, \hat{\boldsymbol{h}}, \hat{u}_2, \hat{v}_2\right)$
$\mathrm{G}_{0,3}$:	Modify$\langle 4 \rangle$, $\langle 9 \rangle$ Change	$\mathrm{sk}'_i \leftarrow \mathrm{KeyGen}\left(\mathrm{msk}, \mathrm{kInd}_i\right)$, $\mathrm{Decaps}\left(\mathrm{sk}'_i, \mathrm{CT}\right)$ Generate keys in Open oracle.
$\mathrm{G}_{k,1}$:	Modify $\langle 3 \rangle$	$\hat{\alpha}_j \leftarrow \mathbb{Z}_N$, $\mathrm{sk}_j \leftarrow \begin{cases} \mathrm{SFKeyGen}\left(\mathrm{msk}, \mathrm{kInd}, 3, \hat{\alpha}_j, g_2, \_\right) & \text{if } j < k \\ \mathrm{SFKeyGen}\left(\mathrm{msk}, \mathrm{kInd}, 1, \_, g_2, \hat{\boldsymbol{h}}\right) & \text{if } j = k \\ \mathrm{KeyGen}\left(\mathrm{msk}, \mathrm{kInd}\right) & \text{if } j > k \end{cases}$
$\mathrm{G}_{k,2}$:	Modify $\langle 3 \rangle$	$\hat{\alpha}_j \leftarrow \mathbb{Z}_N$, $\mathrm{sk}_j \leftarrow \begin{cases} \mathrm{SFKeyGen}\left(\mathrm{msk}, \mathrm{kInd}, 3, \hat{\alpha}_j, g_2, \_\right) & \text{if } j < k \\ \mathrm{SFKeyGen}\left(\mathrm{msk}, \mathrm{kInd}, 2, \hat{\alpha}_j, g_2, \hat{\boldsymbol{h}}\right) & \text{if } j = k \\ \mathrm{KeyGen}\left(\mathrm{msk}, \mathrm{kInd}\right) & \text{if } j > k \end{cases}$
$\mathrm{G}_{k,3}$:	Modify $\langle 3 \rangle$	$\hat{\alpha}_j \leftarrow \mathbb{Z}_N$, $\mathrm{sk}_j \leftarrow \begin{cases} \mathrm{SFKeyGen}\left(\mathrm{msk}, \mathrm{kInd}, 3, \hat{\alpha}_j, g_2, \_\right) & \text{if } j \leq k \\ \mathrm{KeyGen}\left(\mathrm{msk}, \mathrm{kInd}\right) & \text{if } j > k \end{cases}$
G_{q_1+1}:	Modify $\langle 8 \rangle$	$\mathrm{SFKeyGen}\left(\mathrm{msk}, \mathrm{kInd}, 1, \_, g_2, \hat{\boldsymbol{h}}\right)$
G_{q_1+2}:	Insert Modify $\langle 8 \rangle$	$\hat{\alpha} \leftarrow \mathbb{Z}_N$ at the beginning of Phase II $\mathrm{SFKeyGen}\left(\mathrm{msk}, \mathrm{kInd}, 2, \hat{\alpha}, g_2, \hat{\boldsymbol{h}}\right)$
G_{q_1+3}:	Modify $\langle 8 \rangle$	$\mathrm{SFKeyGen}\left(\mathrm{msk}, \mathrm{kInd}, 3, \hat{\alpha}, g_2, \_\right)$
G'_{q_1+3}:	Insert Modify $\langle 4 \rangle$, $\langle 9 \rangle$	$X_2 \leftarrow \mathbb{G}_{p_2}$ in the Setup phase Check consistency, return $\mathrm{e}\left(g_1^{\mathrm{msk}} \cdot X_2, C_1\right)$
$\mathrm{G}_{\mathrm{Final}}$:	Modify $\langle 6 \rangle$	$\mathrm{K}^* \leftarrow \mathbb{G}_{\mathrm{T}}$

Table 5.1.: The probability experiments for the framework in composite-order groups.

The structure of the proof for our CCA-secure construction is similar to the structure of the proof for the CPA-secure construction of [Att14a]. Experiments $\mathrm{G}_{\mathrm{resH}}$, $\mathrm{G}_{\mathrm{resQ}}$, G'_0, and G'_{q_1+3} as well as the four reduction steps denoted by bold edges in Fig. 5.1 are new. The remaining experiments and reductions are from the original CPA-security proof from [Att14a] and require only simple extensions.

Our first reduction $\mathrm{G}_{\mathrm{Real}} \rightarrow \mathrm{G}_{\mathrm{resH}}$ is based on the security of the family of collision-resistant hash functions. In the second reduction $\mathrm{G}_{\mathrm{resH}} \rightarrow \mathrm{G}_{\mathrm{resQ}}$ we separate failure events which enable us to find a non-trivial factor of N, which violates Assumption SD2 by Lemma 4.1. This reduction is an extension of the first reduction step from [Att14a]. These two steps are of a technical nature. Our additional games G'_0 and G'_{q_1+3} and the corresponding new reductions $\mathrm{G}'_0 \rightarrow \mathrm{G}_{0,3}$ and $\mathrm{G}_{q_1+3} \rightarrow \mathrm{G}'_{q_1+3}$ are the most important parts of the CCA-security proof and enable us to deal with decapsulation queries in an elegant way. The major modification in $\mathrm{G}_{0,3}$ is that the decapsulation queries are answered using separately generated *normal keys* which we denote by sk'_i. We do not change these keys to semi-functional in the following games. In particular, using consistency check (4.5) we show that for every (unconditional) $\mathcal{A}$, experiments

G'_0 and $G_{0,3}$ are indistinguishable. The next important observation is that in all reductions between $G_{0,3}$ and G_{q_1+3}, the master secret key is known to the reduction algorithm. Hence, the normal keys for the decapsulation queries can be generated by the key generation algorithm. The final challenge is to answer decapsulation queries without the user secret keys in the last experiment G_{Final}. Experiment G'_{q_1+3} and the corresponding new reduction step $G_{q_1+3} \to G'_{q_1+3}$ allow us to deal with this problem. In the proof of this reduction step we use our additional group element from the encapsulation in order to answer the decapsulation queries. To prove that this modification cannot be noticed, again the consistency checks are crucial (see the proof of Lemma 5.23).

Verifiability of pair encoding schemes. Our framework requires pair encodings that satisfy additional properties defined in Subsection 4.2.1. As mentioned there, the normality of pair encodings is a natural restriction, which is the first required property. In this paragraph we explain how to construct appropriate verification algorithms for pair encoding schemes according to Definition 4.3. Together with our framework, this provides new, fully CCA-secure PE schemes for various predicates. Among these are an IBE scheme, the scheme for regular languages and its dual, new and reviewed key-policy and ciphertext-policy attribute-base schemes, spatial and negated spatial encryption, key-policy over doubly spatial encryption, as well as dual-policy attribute-based schemes. All (nineteen) pair encoding schemes from [Att14a, AY15] satisfy the verifiability property according to Definition 4.3. Almost all of these encoding schemes are *regular*. The following theorem leads to verification algorithms for all these schemes. We refer to Section 5.3 for the constructive proof of this theorem.

Theorem 5.2. *Suppose $\mathcal{R}_{\Omega,\Sigma}$ is a domain-transferable predicate family and* P *is a regular pair encoding scheme for $\mathcal{R}_{\Omega,\Sigma}$. Then,* P *satisfies the verifiability property according to Definition 4.3.*

The two ciphertext-policy attribute-based encryption schemes from [Att14a] achieved using dual scheme conversion are not regular. These schemes were improved in [AY15] and the resulting schemes are regular. In any case, verification algorithms can be constructed using a slightly adapted technique also for these schemes.

5.2. Security Proof of the CCA-Secure Pair Encoding Framework

In this section we present a formal proof of Theorem 5.1. We start with an analysis of semi-functional algorithms and with general lemmata that will be used in the main proof, which is then presented in Subsection 5.2.3.

5.2.1. On the Distribution of Semi-Functional Components

In this subsection we prove some useful lemmata about the output distributions of semi-functional algorithms. The semi-functional algorithms are essential for the main proof, since the normal keys and the normal challenge encapsulation are changed to their semi-functional counterparts. That is, the $\mathbb{G}_{p_2}$ components, which we also call semi-functional, are appended to the group elements of the challenge and to the group elements of the user secret keys. Notice that SFKeyGen and SFEncaps take as input a generator of $\mathbb{G}_{p_2}$. However, if we look at security assumption SD2, which is used for most of reductions in the main proof, we recognize that the generator of $\mathbb{G}_{p_2}$ is not given. The same holds also for assumption SD1. Hence, we cannot use the semi-functional algorithms in the form presented in Subsection 4.2.4 in order to generate

the keys and the challenge encapsulation. However, in experiment SD1 a generator of $\mathbb{G}_{p_1p_2}$ is given, whereas in experiment SD2 a generator of $\mathbb{G}_{p_1p_2}$ and a generator of $\mathbb{G}_{p_2p_3}$ are known. These generators are used to generate the semi-functional components. Hence, first of all we carefully look at the output distributions of semi-functional algorithms and prove that these distributions are independent of the concrete generator of $\mathbb{G}_{p_2}$. This holds mainly due to the linearity of pair encodings (cf. Lemma 1.10). The statements in the following lemmata are essential for the proof, but were not mentioned in the proof of the CPA-framework [Att14a].

The first lemma states that as long as $\hat{\alpha}$ is chosen uniformly at random the output distribution of SFKeyGen is independent of the concrete generator of $\mathbb{G}_{p_2}$ given as input.

Lemma 5.3. *Let $\lambda \in \mathbb{N}$, $\mathrm{des} \in \Omega$, group description $\mathbb{GD} = \left(p_1, p_2, p_3, (g, \mathbb{G}), \mathbb{G}_\mathrm{T}, \mathrm{e}\right) \in \left[\mathcal{G}\left(1^\lambda\right)\right]$, $\left(\mathrm{msk}, \mathrm{pp}_\kappa, g_2, \hat{\boldsymbol{h}}, \_, \_\right) \in \left[\mathrm{SFSetup}\left(1^\lambda, \mathrm{des}; \mathbb{GD}\right)\right]$, $\mathrm{kInd} \in \mathbb{X}_\kappa$, $\mathrm{type} \in \{1,2,3\}$, and generator $\tilde{g}_2 \in \mathbb{G}_{p_2}$ be arbitrary. Then, for every $\mathrm{sk}' \in \mathbb{UK}_{\mathrm{kInd}}$ it holds*

$$\begin{aligned}&\Pr\left[\mathrm{sk} = \mathrm{sk}' : \alpha \leftarrow \mathbb{Z}_N, \mathrm{sk} \leftarrow \mathrm{SFKeyGen}\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{kInd}, \mathrm{type}, \alpha, g_2, \hat{\boldsymbol{h}}\right)\right]\\ &= \Pr\left[\mathrm{sk} = \mathrm{sk}' : \alpha \leftarrow \mathbb{Z}_N, \mathrm{sk} \leftarrow \mathrm{SFKeyGen}\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{kInd}, \mathrm{type}, \alpha, \tilde{g}_2, \hat{\boldsymbol{h}}\right)\right] \quad .\end{aligned}$$

Proof. We prove the lemma for all key types simultaneously. Let $(\boldsymbol{k}, m_2) = \mathrm{Enc1}\left(\kappa, \mathrm{kInd}\right)$, $m_1 = |\boldsymbol{k}|$. By definition of SFSetup element $g_2 \in \mathbb{G}_{p_2}$ is a generator. Element $\tilde{g}_2 \in \mathbb{G}_{p_2}$ is a generator by preconditions. Hence, there exists $x \in \mathbb{Z}_{p_2}^*$ such that $\tilde{g}_2 = g_2^x$.

Let us denote the secret key produced in the first probability experiment by $\widehat{\mathrm{sk}}$ and the random scalar by $\widehat{\alpha}$. For the second experiment we denote the corresponding elements by $\widetilde{\mathrm{sk}}$ and by $\tilde{\alpha}$. By definition of SFKeyGen it holds $\widehat{\mathrm{sk}}, \widetilde{\mathrm{sk}} \in \mathbb{UK}_{\mathrm{kInd}}$. Hence, we can parse $\widehat{\mathrm{sk}} = (\mathrm{kInd}, \boldsymbol{K}_1)$ and $\widetilde{\mathrm{sk}} = (\mathrm{kInd}, \boldsymbol{K}_2)$, where $\boldsymbol{K}_1, \boldsymbol{K}_2 \in \mathbb{G}^{m_1}$. By construction of SFKeyGen, the input values $\hat{\alpha}$, g_2, and $\hat{\boldsymbol{h}}$ affect only the $\mathbb{G}_{p_2}$ components of the group elements in the generated key. Furthermore, these components are generated independently from the $\mathbb{G}_{p_1}$ components and from the $\mathbb{G}_{p_3}$ components. Hence, it is sufficient to consider the distributions of the $\mathbb{G}_{p_2}$ components of group elements in $\boldsymbol{K}_1$ and in $\boldsymbol{K}_2$.

Notice that $\hat{\boldsymbol{h}}$ is fixed. In the first probability space, the $\mathbb{G}_{p_2}$ components $\widehat{\boldsymbol{K}}$ of $\boldsymbol{K}_1$ are determined by the mutually independent random variables $\hat{\boldsymbol{r}}$ (defined by SFKeyGen) and by $\hat{\alpha}$. In the second probability space, the $\mathbb{G}_{p_2}$ components $\widetilde{\boldsymbol{K}}$ of $\boldsymbol{K}_2$ are determined by the mutually independent random variables $\tilde{\boldsymbol{r}}$, (defined by SFKeyGen) and by $\tilde{\alpha}$. Namely, it holds:

$$\widehat{\boldsymbol{K}} = \begin{cases} g_2^{\boldsymbol{k}\left(0, \hat{\boldsymbol{r}}, \hat{\boldsymbol{h}}\right)} & \text{if type} = 1\\ g_2^{\boldsymbol{k}\left(\hat{\alpha}, \hat{\boldsymbol{r}}, \hat{\boldsymbol{h}}\right)} & \text{if type} = 2\\ g_2^{\boldsymbol{k}\left(\hat{\alpha}, \boldsymbol{0}, \boldsymbol{0}\right)} & \text{if type} = 3 \end{cases} \quad \text{and} \quad \widetilde{\boldsymbol{K}} = \begin{cases} \tilde{g}_2^{\boldsymbol{k}\left(0, \tilde{\boldsymbol{r}}, \hat{\boldsymbol{h}}\right)} = g_2^{\boldsymbol{k}\left(0, x\cdot\tilde{\boldsymbol{r}}, \hat{\boldsymbol{h}}\right)} & \text{if type} = 1\\ \tilde{g}_2^{\boldsymbol{k}\left(\tilde{\alpha}, \tilde{\boldsymbol{r}}, \hat{\boldsymbol{h}}\right)} = g_2^{\boldsymbol{k}\left(x\cdot\tilde{\alpha}, x\cdot\tilde{\boldsymbol{r}}, \hat{\boldsymbol{h}}\right)} & \text{if type} = 2\\ \tilde{g}_2^{\boldsymbol{k}\left(\tilde{\alpha}, \boldsymbol{0}, \boldsymbol{0}\right)} = g_2^{\boldsymbol{k}\left(x\cdot\tilde{\alpha}, \boldsymbol{0}, \boldsymbol{0}\right)} & \text{if type} = 3 \end{cases},$$

where we used linearity property (1.1) of pair encodings from Lemma 1.10 to transform elements of $\widetilde{\boldsymbol{K}}$. For the keys of Type 1 and of Type 2 the values $\hat{\boldsymbol{r}}$ and $x \cdot \tilde{\boldsymbol{r}}$ are uniformly distributed over $\mathbb{Z}_{p_2}^{m_2}$ due to the choices of $\hat{\boldsymbol{r}}$ and $\tilde{\boldsymbol{r}}$ respectively, since $x \neq 0 \pmod{p_2}$. Additionally, for the keys of Type 2 and of Type 3 the values $\hat{\alpha}$ and $x \cdot \tilde{\alpha}$ are uniformly distributed over $\mathbb{Z}_{p_2}$ due to the choices of $\hat{\alpha}$ and $\tilde{\alpha}$ respectively, since $x \neq 0 \pmod{p_2}$. Hence, we deduce that the $\mathbb{G}_{p_2}$ components of the group elements in the keys are identically distributed in both probability experiments. As mentioned above, this implies that $\boldsymbol{K}_1$ and $\boldsymbol{K}_2$ are identically distributed Consequently, $\widehat{\mathrm{sk}}$ and $\widetilde{\mathrm{sk}}$ are identically distributed. □

The next lemma is very similar and states that the output distribution of the semi-functional encapsulation algorithm is independent of the concrete generator of $\mathbb{G}_{p_2}$ given as input.

Lemma 5.4. *Let* $\lambda \in \mathbb{N}$, $\mathrm{des} \in \Omega$, *group description* $\mathbb{GD} = \left(p_1, p_2, p_3, (g, \mathbb{G}), \mathbb{G}_\mathrm{T}, \mathrm{e}\right) \in \left[\mathcal{G}\left(1^\lambda\right)\right]$, $\left(\mathrm{msk}, \mathrm{pp}_\kappa, g_2, \hat{\boldsymbol{h}}, \hat{u}_2, \hat{v}_2\right) \in \left[\mathrm{SFSetup}\left(1^\lambda, \mathrm{des}; \mathbb{GD}\right)\right]$, $\mathrm{cInd} \in \mathbb{Y}_\kappa$, *and generator* $\tilde{g}_2 \in \mathbb{G}_{p_2}$ *be arbitrary. Then, for every* $(\mathrm{K}', \mathrm{CT}') \in \left[\mathrm{SFEncaps}\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{cInd}, g_2, \hat{\boldsymbol{h}}, \hat{u}_2, \hat{v}_2\right)\right]$ *it holds*

$$\begin{aligned}&\Pr\left[(\mathrm{K}, \mathrm{CT}) = \left(\mathrm{K}', \mathrm{CT}'\right) : (\mathrm{K}, \mathrm{CT}) \leftarrow \mathrm{SFEncaps}\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{cInd}, g_2, \hat{\boldsymbol{h}}, \hat{u}_2, \hat{v}_2\right)\right]\\ &= \Pr\left[(\mathrm{K}, \mathrm{CT}) = \left(\mathrm{K}', \mathrm{CT}'\right) : (\mathrm{K}, \mathrm{CT}) \leftarrow \mathrm{SFEncaps}\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{cInd}, \tilde{g}_2, \hat{\boldsymbol{h}}, \hat{u}_2, \hat{v}_2\right)\right] \quad .\end{aligned}$$

Proof. Let $(\boldsymbol{c}, w_2) = \mathrm{Enc2}\left(\kappa, \mathrm{cInd}\right)$, $w_1 = |\boldsymbol{c}|$. By definition of SFSetup element $g_2 \in \mathbb{G}_{p_2}$ is a generator and element $\tilde{g}_2 \in \mathbb{G}_{p_2}$ is a generator by preconditions. Hence, there exists $x \in \mathbb{Z}_{p_2}^*$ such that $\tilde{g}_2 = g_2^x$. The encapsulated keys are uniquely defined by their encapsulations by construction of SFEncaps. Hence, it is sufficient to consider the distributions of the corresponding encapsulations.

Let us denote the encapsulations produced in the first and in the second probability experiment by $\widehat{\mathrm{CT}}$ and by $\widetilde{\mathrm{CT}}$, respectively. By definition of SFEncaps it holds $\widehat{\mathrm{CT}}, \widetilde{\mathrm{CT}} \in \mathbb{C}_{\mathrm{cInd}}$. Hence, we can parse $\widehat{\mathrm{CT}} = (\mathrm{cInd}, \boldsymbol{C}_1, C_1'')$ and $\widetilde{\mathrm{CT}} = (\mathrm{cInd}, \boldsymbol{C}_2, C_2'')$, where $\boldsymbol{C}_1, \boldsymbol{C}_2 \in \mathbb{G}^{w_1}$, $C_1'', C_2'' \in \mathbb{G}$. We additionally denote $\boldsymbol{C}_1 = (C_{1,1}, \ldots, C_{1,w_1})$ and $\boldsymbol{C}_2 = (C_{2,1}, \ldots, C_{2,w_1})$.

First of all we claim that $\boldsymbol{C}_1 = \boldsymbol{C}_2$ implies $C_1'' = C_2''$. By construction of SFEncaps it holds

$$C_1'' = \left(U_1^t \cdot V_1\right)^s \cdot \left(g_2^{\hat{u}_2 \cdot t} \cdot g_2^{\hat{v}_2}\right)^{\hat{s}} = \left(g_1^s\right)^{u_1 \cdot t + v_1} \cdot \left(g_2^{\hat{s}}\right)^{\hat{u}_2 \cdot t + \hat{v}_2} \quad ,$$

where g_1^s is the $\mathbb{G}_{p_1}$ component of $C_{1,1}$, $g_2^{\hat{s}}$ is the $\mathbb{G}_{p_2}$ component of $C_{1,1}$, t is the hash value uniquely determined by cInd and $\boldsymbol{C}_1$, and u_1 and v_1 are fixed by $U_1, V_1 \in \mathrm{pp}_\kappa$. Hence, if $\boldsymbol{C}_1 = \boldsymbol{C}_2$, then $C_{1,1} = C_{2,1}$ which in turn implies $C_1'' = C_2''$. Using this result, it is sufficient to prove that the distributions of $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$ are identical.

By the definition of SFEncaps, the input values g_2 and $\hat{\boldsymbol{h}}$ only affects the $\mathbb{G}_{p_2}$ components of the group elements in the generated encapsulation. Furthermore, these components are generated independently from the $\mathbb{G}_{p_1}$ components and all these elements do not contain the $\mathbb{G}_{p_3}$ components. Hence, it is sufficient to consider the distributions of the $\mathbb{G}_{p_2}$ components of $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$.

In the first probability space, the $\mathbb{G}_{p_2}$ components $\widehat{\boldsymbol{C}}$ of $\boldsymbol{C}_1$ are determined by the mutually independent random variables $\hat{s}$ and $\hat{\boldsymbol{s}}$ (defined by SFEncaps). In the second probability space, the $\mathbb{G}_{p_2}$ components $\widetilde{\boldsymbol{C}}$ of $\boldsymbol{C}_2$ are determined by the mutually independent random variables $\tilde{s}$ and $\tilde{\boldsymbol{s}}$ (defined by SFEncaps). Namely, it holds:

$$\widehat{\boldsymbol{C}} = g_2^{\boldsymbol{c}\left(\hat{s}, \hat{\boldsymbol{s}}, \hat{\boldsymbol{h}}\right)} \quad \text{and} \quad \widetilde{\boldsymbol{C}} = \tilde{g}_2^{\boldsymbol{c}\left(\tilde{s}, \tilde{\boldsymbol{s}}, \hat{\boldsymbol{h}}\right)} = g_2^{\boldsymbol{c}\left(x \cdot \tilde{s}, x \cdot \tilde{\boldsymbol{s}}, \hat{\boldsymbol{h}}\right)} \quad ,$$

where we used linearity property (1.2) of pair encodings from Lemma 1.10 to transform elements of $\widetilde{\boldsymbol{C}}$. The values $\hat{s}, x \cdot \tilde{s} \in \mathbb{Z}_{p_2}$, and $\hat{\boldsymbol{s}}, x \cdot \tilde{\boldsymbol{s}} \in \mathbb{Z}_{p_2}^{w_2}$ are uniformly distributed due to the choice of the corresponding random values since $x \neq 0 \pmod{p_2}$. Hence, we deduce that $\widehat{\boldsymbol{C}}$ and $\widetilde{\boldsymbol{C}}$ are identically distributed. As mentioned above, this implies that $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$ are identically distributed too. From this we finally deduce that the considered output distributions of SFEncaps are identical. □

5.2.2. Supplementary Algorithms

In this subsection we show how to generate correctly distributed elements of the scheme in different settings. All algorithms from this subsection are used in several reductions. Partially,

these algorithms were (implicitly) presented in the proof of the original CPA-secure framework from [Att14a]. We separately define these algorithms in order to avoid repetitions. Furthermore, we formally state and prove the achieved properties, which makes the corresponding reductions in the main proof more comprehensible and brings the main parts of the proofs into focus.

Simulation of the Semi-Functional Public Parameters

By the definition of SFSetup, the semi-functional public parameters for the predicate R_κ consist of the normal public parameters pp_κ for composite-order bilinear groups

$$\mathbb{GD} = \left(p_1, p_2, p_3, (g, \mathbb{G}), \mathbb{G}_\mathrm{T}, \mathrm{e}\right) \ ,$$

a generator $\hat{g}_2 \in \mathbb{G}_{p_2}$, a vector $\hat{\boldsymbol{h}} \in \mathbb{Z}_{p_2}^n$ ($n = \mathrm{Param}(\kappa)$), and two additional elements $\hat{u}_2, \hat{v}_2 \in \mathbb{Z}_{p_2}$. For fixed $\mathbb{GD}$ all additional elements are generated independently of the normal public parameters pp_κ and remain hidden in the realization of the scheme. In this subsection we show how to generate properly distributed semi-functional public parameters (except for $g_2 \in \mathbb{G}_{p_2}$) given $\mathrm{des} \in \Omega$, the restricted group description $\mathbb{GD}_N$, $g_1 \in \mathbb{G}_{p_1}$, and $g_3 \in \mathbb{G}_{p_3}$. Consequently, this simulation algorithm can be used with all three security assumptions.

Consider the following ppt algorithm, which we call SimPP:

Algorithm 1: SimPP

Input : $(\mathrm{des}, \mathbb{GD}_N, g_1, g_3)$.

1 Set $\kappa := (\mathrm{des}, N)$ and compute $n := \mathrm{Param}(\kappa)$.
2 Pick $\alpha \leftarrow \mathbb{Z}_N$ and compute $Y := \mathrm{e}(g_1, g_1)^\alpha$.
3 Pick $\boldsymbol{h} \leftarrow \mathbb{Z}_N^n$ and $u, v \leftarrow \mathbb{Z}_N$. Compute $g_1^{\boldsymbol{h}}$, $U_1 := g_1^u$ and $V_1 := g_1^v$.
4 Choose a hash function $\mathrm{H} \leftarrow \mathcal{H}\left(1^\lambda\right)$.
5 Define $\mathrm{msk} := \alpha$ and $\mathrm{pp}_\kappa := \left(\mathrm{des}, \mathbb{GD}_N, g_1, g_1^{\boldsymbol{h}}, U_1, V_1, g_3, Y, \mathrm{H}\right)$.

Output : $(\mathrm{msk}, \mathrm{pp}_\kappa, \boldsymbol{h}, u, v)$.

Recall that 1^λ can be computed from $\mathbb{GD}_N$ by our convention, since $|p_i| = \lambda$. We deduce that Algorithm SimPP is a ppt algorithm with respect to λ by construction. Furthermore, notice that SimPP outputs $\boldsymbol{h} \in \mathbb{Z}_N^n$ and $u, v \in \mathbb{Z}_N$, whereas the semi-functional components are $\hat{\boldsymbol{h}} \in \mathbb{Z}_{p_2}^n$, $\hat{u}_2, \hat{v}_2 \in \mathbb{Z}_{p_2}$. Indeed, in the corresponding proofs we will set $\hat{\boldsymbol{h}} := \boldsymbol{h} \pmod{p_2}$, $\hat{u}_2 := u \pmod{p_2}$, and $\hat{v}_2 := v \pmod{p_2}$. With this setting in mind we prove the following lemma.

Lemma 5.5 (Algorithm SimPP)**.** *For every $\lambda \in \mathbb{N}$ and every $\mathrm{des} \in \Omega$ the following properties are satisfied:*

1. (Assignments) *For every $\mathbb{GD} = \left(p_1, p_2, p_3, (g, \mathbb{G}), \mathbb{G}_\mathrm{T}, \mathrm{e}\right) \in \left[\mathcal{G}\left(1^\lambda\right)\right]$, every $g_1 \in \mathbb{G}_{p_1}$, $g_3 \in \mathbb{G}_{p_3}$, and every $(\mathrm{msk}, \mathrm{pp}_\kappa, \boldsymbol{h}, u, v) \in [\mathrm{SimPP}(\mathrm{des}, \mathbb{GD}_N, g_1, g_3)]$ it holds*

$$\mathrm{pp}_\kappa = \left(\mathrm{des}, \mathbb{GD}_N, g_1, g_1^{\boldsymbol{h}}, U_1, V_1, g_3, Y, \mathrm{H}\right) \ ,$$

   $u, v \in \mathbb{Z}_N$, $\boldsymbol{h} \in \mathbb{Z}_N^n$, where $n = \mathrm{Param}(\kappa)$. Furthermore, $U_1 = g_1^u$, $V_1 = g_1^v$.

2. (Distribution of public parameter) *For every $\left(\mathrm{msk}', \mathrm{pp}'_\kappa\right) \in \left[\mathrm{Setup}\left(1^\lambda, \mathrm{des}\right)\right]$ it holds*

$$\begin{aligned}
&\Pr\left[(\mathrm{msk}, \mathrm{pp}_\kappa) = \left(\mathrm{msk}', \mathrm{pp}'_\kappa\right) : \begin{array}{l}\mathbb{GD} \leftarrow \mathcal{G}\left(1^\lambda\right), g_1 \leftarrow \mathbb{G}_{p_1}, g_3 \leftarrow \mathbb{G}_{p_3}, \\ (\mathrm{msk}, \mathrm{pp}_\kappa, _, _, _) \leftarrow \mathrm{SimPP}(\mathrm{des}, \mathbb{GD}_N, g_1, g_3)\end{array}\right] \\
&= \Pr\left[(\mathrm{msk}, \mathrm{pp}_\kappa) = \left(\mathrm{msk}', \mathrm{pp}'_\kappa\right) : (\mathrm{msk}, \mathrm{pp}_\kappa) \leftarrow \mathrm{Setup}\left(1^\lambda, \mathrm{des}\right)\right] \\
&= \Pr\left[(\mathrm{msk}, \mathrm{pp}_\kappa) = \left(\mathrm{msk}', \mathrm{pp}'_\kappa\right) : (\mathrm{msk}, \mathrm{pp}_\kappa, _, _, _, _) \leftarrow \mathrm{SFSetup}\left(1^\lambda, \mathrm{des}\right)\right] \ .
\end{aligned}$$

3. (Distribution of semi-functional components) *For every* $\mathbb{GD} = \left(p_1, p_2, p_3, (g, \mathbb{G}), \mathbb{G}_\mathrm{T}, \mathrm{e}\right) \in \mathcal{G}\left(1^\lambda\right)$*, all generators* $g_1 \in \mathbb{G}_{p_1}$*,* $g_3 \in \mathbb{G}_{p_3}$*, every* $\left(\mathrm{msk}', \mathrm{pp}'_\kappa\right) \in \mathrm{Setup}\left(1^\lambda, \mathrm{des}; \mathbb{GD}, g_1, g_3\right)$*, and every* $\boldsymbol{v} \in \mathbb{Z}_{p_2}^{n+2}$*, where* $n = \mathrm{Param}(\kappa)$*, it holds*

$$\begin{aligned}
&\Pr\left[(\boldsymbol{h}, u, v) = \boldsymbol{v} \pmod{p_2} \;\middle|\; \mathrm{pp}_\kappa = \mathrm{pp}'_\kappa : (_, \mathrm{pp}_\kappa, \boldsymbol{h}, u, v) \leftarrow \mathrm{SimPP}\left(\mathrm{des}, \mathbb{GD}_N, g_1, g_3\right)\right] \\
&= \Pr\left[(\boldsymbol{h}, u, v) = \boldsymbol{v} \pmod{p_2} : (_, _, \boldsymbol{h}, u, v) \leftarrow \mathrm{SimPP}\left(\mathrm{des}, \mathbb{GD}_N, g_1, g_3\right)\right] \\
&= \Pr\left[\left(\hat{\boldsymbol{h}}, \hat{u}_2, \hat{v}_2\right) = \boldsymbol{v} : \left(_, _, _, \hat{\boldsymbol{h}}, \hat{u}_2, \hat{v}_2\right) \leftarrow \mathrm{SFSetup}\left(1^\lambda, \mathrm{des}; \mathbb{GD}, \mathrm{pp}'_\kappa\right)\right] \\
&= \left(\frac{1}{p_2}\right)^{n+2} .
\end{aligned}$$

That is, $\boldsymbol{h}$*,* u*, and* v *modulo* p_2 *in the output of* SimPP *are uncorrelated with generated public parameters* pp_κ*.*

Proof. The first property in the lemma holds by construction of SimPP. Consider the second property defined in the lemma. In the first probability space, $\mathbb{GD}$, g_1 and g_3 are generated identically to the generation of $\mathbb{GD}$, g_1 and g_3 in the algorithm Setup. All other elements of the public parameters and the master secret key are properly distributed by construction of SimPP. The last equation holds by the definition of SFSetup, which generated the master secret key and the public parameters using Setup.

Finally, consider the last property. By definition of SFSetup vector $\boldsymbol{h} \in \mathbb{Z}_{p_2}^n$, and $\hat{u}, \hat{v} \in \mathbb{Z}_{p_2}$ are mutually independent and uniformly distributed, which implies the last equation. Next, by construction of SimPP the values $(\boldsymbol{h}, u, v) = \boldsymbol{v} \pmod{p_2}$ are uniformly distributed, which implies the second equation. In turn, due to the first property in the lemma elements $\boldsymbol{h}$, u and v modulo p_1 in the output of SimPP are correlated with pp_κ because of $g_1^{\boldsymbol{h}}$, U_1 and V_1. However, by the Chinese Remainder Theorem these values are uncorrelated with $\boldsymbol{h}$, u and v modulo p_2 and modulo p_3. In particular, $(\boldsymbol{h}, u, v) \pmod{p_2}$ are not correlated with public parameters and we conclude that the first equation holds. □

Note that SimPP outputs not only properly distributed semi-functional public parameters, but also the exponents which are correlated with the public parameters modulo p_1. This will be exploited in the following reductions in order to generate properly distributed semi-functional keys and semi-functional encapsulations without a generator of $\mathbb{G}_{p_2}$.

Simulation of the Semi-Functional Encapsulation

From the previous section we know that using SimPP we can generate properly distributed (semi-functional) public parameters (except for $g_2 \in \mathbb{G}_{p_2}$). In this section we show how to generate correctly distributed semi-functional encapsulation (see the definition on page 80) given a generator of $\mathbb{G}_{p_1 p_2}$.

Consider the following algorithm SimSFChlg:

Algorithm 2: SimSFChlg - simulation of semi-functional challenge from $\mathrm{SD2}(\lambda)$

Input : $(\mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{cInd}, \boldsymbol{h}', u, v, X_1 X_2)$.
Require : $\mathrm{pp}_\kappa = \left(\mathrm{des}, \mathbb{GD}_N, g_1, g_1^{\boldsymbol{h}}, U_1, V_1, g_3, Y, \mathrm{H}\right)$.
1 Compute $(\boldsymbol{c}, w_2) := \mathrm{Enc2}(\kappa, \mathrm{cInd})$.
2 Pick $s' \leftarrow \mathbb{Z}_N$ and $\boldsymbol{s}' = \left(s'_1, \ldots, s'_{w_2}\right) \leftarrow \mathbb{Z}_N^{w_2}$. Compute $\boldsymbol{C} := (X_1 X_2)^{\mathbf{c}(s', \boldsymbol{s}', \boldsymbol{h}')}$.
3 Compute $t := \mathrm{H}(\mathrm{HInput}(\mathrm{cInd}, \boldsymbol{C}, \_))$ and $C'' := (X_1 X_2)^{s' \cdot (u \cdot t + v)}$.
4 Compute $\mathrm{K} := \mathrm{e}(X_1 X_2, g_1)^{\mathrm{msk} \cdot s'}$ and set $\mathrm{CT} := (\mathrm{cInd}, \boldsymbol{C}, C'')$.
Output : $(\mathrm{K}, \mathrm{CT})$.

Recall that 1^λ can be computed from $\mathbb{GD}_N$ by our convention, since $|p_i| = \lambda$. We deduce that SimSFChlg is a ppt algorithm with respect to λ by construction. In particular, all exponents in the description of the algorithm can be computed explicitly. Algorithm SimSFChlg will be used in almost all following reductions for the generation of the semi-functional challenge. It is important to notice that the output distribution of SimSFChlg is independent of the generator $X_1X_2 \in \mathbb{G}_{p_1p_2}$. Hence, we state and prove the following lemma.

Lemma 5.6 (Algorithm SimSFChlg)**.** *Let* $\lambda \in \mathbb{N}$, $\mathrm{des} \in \Omega$, $\mathbb{GD} = \left(p_1, p_2, p_3, (g, \mathbb{G}), \mathbb{G}_\mathrm{T}, \mathrm{e}\right) \in \left[\mathcal{G}\left(1^\lambda\right)\right]$, *generators* $g_1, X_1 \in \mathbb{G}_{p_1}$, $X_2 \in \mathbb{G}_{p_2}$, $g_3 \in \mathbb{G}_{p_3}$, *parameters* $\left(\mathrm{msk}', \mathrm{pp}'_\kappa, g_2, \hat{\boldsymbol{h}}, \hat{u}_2, \hat{v}_2\right) \in \left[\mathrm{SFSetup}\left(1^\lambda, \mathrm{des}; \mathbb{GD}, g_1, g_3\right)\right]$, *and index* $\mathrm{cInd} \in \mathbb{Y}_\kappa$ *be arbitrary. Then, for every* $(\mathrm{K}', \mathrm{CT}') \in \left[\mathrm{SFEncaps}\left(\mathrm{pp}'_\kappa, \mathrm{cInd}, g_2, \hat{\boldsymbol{h}}, \hat{u}_2, \hat{v}_2\right)\right]$ *it holds:*

$$\begin{aligned}
&\Pr\left[(\mathrm{K}, \mathrm{CT}) = \left(\mathrm{K}', \mathrm{CT}'\right) : (\mathrm{K}, \mathrm{CT}) \leftarrow \mathrm{SFEncaps}\left(\mathrm{pp}'_\kappa, \mathrm{cInd}, g_2, \hat{\boldsymbol{h}}, \hat{u}_2, \hat{v}_2\right)\right] \\
&= \Pr\left[(\mathrm{K}, \mathrm{CT}) = \left(\mathrm{K}', \mathrm{CT}'\right) \;\middle|\; \begin{array}{c} \mathrm{pp}_\kappa = \mathrm{pp}'_\kappa, \\ \boldsymbol{h}' = \hat{\boldsymbol{h}} \pmod{p_2}, \\ u = \hat{u}_2 \pmod{p_2}, \\ v = \hat{v}_2 \pmod{p_2} \end{array}\right],
\end{aligned}$$

where the second probability distribution is determined by experiment $(\mathrm{msk}, \mathrm{pp}_\kappa, \boldsymbol{h}', u, v) \leftarrow \mathrm{SimPP}\left(\mathrm{des}, \mathbb{GD}_N, g_1, g_3\right)$, *and* $(\mathrm{K}, \mathrm{CT}) \leftarrow \mathrm{SimSFChlg}\left(\mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{cInd}, \boldsymbol{h}', u, v, X_1X_2\right)$.

Proof. Let $(\boldsymbol{c}, w_2) = \mathrm{Enc2}\left(\kappa, \mathrm{cInd}\right)$, $w_1 = |\boldsymbol{c}|$. Elements g_1, g_2, X_1 and X_2 are generators of the corresponding subgroups by preconditions. Hence, there exists $x_1 \in \mathbb{Z}^*_{p_1}$ and $x_2 \in \mathbb{Z}^*_{p_2}$ such that $X_1 = g_1^{x_1}$ and $X_2 = g_2^{x_2}$. Furthermore, notice that the public parameters are fixed and equal in both probability distributions. Hence, we denote these common public parameters by $\mathrm{pp}_\kappa = \left(\mathrm{des}, \mathbb{GD}_N, g_1, g_1^{\boldsymbol{h}}, U_1, V_1, g_3, Y, \mathrm{H}\right)$, where for the second distribution it additionally holds $g_1^{\boldsymbol{h}} = g_1^{\boldsymbol{h}'}$, $U_1 = g_1^u$ and $V_1 = g_1^v$ by the first property of Lemma 5.5.

The first probability distribution is over the interior choices of SFEncaps. On the one hand, these are the random choices of Encaps on input pp_κ and cInd: $s \leftarrow \mathbb{Z}_{p_1}$, and $\boldsymbol{s} \leftarrow \mathbb{Z}_{p_1}^{w_2}$. On the other hand, these are $\hat{s} \leftarrow \mathbb{Z}_{p_2}$, and $\hat{\boldsymbol{s}} \leftarrow \mathbb{Z}_{p_2}^{w_2}$ chosen by SFEncaps itself. Key K and its encapsulation $\mathrm{CT} = (\mathrm{cInd}, \boldsymbol{C}, C'')$ are completely determined by these random variables and by the semi-functional public parameters. Namely, it holds

$$\begin{aligned}
\mathrm{K} &= Y^s \;, \\
\boldsymbol{C} &= g_1^{\boldsymbol{c}(s, \boldsymbol{s}, \boldsymbol{h})} \cdot g_2^{\boldsymbol{c}\left(\hat{s}, \hat{\boldsymbol{s}}, \hat{\boldsymbol{h}}\right)} \;, \\
C'' &= \left(U_1^t \cdot V_1\right)^s \cdot \left(g_2^{\hat{u}_2 \cdot t + \hat{v}_2}\right)^{\hat{s}} \;,
\end{aligned}$$

where $t = \mathrm{H}\left(\mathrm{HInput}\left(\mathrm{cInd}, \boldsymbol{C}, \_\right)\right)$.

Now, consider SimSFChlg in the context of the conditional distribution defined in the lemma. All input values of SimSFChlg except for the values $\boldsymbol{h}', u, v \pmod{p_3}$ are fixed. Namely, the public parameters uniquely define the master secret key msk, $\boldsymbol{h}' \pmod{p_1}$, $u \pmod{p_1}$, and $v \pmod{p_1}$ by Lemma 5.5. Furthermore, by construction of SimSFChlg its computations are independent of $\boldsymbol{h}', u, v \pmod{p_3}$. Hence, the second probability distribution is over the interior choices of SimSFChlg. By construction of SimSFChlg the encapsulated key K is as follows

$$\begin{aligned}
\mathrm{K} &= \mathrm{e}\left(X_1X_2, g_1\right)^{\widetilde{\mathrm{msk}} \cdot s'} \\
&= \mathrm{e}\left(g_1^{x_1}, g_1\right)^{\mathrm{msk} \cdot s'} \\
&= Y^{x_1 \cdot s'} \;.
\end{aligned}$$

Hence, K is a key with $s = x_1 \cdot s' \pmod{p_1}$. Furthermore, it holds

$$\begin{aligned} \boldsymbol{C} &= (X_1X_2)^{\boldsymbol{c}(s',\boldsymbol{s}',\boldsymbol{h}')} \\ &= g_1^{x_1\cdot\boldsymbol{c}(s',\boldsymbol{s}',\boldsymbol{h}')} \cdot g_2^{x_2\cdot\boldsymbol{c}(s',\boldsymbol{s}',\boldsymbol{h}')} \\ &= g_1^{\boldsymbol{c}(x_1\cdot s',x_1\cdot\boldsymbol{s}',\boldsymbol{h})} \cdot g_2^{\boldsymbol{c}\left(x_2\cdot s',x_2\cdot\boldsymbol{s}',\hat{\boldsymbol{h}}\right)}, \end{aligned} \qquad \begin{aligned} C'' &= (X_1X_2)^{s'(u\cdot t+v)} \\ &= g_1^{x_1\cdot s'\cdot(u\cdot t+v)} \cdot g_2^{x_2\cdot s'\cdot(u\cdot t+v)} \\ &= \left(U_1^t \cdot V_1\right)^{x_1\cdot s'} \cdot \left(g_2^{\hat{u}_2\cdot t+\hat{v}_2}\right)^{x_2\cdot s'}, \end{aligned}$$

where $t = \mathrm{H}\left(\mathrm{HInput}\left(\mathrm{cInd}, \boldsymbol{C}, \_\right)\right)$. In particular, we used $\boldsymbol{h}' = \hat{\boldsymbol{h}} \pmod{p_2}$, $u = \hat{u}_2 \pmod{p_2}$, and $v = \hat{v}_2 \pmod{p_2}$ in the last equations. Hence, $\mathrm{CT} = (\mathrm{cInd}, \boldsymbol{C}, C'')$ is a semi-functional encapsulation of K with $s = x_1 \cdot s' \pmod{p_1}$, $\boldsymbol{s} = x_1 \cdot \boldsymbol{s}' \pmod{p_1}$, $\hat{s} = x_2 \cdot s' \pmod{p_2}$, and $\hat{\boldsymbol{s}} = x_2 \cdot \boldsymbol{s}' \pmod{p_2}$. Since $x_1 \neq 0 \pmod{p_1}$, $x_2 \neq 0 \pmod{p_2}$, all these elements are properly distributed due to the choice of s' and $\boldsymbol{s}'$, and due to the Chinese Remainder Theorem. This completes the proof. □

Simulation of the Semi-Functional Keys of Type 3

In this subsection, analogously to the previous subsection, we show how to generate correctly distributed semi-functional secret keys of Type 3 (see the definition on page 80) given a generator of $\mathbb{G}_{p_2p_3}$.

Consider the following algorithm SimSFKeyT3:

Algorithm 3: SimSFKeyT3 - simulation of semi-functional keys of Type 3

Input : $(\mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{kInd}, Y_2Y_3, \alpha')$.
Require : $\mathrm{pp}_\kappa = \left(\mathrm{des}, \mathbb{GD}_N, g_1, g_1^{\boldsymbol{h}}, U_1, V_1, g_3, Y, \mathrm{H}\right)$.
1 Compute $(\boldsymbol{k}, m_2) := \mathrm{Enc1}\left(\kappa, \mathrm{kInd}\right)$. Let $m_1 := |\boldsymbol{k}|$.
2 Pick $\boldsymbol{r} \leftarrow \mathbb{Z}_N^{m_2}$, $\boldsymbol{R}_3' \leftarrow \mathbb{G}_{p_3}^{m_1}$ and compute $\boldsymbol{K} := g_1^{\boldsymbol{k}(\mathrm{msk},\boldsymbol{r},\boldsymbol{h})} \cdot (Y_2Y_3)^{\boldsymbol{k}(\alpha',\boldsymbol{0},\boldsymbol{0})} \cdot \boldsymbol{R}_3'$.
Output : $\mathrm{sk} = (\mathrm{kInd}, \boldsymbol{K})$.

Recall that 1^λ can be computed from $\mathbb{GD}_N$ by our convention, since $|p_i| = \lambda$. We deduce that SimSFKeyT3 is a ppt algorithm with respect to λ by construction. In particular, $\boldsymbol{k}\left(\alpha', \boldsymbol{0}, \boldsymbol{0}\right)$ can be computed explicitly, whereas $g_1^{\boldsymbol{k}(\mathrm{msk},\boldsymbol{r}',\boldsymbol{h})}$ can be computed by Lemma 4.5. Algorithm SimSFKeyT3 will be used in almost all following reductions for the generation of the semi-functional keys of type three. It is important to notice that the output distribution of the algorithm SimSFKeyT3 is independent of the generator $Y_2Y_3 \in \mathbb{G}_{p_2p_3}$. Hence, we state and prove the following lemma.

Lemma 5.7 (Algorithm SimSFKeyT3)**.** *Let* $\lambda \in \mathbb{N}$, $\mathrm{des} \in \Omega$, $\mathbb{GD} = \left(p_1, p_2, p_3, (g, \mathbb{G}), \mathbb{G}_\mathrm{T}, \mathrm{e}\right) \in \left[\mathcal{G}\left(1^\lambda\right)\right]$, *generators* $g_1 \in \mathbb{G}_{p_1}$, $Y_2 \in \mathbb{G}_{p_2}$, $g_3, Y_3 \in \mathbb{G}_{p_3}$, *parameters* $\left(\mathrm{msk}', \mathrm{pp}_\kappa', g_2, \_, \_, \_\right) \in \left[\mathrm{SFSetup}\left(1^\lambda, \mathrm{des}; \mathbb{GD}, g_1, g_3\right)\right]$, *and* $\mathrm{kInd} \in \mathbb{X}_\kappa$ *be arbitrary. Then, for every* $\mathrm{sk}' \in \mathbb{UK}_{\mathrm{kInd}}$ *it holds*

$$\begin{aligned} &\Pr\left[\mathrm{sk} = \mathrm{sk}' : \hat{\alpha} \leftarrow \mathbb{Z}_N, \mathrm{sk} \leftarrow \mathrm{SFKeyGen}\left(\mathrm{pp}_\kappa', \mathrm{msk}', \mathrm{kInd}, 3, \hat{\alpha}, g_2, \_\right)\right] \\ &= \Pr\left[\mathrm{sk} = \mathrm{sk}' : \alpha' \leftarrow \mathbb{Z}_N, \mathrm{sk} \leftarrow \mathrm{SimSFKeyT3}\left(\mathrm{pp}_\kappa', \mathrm{msk}', \mathrm{kInd}, Y_2Y_3, \alpha'\right)\right] \\ &= \Pr\left[\mathrm{sk} = \mathrm{sk}' \;\middle|\; \mathrm{pp}_\kappa = \mathrm{pp}_\kappa' : \begin{array}{l} (\mathrm{msk}, \mathrm{pp}_\kappa, \_, \_, \_) \leftarrow \mathrm{SimPP}\left(\mathrm{des}, \mathbb{GD}_N, g_1, g_3\right), \\ \alpha' \leftarrow \mathbb{Z}_N, \mathrm{sk} \leftarrow \mathrm{SimSFKeyT3}\left(\mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{kInd}, Y_2Y_3, \alpha'\right) \end{array}\right]. \end{aligned}$$

Furthermore, SimSFKeyT3 *sets the* $\mathbb{G}_{p_2}$ *component of* $\boldsymbol{K}$ *in* $\mathrm{sk} = (\mathrm{kInd}, \boldsymbol{K})$ *to* $Y_2^{\boldsymbol{k}(\alpha',\boldsymbol{0},\boldsymbol{0})}$, *where* $(\boldsymbol{k}, \_) = \mathrm{Enc1}\left(\kappa, \mathrm{kInd}\right)$.

Proof. We will only prove the first equation, since the second equation is then implied by property 2 of Lemma 5.5. Let $(\boldsymbol{k}, m_2) = \mathrm{Enc1}\left(\kappa, \mathrm{kInd}\right)$, and $m_1 = |\boldsymbol{k}|$. Elements g_2 and Y_2 are

generators of $\mathbb{G}_{p_2}$. Hence, there exists $y \in \mathbb{Z}_{p_2}^*$ such that $Y_2 = g_2^y$. Let public parameters be $\mathrm{pp}'_\kappa = \big(\mathrm{des}, \mathbb{GD}_N, g_1, g_1^{\boldsymbol{h}}, U_1, V_1, g_3, Y, \mathrm{H}\big)$.

The first probability space is determined by $\hat{\alpha}$ and by the random variables $\boldsymbol{r} \leftarrow \mathbb{Z}_{p_1}^{m_2}$, and $\boldsymbol{R}_3 \leftarrow \mathbb{G}_{p_3}^{m_1}$ defined by SFKeyGen (or rather KeyGen as a subroutine of SFKeyGen). Vector $\boldsymbol{r}$ is uniformly distributed over $\mathbb{Z}_{p_1}^{m_2}$, whereas vector $\boldsymbol{R}_3$ is uniformly distributed over $\mathbb{G}_{p_3}^{m_1}$. The output of SFKeyGen is a secret key $\mathrm{sk} = (\mathrm{kInd}, \boldsymbol{K})$ such that:

$$\boldsymbol{K} = g_1^{\boldsymbol{k}(\mathrm{msk},\boldsymbol{r},\boldsymbol{h})} \cdot g_2^{\boldsymbol{k}(\hat{\alpha},\boldsymbol{0},\boldsymbol{0})} \cdot \boldsymbol{R}_3 \ .$$

Notice that the $\mathbb{G}_{p_2}$ components of $\boldsymbol{K}$ are fixed by the input values and $\hat{\alpha} \pmod{p_2}$ is uniformly distributed.

Now, consider SimSFKeyT3 in the context of the second probability distribution. Algorithm SimSFKeyT3 outputs $\mathrm{sk} = (\mathrm{kInd}, \boldsymbol{K})$, where $\boldsymbol{K} \in \mathbb{G}^{m_1}$ (that is $\mathrm{sk} \in \mathbb{UK}_{\mathrm{kInd}}$) and it holds

$$\begin{aligned}
\boldsymbol{K} &= g_1^{\boldsymbol{k}(\mathrm{msk},\boldsymbol{r},\boldsymbol{h})} \cdot (Y_2 Y_3)^{\boldsymbol{k}(\alpha',\boldsymbol{0},\boldsymbol{0})} \cdot \boldsymbol{R}'_3 \\
&= g_1^{\boldsymbol{k}(\mathrm{msk},\boldsymbol{r},\boldsymbol{h})} \cdot Y_2^{\boldsymbol{k}(\alpha',\boldsymbol{0},\boldsymbol{0})} \cdot Y_3^{\boldsymbol{k}(\alpha',\boldsymbol{0},\boldsymbol{0})} \cdot \boldsymbol{R}'_3 \\
&= g_1^{\boldsymbol{k}(\mathrm{msk},\boldsymbol{r},\boldsymbol{h})} \cdot g_2^{\boldsymbol{k}(y\cdot\alpha',\boldsymbol{0},\boldsymbol{0})} \cdot Y_3^{\boldsymbol{k}(\alpha',\boldsymbol{0},\boldsymbol{0})} \cdot \boldsymbol{R}'_3 \ .
\end{aligned}$$

The $\mathbb{G}_{p_1}$ components of the group elements in $\boldsymbol{K}$ are computed as defined in the (semi-functional) key generation algorithm. The $\mathbb{G}_{p_2}$ components are properly distributed, since $y \cdot \alpha' \pmod{p_2}$ is uniformly distributed due to the choice of α' and since $y \neq 0 \pmod{p_2}$. Finally, the $\mathbb{G}_{p_3}$ components of $\boldsymbol{K}$ are uniformly distributed due to the choice of the group elements in $\boldsymbol{R}'_3$.

Furthermore, SimSFKeyT3 sets the $\mathbb{G}_{p_2}$ component of $\boldsymbol{K}$ to $Y_2^{\boldsymbol{k}(\alpha',\boldsymbol{0},\boldsymbol{0})}$ as shown above in an intermediate step. This completes the proof. □

We notice that the proof of this lemma demonstrates the task of the $\mathbb{G}_{p_3}$ subgroup in the Dual System Encryption Methodology. The $\mathbb{G}_{p_3}$ components in the group elements of the key do not collude with the ciphertext during the decryption anyway, since neither normal ciphertext nor semi-functional ciphertext contain $\mathbb{G}_{p_3}$ components. In our CCA-secure construction this is even ensured by our consistency check in (4.4). Indeed, this subgroup is only used to add additional randomness to the key. In the security proof this is essential in order to generate semi-functional keys using a generator of $\mathbb{G}_{p_2 p_3}$ instead of a generator of $\mathbb{G}_{p_2}$.

Difference Lemma

The following general lemma will be used in almost all reduction steps of the main proof.

Lemma 5.8 (Difference Lemma [Sho04])**.** *Let E_1, E_2 and F be events defined in a probability space, and suppose that $E_1 \wedge \neg F \Leftrightarrow E_2 \wedge \neg F$. Then $\big|\Pr[E_1] - \Pr[E_2]\big| \leq \Pr[F]$.*

5.2.3. Proof of the Main Theorem

In this section we provide the formal proof of our main theorem. As explained in Section 5.1, some of the reductions from the original CPA-secure framework of [Att14a] will require only a few simple modifications. For the sake of completeness, we present the complete proof and highlight new parts of the proof.

Remark 5.9. Formally, we have to show that the statement of our main theorem holds for every $\mathrm{des} \in \Omega$. However, we will not present different reduction algorithms for different description parameters des. Rather, the reduction algorithms in the proof will get des as an additional input.

From G_{Real} to G_{resH}

The first game G_{Real} in the proof sequence of probability experiments (cf. Fig. 5.1) is the CCA-security experiment $\text{aP-KEM}^{\text{CCA2}}_{\Pi,\mathcal{A}}(\lambda, \text{des})$ from Fig. 4.2. The restricted hash game G_{resH} is defined as G_{Real} except for the guess phase, where the output $\langle 10 \rangle$ is modified. Recall that $\text{HInput}(\cdot)$ is defined on page 76 as a part of the encapsulation algorithm. It takes as input an encapsulation and computes the corresponding input for the hash function. The last group element C'' of the encapsulation does not affect the hash input.

Changes in G_{resH} in comparison to G_{Real}:
Exchange $\langle 10 \rangle$ for:

1. The output is 0, if $\mathcal{A}$ queried the decapsulation of $\text{CT} \in \mathbb{C}_{\text{cInd}} \subseteq \mathbb{C}_{\text{pp}_\kappa}$ such that

$$\text{HInput}(\text{CT}) \neq \text{HInput}(\text{CT}^*) \qquad \text{and} \qquad t = t^* \pmod{N} \ ,$$

   where CT^* is the challenge encapsulation, t and t^* are the hash values of CT and CT^* respectively.
2. Otherwise, the output is as defined in $\text{aP-KEM}^{\text{CCA2}}_{\Pi,\mathcal{A}}(\lambda, \text{des})$.

We call by HashAbort the event that a query, as defined in Step 1 above, occurs. The probability for this event is negligible due to the collision-resistance of $\mathcal{H}$ as stated in the following lemma.

Lemma 5.10. *For every* $\text{des} \in \Omega$ *and every ppt algorithm* $\mathcal{A}$ *there exists a ppt algorithm* $\mathcal{B}$ *such that for every* $\lambda \in \mathbb{N}$ *it holds*

$$\left|\text{Adv}^{G_{Real}}_{\Pi,\mathcal{A}}(\lambda, \text{des}) - \text{Adv}^{G_{resH}}_{\Pi,\mathcal{A}}(\lambda, \text{des})\right| \leq \text{Adv}^{\text{CR}}_{\mathcal{H},\mathcal{B}}(\lambda) \ .$$

The running time of $\mathcal{B}$ *is essentially the same as the running time of* $\mathcal{A}$.

Proof. Given a ppt adversary $\mathcal{A}$ that can distinguish between G_{Real} and G_{resH}, we construct a ppt algorithm $\mathcal{B}$ which uses $\mathcal{A}$ and breaks the security property of the collision-resistant hash function family $\mathcal{H}$.

Let $\mathcal{A}$ and $\text{des} \in \Omega$ be arbitrary, but fixed. Both probability experiments are identical as long as the event HashAbort does not occur. Hence, by Lemma 5.8 it holds for every $\lambda \in \mathbb{N}$

$$\begin{aligned}\Pr[\text{HashAbort}] &\geq \left|\Pr\left[G_{\text{Real},\Pi,\mathcal{A}}(\lambda, \text{des}) = 1\right] - \Pr\left[G_{\text{resH},\Pi,\mathcal{A}}(\lambda, \text{des}) = 1\right]\right| \\ &\overset{\text{by def.}}{=} \left|\text{Adv}^{G_{Real}}_{\Pi,\mathcal{A}}(\lambda, \text{des}) - \text{Adv}^{G_{resH}}_{\Pi,\mathcal{A}}(\lambda, \text{des})\right|\end{aligned}$$

But if this event occurs, we get a collision (x_1, x_2) for H:

$$\begin{aligned}x_1 = \text{HInput}(\text{CT}) &\neq \text{HInput}(\text{CT}^*) = x_2 \ , \\ \text{H}(x_1) = t &= t^* = \text{H}(x_2) \ ,\end{aligned}$$

which violates the security property of $\mathcal{H}$. More formally, we can construct a ppt algorithm $\mathcal{B}$ against $\mathcal{H}$ as follows. $\mathcal{B}$ on input $\left(1^\lambda, \text{H}\right)$ (as defined in experiment $\text{CR}_{\mathcal{H},\mathcal{A}}(\lambda)$ on page 5) simulates $\mathcal{A}$ using H and correctly generated public parameters for des. If HashAbort occurs, $\mathcal{B}$ outputs the collision (x_1, x_2) from above for H and wins. We deduce that for every $\text{des} \in \Omega$ and every ppt $\mathcal{A}$ it holds $\text{Adv}^{\text{CR}}_{\mathcal{H},\mathcal{B}}(\lambda) = \Pr[\text{HashAbort}]$. This completes the proof. □

From $\mathrm{G}_{\mathrm{resH}}$ to $\mathrm{G}_{\mathrm{resQ}}$

The game with restricted queries $\mathrm{G}_{\mathrm{resQ}}$ is defined as $\mathrm{G}_{\mathrm{resH}}$ except for the Guess phase, where we again modify the output $\langle 10 \rangle$. We keep the modification from $\mathrm{G}_{\mathrm{resH}}$ and add two additional checks:

Changes in $\mathrm{G}_{\mathrm{resQ}}$ in comparison to $\mathrm{G}_{\mathrm{resH}}$:
Exchange $\langle 10 \rangle$ for:

1. The same as Step 1 in $\mathrm{G}_{\mathrm{resH}}$.
2. The output is 0, if $\mathcal{A}$ queried the covered key generation oracle in Phase I or in Phase II on key index kInd with

$$\mathrm{Factor}\left(\kappa, \mathrm{kInd}, \mathrm{cInd}^*\right) = F \neq \perp \ ,$$

   where Factor is the algorithm from the domain-transferability property of $\mathcal{R}$ (see Definition 1.5).
3. The output is 0, if $\mathcal{A}$ queried the decapsulation oracle on $\mathrm{CT} \in \mathbb{C}_{\mathrm{cInd}} \subseteq \mathrm{pp}_\kappa$ such that for the corresponding hash value t it holds

$$t \neq t^* \pmod{N} \qquad \text{and} \qquad \gcd\left(t - t^*, N\right) \neq 1 \ ,$$

   where t^* is the hash value for the challenge CT^*.
4. Otherwise, the output is as defined in $\text{aP-KEM}^{\mathrm{CCA2}}_{\Pi,\mathcal{A}}\left(\lambda, \mathrm{des}\right)$ (the same as Step 2 in $\mathrm{G}_{\mathrm{resH}}$).

We call by FactorAbort the event that a query, as defined in Step 2 or a query as defined in Step 3 above, occurs whereas the event HashAbort does not occur. The probability for this event is negligible, since in both cases we can compute a non-trivial factor of N, which violates Assumption SD2 by Lemma 4.1 as stated in the following lemma.

Lemma 5.11. *For every* $\mathrm{des} \in \Omega$ *and every ppt algorithm* $\mathcal{A}$ *there exists a ppt algorithm* $\mathcal{B}$ *such that for every security parameter* λ *it holds*

$$\left|\mathrm{Adv}^{\mathrm{G}_{\mathrm{resH}}}_{\Pi,\mathcal{A}}\left(\lambda, \mathrm{des}\right) - \mathrm{Adv}^{\mathrm{G}_{\mathrm{resQ}}}_{\Pi,\mathcal{A}}\left(\lambda, \mathrm{des}\right)\right| \leq \mathrm{Adv}^{\mathrm{SD2}}_{\mathcal{B}}\left(\lambda\right) \ .$$

The running time of $\mathcal{B}$ *is essentially the same as the running time of* $\mathcal{A}$*.*

Proof. Let $\mathrm{des} \in \Omega$ be arbitrary, but fixed. Experiments $\mathrm{G}_{\mathrm{resH}}$ and $\mathrm{G}_{\mathrm{resQ}}$ are identical as long as the event FactorAbort does not occur. Hence, by Lemma 5.8 it holds for every $\lambda \in \mathbb{N}$

$$\left|\mathrm{Adv}^{\mathrm{G}_{\mathrm{resH}}}_{\Pi,\mathcal{A}}\left(\lambda, \mathrm{des}\right) - \mathrm{Adv}^{\mathrm{G}_{\mathrm{resQ}}}_{\Pi,\mathcal{A}}\left(\lambda, \mathrm{des}\right)\right| \leq \Pr\left[\mathrm{FactorAbort}\right] \ .$$

Next we analyze the probability for this event. We construct a ppt algorithm $\mathcal{B}$ which uses $\mathcal{A}$ and breaks Assumption SD2 if the event FactorAbort occurs. $\mathcal{B}$ is given $\mathrm{des} \in \Omega$ in addition to its input (D, Z) from experiment SD2 as explained in Remark 5.9 and is presented in Algorithm 4.

In the first step, $\mathcal{B}$ generates properly distributed public parameters and the corresponding master secret key by Lemma 5.5. Hence, $\mathcal{B}$ can simulate the adversary as defined in the experiment. Note that the challenge Z of $\mathcal{B}$ is not used in the simulation of $\mathcal{A}$.

If the event FactorAbort does not occur, $\mathcal{B}$ outputs a guess, and hence it outputs 1 with probability $\frac{1}{2}$ independently of the value Z. If the event FactorAbort occurs, $\mathcal{B}$ computes a non-trivial factor of N and breaks Experiment SD2 with success probability 1 by Lemma 4.1.

Algorithm 4: $\mathcal{B}$ against Assumption SD2

Input : (D, Z, des).
Require: $D = (\mathbb{GD}_N, g_1, X_1X_2, Y_2Y_3, g_3)$, $Z \in \mathbb{G}$, $\text{des} \in \Omega$.

1 Compute $(\text{msk}, \text{pp}_\kappa, \_, \_, \_) \leftarrow \text{SimPP}(\text{des}, \mathbb{GD}_N, g_1, g_3)$ and simulate $\mathcal{A}$ as defined in the experiment G_{resH} until its output.
2 Perform Step 1 from the Guess Phase. Output a guess $\nu \leftarrow \{0,1\}$ if the result of the experiment is defined to be 0 in this step.
3 (Instead of Step 2) For every kInd_i used as input for the covered key generation oracle in Phase I or in Phase II compute $F_i := \text{Factor}(\kappa, \text{kInd}_i, \text{cInd}^*)$.
4 **if** *there exists* F_i *such that* $F_i \neq \perp$ **then**
5 | Break the own challenge as shown in the proof of Lemma 4.1 using (D, Z, F_i).
6 (Instead of Step 3) For every decapsulation query on $\text{CT} \in \mathbb{C}_{\text{cInd}}$, $\text{cInd} \in \mathbb{Y}_\kappa$ compute the corresponding hash value t and check if $t \neq t^* \pmod N$ and $\gcd(t - t^*, N) \neq 1$, where t^* is the hash value for the challenge CT^*.
7 **if** t *with required property is found* **then**
8 | Compute a factor $F = \gcd(t - t^*, N)$ of N and break the own challenge as shown in the proof of Lemma 4.1, using (D, Z, F).
9 Output a guess $\nu \leftarrow \{0,1\}$.

Formally, for every $\text{des} \in \Omega$ and every ppt algorithm $\mathcal{A}$ there exists a ppt algorithm $\mathcal{B}' = \mathcal{B}_{\mathcal{A}}(\cdot, \cdot, \text{des})$ such that for every security parameter λ it holds

$$\begin{aligned}
\text{Adv}^{\text{SD2}}_{\mathcal{B}'}(\lambda) &= \left|\Pr\left[\mathcal{B}'(D, Z_0) = 1\right] - \Pr\left[\mathcal{B}'(D, Z_1) = 1\right]\right| \\
&= \Pr[\text{FactorAbort}] \cdot |\Pr[\mathcal{B}_{\mathcal{A}}(D, Z_0, \text{des}) = 1 \mid \text{FactorAbort}] \\
&\qquad - \Pr[\mathcal{B}_{\mathcal{A}}(D, Z_1, \text{des}) = 1 \mid \text{FactorAbort}]| \\
&= \Pr[\text{FactorAbort}] \cdot \left|\Pr\left[\mathcal{B}''(D, Z_0, F) = 1\right] - \Pr\left[\mathcal{B}''(D, Z_1, F) = 1\right]\right| \\
&= \Pr[\text{FactorAbort}] \ ,
\end{aligned}$$

where $\mathcal{B}''$ is the algorithm from Lemma 4.1. This proves the lemma. □

Supplementary Lemmata

One can efficiently check if the events HashAbort or FactorAbort occur, as shown in the definition of G_{resH} and in the definition of G_{resQ}. In the following experiments, the output will be 0 if one of these events occurs. Equivalently, we can assume that these events never happen. We obtain the following corollaries.

Lemma 5.12. *Suppose that events* HashAbort *and FactorAbort do not occur. Then, for every* $p_i \mid N$ *and every encapsulation* CT*, used by* $\mathcal{A}$ *as input for the decapsulation oracle, it holds*

$$\text{HInput}(\text{CT}) \neq \text{HInput}(\text{CT}^*) \qquad \textit{implies} \qquad t \neq t^* \pmod{p_i} \ ,$$

where $t = \text{H}(\text{HInput}(\text{CT}))$ *and* $t^* = \text{H}(\text{HInput}(\text{CT}^*))$.

Proof. By the definition of H it holds $t, t^* \in \mathbb{Z}_N$. If the event HashAbort does not occur, it holds $t \neq t^* \pmod N$ for every CT which satisfies $\text{HInput}(\text{CT}) \neq \text{HInput}(\text{CT}^*)$. W.l.o.g. let $0 < t - t^* < N$ and $p_i \mid N$ be arbitrary but fixed. Assume that $t = t^* \pmod{p_i}$. Then we deduce that $\gcd(t - t^*, N) \geq p_i > 1$, which causes the FactorAbort event (Step 3). Hence, it holds $t \neq t^* \pmod{p_i}$ for every CT which satisfies $\text{HInput}(\text{CT}) \neq \text{HInput}(\text{CT}^*)$. □

Lemma 5.13. *Suppose that event FactorAbort does not occur. Then, for every* kInd*, used by* $\mathcal{A}$ *in covered key generation queries, it holds*

$$\mathrm{R}_N\left(\mathrm{kInd}, \mathrm{cInd}^*\right) = 0 \qquad \textit{implies} \qquad \mathrm{R}_{p_2}\left(f_1\left(\mathrm{kInd}\right), f_2\left(\mathrm{cInd}^*\right)\right) = 0 \ ,$$

where f_1 *and* f_2 *are the projection maps from the domain-transferability property of* $\mathcal{R}$*.*

Proof. The implication is guaranteed by Step 2 in the Guess phase from the definition of $\mathrm{G}_{\mathrm{resQ}}$, since otherwise algorithm Factor outputs a non-trivial factor F of N. □

Together with the properties of selective master key hiding and co-selective master key hiding of the underlying pair encoding schemes, Lemma 5.13 is crucial for the analysis of the reduction steps leading from $\mathrm{G}_{k,1}$ to $\mathrm{G}_{k,2}$ and from G_{q_1+1} to G_{q_1+2}. In turn, Lemma 5.12 is crucial for our last additional reduction where we prove that G_{q_1+3} and G'_{q_1+3} are indistinguishable.

From $\mathrm{G}_{\mathrm{resQ}}$ to G'_0

The experiment G'_0 is as $\mathrm{G}_{\mathrm{resQ}}$, but the challenge encapsulation is semi-functional:

Changes in G'_0 in comparison to $\mathrm{G}_{\mathrm{resQ}}$:
Exchange $\langle 1 \rangle$ for:

$$\left(\mathrm{msk}, \mathrm{pp}_\kappa, g_2, \hat{\boldsymbol{h}}, \hat{u}_2, \hat{v}_2\right) \leftarrow \mathrm{SFSetup}\left(1^\lambda, \mathrm{des}\right) \ .$$

Exchange $\langle 5 \rangle$ for:

$$\left(\mathrm{K}_0, \mathrm{CT}^*\right) \leftarrow \mathrm{SFEncaps}\left(\mathrm{cInd}^*, g_2, \hat{\boldsymbol{h}}, \hat{u}_2, \hat{v}_2\right) \ .$$

The following lemma corresponds to Lemma 28 from [Att14b] and requires only a few modifications. We use our supplementary algorithms from Subsection 5.2.2, which simplifies the description of the reduction algorithm and the arguments for the proof. Furthermore, we extend the algorithm by the computation of our additional element C'' in the challenge encapsulation.

Lemma 5.14. *(cf. Lemma 28 in [Att14b]) For every* $\mathrm{des} \in \Omega$ *and every ppt algorithm* $\mathcal{A}$ *there exists a ppt algorithm* $\mathcal{B}$ *such that for every security parameter* λ *it holds*

$$\left|\mathrm{Adv}^{\mathrm{G}_{\mathrm{resQ}}}_{\Pi,\mathcal{A}}\left(\lambda, \mathrm{des}\right) - \mathrm{Adv}^{\mathrm{G}'_0}_{\Pi,\mathcal{A}}\left(\lambda, \mathrm{des}\right)\right| = \mathrm{Adv}^{\mathrm{SD1}}_{\mathcal{B}}\left(\lambda\right) \ .$$

The running time of $\mathcal{B}$ *is essentially the same as the running time of* $\mathcal{A}$*.*

Proof. Given a ppt adversary $\mathcal{A}$ that can distinguish between $\mathrm{G}_{\mathrm{resQ}}$ and G'_0, we construct a ppt algorithm $\mathcal{B}$ which uses $\mathcal{A}$ and breaks Assumption SD1 with the same success probability. Let $\mathrm{des} \in \Omega$ be arbitrary but fixed. $\mathcal{B}$ is given des in addition to its input (D, Z) from experiment SD1 as explained in Remark 5.9 and is presented in Algorithm 5. $\mathcal{B}$ is a ppt algorithm with respect to λ by construction. In particular, all exponents in the description of the algorithm can be computed explicitly.

Next, we analyze the view of $\mathcal{A}$ and the success probability of $\mathcal{B}$. By construction of $\mathcal{B}$ and by Statement 2 of Lemma 5.5 the public parameters $\mathrm{pp}_\kappa = \left(\mathrm{des}, \mathbb{GD}_N, g_1, g_1^{\boldsymbol{h}}, U_1, V_1, g_3, Y, \mathrm{H}\right)$ and

Algorithm 5: $\mathcal{B}$ against Assumption SD1

Input : (D, Z, des).
Require: $D = (\mathbb{GD}_N, g_1, g_3)$, $Z \in \mathbb{G}$, $\mathrm{des} \in \Omega$.

Setup
  Compute $(\mathrm{msk}, \mathrm{pp}_\kappa, \boldsymbol{h'}, u, v) \leftarrow \mathrm{SimPP}(\mathrm{des}, \mathbb{GD}_N, g_1, g_3)$ and simulate $\mathcal{A}\left(1^\lambda, \mathrm{pp}_\kappa\right)$.
Phase I
  As defined in the experiments using $\mathrm{KeyGen}(\mathrm{msk}, \cdot)$ for key generation.
Challenge (given $\mathrm{cInd}^* \in \mathbb{Y}_\kappa$ from $\mathcal{A}$)
  Compute $(\boldsymbol{c}, w_2) := \mathrm{Enc2}(\kappa, \mathrm{cInd}^*)$.
  Pick $s' \leftarrow \mathbb{Z}_N$, $\boldsymbol{s'} \leftarrow \mathbb{Z}_N^{w_2}$ and compute

$$\boldsymbol{C}^* := Z^{\boldsymbol{c}(s', \boldsymbol{s'}, \boldsymbol{h'})} \ .$$

  Compute $t^* := \mathrm{H}\big(\mathrm{HInput}(\mathrm{cInd}^*, \boldsymbol{C}^*, \_)\big)$ and

$$C''^* := Z^{s' \cdot (u \cdot t^* + v)} \ .$$

  Set $\mathrm{CT}^* := (\mathrm{cInd}^*, \boldsymbol{C}^*, C''^*)$ and $\mathrm{K}_0 := \mathrm{e}(Z, g_1)^{\mathrm{msk} \cdot s'}$.
  Pick $\mathrm{K}_1 \leftarrow \mathbb{G}_\mathrm{T}$, flip a coin $b \leftarrow \{0, 1\}$, set $\mathrm{K}^* := \mathrm{K}_b$, and return $(\mathrm{K}^*, \mathrm{CT}^*)$.
Phase II
  As defined in the experiments using $\mathrm{KeyGen}(\mathrm{msk}, \cdot)$ for key generation.
Guess
  Output 1 if and only if $\mathcal{A}$ wins according to the definitions of the experiments.

the master secret msk are distributed as defined in the experiments. Note that $\mathbb{GD}_N$, g_1 and g_3 are distributed as required in Lemma 5.5 due to the definition of experiment SD1. Furthermore, by Statement 1 of Lemma 5.5 it holds $g_1^{\boldsymbol{h'}} = g_1^{\boldsymbol{h}}$, $U_1 = g_1^u$ and $V_1 = g_1^v$. $\mathcal{B}$ implicitly sets the semi-functional elements to $\hat{\boldsymbol{h}} = \boldsymbol{h'} \pmod{p_2}$, $\hat{u}_2 = u \pmod{p_2}$, and $\hat{v}_2 = v \pmod{p_2}$. These elements are properly distributed by Statement 3 of Lemma 5.5.

All secret keys generated in Phase I and in Phase II are normal in both experiments, and hence by construction of $\mathcal{B}$ all user secret keys are properly generated using $\mathrm{KeyGen}(\mathrm{msk}, \cdot)$. Let g_2 be an arbitrary but fixed generator of $\mathbb{G}_{p_2}$. By the definition of probability experiment SD1 it holds $Z = g_1^{z_1} g_2^{z_2}$, where z_1 is uniformly distributed over $\mathbb{Z}_{p_1}^*$, and z_2 is either uniformly distributed over $\mathbb{Z}_{p_2}^*$ (if $Z = Z_1$) or $z_2 = 0 \pmod{p_2}$ (if $Z = Z_0$).

Next, consider the challenge phase and the generated challenge. It is important to notice that by Lemma 5.4, we can consider the distribution of the encapsulation for any fixed generator of $\mathbb{G}_{p_2}$. By construction of $\mathcal{B}$ it holds

$$\begin{aligned} \mathrm{K}_0 &= \mathrm{e}(Z, g_1)^{\mathrm{msk} \cdot s'} \\ &= \mathrm{e}(g_1^{z_1} g_2^{z_2}, g_1)^{\mathrm{msk} \cdot s'} \\ &= Y^{z_1 \cdot s'} \ , \end{aligned}$$

and

$$\begin{aligned} \boldsymbol{C}^* &= Z^{\boldsymbol{c}(s', \boldsymbol{s'}, \boldsymbol{h'})} \\ &= g_1^{z_1 \cdot \boldsymbol{c}(s', \boldsymbol{s'}, \boldsymbol{h'})} \cdot g_2^{z_2 \cdot \boldsymbol{c}(s', \boldsymbol{s'}, \boldsymbol{h'})} \\ &= g_1^{\boldsymbol{c}(z_1 \cdot s', z_1 \cdot \boldsymbol{s'}, \boldsymbol{h})} \cdot g_2^{\boldsymbol{c}\left(z_2 \cdot s', z_2 \cdot \boldsymbol{s'}, \hat{\boldsymbol{h}}\right)} \ , \end{aligned} \qquad \begin{aligned} C''^* &= Z^{s' \cdot (u \cdot t^* + v)} \\ &= (g_1^{z_1} g_2^{z_2})^{s' \cdot (u \cdot t^* + v)} \\ &= \left(U_1^{t^*} V_1\right)^{z_1 \cdot s'} \cdot \left(g_2^{\hat{u}_2 \cdot t^*} g_2^{\hat{v}_2}\right)^{z_2 \cdot s'} \ . \end{aligned}$$

We claim that $\mathrm{CT}^* = (\mathrm{cInd}^*, \boldsymbol{C}^*, C''^*)$ is a properly distributed encapsulation of K_0, which is either normal (if $Z = Z_0$) or semi-functional, as defined on page 80 (if $Z = Z_1$). Namely, $\mathcal{B}$

implicitly sets the random values of the normal ($\mathbb{G}_{p_1}$) components to $s := z_1 \cdot s' \pmod{p_1}$, and to $\boldsymbol{s} := z_1 \cdot \boldsymbol{s'} \pmod{p_1}$. The random values of semi-functional ($\mathbb{G}_{p_2}$) components are set as $\hat{s} := z_2 \cdot s' \pmod{p_2}$, $\hat{\boldsymbol{s}} := z_2 \cdot s' \pmod{p_2}$. These values are properly distributed due to the choice of s' and $\boldsymbol{s'}$. Thereby, we use the fact that $z_1 \neq 0 \pmod{p_1}$ in both cases and $z_2 \neq 0 \pmod{p_2}$ in the case of $Z = Z_1$. Furthermore, the value s' and all values in $\boldsymbol{s'}$ modulo p_1 and modulo p_2 are uncorrelated by the Chinese Remainder Theorem.

We deduce that $\mathcal{B}$ perfectly simulates experiment $\mathrm{G}_{\mathrm{resQ}}$ if $Z = Z_0$ and experiment G'_0 if $Z = Z_1$. Furthermore, the output of $\mathcal{B}$ is 1 if and only if $\mathcal{A}$ wins in the corresponding experiment. Hence, for every $\mathrm{des} \in \Omega$ and every $\mathcal{A}$ there exists a ppt algorithm $\mathcal{B}' = \mathcal{B}_{\mathcal{A}}(\cdot, \cdot, \mathrm{des})$ such that for every security parameter λ it holds

$$\begin{aligned}
\mathrm{Adv}^{\mathrm{SD1}}_{\mathcal{B}'}(\lambda) &= \left|\Pr\left[\mathcal{B}'(D, Z_0) = 1\right] - \Pr\left[\mathcal{B}'(D, Z_1) = 1\right]\right| \\
&= \left|\frac{1}{2} + \mathrm{Adv}^{\mathrm{G}_{\mathrm{resQ}}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) - \left(\frac{1}{2} + \mathrm{Adv}^{\mathrm{G}'_0}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})\right)\right| \\
&= \left|\mathrm{Adv}^{\mathrm{G}_{\mathrm{resQ}}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) - \mathrm{Adv}^{\mathrm{G}'_0}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})\right| \quad .
\end{aligned}$$

This proves the lemma. □

From G'_0 to $\mathrm{G}_{0,3}$

The main modification in $\mathrm{G}_{0,3}$ is that the decapsulation queries are answered using separately generated normal keys, which we denote by sk'_i. Hence, the keys generated in the covered key generation queries and denoted by sk_i, will be used only in the opening oracle. Consequently, we do not have to generate these keys in the covered key generation queries anymore, instead they are generated in the opening oracle, when the keys are given to the adversary. This last change is only conceptual at this point, but is crucial for the following reductions and the resulting security guaranties.

Changes in $\mathrm{G}_{0,3}$ in comparison to G'_0:
CKGen (kInd_i) - Do not generate keys in $^{\langle 2 \rangle}$ and in $^{\langle 7 \rangle}$, just store (i, kInd_i).
Open (i) - Instead of $^{\langle 3 \rangle}$ and $^{\langle 8 \rangle}$, generate $^{\langle 11 \rangle}$ $\boxed{\mathrm{sk}_i \leftarrow \mathrm{KeyGen}(\mathrm{msk}, \mathrm{kInd}_i)}$ if sk_i was not generated yet. Return sk_i.
Decaps (CT, i) instead of $^{\langle 4 \rangle}$ and $^{\langle 9 \rangle}$:

- For the first decapsulation query with index i generate and store a secret key $\mathrm{sk}'_i \leftarrow \mathrm{KeyGen}(\mathrm{msk}, \mathrm{kInd}_i)$.
- Return $\mathrm{Decaps}\left(\mathrm{sk}'_i, \mathrm{CT}\right)$.

The following lemma is new in the original sequence of probability experiments (see Fig. 5.1 on page 83). We show that both experiments are unconditionally indistinguishable because of the consistency checks and especially because of the verifiability property of the underlying pair encoding scheme.

Lemma 5.15. *For every security parameter λ, every $\mathrm{des} \in \Omega$ and every algorithm $\mathcal{A}$ it holds*

$$\mathrm{Adv}^{\mathrm{G}'_0}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) = \mathrm{Adv}^{\mathrm{G}_0}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) \quad .$$

Proof. All generated keys are normal in both experiments by definition. By construction, the view of $\mathcal{A}$ in both experiments can only differ if there is a decapsulation query on CT and $i \in \mathbb{N}$ such that the probability distributions defined by $\text{Decaps}(\text{sk}_i, \text{CT})$ and $\text{Decaps}(\text{sk}'_i, \text{CT})$ are not equal. This cannot happen due to Lemma 4.8, which proves the lemma. □

Remark 5.16. As mentioned above, all reduction steps between $\text{G}_{0,3}$ and G_{q_1+3} are similar to the original CPA-secure construction of [Att14a]. In all reduction steps between these two experiments, the master secret key in known to the reduction algorithm. Hence, all normal keys used to answer decapsulation queries can be generated using $\text{KeyGen}(\text{msk}, \cdot)$. Furthermore, as mentioned before, we have to show that the additional element C''^* for the challenge encapsulation can be generated. For those steps, which are based on the subgroup decision assumptions, this is already covered by Lemma 5.6 and by the algorithm SimSFChlg. The steps based on the security properties of the underlying pair encoding schemes require further explanations (Lemma 5.18 and Lemma 5.21). For the sake of completeness we present all reductions using our supplementary algorithms.

From $\text{G}_{k-1,3}$ **to** $\text{G}_{k,1}$ **for** $k \in [q_1]$

Experiment $\text{G}_{k-1,3}$ is defined as experiment $\text{G}_{0,3}$, but the first $k-1$ keys, corrupted in Phase I, are semi-functional of Type 3. Hence, $\text{G}_{0,3}$ is a special case of $\text{G}_{k-1,3}$ for $k = 1$. The following experiments include an index $j \in \mathbb{N}$, which denotes the current number of corrupted keys in Phase I.

Experiment $\text{G}_{k-1,3}$ for $k \in [q_1]$ as generalization of $\text{G}_{0,3}$:

- Set $j := 0$ in the Setup phase.

Open (i)[a]: Exchange $\langle 11 \rangle$ in Phase I (defined in $\text{G}_{0,3}$ on page 98) for:

- Set $j := j + 1$;
- If $j < k$, choose $\hat{\alpha}_j \leftarrow \mathbb{Z}_N$ and return $\text{sk}_i \leftarrow \text{SFKeyGen}\left(\text{msk}, \text{kInd}_i, 3, \hat{\alpha}_j, g_2, \hat{\boldsymbol{h}}\right)$.
- If $j \geq k$, return $\text{sk}_i \leftarrow \text{KeyGen}(\text{msk}, \text{kInd}_i)$.

[a] W.l.o.g. assume that $\mathcal{A}$ never asks for the same index. Otherwise, just store the keys.

$\text{G}_{k,1}$ is as $\text{G}_{k-1,3}$, but the k's key corrupted in the first phase is semi-functional of Type 1:

Changes in $\text{G}_{k,1}$ in comparison to $\text{G}_{k-1,3}$:
Open (i): modify $\langle 11 \rangle$ in Phase I by

- If $j = k$, return $\text{sk}_i \leftarrow \text{SFKeyGen}\left(\text{msk}, \text{kInd}_i, 1, \_, g_2, \hat{\boldsymbol{h}}\right)$.

Lemma 5.17. *(cf. Lemma 29 in [Att14b]) Suppose $q_1 \in \mathbb{N}$ is the upper bound for the number of corrupted keys in Phase I. Let $k \in [q_1]$ be arbitrary. For every $\text{des} \in \Omega$ and every ppt algorithm $\mathcal{A}$ there exists a ppt algorithm $\mathcal{B}$ such that for every security parameter λ it holds*

$$\left|\text{Adv}^{\text{G}_{k-1,3}}_{\Pi,\mathcal{A}}(\lambda, \text{des}) - \text{Adv}^{\text{G}_{k,1}}_{\Pi,\mathcal{A}}(\lambda, \text{des})\right| = \text{Adv}^{\text{SD2}}_{\mathcal{B}}(\lambda) \quad .$$

The running time of $\mathcal{B}$ is essentially the same as the running time of $\mathcal{A}$.

Algorithm 6: $\mathcal{B}$ against Assumption SD2

Input : (D, Z, des).
Require: $D = (\mathbb{GD}_N, g_1, X_1X_2, Y_2Y_3, g_3)$, $Z \in \mathbb{G}$, $\mathrm{des} \in \Omega$.

Setup
  Compute $(\mathrm{msk}, \mathrm{pp}_\kappa, \boldsymbol{h'}, u, v) \leftarrow \mathrm{SimPP}(\mathbb{GD}_N, g_1, g_3, \mathrm{des})$.
  Set $j := 0$.
Phase I
  CKGen (kInd_i) with $\mathrm{kInd}_i \in \mathbb{X}_\kappa$:
    Store (i, kInd_i).
  Open (i) with $i \in \mathbb{N}$:
    Set $j := j + 1$.
    case $j < k$ **do**
      Pick $\alpha'_j \leftarrow \mathbb{Z}_N$ and return $\mathrm{sk}_i \leftarrow \mathrm{SimSFKeyT3}(\mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{kInd}_i, Y_2Y_3, \alpha_j)$.
    case $j = k$ **do**
      Compute $(\boldsymbol{k}, m_2) := \mathrm{Enc1}(\mathrm{kInd}_i)$. Let $m_1 := |\boldsymbol{k}|$.
      Pick $\boldsymbol{r'}, \hat{\boldsymbol{r}}' \leftarrow \mathbb{Z}_N^{m_2}$ and $\boldsymbol{R}'_3 \leftarrow \mathbb{G}_{p_3}^{m_1}$ and compute

$$\boldsymbol{K} := g_1^{\boldsymbol{k}(\mathrm{msk}, \boldsymbol{r'}, \boldsymbol{h})} \cdot Z^{\boldsymbol{k}(0, \hat{\boldsymbol{r}}', \boldsymbol{h'})} \cdot \boldsymbol{R}'_3 \ .$$

      Return $\mathrm{sk}_i := (\mathrm{kInd}_i, \boldsymbol{K})$.
    case $j > k$ **do**
      Return $\mathrm{sk}_i \leftarrow \mathrm{KeyGen}(\mathrm{msk}, \mathrm{kInd}_i)$.
  Decaps (CT, i) with $\mathrm{CT} \in \mathbb{C}_{\mathrm{cInd}}$, $\mathrm{cInd} \in \mathbb{Y}_\kappa$, $i \in \mathbb{N}$:
    As defined in the experiment using normal secret key
    $\mathrm{sk}'_i \leftarrow \mathrm{KeyGen}(\mathrm{msk}, \mathrm{kInd}_i)$, generated once.
Challenge (given $\mathrm{cInd}^* \in \mathbb{Y}_\kappa$ from $\mathcal{A}$)
  Generate $(\mathrm{K}_0, \mathrm{CT}^*) \leftarrow \mathrm{SimSFChlg}(\mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{cInd}^*, \boldsymbol{h'}, u, v, X_1X_2)$
  Pick $\mathrm{K}_1 \leftarrow \mathbb{G}_\mathrm{T}$, flip a coin $b \leftarrow \{0, 1\}$, set $\mathrm{K}^* := \mathrm{K}_b$, and return the challenge $(\mathrm{K}^*, \mathrm{CT}^*)$.
Phase II
  Simulates this phase as defined in the experiment using msk.
Guess
  Output 1 if and only if $\mathcal{A}$ wins according to the definitions of the experiments.

Proof. Let $k \in [q_1]$ be arbitrary, but fixed. Given a ppt adversary $\mathcal{A}$ that can distinguish between $\mathrm{G}_{k-1,3}$ and $\mathrm{G}_{k,1}$, we construct a ppt algorithm $\mathcal{B}$ which uses $\mathcal{A}$ and breaks Assumption SD2 with the same success probability. Let $\mathrm{des} \in \Omega$ be arbitrary. $\mathcal{B}$ is given des in addition to its input (D, Z) from experiment SD2 as explained in Remark 5.9 and is presented in Algorithm 6. $\mathcal{B}$ is a ppt algorithm by construction. In particular, random elements from $\mathbb{G}_{p_3}$ can be chosen using g_3, the elements from $g_1^{\boldsymbol{k}(\mathrm{msk}, \boldsymbol{r'}, \boldsymbol{h})}$ can be computed as shown in Lemma 4.5, and $\boldsymbol{k}\left(0, \hat{\boldsymbol{r}}', \boldsymbol{h'}\right)$ can be computed explicitly.

Next, we analyze the view of $\mathcal{A}$ and the success probability of $\mathcal{B}$. By construction of $\mathcal{B}$ and by Statement 2 of Lemma 5.5 the public parameters $\mathrm{pp}_\kappa = \left(\mathrm{des}, \mathbb{GD}_N, g_1, g_1^{\boldsymbol{h}}, U_1, V_1, g_3, Y, \mathrm{H}\right)$ and the master secret msk are distributed as defined in the experiments. Note that $\mathbb{GD}_N$, g_1 and g_3 are distributed as required in Lemma 5.5 due to the definition of experiment SD2. Furthermore, by Statement 1 of Lemma 5.5 it holds $g_1^{\boldsymbol{h'}} = g_1^{\boldsymbol{h}}$, $U_1 = g_1^u$ and $V_1 = g_1^v$. $\mathcal{B}$ implicitly sets the semi-functional elements as $\hat{\boldsymbol{h}} = \boldsymbol{h'} \pmod{p_2}$, $\hat{u}_2 = u \pmod{p_2}$, and $\hat{v}_2 = v \pmod{p_2}$. These elements are properly distributed by Statement 3 of Lemma 5.5.

Let g_2 be an arbitrary but fixed generator of $\mathbb{G}_{p_2}$. Then, by the definition of probability

experiment SD2 it holds $Z = g_1^{z_1} g_2^{z_2} g_3^{z_3}$, where $z_1 \in \mathbb{Z}_{p_1}^*$, $z_3 \in \mathbb{Z}_{p_3}^*$ are uniformly distributed, and z_2 is either uniformly distributed in $\mathbb{Z}_{p_2}^*$ (if $Z = Z_1$) or $z_2 = 0 \pmod{p_2}$ (if $Z = Z_0$). Furthermore, $X_1 = g_1^{x_1}$, $X_2 = g_2^{x_2}$, $Y_2 = g_2^{y_2}$ and $Y_3 = g_3^{y_3}$, where $x_1 \in \mathbb{Z}_{p_1}^*$, $x_2, y_2 \in \mathbb{Z}_{p_2}^*$ and $y_3 \in \mathbb{Z}_{p_3}^*$ are uniformly distributed and independent.

The semi-functional challenge and all semi-functional keys of Type 3 are generated as required in the experiment by Lemma 5.6, and by Lemma 5.7 respectively. The normal keys are correctly generated using $\mathrm{KeyGen}(\mathrm{msk}, \cdot)$.

Consider the corrupted key $\mathrm{sk}_i = (\mathrm{kInd}_i, \boldsymbol{K})$ generated for $j = k$. By construction of the algorithm $\mathcal{B}$ it holds:

$$\begin{aligned}
\boldsymbol{K} &= g_1^{\boldsymbol{k}(\mathrm{msk}, \boldsymbol{r}', \boldsymbol{h})} \cdot Z^{\boldsymbol{k}\left(0, \hat{\boldsymbol{r}}', \boldsymbol{h}'\right)} \cdot \boldsymbol{R}_3' \\
&= g_1^{\boldsymbol{k}(\mathrm{msk}, \boldsymbol{r}', \boldsymbol{h})} \cdot \left(g_1^{z_1} g_2^{z_2} g_3^{z_3}\right)^{\boldsymbol{k}\left(0, \hat{\boldsymbol{r}}', \boldsymbol{h}'\right)} \cdot \boldsymbol{R}_3' \\
&= g_1^{\boldsymbol{k}(\mathrm{msk}, \boldsymbol{r}', \boldsymbol{h}) + z_1 \cdot \boldsymbol{k}\left(0, \hat{\boldsymbol{r}}', \boldsymbol{h}'\right)} \cdot g_2^{z_2 \cdot \boldsymbol{k}\left(0, \hat{\boldsymbol{r}}', \boldsymbol{h}'\right)} \cdot g_3^{z_3 \cdot \boldsymbol{k}\left(0, \hat{\boldsymbol{r}}', \boldsymbol{h}'\right)} \cdot \boldsymbol{R}_3' \\
&= g_1^{\boldsymbol{k}\left(\mathrm{msk}, \boldsymbol{r}' + z_1 \cdot \hat{\boldsymbol{r}}', \boldsymbol{h}\right)} \cdot g_2^{\boldsymbol{k}\left(0, z_2 \cdot \hat{\boldsymbol{r}}', \hat{\boldsymbol{h}}\right)} \cdot g_3^{z_3 \cdot \boldsymbol{k}\left(0, \hat{\boldsymbol{r}}', \boldsymbol{h}'\right)} \cdot \boldsymbol{R}_3' \ .
\end{aligned}$$

We claim that sk_i is either a properly distributed normal secret key (if $Z = Z_0$) or a properly distributed semi-functional secret key of Type 1 (if $Z = Z_1$). Namely, $\mathcal{B}$ implicitly sets the random values of the normal ($\mathbb{G}_{p_1}$) components as $\boldsymbol{r} = z_1 \cdot \hat{\boldsymbol{r}}' + \boldsymbol{r}' \pmod{p_1}$, which are properly distributed due to the choice of $\boldsymbol{r}'$. The random values of the semi-functional ($\mathbb{G}_{p_2}$) components are set as $\hat{\boldsymbol{r}} = z_2 \cdot \hat{\boldsymbol{r}}'$, which are properly distributed (for Type 1 keys) due to the choice of $\hat{\boldsymbol{r}}'$ if $Z = Z_1$, and which disappear if $Z = Z_0$, since $z_2 = 0 \pmod{p_2}$. Finally, the $\mathbb{G}_{p_3}$ components are set as $\boldsymbol{R}_3 = g_3^{z_3 \cdot \boldsymbol{k}\left(0, \hat{\boldsymbol{r}}', \boldsymbol{h}\right)} \cdot \boldsymbol{R}_3'$ and are properly distributed by the choice of $\boldsymbol{R}_3'$.

Hence, for every $\mathrm{des} \in \Omega$ and every $\mathcal{A}$ there exists a ppt algorithm $\mathcal{B}' = \mathcal{B}_{\mathcal{A}}(\cdot, \cdot, \mathrm{des})$ such that for every security parameter λ it holds

$$\begin{aligned}
\mathrm{Adv}_{\mathcal{B}'}^{\mathrm{SD2}}(\lambda) &= \left|\Pr\left[\mathcal{B}'(D, Z_0) = 1\right] - \Pr\left[\mathcal{B}'(D, Z_1) = 1\right]\right| \\
&= \left|\frac{1}{2} + \mathrm{Adv}_{\Pi, \mathcal{A}}^{\mathrm{G}_{k-1,3}}(\lambda, \mathrm{des}) - \left(\frac{1}{2} + \mathrm{Adv}_{\Pi, \mathcal{A}}^{\mathrm{G}_{k,1}}(\lambda, \mathrm{des})\right)\right| \\
&= \left|\mathrm{Adv}_{\Pi, \mathcal{A}}^{\mathrm{G}_{k-1,3}}(\lambda, \mathrm{des}) - \mathrm{Adv}_{\Pi, \mathcal{A}}^{\mathrm{G}_{k,1}}(\lambda, \mathrm{des})\right| \ .
\end{aligned}$$

The second equation holds since $\mathcal{B}$ perfectly simulates $\mathrm{G}_{k-1,3}$ and $\mathrm{G}_{k,1}$ if $Z = Z_0$ and $Z = Z_1$ respectively. Furthermore, $\mathcal{B}$ outputs 1 if and only if $\mathcal{A}$ wins the corresponding experiment. This proves the lemma. □

From $\mathrm{G}_{k,1}$ to $\mathrm{G}_{k,2}$ for $k \in [q_1]$

$\mathrm{G}_{k,2}$ is as $\mathrm{G}_{k,1}$, but the k's key is semi-functional of Type 2:

Changes in $\mathrm{G}_{k,2}$ in comparison to $\mathrm{G}_{k,1}$:
Open (i): modify $^{\langle 11 \rangle}$ in Phase I (defined in $\mathrm{G}_{0,3}$) for the key with $j = k$

- If $j = k$, choose $\hat{\alpha}_k \leftarrow \mathbb{Z}_N$ and return $\mathrm{sk}_i \leftarrow \mathrm{SFKeyGen}\left(\mathrm{msk}, \mathrm{kInd}_i, 2, \hat{\alpha}_k, g_2, \hat{\boldsymbol{h}}\right)$.

Lemma 5.18. *(cf. Lemma 30 in [Att14b]) Suppose $q_1 \in \mathbb{N}$ is the upper bound for the number of corrupted keys in Phase I. Let $k \in [q_1]$ be arbitrary. For every ppt algorithm $\mathcal{A}$ there exists a ppt algorithm $\mathcal{B}$ such that for every security parameter λ and every $\mathrm{des} \in \Omega$ it holds*

$$\left|\mathrm{Adv}_{\Pi, \mathcal{A}}^{\mathrm{G}_{k,1}}(\lambda, \mathrm{des}) - \mathrm{Adv}_{\Pi, \mathcal{A}}^{\mathrm{G}_{k,2}}(\lambda, \mathrm{des})\right| = \mathrm{Adv}_{\mathrm{P}, \mathcal{B}}^{\mathrm{CMH}}(\lambda, \mathrm{des}) \ .$$

The running time of $\mathcal{B}$ is essentially the same as the running time of $\mathcal{A}$.

Proof. Let $k \in [q_1]$ be arbitrary, but fixed. Given a ppt adversary $\mathcal{A}$ that can distinguish between $\mathrm{G}_{k,1}$ and $\mathrm{G}_{k,2}$, we construct a ppt algorithm $\mathcal{B}$ which uses $\mathcal{A}$ and breaks the co-selective master-key hiding security property of the underlining pair encoding scheme with the same advantage. $\mathcal{B}$ on input $(\mathrm{des}, \mathbb{GD}_N, g_1, g_2, g_3)$, as defined in $\mathrm{Exp}^{\mathrm{CMH}}_{\mathrm{P},\mathcal{G},\nu,\mathcal{A}}(\lambda, \mathrm{des})$ from Fig. 1.3, is as presented in Algorithm 7. $\mathcal{B}$ is a ppt algorithm with respect to λ by construction. It uses different supplementary ppt algorithms, the own oracles, and performs besides only simple computation. Next we analyze the view of $\mathcal{A}$ and the success probability of $\mathcal{B}$.

Let $\mathrm{des} \in \Omega$ be arbitrary, but fixed. By the definition of the Experiment $\mathrm{Exp}^{\mathrm{CMH}}_{\mathrm{P},\mathcal{G},\nu,\mathcal{A}}(\lambda, \mathrm{des})$, $\mathbb{GD}_N$ is the restricted group description of $\mathbb{GD}$ generated by $\mathcal{G}\left(1^\lambda\right)$. Furthermore, the generators $g_i \in \mathbb{G}_{p_i}$ are chosen uniformly at random. Hence, by construction of $\mathcal{B}$ and by Statement 2 of Lemma 5.5 the public parameters $\mathrm{pp}_\kappa = \left(\mathrm{des}, \mathbb{GD}_N, g_1, g_1^{\boldsymbol{h}}, U_1, V_1, g_3, Y, \mathrm{H}\right)$ and the master secret msk are distributed as defined in the experiments. Furthermore, by Statement 1 of Lemma 5.5 it holds $U_1 = g_1^u$ and $V_1 = g_1^v$. $\mathcal{B}$ implicitly sets the semi-functional elements as $\hat{u}_2 = u \pmod{p_2}$, and $\hat{v}_2 = v \pmod{p_2}$. These elements are properly distributed by Statement 3 of Lemma 5.5. Furthermore, $\mathcal{B}$ implicitly sets the input generator g_2 as the generator of $\mathbb{G}_{p_2}$. This generator is properly distributed as mentioned above. Vector $\hat{\boldsymbol{h}} \pmod{p_2}$ of the semi-functional public parameters will be defined below.

It is important to notice that all oracle queries made by $\mathcal{B}$ are permissible if all corruption queries of $\mathcal{A}$ are permissible, since $\mathrm{R}_N(\mathrm{kInd}, \mathrm{cInd}^*) = 0$ implies $\mathrm{R}_{p_2}(f_1(\mathrm{kInd}), f_2(\mathrm{cInd}^*)) = 0$ by Lemma 5.13. The normal keys and the semi-functional keys of Type 3 are generated using msk and g_2 as defined in the experiments.

Next, we claim that the challenge encapsulation is a properly distributed semi-functional encapsulation. Furthermore, the k's corrupted key is either a properly distributed semi-functional key of Type 1 (if $\nu = 0$) or a properly distributed semi-functional key of Type 2 (if $\nu = 1$). Namely, by the definition of the Experiment $\mathrm{Exp}^{\mathrm{CMH}}_{\mathrm{P},\mathcal{G},\nu,\mathcal{A}}(\lambda, \mathrm{des})$, the challenger choose $\tilde{\alpha} \leftarrow \mathbb{Z}_N$ and $\tilde{\boldsymbol{h}} \leftarrow \mathbb{Z}_N^n$, where $n = \mathrm{Param}(\kappa)$. Then, $\mathcal{B}$ receives

$$\widehat{\boldsymbol{K}} = \mathcal{O}^1_{\mathrm{CMH},\nu,\hat{\alpha},\hat{\boldsymbol{h}}}(\mathrm{kInd}_i) = \begin{cases} g_2^{\boldsymbol{k}(0,\tilde{\boldsymbol{r}},\tilde{\boldsymbol{h}})} & \text{if } \nu = 0 \\ g_2^{\boldsymbol{k}(\tilde{\alpha},\tilde{\boldsymbol{r}},\tilde{\boldsymbol{h}})} & \text{if } \nu = 1 \end{cases},$$

where $\tilde{\boldsymbol{r}} \in \mathbb{Z}_{p_2}^{m_2}$ is chosen uniformly at random, $(\boldsymbol{k}, m_2) = \mathrm{Enc1}(\kappa, \mathrm{kInd}_i)$. Hence, $\widehat{\boldsymbol{K}}$ is a properly distributed semi-functional part of either Type 1 key (if $\nu = 0$) or Type 2 key (if $\nu = 1$) with random values $\hat{\boldsymbol{h}} = \tilde{\boldsymbol{h}} \pmod{p_2}$, $\hat{\boldsymbol{r}} = \tilde{\boldsymbol{r}} \pmod{p_2}$, and $\hat{\alpha} = \tilde{\alpha} \pmod{p_2}$. Furthermore, $\mathcal{B}$ receives

$$\widehat{\boldsymbol{C}} = \mathcal{O}^2_{\mathrm{CMH},\nu,\hat{\alpha},\hat{\boldsymbol{h}}}(\mathrm{cInd}^*) = g_2^{\boldsymbol{c}\left(\tilde{s},\tilde{\boldsymbol{s}},\hat{\boldsymbol{h}}\right)},$$

where $\tilde{s} \in \mathbb{Z}_N$ and $\tilde{\boldsymbol{s}} \in \mathbb{Z}_N^{w_2}$ are chosen uniformly at random, $(\boldsymbol{c}, w_2) = \mathrm{Enc2}(\kappa, \mathrm{cInd}^*)$. Hence, $\widehat{\boldsymbol{C}}$ is a properly distributed semi-functional part of an encapsulation with random values $\hat{s} = \tilde{s} \pmod{p_2}$, $\hat{\boldsymbol{s}} = \tilde{\boldsymbol{s}} \pmod{p_2}$, and $\hat{\boldsymbol{h}} = \tilde{\boldsymbol{h}} \pmod{p_2}$.

Finally, we show that the last group element in the challenge encapsulation is correctly generated:

$$\begin{aligned} C''^* &= (C_1)^{u \cdot t^* + v} \cdot \left(\widehat{C}_1\right)^{u \cdot t^* + v} \\ &= (g_1^s)^{u \cdot t^* + v} \cdot \left(g_2^{\hat{s}}\right)^{u \cdot t^* + v} \\ &= \left(U_1^{t^*} \cdot V_1\right)^s \cdot \left(g_2^{\hat{u}_2 \cdot t^* + \hat{v}_2}\right)^{\hat{s}}, \end{aligned}$$

Algorithm 7: $\mathcal{B}$ against co-selective master-key hiding security property of P

Input : $(\mathrm{des}, \mathbb{GD}_N, g_1, g_2, g_3)$.

Setup

Compute the public parameters and the master secret key $(\mathrm{msk}, \mathrm{pp}_\kappa, \_, u, v) \leftarrow \mathrm{SimPP}\left(1^\lambda, \mathbb{GD}_N, g_1, g_3, \mathrm{des}\right)$. Set $j := 0$.

Phase I

CKGen (kInd_i) with $\mathrm{kInd}_i \in \mathbb{X}_\kappa$:

Store (i, kInd_i).

Open (i)**:**

Set $j := j + 1$.

case $j < k$ **do**

Pick $\alpha_j \leftarrow \mathbb{Z}_N$ and return $\mathrm{sk}_i \leftarrow \mathrm{SFKeyGen}\left(\mathrm{msk}, \mathrm{kInd}_i, 3, \alpha_j, g_2, \_\right)$.

case $j = k$ **do**

Generate a normal key $(\mathrm{kInd}_i, \boldsymbol{K}) \leftarrow \mathrm{KeyGen}\left(\mathrm{msk}, \mathrm{kInd}_i\right)$.

Query the own oracle:

$$\widehat{\boldsymbol{K}} := \mathcal{O}^1_{\mathrm{CMH},\nu,\hat{\alpha},\hat{\boldsymbol{h}}}\left(\mathrm{kInd}_i\right) \quad .$$

Output $\mathrm{sk}_i = \left(\mathrm{kInd}_i, \boldsymbol{K} \cdot \widehat{\boldsymbol{K}}\right)$.

case $j > k$ **do**

Output $\mathrm{sk}_i \leftarrow \mathrm{KeyGen}\left(\mathrm{msk}, \mathrm{kInd}_i\right)$.

Decaps (CT, i)**:**

As defined in the experiment using $\mathrm{sk}'_i \leftarrow \mathrm{KeyGen}\left(\mathrm{msk}, \mathrm{kInd}_i\right)$, generated once.

Challenge (given cInd^* from $\mathcal{A}$)

Compute $(\mathrm{K}_0, (\mathrm{cInd}^*, \boldsymbol{C}, \_)) \leftarrow \mathrm{Encaps}\left(\mathrm{cInd}^*\right)$.

Query the own oracle:

$$\widehat{\boldsymbol{C}} := \mathcal{O}^2_{\mathrm{CMH},\nu,\hat{\alpha},\hat{\boldsymbol{h}}}\left(\mathrm{cInd}^*\right) \quad ,$$

set $\boldsymbol{C}^* := \boldsymbol{C} \cdot \widehat{\boldsymbol{C}}$.

Compute $t^* := \mathrm{H}\left(\mathrm{HInput}\left(\mathrm{cInd}^*, \boldsymbol{C}^*, \_\right)\right)$ and

$$C''^* := \left(C^*_1\right)^{u \cdot t^* + v} \quad .$$

Choose $\mathrm{K}_1 \leftarrow \mathbb{G}_\mathrm{T}$, pick $b \leftarrow \{0, 1\}$, set $\mathrm{K}^* := \mathrm{K}_b$ and return $\left(\mathrm{K}^*, \mathrm{CT}^*_{\mathrm{cInd}^*} = (\mathrm{cInd}^*, \boldsymbol{C}^*, C''^*)\right)$.

Phase II

Simulates this phase as defined in the experiment using msk.

Guess

Output 1 if and only if $\mathcal{A}$ wins according to the definitions of the experiments.

where s is the random element fixed by C_1, which is chosen as the first element of $\boldsymbol{C}$ in Line 19 and $\hat{s}$ is fixed by $\widehat{C}_1$ as defined above. In the second equation we used the normality of P. Hence, C''^* is exactly as defined in SFEncaps.

We deduce that for every $\mathcal{A}$, every security parameter λ and every $\mathrm{des} \in \Omega$ it holds

$$\begin{aligned}\mathrm{Adv}^{\mathrm{CMH}}_{\mathrm{P},\mathcal{B}}(\lambda,\mathrm{des}) &= \left|\mathrm{Exp}^{\mathrm{CMH}}_{\mathrm{P},\mathcal{G},0,\mathcal{B}}(\lambda,\mathrm{des}) - \mathrm{Exp}^{\mathrm{CMH}}_{\mathrm{P},\mathcal{G},1,\mathcal{B}}(\lambda,\mathrm{des})\right| \\ &= \left|\frac{1}{2} + \mathrm{Adv}^{\mathrm{G}_{k,1}}_{\Pi,\mathcal{A}}(\lambda,\mathrm{des}) - \left(\frac{1}{2} + \mathrm{Adv}^{\mathrm{G}_{k,2}}_{\Pi,\mathcal{A}}(\lambda,\mathrm{des})\right)\right| \\ &= \left|\mathrm{Adv}^{\mathrm{G}_{k,1}}_{\Pi,\mathcal{A}}(\lambda,\mathrm{des}) - \mathrm{Adv}^{\mathrm{G}_{k,2}}_{\Pi,\mathcal{A}}(\lambda,\mathrm{des})\right| \quad .\end{aligned}$$

The second equation holds since $\mathcal{B}$ correctly simulates $\mathrm{G}_{k,1}$ and $\mathrm{G}_{k,2}$ if $\nu = 0$ and if $\nu = 1$ respectively. Furthermore, $\mathcal{B}$ outputs 1 if and only if $\mathcal{A}$ wins the corresponding game. This proves the lemma. □

From $\mathrm{G}_{k,2}$ to $\mathrm{G}_{k,3}$ for $k \in [q_1]$

$\mathrm{G}_{k,3}$ is as $\mathrm{G}_{k,2}$, but the k's corrupted key is semi-functional of Type 3.

Changes in $\mathrm{G}_{k,3}$ in comparison to $\mathrm{G}_{k,2}$:
Open (i): modify $^{\langle 11\rangle}$ in Phase I (defined in $\mathrm{G}_{0,3}$) for the key with $j = k$

- If $j = k$, choose $\hat{\alpha}_k \leftarrow \mathbb{Z}_N$ and return $\mathrm{sk}_k \leftarrow \mathrm{SFKeyGen}(\mathrm{msk}, \mathrm{kInd}_i, 3, \hat{\alpha}_k, g_2, \_)$.

Lemma 5.19. *(cf. Lemma 31 in [Att14b]) Suppose $q_1 \in \mathbb{N}$ is the upper bound for the number of corrupted keys in Phase I. Let $k \in [q_1]$ be arbitrary. For every $\mathrm{des} \in \Omega$ and every ppt algorithm $\mathcal{A}$ there exists a ppt algorithm $\mathcal{B}$ such that for every security parameter λ it holds*

$$\left|\mathrm{Adv}^{\mathrm{G}_{k,2}}_{\Pi,\mathcal{A}}(\lambda,\mathrm{des}) - \mathrm{Adv}^{\mathrm{G}_{k,3}}_{\Pi,\mathcal{A}}(\lambda,\mathrm{des})\right| = \mathrm{Adv}^{\mathrm{SD2}}_{\mathcal{A}}(\lambda) \quad .$$

The running time of $\mathcal{B}$ is essentially the same as the running time of $\mathcal{A}$.

Proof. Given a ppt adversary $\mathcal{A}$ that can distinguish between $\mathrm{G}_{k,2}$ and $\mathrm{G}_{k,3}$, we construct a ppt algorithm $\mathcal{B}$ which uses $\mathcal{A}$ and breaks Assumption SD2 with the same advantage. Let $\mathrm{des} \in \Omega$ be arbitrary. $\mathcal{B}$ is given des in addition to its input (D, Z) from experiment SD2 as explained in Remark 5.9.

$\mathcal{B}$ (presented in Algprorithm 8) is almost the same as Algorithm 6. We only change the simulation of corrupted key number k. $\mathcal{B}$ is a ppt algorithm by construction. Next we analyze the view of $\mathcal{A}$ and the success probability of algorithm $\mathcal{B}$.

Let all elements be as defined in the proof of Lemma 5.17. In particular, $Z = g_1^{z_1} g_2^{z_2} g_3^{z_3}$, $Y_2 = g_2^{y_2}$, $Y_3 = g_3^{y_3}$, $\boldsymbol{h} = \boldsymbol{h'} \pmod{p_1}$ and $\hat{\boldsymbol{h}} = \boldsymbol{h'} \pmod{p_2}$. Here, we only analyze the distribution of the secret key $\mathrm{sk}_i = (\mathrm{kInd}_i, \boldsymbol{K})$ generated for $j = k$. By construction of $\mathcal{B}$ it holds

$$\begin{aligned}\boldsymbol{K} &= g_1^{\boldsymbol{k}(\mathrm{msk},\boldsymbol{r'},\boldsymbol{h})} \cdot (Y_2 Y_3)^{\boldsymbol{k}(\hat{\alpha}',\boldsymbol{0},\boldsymbol{0})} \cdot Z^{\boldsymbol{k}\left(0,\hat{\boldsymbol{r}}',\boldsymbol{h'}\right)} \cdot \boldsymbol{R}'_3 \\ &= g_1^{\boldsymbol{k}(\mathrm{msk},\boldsymbol{r'},\boldsymbol{h})} \cdot (g_2^{y_2} g_3^{y_3})^{\boldsymbol{k}(\hat{\alpha}',\boldsymbol{0},\boldsymbol{0})} \cdot (g_1^{z_1} g_2^{z_2} g_3^{z_3})^{\boldsymbol{k}\left(0,\hat{\boldsymbol{r}}',\boldsymbol{h'}\right)} \cdot \boldsymbol{R}'_3 \\ &= g_1^{\boldsymbol{k}(\mathrm{msk},\boldsymbol{r'},\boldsymbol{h}) + z_1 \cdot \boldsymbol{k}\left(0,\hat{\boldsymbol{r}}',\boldsymbol{h}\right)} \cdot g_2^{y_2 \cdot \boldsymbol{k}(\hat{\alpha}',\boldsymbol{0},\boldsymbol{0}) + z_2 \cdot \boldsymbol{k}\left(0,\hat{\boldsymbol{r}}',\hat{\boldsymbol{h}}\right)} \cdot g_3^{y_3 \cdot \boldsymbol{k}(\hat{\alpha}',\boldsymbol{0},\boldsymbol{0}) + z_3 \cdot \boldsymbol{k}\left(0,\hat{\boldsymbol{r}}',\boldsymbol{h'}\right)} \cdot \boldsymbol{R}'_3 \\ &= g_1^{\boldsymbol{k}\left(\mathrm{msk},\boldsymbol{r'} + z_1 \cdot \hat{\boldsymbol{r}}',\boldsymbol{h}\right)} \cdot g_2^{\boldsymbol{k}\left(y_2 \cdot \hat{\alpha}', z_2 \cdot \hat{\boldsymbol{r}}', \hat{\boldsymbol{h}}\right)} \cdot g_3^{\boldsymbol{k}\left(y_3 \cdot \hat{\alpha}', z_3 \cdot \hat{\boldsymbol{r}}', \boldsymbol{h}\right)} \cdot \boldsymbol{R}'_3 \quad .\end{aligned}$$

Algorithm 8: $\mathcal{B}$ against Assumption SD2 as modification of Algorithm 6

Input : (D, Z, des).
Require: $D = (\mathbb{GD}_N, g_1, X_1X_2, Y_2Y_3, g_3)$, $Z \in \mathbb{G}$, $\mathrm{des} \in \Omega$.

...
Phase I
  ...
  Open (i) with $i \in \mathbb{N}$:
    ...
    case $j = k$ **do**
      Compute $(\boldsymbol{k}, m_2) := \mathrm{Enc1}(\mathrm{kInd}_k)$. Let $m_1 := |\boldsymbol{k}|$.
      Pick $\boldsymbol{r}', \hat{\boldsymbol{r}}', \hat{\alpha}' \leftarrow \mathbb{Z}_N^{m_2}$ and $\boldsymbol{R}_3' \leftarrow \mathbb{G}_{p_3}^{m_1}$. Compute

$$\boldsymbol{K} := g_1^{\boldsymbol{k}(\mathrm{msk}, \boldsymbol{r}', \boldsymbol{h})} \cdot (Y_2Y_3)^{\boldsymbol{k}(\hat{\alpha}', \boldsymbol{0}, \boldsymbol{0})} \cdot Z^{\boldsymbol{k}\left(0, \hat{\boldsymbol{r}}', \boldsymbol{h}'\right)} \cdot \boldsymbol{R}_3' \ .$$

      Return $\mathrm{sk}_i = (\mathrm{kInd}_i, \boldsymbol{K})$.
...

We claim that this key is either a properly distributed semi-functional key of Type 2 (if $Z = Z_1$) or a properly distributed semi-functional key of Type 3 (if $Z = Z_0$). Namely, $\mathcal{B}$ implicitly sets the random values of the $\mathbb{G}_{p_1}$ components as $\boldsymbol{r} = z_1 \cdot \hat{\boldsymbol{r}}' + \boldsymbol{r}' \pmod{p_1}$, which are properly distributed due to the choice of $\boldsymbol{r}'$. The random values of $\mathbb{G}_{p_2}$ components are set as $\hat{\alpha} = y_2 \cdot \hat{\alpha}' \pmod{p_2}$ and $\hat{\boldsymbol{r}} = z_2 \cdot \hat{\boldsymbol{r}}' \pmod{p_2}$. If $Z = Z_0$, it holds $z_2 = 0 \pmod{p_2}$ and thus

$$\boldsymbol{k}\left(y_2 \cdot \hat{\alpha}', z_2 \cdot \hat{\boldsymbol{r}}', \hat{\boldsymbol{h}}\right) = \boldsymbol{k}\left(y_2 \cdot \hat{\alpha}', \boldsymbol{0}, \hat{\boldsymbol{h}}\right) = \boldsymbol{k}\left(y_2 \cdot \hat{\alpha}', \boldsymbol{0}, \boldsymbol{0}\right) \pmod{p_2} \ .$$

Hence, if $Z = Z_0$ the $\mathbb{G}_{p_2}$ components are properly distributed as defined for Type 2 keys due to the choice of $\hat{\alpha}'$ (since $y_2 \neq 0 \pmod{p_2}$). If $Z = Z_1$, then $\hat{\alpha}$ and $\hat{\boldsymbol{r}}$ are properly distributed, as defined for Type 3 keys, due to the choice of $\hat{\alpha}' \pmod{p_2}$ and $\hat{\boldsymbol{r}}' \pmod{p_2}$ (since $y_2, z_2 \neq 0 \pmod{p_2}$), respectively. Finally, the $\mathbb{G}_{p_3}$ components are set as $\boldsymbol{R}_3 = g_3^{z_3 \cdot \boldsymbol{k}\left(0, \hat{\boldsymbol{r}}', \boldsymbol{h}\right)} \cdot \boldsymbol{R}_3'$ and are properly distributed by the choice of $\boldsymbol{R}_3'$.

We deduce that for every $\mathrm{des} \in \Omega$ and every $\mathcal{A}$ there exists a ppt algorithm $\mathcal{B}' = \mathcal{B}_{\mathcal{A}}(\cdot, \cdot, \mathrm{des})$ such that for every security parameter λ it holds

$$\begin{aligned}
\mathrm{Adv}_{\mathcal{B}'}^{\mathrm{SD2}}(\lambda) &= \left|\Pr\left[\mathcal{B}'(D, Z_0) = 1\right] - \Pr\left[\mathcal{B}'(D, Z_1) = 1\right]\right| \\
&= \left|\frac{1}{2} + \mathrm{Adv}_{\Pi, \mathcal{A}}^{\mathrm{G}_{k,2}}(\lambda, \mathrm{des}) - \left(\frac{1}{2} + \mathrm{Adv}_{\Pi, \mathcal{A}}^{\mathrm{G}_{k,3}}(\lambda, \mathrm{des})\right)\right| \\
&= \left|\mathrm{Adv}_{\Pi, \mathcal{A}}^{\mathrm{G}_{k,2}}(\lambda, \mathrm{des}) - \mathrm{Adv}_{\Pi, \mathcal{A}}^{\mathrm{G}_{k,3}}(\lambda, \mathrm{des})\right| \ .
\end{aligned}$$

The second equation holds since $\mathcal{B}$ perfectly simulates $\mathrm{G}_{k,2}$ and $\mathrm{G}_{k,3}$ if $Z = Z_0$ and if $Z = Z_1$ respectively. Furthermore, $\mathcal{B}$ outputs 1 if and only if $\mathcal{A}$ wins the corresponding experiment. This proves the lemma. □

From $\mathrm{G}_{q_1,3}$ to G_{q_1+1}

Experiment $\mathrm{G}_{q_1,3}$ is a special case of $\mathrm{G}_{k,3}$ for $k = q_1$. In $\mathrm{G}_{q_1,3}$ all corrupted keys in Phase I are semi-functional of Type 3 and in Phase II all corrupted keys are normal. We simplify the description of the experiment as follows:

$G_{q_1,3}$:
Open (i) in Phase I:

- If sk_i is not generated yet, choose $\hat{\alpha}_j \leftarrow \mathbb{Z}_N$ and return the secret key $\mathrm{sk}_i \leftarrow \mathrm{SFKeyGen}\left(\mathrm{msk}, \mathrm{kInd}_i, 3, \hat{\alpha}_j, g_2, \hat{\boldsymbol{h}}\right)$.

Open (i) in Phase II:

- If sk_i is not generated yet, return $\mathrm{sk}_i \leftarrow \mathrm{KeyGen}\left(\mathrm{msk}, \mathrm{kInd}_i\right)$.

G_{q_1+1} is as $G_{q_1,3}$, but the corrupted keys in Phase II are semi-functional of Type 1:

Changes in G_{q_1+1} in comparison to $G_{q_1,3}$:
Open (i) in Phase II:

- If sk_i is not generated yet, return $\mathrm{sk}_i \leftarrow \mathrm{SFKeyGen}\left(\mathrm{msk}, \mathrm{kInd}_i, 1, \_, g_2, \hat{\boldsymbol{h}}\right)$.

Lemma 5.20. *(cf. Lemma 32 in [Att14b]) For every* $\mathrm{des} \in \Omega$ *and every ppt algorithm* $\mathcal{A}$ *there exists a ppt algorithm* $\mathcal{B}$ *such that for every security parameter* λ *it holds*

$$\left|\mathrm{Adv}_{\Pi,\mathcal{A}}^{G_{q_1,3}}(\lambda, \mathrm{des}) - \mathrm{Adv}_{\Pi,\mathcal{A}}^{G_{q_1+1}}(\lambda, \mathrm{des})\right| = \mathrm{Adv}_{\mathcal{B}}^{\mathrm{SD2}}(\lambda) \quad .$$

The running time of $\mathcal{B}$ *is essentially the same as the running time of* $\mathcal{A}$.

Proof. Given a ppt adversary $\mathcal{A}$ that can distinguish between $G_{q_1,3}$ and G_{q_1+1}, we construct a ppt algorithm $\mathcal{B}$ which uses $\mathcal{A}$ and breaks Assumption SD2 with the same advantage. Let $\mathrm{des} \in \Omega$ be arbitrary. $\mathcal{B}$ is given des in addition to its input (D, Z) from experiment SD2 as explained in Remark 5.9 and is presented in Algorithm 9. $\mathcal{B}$ is again similar to Algorithm 6, but there is no distinction of cases in the opening oracle. Thus, we have presented the complete algorithm. $\mathcal{B}$ is a ppt algorithm with respect to 1^λ by construction. In particular, random elements from $\mathbb{G}_{p_3}$ can be chosen using g_3, the elements from $g_1^{\boldsymbol{k}(\mathrm{msk}, \boldsymbol{r}', \boldsymbol{h})}$ can be computed as shown in Lemma 4.5, and $\boldsymbol{k}\left(0, \hat{\boldsymbol{r}}', \boldsymbol{h}'\right)$ can be computed explicitly.

Next, we analyze the view of $\mathcal{A}$ and the success probability of $\mathcal{B}$. By construction of $\mathcal{B}$ and by Statement 2 of Lemma 5.5 the public parameters $\mathrm{pp}_\kappa = \left(\mathrm{des}, \mathbb{GD}_N, g_1, g_1^{\boldsymbol{h}}, U_1, V_1, g_3, Y, \mathrm{H}\right)$ and the master secret msk are distributed as defined in the experiments. Note that $\mathbb{GD}_N$, g_1 and g_3 are distributed as required by Lemma 5.5 due to the definition of experiment SD2. Furthermore, by Statement 1 of Lemma 5.5 it holds $g_1^{\boldsymbol{h}'} = g_1^{\boldsymbol{h}}$, $U_1 = g_1^u$ and $V_1 = g_1^v$. $\mathcal{B}$ implicitly sets the semi-functional elements as $\hat{\boldsymbol{h}} = \boldsymbol{h}' \pmod{p_2}$, $\hat{u}_2 = u \pmod{p_2}$, and $\hat{v}_2 = v \pmod{p_2}$. These elements are properly distributed by Statement 3 of Lemma 5.5.

Let g_2 be an arbitrary but fixed generator of $\mathbb{G}_{p_2}$. Then, by the definition of probability experiment SD2 it holds $Z = g_1^{z_1} g_2^{z_2} g_3^{z_3}$, where $z_1 \in \mathbb{Z}_{p_1}^*$, $z_3 \in \mathbb{Z}_{p_3}^*$ are uniformly distributed, and z_2 is either uniformly distributed in $\mathbb{Z}_{p_2}^*$ (if $Z = Z_1$) or $z_2 = 0 \pmod{p_2}$ (if $Z = Z_0$). Furthermore, $X_1 = g_1^{x_1}$, $X_2 = g_2^{x_2}$, $Y_2 = g_2^{y_2}$ and $Y_3 = g_3^{y_3}$, where $x_1 \in \mathbb{Z}_{p_1}^*$, $x_2, y_2 \in \mathbb{Z}_{p_2}^*$ and $y_3 \in \mathbb{Z}_{p_3}^*$ are uniformly distributed and mutually independent.

The challenge and all semi-functional keys of Type 3 in Phase I are generated as required in the experiments by Lemma 5.6, and by Lemma 5.7 respectively.

Consider the corrupted keys in Phase II. By construction of the algorithm $\mathcal{B}$, for every i and

Algorithm 9: $\mathcal{B}$ against Assumption SD2

Input : (D, Z, des).
Require: $D = (\mathbb{GD}_N, g_1, X_1X_2, Y_2Y_3, g_3)$, $Z \in \mathbb{G}$, $\mathrm{des} \in \Omega$.

Setup
  Compute the public parameters and the master secret key $(\mathrm{msk}, \mathrm{pp}_\kappa, \boldsymbol{h}', u, v) \leftarrow \mathrm{SimPP}\left(1^\lambda, \mathbb{GD}_N, g_1, g_3, \mathrm{des}\right)$. Set $j := 0$.
Phase I
  CKGen (kInd_i) with $\mathrm{kInd}_i \in \mathbb{X}_\kappa$:
    Store (i, kInd_i).
  Open (i)**:**
    Set $j := j + 1$.
    Pick $\alpha'_j \leftarrow \mathbb{Z}_N$ and return $\mathrm{sk}_i \leftarrow \mathrm{SimSFKeyT3}\left(\mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{kInd}_i, Y_2Y_3, \alpha'_j\right)$.
  Decaps (CT, i)**:**
    As defined in the experiment using $\mathrm{sk}'_i \leftarrow \mathrm{KeyGen}\,(\mathrm{msk}, \mathrm{kInd}_i)$ generated once.
Challenge (given cInd^* from $\mathcal{A}$)
  Generate $(\mathrm{K}_0, \mathrm{CT}^*) \leftarrow \mathrm{SimSFChlg}\,(\mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{cInd}^*, \boldsymbol{h}', u, v, X_1X_2)$.
  Pick $\mathrm{K}_1 \leftarrow \mathbb{G}_\mathrm{T}$, flip a coin $b \leftarrow \{0,1\}$, set $\mathrm{K}^* := \mathrm{K}_b$, and return the challenge $(\mathrm{K}^*, \mathrm{CT}^*)$.
Phase II
  CKGen (kInd_i) with $\mathrm{kInd}_i \in \mathbb{X}_\kappa$:
    Store (i, kInd_i).
  Open (i)**:**
    Compute $(\boldsymbol{k}, m_2) := \mathrm{Enc1}\,(\mathrm{kInd}_i)$. Let $m_1 = |\boldsymbol{k}|$. Pick $\boldsymbol{r}', \hat{\boldsymbol{r}}' \leftarrow \mathbb{Z}_N^{m_2}$ and $\boldsymbol{R}'_3 \leftarrow \mathbb{G}_{p_3}^{m_1}$ and compute

$$\boldsymbol{K} \;:=\; g_1^{\boldsymbol{k}(\mathrm{msk}, \boldsymbol{r}', \boldsymbol{h})} \cdot Z^{\boldsymbol{k}\left(0, \hat{\boldsymbol{r}}', \boldsymbol{h}'\right)} \cdot \boldsymbol{R}'_3 \;.$$

    Return $\mathrm{sk}_i = (\mathrm{kInd}_i, \boldsymbol{K})$.
  Decaps (CT, i)**:**
    As defined in the experiment using $\mathrm{sk}'_i \leftarrow \mathrm{KeyGen}\,(\mathrm{msk}, \mathrm{kInd}_i)$ generated once.
Guess
  Output 1 if and only if $\mathcal{A}$ wins according to the definitions of the experiments.

every generated secret key $\mathrm{sk}_i = (\mathrm{kInd}_i, \boldsymbol{K})$ it holds:

$$\begin{aligned}
\boldsymbol{K} &= g_1^{\boldsymbol{k}(\mathrm{msk}, \boldsymbol{r}', \boldsymbol{h})} \cdot Z^{\boldsymbol{k}\left(0, \hat{\boldsymbol{r}}', \boldsymbol{h}'\right)} \cdot \boldsymbol{R}'_3 \\
&= g_1^{\boldsymbol{k}(\mathrm{msk}, \boldsymbol{r}', \boldsymbol{h})} \cdot \left(g_1^{z_1} g_2^{z_2} g_3^{z_3}\right)^{\boldsymbol{k}\left(0, \hat{\boldsymbol{r}}', \boldsymbol{h}'\right)} \cdot \boldsymbol{R}'_3 \\
&= g_1^{\boldsymbol{k}(\mathrm{msk}, \boldsymbol{r}', \boldsymbol{h}) + z_1 \cdot \boldsymbol{k}\left(0, \hat{\boldsymbol{r}}', \boldsymbol{h}'\right)} g_2^{z_2 \cdot \boldsymbol{k}\left(0, \hat{\boldsymbol{r}}', \boldsymbol{h}'\right)} \cdot g_3^{z_3 \cdot \boldsymbol{k}\left(0, \hat{\boldsymbol{r}}', \boldsymbol{h}'\right)} \cdot \boldsymbol{R}'_3 \\
&= g_1^{\boldsymbol{k}\left(\mathrm{msk}, \boldsymbol{r}' + z_1 \cdot \hat{\boldsymbol{r}}', \boldsymbol{h}\right)} \cdot g_2^{\boldsymbol{k}\left(0, z_2 \cdot \hat{\boldsymbol{r}}', \hat{\boldsymbol{h}}\right)} \cdot g_3^{z_3 \cdot \boldsymbol{k}\left(0, \hat{\boldsymbol{r}}', \boldsymbol{h}'\right)} \cdot \boldsymbol{R}'_3 \;.
\end{aligned}$$

We claim that for every i, the corresponding secret key sk is either a properly distributed normal secret key (if $Z = Z_0$) or a properly distributed semi-functional secret key of Type 1 (if $Z = Z_1$). Namely, $\mathcal{B}$ implicitly sets the random values of the normal components (in $\mathbb{G}_{p_1}$) as $\boldsymbol{r} = z_1 \cdot \hat{\boldsymbol{r}}' + \boldsymbol{r}' \pmod{p_1}$, which are properly distributed due to the choice of $\boldsymbol{r}'$. The random values of the semi-functional components (in $\mathbb{G}_{p_2}$) are set as $\hat{\boldsymbol{r}} = z_2 \cdot \hat{\boldsymbol{r}}'$, which are properly distributed (for Type 1 keys) due to the choice of $\hat{\boldsymbol{r}}'$ if $Z = Z_1$, and which disappear if $Z = Z_0$, and hence $z_2 = 0 \pmod{p_2}$. Finally, the $\mathbb{G}_{p_3}$ components are set as $\boldsymbol{R}_3 = g_3^{z_3 \cdot \boldsymbol{k}\left(0, \hat{\boldsymbol{r}}', \boldsymbol{h}'\right)} \cdot \boldsymbol{R}'_3$ and are properly distributed by the choice of $\boldsymbol{R}'_3$.

Hence, for every $\mathrm{des} \in \Omega$ and every $\mathcal{A}$ there exists a ppt algorithm $\mathcal{B}' = \mathcal{B}_{\mathcal{A}}(\cdot, \cdot, \mathrm{des})$ such that for every security parameter λ it holds

$$\begin{aligned}\mathrm{Adv}^{\mathrm{SD2}}_{\mathcal{B}'}(\lambda) &= \left|\Pr\left[\mathcal{B}'(D, Z_0) = 1\right] - \Pr\left[\mathcal{B}'(D, Z_1) = 1\right]\right| \\ &= \left|\frac{1}{2} + \mathrm{Adv}^{\mathrm{G}_{q_1,3}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) - \left(\frac{1}{2} + \mathrm{Adv}^{\mathrm{G}_{q_1+1}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})\right)\right| \\ &= \left|\mathrm{Adv}^{\mathrm{G}_{q_1,3}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) - \mathrm{Adv}^{\mathrm{G}_{q_1+1}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})\right|.\end{aligned}$$

The second equation holds since $\mathcal{B}$ perfectly simulates $\mathrm{G}_{q_1,3}$ and G_{q_1+1} if $Z = Z_0$ and $Z = Z_1$ respectively. Furthermore, $\mathcal{B}$ outputs 1 if and only if $\mathcal{A}$ wins the corresponding experiment. This proves the lemma. □

From G_{q_1+1} to G_{q_1+2}

G_{q_1+2} is as G_{q_1+1}, but the key in Phase II are semi-functional of Type 2.

Changes in G_{q_1+2} in comparison to G_{q_1+1}:

- Choose $\hat{\alpha} \leftarrow \mathbb{Z}_N$ at the beginning of Phase II.

Open (i) in Phase II

- Return $\mathrm{sk}_i \leftarrow \mathrm{SFKeyGen}\left(\mathrm{msk}, \mathrm{kInd}_i, 2, \hat{\alpha}, g_2, \hat{\boldsymbol{h}}\right)$.

Lemma 5.21. *(cf. Lemma 33 in [Att14b]) For every ppt algorithm $\mathcal{A}$ there exists a ppt algorithm $\mathcal{B}$ such that for every security parameter λ and every $\mathrm{des} \in \Omega$ it holds*

$$\left|\mathrm{Adv}^{\mathrm{G}_{q_1+1}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) - \mathrm{Adv}^{\mathrm{G}_{q_1+2}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})\right| = \mathrm{Adv}^{\mathrm{SMH}}_{\mathrm{P},\mathcal{B}}(\lambda, \mathrm{des}) \quad .$$

The running time of $\mathcal{B}$ is essentially the same as the running time of $\mathcal{A}$.

Proof. Given a ppt adversary $\mathcal{A}$ that can distinguish between G_{q_1+1} and G_{q_1+2}, we construct a ppt algorithm $\mathcal{B}$ which uses $\mathcal{A}$ and breaks the selective master-key hiding security property of the underlining pair encoding scheme with the same advantage. $\mathcal{B}$ on input $(\mathbb{GD}_N, g_1, g_2, g_3, \mathrm{des})$ as defined in $\mathrm{Exp}^{\mathrm{SMH}}_{\mathrm{P},\mathcal{G},\nu,\mathcal{A}}(\lambda, \mathrm{des})$ in Fig. 1.2 is presented in Algorithm 10. Note that $\mathcal{B}$ is similar to Algorithm 7, but the oracle is used in Phase II to generate all corrupted keys. $\mathcal{B}$ is a ppt algorithm with respect to λ by construction. It uses different supplementary ppt algorithms and performs besides only simple computation. Next we analyze the view of $\mathcal{A}$ and the success probability of $\mathcal{B}$.

Let security parameter λ and $\mathrm{des} \in \Omega$ be arbitrary, but fixed. By the definition of the Experiment $\mathrm{Exp}^{\mathrm{SMH}}_{\mathrm{P},\mathcal{G},\nu,\mathcal{A}}(\lambda, \mathrm{des})$, $\mathbb{GD}_N$ is the restricted group description of $\mathbb{GD}$ generated by $\mathcal{G}\left(1^\lambda\right)$. Furthermore, the generators $g_i \in \mathbb{G}_{p_i}$ are chosen uniformly at random. Hence, by construction of $\mathcal{B}$ and by Statement 2 of Lemma 5.5 the master secret msk and the public parameters $\mathrm{pp}_\kappa = \left(\mathrm{des}, \mathbb{GD}_N, g_1, g_1^{\boldsymbol{h}}, U_1, V_1, g_3, Y, \mathrm{H}\right)$ are distributed as defined in the experiments. Furthermore, by Statement 1 of Lemma 5.5 it holds $U_1 = g_1^u$ and $V_1 = g_1^v$. $\mathcal{B}$ implicitly sets the semi-functional elements as $\hat{u}_2 = u \pmod{p_2}$, and $\hat{v}_2 = v \pmod{p_2}$. These elements are properly distributed by Statement 3 of Lemma 5.5. Furthermore, $\mathcal{B}$ implicitly sets the input

Algorithm 10: $\mathcal{B}$ against selective master-key hiding security property

Input : $(\mathbb{GD}_N, g_1, g_2, g_3, \mathrm{des})$.

Setup

Compute the public parameters and the master secret key $(\mathrm{msk}, \mathrm{pp}_\kappa, \_, u, v) := \mathrm{SimPP}\left(1^\lambda, \mathbb{GD}_N, g_1, g_3, \mathrm{des}\right)$. Set $j := 0$.

Phase I

CKGen (kInd_i) with $\mathrm{kInd}_i \in \mathbb{X}_\kappa$:

Store (i, kInd_i).

Open (i)**:**

Set $j := j + 1$.

Pick $\alpha_j \leftarrow \mathbb{Z}_N$ and output $\mathrm{sk} \leftarrow \mathrm{SFKeyGen}\left(\mathrm{msk}, \mathrm{kInd}_i, 3, \alpha_j, g_2, \_\right)$.

Decaps (CT, i)**:**

As defined in the experiment using $\mathrm{sk}'_i \leftarrow \mathrm{KeyGen}\left(\mathrm{msk}, \mathrm{kInd}_i\right)$ generated once.

Challenge (given cInd^* from $\mathcal{A}$)

Compute $(\mathrm{K}_0, (\mathrm{cInd}^*, \boldsymbol{C}, \_)) \leftarrow \mathrm{Encaps}\left(\mathrm{cInd}^*\right)$.

Query the own oracle

$$\widehat{\boldsymbol{C}} := \mathcal{O}^1_{\mathrm{SMH},\nu,\hat{\alpha},\hat{\boldsymbol{h}}}\left(\mathrm{cInd}^*\right) \quad ,$$

Set $\boldsymbol{C}^* := \boldsymbol{C} \cdot \widehat{\boldsymbol{C}}$.

Compute $t^* := \mathrm{H}\left(\mathrm{HInput}\left(\mathrm{cInd}^*, \boldsymbol{C}^*, \_\right)\right)$ and

$$C''^* := \left(C_1^*\right)^{u \cdot t^* + v} \quad .$$

Choose $\mathrm{K}_1 \leftarrow \mathbb{G}_\mathrm{T}$, pick $b \leftarrow \{0, 1\}$, set $\mathrm{K}^* := \mathrm{K}_b$ and return $(\mathrm{K}^*, (\mathrm{cInd}^*, \boldsymbol{C}^*, C''^*))$.

Phase II

CKGen (kInd_i) with $\mathrm{kInd}_i \in \mathbb{X}_\kappa$:

Store (i, kInd_i).

Open (i)**:**

Query the oracle

$$\widehat{\boldsymbol{K}} := \mathcal{O}^2_{\mathrm{SMH},\nu,\hat{\alpha},\hat{\boldsymbol{h}}}\left(\mathrm{kInd}_i\right) \quad .$$

Compute a normal key $(\mathrm{kInd}_i, \boldsymbol{K}) \leftarrow \mathrm{KeyGen}\left(\mathrm{msk}, \mathrm{kInd}_i\right)$ and return

$$\mathrm{sk} := \left(\mathrm{kInd}_i, \boldsymbol{K} \cdot \widehat{\boldsymbol{K}}\right).$$

Decaps (CT, i)**:**

As defined in the experiment using $\mathrm{sk}'_i \leftarrow \mathrm{KeyGen}\left(\mathrm{msk}, \mathrm{kInd}_i\right)$ generated once.

Guess

Output 1 if and only if $\mathcal{A}$ wins according to the definitions of the experiments.

generator g_2 as the generator of $\mathbb{G}_{p_2}$. This generator is properly distributed as mentioned above. Vector $\hat{\boldsymbol{h}} \pmod{p_2}$ of the semi-functional public parameters will be defined below.

It is important to notice that all oracle queries made by $\mathcal{B}$ are permissible if all corruption queries of $\mathcal{A}$ are permissible, since $\mathrm{R}_N(\mathrm{kInd}, \mathrm{cInd}^*) = 0$ implies $\mathrm{R}_{p_2}(f_1(\mathrm{kInd}), f_2(\mathrm{cInd}^*)) = 0$ by Lemma 5.13. The normal keys and the semi-functional keys of Type 3 are generated using msk and g_2 as defined in the experiments.

By the definition of the Experiment $\mathrm{Exp}^{\mathrm{SMH}}_{\mathrm{P},\mathcal{G},\nu,\mathcal{A}}(\lambda, \mathrm{des})$, the challenger choose $\hat{\alpha} \leftarrow \mathbb{Z}_N$ and $\hat{\boldsymbol{h}} \leftarrow \mathbb{Z}_N^n$, where $n = \mathrm{Param}(\kappa)$. Then, $\mathcal{B}$ receives

$$\widehat{\boldsymbol{K}} = \mathcal{O}^2_{\mathrm{SMH},\nu,\hat{\alpha},\hat{\boldsymbol{h}}}(\mathrm{kInd}_i) = \begin{cases} g_2^{\boldsymbol{k}(0,\hat{\boldsymbol{r}},\hat{\boldsymbol{h}})} & \text{if } \nu = 0 \\ g_2^{\boldsymbol{k}(\hat{\alpha},\hat{\boldsymbol{r}},\hat{\boldsymbol{h}})} & \text{if } \nu = 1 \end{cases},$$

where $\hat{r} \in \mathbb{Z}_{p_2}^{m_2}$ is chosen uniformly at random for every key. Hence, all corrupted keys in Phase II are either properly distributed semi-functional keys of Type 1 (if $\nu = 0$) or properly distributed semi-functional keys of Type 2 (if $\nu = 1$) by construction. If $\nu = 1$, the element $\hat{\alpha}$ is the same for all these keys, as defined in the experiment G_{q_1+2}.

Furthermore, $\mathcal{B}$ receives

$$\widehat{\boldsymbol{C}} = \mathcal{O}^1_{\mathrm{SMH},\nu,\hat{\alpha},\hat{\boldsymbol{h}}}(\mathrm{cInd}^*) = g_2^{\boldsymbol{c}(\tilde{s},\tilde{\boldsymbol{s}},\hat{\boldsymbol{h}})},$$

where $\tilde{s} \in \mathbb{Z}_{p_2}$ and $\tilde{\boldsymbol{s}} \in \mathbb{Z}_{p_2}^{w_2}$ are chosen uniformly at random, $\hat{\boldsymbol{h}}$ is as above. Hence, $\widehat{\boldsymbol{C}}$ are properly distributed semi-functional components with $\hat{s} = \tilde{s} \pmod{p_2}$ and $\hat{\boldsymbol{s}} = \tilde{\boldsymbol{s}} \pmod{p_2}$.

Finally, we show that the last group element in the challenge encapsulation is correctly generated:

$$\begin{aligned} C''^* &= (C_1)^{u \cdot t^* + v} \cdot \left(\widehat{C}_1\right)^{u \cdot t^* + v} \\ &= (g_1^s)^{u \cdot t^* + v} \cdot \left(g_2^{\hat{s}}\right)^{u \cdot t^* + v} \\ &= \left(U_1^{t^*} \cdot V_1\right)^s \cdot \left(g_2^{\hat{u}_2 \cdot t^* + \hat{v}_2}\right)^{\hat{s}}, \end{aligned}$$

where s is the random element fixed by C_1, which is chosen as the first element of $\boldsymbol{C}$ in Line 12 of Algorithm 10 and $\hat{s}$ is fixed by $\widehat{C}_1$ as defined above. In the second equation we used the normality of P. Hence, C''^* is exactly as defined in SFEncaps.

We deduce that for every $\mathcal{A}$, there exist a ppt algorithm $\mathcal{B}$ such that for every security parameter λ and every $\mathrm{des} \in \Omega$ it holds

$$\begin{aligned} \mathrm{Adv}^{\mathrm{SMH}}_{\mathrm{P},\mathcal{B}}(\lambda, \mathrm{des}) &= \left|\mathrm{Exp}^{\mathrm{SMH}}_{\mathrm{P},\mathcal{G},0,\mathcal{B}}(\lambda, \mathrm{des}) - \mathrm{Exp}^{\mathrm{SMH}}_{\mathrm{P},\mathcal{G},1,\mathcal{B}}(\lambda, \mathrm{des})\right| \\ &= \left|\frac{1}{2} + \mathrm{Adv}^{\mathrm{G}_{q_1+1}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) - \left(\frac{1}{2} + \mathrm{Adv}^{\mathrm{G}_{q_1+2}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})\right)\right| \\ &= \left|\mathrm{Adv}^{\mathrm{G}_{q_1+1}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) - \mathrm{Adv}^{\mathrm{G}_{q_1+2}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})\right|. \end{aligned}$$

The second equation holds since $\mathcal{B}$ correctly simulates G_{q_1+1} and G_{q_1+2} if $\nu = 0$ and if $\nu = 1$ respectively. Furthermore, $\mathcal{B}$ outputs 1 if and only if $\mathcal{A}$ wins the corresponding game. This proves the lemma. □

From G_{q_1+2} to G_{q_1+3}

G_{q_1+3} is as G_{q_1+2}, but the key in Phase II are semi-functional of Type 3.

Changes in G_{q_1+3} in comparison to G_{q_1+2}:
Open (i) in Phase II:

- $\mathrm{sk}_i \leftarrow \mathrm{SFKeyGen}(\mathrm{msk}, \mathrm{kInd}_i, 3, \hat{\alpha}, g_2, \_)$. (Where $\hat{\alpha}$ as defined in G_{q_1+1})

Lemma 5.22. *(cf. Lemma 34 in [Att14b]) For every* $\mathrm{des} \in \Omega$ *and every ppt algorithm* $\mathcal{A}$ *there exists a ppt algorithm* $\mathcal{B}$ *such that for every security parameter* λ *it holds*

$$\left|\mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{q_1+2}}(\lambda, \mathrm{des}) - \mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{q_1+3}}(\lambda, \mathrm{des})\right| = \mathrm{Adv}_{\mathcal{A}}^{\mathrm{SD2}}(\lambda) \quad .$$

The running time of $\mathcal{B}$ *is essentially the same as the running time of* $\mathcal{A}$.

Proof. Given a ppt adversary $\mathcal{A}$ that can distinguish between G_{q_1+2} and G_{q_1+3}, we construct a ppt algorithm $\mathcal{B}$ which uses $\mathcal{A}$ and breaks Assumption SD2 with the same advantage. Let $\mathrm{des} \in \Omega$ be arbitrary. $\mathcal{B}$ is given des in addition to its input (D, Z) from experiment SD2 as explained in Remark 5.9. Note that $\mathcal{B}$ is almost the same as Algorithm 9. Hence, next we present only the simulation of corrupted keys in Phase II:

Algorithm 11: $\mathcal{B}$ against Assumption SD2 as modification of Algorithm 9

Input : (D, Z, des).
Require: $D = (\mathbb{GD}_N, g_1, X_1X_2, Y_2Y_3, g_3)$, $Z \in \mathbb{G}$, $\mathrm{des} \in \Omega$.

Setup
  ...
...
Phase II
  Pick $\hat{\alpha}' \leftarrow \mathbb{Z}_N$.
  ...
  Open (i)**:**
    Compute $(\boldsymbol{k}, m_2) := \mathrm{Enc1}(\mathrm{kInd}_i)$. Let $m_1 := |\boldsymbol{k}|$.
    Pick $\boldsymbol{r}', \hat{\boldsymbol{r}}', \leftarrow \mathbb{Z}_N^{m_2}$ and $\boldsymbol{R}_3' \leftarrow \mathbb{G}_{p_3}^{m_1}$ and compute

$$\boldsymbol{K} := g_1^{\boldsymbol{k}(\alpha, \boldsymbol{r}', \boldsymbol{h})} \cdot (Y_2Y_3)^{\boldsymbol{k}(\hat{\alpha}', 0, 0)} \cdot Z^{\boldsymbol{k}\left(0, \hat{\boldsymbol{r}}', \boldsymbol{h}\right)} \cdot \boldsymbol{R}_3'$$

    Return $(\mathrm{kInd}_i, \boldsymbol{K})$.
  ...
...

According to the analysis of Algorithm 9 we have to consider only the corrupted keys in Phase 2. Note that these keys are generated as the semi-functional keys of Type 3 in Phase I in Algorithm 8 except for the choice of $\hat{\alpha}'$, which is the same for all keys in this phase.

Let $Y_2 = g_2^{y_2}$ and $Z = g_1^{z_1} g_2^{z_2}$, where g_2 is an arbitrary but fixed generator of $\mathbb{G}_{p_2}$, $y_2 \in \mathbb{Z}_{p_2}^*$, $z_1 \in \mathbb{Z}_{p_1}^*$, and either $z_2 = 0 \pmod{p_2}$ if $Z = Z_0$ or $z_2 \in \mathbb{Z}_{p_2}^*$ if $Z = Z_1$, as defined in SD2. As already shown in the analysis of Algorithm 8, the semi-functional ($\mathbb{G}_{p_2}$) components of the keys are set to:

$$g_2^{\boldsymbol{k}\left(y_2 \cdot \hat{\alpha}', z_2 \cdot \hat{\boldsymbol{r}}', \hat{\boldsymbol{h}}\right)} \quad .$$

The elements in $\hat{\boldsymbol{r}}'$ are chosen uniformly at random for every key, whereas $\hat{\alpha}'$ is chosen once in the Setup Phase. Hence, the corresponding key is either a properly distributed semi-functional key of Type 2 (if $Z = Z_1$) or a properly distributed semi-functional key of Type 3 (if $Z = Z_0$) as already explained in the analysis of Algorithm 8.

Hence, for every $\mathrm{des} \in \Omega$ and every $\mathcal{A}$ there exists a ppt algorithm $\mathcal{B}' = \mathcal{B}_{\mathcal{A}}(\cdot,\cdot,\mathrm{des})$ such that for every security parameter λ it holds

$$\begin{aligned}\mathrm{Adv}^{\mathrm{SD2}}_{\mathcal{B}'}(\lambda) &= \left|\Pr\left[\mathcal{B}'(D,Z_0)=1\right] - \Pr\left[\mathcal{B}'(D,Z_1)=1\right]\right| \\ &= \left|\frac{1}{2} + \mathrm{Adv}^{\mathrm{G}_{q_1+3}}_{\Pi,\mathcal{A}}(\lambda,\mathrm{des}) - \left(\frac{1}{2} + \mathrm{Adv}^{\mathrm{G}_{q_1+2}}_{\Pi,\mathcal{A}}(\lambda,\mathrm{des})\right)\right| \\ &= \left|\mathrm{Adv}^{\mathrm{G}_{q_1+3}}_{\Pi,\mathcal{A}}(\lambda,\mathrm{des}) - \mathrm{Adv}^{\mathrm{G}_{q_1+2}}_{\Pi,\mathcal{A}}(\lambda,\mathrm{des})\right| \ .\end{aligned}$$

The second equation holds since $\mathcal{B}$ perfectly simulates G_{q_1+3} and G_{q_1+2} if $Z = Z_0$ and $Z = Z_1$ respectively. Furthermore, $\mathcal{B}$ outputs 1 if and only if $\mathcal{A}$ wins the corresponding experiment. This proves the lemma. □

From G_{q_1+3} to G'_{q_1+3}

In experiment G_{q_1+3} all keys generated in the opening oracle are semi-functional of Type 3 and all keys used in the decapsulation queries are normal. Experiment G'_{q_1+3} is defined as G_{q_1+3} except for decapsulation queries, which are answered without user secret keys. Namely, an additional generator X_2 of $\mathbb{G}_{p_2}$ is chosen uniformly at random in the setup phase and the decapsulation queries on $\mathrm{CT} = (\mathrm{cInd}, \boldsymbol{C}, C'')$, which passes the consistency checks, are answered with $\mathrm{e}\left(g_1^{\mathrm{msk}} \cdot X_2, C_1\right)$, where C_1 is the first element of $\boldsymbol{C}$.

Changes in G'_{q_1+3} in comparison to G_{q_1+3}:

- Pick $X_2 \leftarrow \mathbb{G}_{p_2}$ in the Setup phase.

Exchange $\langle 4 \rangle$ and $\langle 9 \rangle$ for:

- **Decaps** (CT, i):
  - Check that it holds $\mathrm{CT} \in \mathbb{C}_{\mathrm{cInd}} \subseteq \mathbb{C}_{\mathrm{pp}_\kappa}$ Perform the (implicit) syntactic checks. Return $\perp$ if these are not satisfied. Otherwise $\mathrm{CT} = (\mathrm{cInd}, \boldsymbol{C}, C'')$ for $\mathrm{cInd} \in \mathbb{Y}_\kappa$. Return $\perp$ if $\mathrm{R}(\mathrm{kInd}_i, \mathrm{cInd}) = 0$.
  - Return $\perp$ if the consistency checks in Decaps are not satisfied for CT.
  - Return $\mathrm{e}\left(g_1^{\mathrm{msk}} \cdot X_2, C_1\right)$, where $C_1 \in \boldsymbol{C}$.

Lemma 5.23. *For every* $\mathrm{des} \in \Omega$ *and every ppt algorithm* $\mathcal{A}$ *there exists a ppt algorithm* $\mathcal{B}$ *such that for every security parameter* λ *it holds*

$$\left|\mathrm{Adv}^{\mathrm{G}_{q_1+3}}_{\Pi,\mathcal{A}}(\lambda,\mathrm{des}) - \mathrm{Adv}^{\mathrm{G}'_{q_1+3}}_{\Pi,\mathcal{A}}(\lambda,\mathrm{des})\right| \leq \mathrm{Adv}^{\mathrm{SD2}}_{\mathcal{B}}(\lambda) + \frac{q_{\mathrm{dec1}}}{2^\lambda} + \frac{2}{2^\lambda} \ ,$$

where q_{dec1} *is the number of decapsulation queries in Phase I. The running time of* $\mathcal{B}$ *is essentially the same as the running time of* $\mathcal{A}$*.*

Proof. Experiments G_{q_1+3} and G'_{q_1+3} differ only in the realization of the decapsulation oracles. First, let us consider experiment G_{q_1+3}. Suppose that $\mathcal{A}$ queries the decapsulation oracle on (CT, i) with $\mathrm{CT} = (\mathrm{cInd}, \boldsymbol{C}, C'') \in \mathbb{C}_{\mathrm{cInd}}$ and $i \in \mathbb{N}$, and suppose that CT passes the consistency checks. Then, due to the checks in (4.4), the elements in $\boldsymbol{C}$ do not contain $\mathbb{G}_{p_3}$ components. By the definition of experiment G_{q_1+3}, the decapsulation queries are answered using normal keys. Let $\mathrm{sk}'_i = (\mathrm{kInd}_i, \boldsymbol{K})$, $\boldsymbol{K} = g_1^{\boldsymbol{k}(\mathrm{msk},\boldsymbol{r},\boldsymbol{h})} \cdot \boldsymbol{R}_3$ be the corresponding normal secret key generated to

answer the decapsulation query. Recall that the group elements of normal keys do not contain $\mathbb{G}_{p_2}$ components. Hence, during the decapsulation of CT using sk'_i the $\mathbb{G}_{p_2}$ components of $\boldsymbol{C}$ and the $\mathbb{G}_{p_3}$ components of $\boldsymbol{K}$ disappear. Namely, it holds

$$\mathrm{K} = \mathrm{e}\left(\left(g_1^{\boldsymbol{k}(\mathrm{msk},\boldsymbol{r},\boldsymbol{h})} \cdot \boldsymbol{R}_3\right)^{\boldsymbol{E}}, \boldsymbol{C}\right) = \mathrm{e}\left(\left(g_1^{\boldsymbol{k}(\mathrm{msk},\boldsymbol{r},\boldsymbol{h})}\right)^{\boldsymbol{E}}, \boldsymbol{C}\right) = \mathrm{e}\left(g_1, C_1\right)^{\mathrm{msk}} \quad ,$$

where the last equation holds since CT pass the check in (4.5) and due to the soundness of algorithm Vrfy.

In turn, by the definition of experiment G'_{q_1+3}, the decapsulation oracle on CT as above returns

$$\mathrm{K} = \mathrm{e}\left(g_1^{\mathrm{msk}} \cdot X_2, C_1\right) = \mathrm{e}\left(g_1, C_1\right)^{\mathrm{msk}} \cdot \mathrm{e}\left(X_2, C_1\right) \quad .$$

We deduce that the view of $\mathcal{A}$, and hence its success probabilities in the experiments G'_{q_1+3} and G_{q_1+3}, can differ if and only if $\mathcal{A}$ queries the decapsulation oracle on $(\mathrm{CT}, i) \in \mathbb{C}_{\mathrm{cInd}} \times \mathbb{N}$ such that CT pass the consistency checks and $C_1 \in \mathrm{CT}$ contains $\mathbb{G}_{p_2}$ component. At the same time the challenger should not output 0 in the guess phase because of an abort event (recall that in this case the output of both experiments is 0). Hence, we call by CipherAbort the event that a decapsulation query on (CT, i) with property from above (C_1 contains $\mathbb{G}_{p_2}$ component) exist and the events HashAbort and FactorAbort do not occur. By Lemma 5.8 it holds

$$\left|\mathrm{Adv}^{\mathrm{G}_{q_1+3}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) - \mathrm{Adv}^{\mathrm{G}'_{q_1+3}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})\right| \leq \Pr\left[\mathrm{CipherAbort}\right] \quad .$$

In order to analyze CipherAbort event, we construct an algorithm $\mathcal{B}$ against Experiment SD2, which uses $\mathcal{A}$ as a subroutine. $\mathcal{B}$ simulates game G_{q_1+3} for $\mathcal{A}$ and if the event CipherAbort occurs, $\mathcal{B}$ breaks Experiment SD2 which violates Assumption SD2.

The main observation is that if the event CipherAbort occurs and $t \neq t^* \pmod N$, then with overwhelming probability we can use the elements C_1 and C'' in order to compute a generator of $\mathbb{G}_{p_2}$. This is due to the fact that $\mathcal{A}$ gets information about $\hat{u}$ and $\hat{v} \pmod{p_2}$ only from $\hat{u} \cdot t^* + \hat{v} \pmod{p_2}$ contained in $C''^* \in \mathrm{CT}^*$. This value is independent of $\hat{u} \cdot t + \hat{v} \pmod{p_2}$ as long as $t \neq t^* \pmod{p_2}$ (which can be deduced from $t \neq t^* \pmod N$). Hence, over the random choice of $\hat{u}$ and $\hat{v} \pmod{p_2}$ the probability that $\mathcal{A}$ uses the correct value $\hat{u} \cdot t + \hat{v} \pmod{p_2}$ in CT is negligible. Furthermore, if the event CipherAbort occurs and $t = t^* \pmod N$ then, the $\mathbb{G}_{p_1}$ components of CT and CT^* must be the same. Hence, we will found an index l such that C_l and C^*_l differ only in the $\mathbb{G}_{p_2}$ components, and hence we again compute a generator of $\mathbb{G}_{p_2}$. Using a generator of $\mathbb{G}_{p_2}$ algorithm $\mathcal{B}$ can break the own challenge.

Let $\mathrm{des} \in \Omega$ be arbitrary, but fixed. $\mathcal{B}$ is given $\mathrm{des} \in \Omega$ in addition to its input (D, Z) from experiment SD2 as explained in Remark 5.9 and is presented in Algorithm 12. $\mathcal{B}$ is a ppt algorithm with respect to λ by construction. It uses different supplementary ppt algorithms and performs besides these only simple computation. Next, we analyze the view of $\mathcal{A}$ and the success probability of $\mathcal{B}$. $\mathcal{B}$ simulates $\mathcal{A}$ until her output (which is ignored) using supplementary algorithms. We are not interested in the output of the experiment, but in the success probability of $\mathcal{B}$ related to the probability that the event CipherAbort occurs. Hence, consider the computation of $\mathcal{B}$ in the guess phase.

At first, let us consider the computation of $\mathcal{B}$ independently of the event CipherAbort. Recall from page 76 that for $\mathrm{CT} \in \mathbb{C}_{\mathrm{cInd}}$ the input of the hash function is $\mathrm{HInput}\left(\mathrm{CT}\right) = \left(\mathrm{cInd}, \mathrm{e}\left(g_1, C_1\right), \ldots, \mathrm{e}\left(g_1, C_{w_1}\right)\right)$. Let (CT, i) be an encapsulation, considered in the foreach loop. Due to the checks in Line 22 of Algorithm 12 we can assume that event HashAbort does not occur. Hence, in Line 25 of Algorithm 12, $t = t^* \pmod N$ implies $\mathrm{HInput}\left(\mathrm{CT}\right) = \mathrm{HInput}\left(\mathrm{CT}^*\right)$. If an encapsulation with these properties occurs in Phase I, $\mathcal{B}$ aborts and outputs a guess (event Abort). However, in Phase I, $\mathcal{A}$ gets no information about C^*_1, and therefore it gets no information about its $\mathbb{G}_{p_1}$ component $g_1^{s^*}$. In turn, $s^* \pmod{p_1}$ is uniformly distributed

Algorithm 12: $\mathcal{B}$ against Assumption SD2

Input : (D, Z, des).
Require: $D = (\mathbb{GD}_N, g_1, X_1X_2, Y_2Y_3, g_3)$, $Z \in \mathbb{G}$, $\mathrm{des} \in \Omega$.

Setup
  Compute $(\mathrm{msk}, \mathrm{pp}_\kappa, \boldsymbol{h}', u, v) \leftarrow \mathrm{SimPP}(\mathbb{GD}_N, g_1, g_3, \mathrm{des})$. Pick $\hat{\alpha}' \leftarrow \mathbb{Z}_N$.
Phase I
  CKGen (kInd_i) with $\mathrm{kInd}_i \in \mathbb{X}_\kappa$:
    Store (i, kInd_i).
  Open (i):
    Pick $\hat{\alpha} \leftarrow \mathbb{Z}_N$ and return $\mathrm{sk} \leftarrow \mathrm{SimSFKeyT3}(\mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{kInd}_i, Y_2Y_3, \hat{\alpha})$.
  Decaps (CT, i):
    As defined in the experiment using normal secret key $\mathrm{sk}'_i \leftarrow \mathrm{KeyGen}(\mathrm{msk}, \mathrm{kInd}_i)$ generated once.
Challenge (given cInd^* from $\mathcal{A}$)
  Generate $(\mathrm{K}_0, \mathrm{CT}^*) \leftarrow \mathrm{SimSFChlg}(\mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{cInd}^*, \boldsymbol{h}', u, v, X_1X_2)$.
  Pick $\mathrm{K}_1 \leftarrow \mathbb{G}_\mathrm{T}$, flip a coin $b \leftarrow \{0,1\}$, set $\mathrm{K}^* := \mathrm{K}_b$, and return the challenge $(\mathrm{K}^*, \mathrm{CT}^*)$.
Phase II
  CKGen (kInd_i) with $\mathrm{kInd}_i \in \mathbb{X}_\kappa$:
    As before.
  Open (i):
    Return $\mathrm{sk} \leftarrow \mathrm{SimSFKeyT3}(\mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{kInd}_i, Y_2Y_3, \hat{\alpha}')$.
  Decaps (CT, i):
    As defined in the experiment using normal secret key $\mathrm{sk}'_i \leftarrow \mathrm{KeyGen}(\mathrm{msk}, \mathrm{kInd}_i)$ generated once.
Guess
  Ignore the output of $\mathcal{A}$. Choose $\nu' \leftarrow \{0,1\}$.
  Perform the original checks of bad events and output ν' if one of these events occurs.
  foreach *decapsulation query on* (CT, i) *with* $\mathrm{CT} \in \mathbb{C}_{\mathrm{cInd}}$ *and* $\mathrm{R}(\mathrm{kInd}_i, \mathrm{cInd}) = 1$, *where* CT *pass the consistency checks (let* t *be the corresponding hash value)* **do**
    if $t = t^* \pmod N$ **then**
      if *the query is made in Phase I* **then** output ν';
      else
        Look for an index $l \in [w_1 + 1]$ (index $w_1 + 1$ corresponds to the element C'') such that $C_l \neq C_l^*$. Then, compute $G_1 := \frac{C_l}{C_l^*}$.
        if $\mathrm{e}(G_1, Z) \neq 1_{\mathbb{G}_\mathrm{T}}$ **then** output 1;
        else output 0;
      end
    else
      Compute $G_2 := C'' \cdot C_1^{-(u \cdot t + v)}$.
      if $G_2 \neq 1_\mathbb{G}$ **then**
        if $\mathrm{e}(G_2, Z) \neq 1_{\mathbb{G}_\mathrm{T}}$ **then** output 1;
        else output 0;
      end
    end
  end
  Output ν'.

over $\mathbb{Z}_{p_1}$ and fixes the second element of $\mathrm{HInput}(\mathrm{CT}^*)$. Hence, the overall probability that $\mathcal{B}$ aborts and outputs a guess in Line 25 is at most $\frac{q_{\mathrm{dec1}}}{p_1}$, where q_{dec1} is the number of decapsulation queries in Phase I:

$$\Pr[\mathrm{Abort}] \leq \frac{q_{\mathrm{dec1}}}{p_1} \ .$$

Next we consider $\mathcal{B}$'s computation in Line 27 of Algorithm 12. In this case a query with $t = t^* \pmod N$ is made in the second phase and consequently it holds $\mathrm{HInput}(\mathrm{CT}) = \mathrm{HInput}(\mathrm{CT}^*)$. However, in Phase II the adversary is not allowed to query the decapsulation of CT^* – that is, $\mathrm{CT} \neq \mathrm{CT}^*$. We deduce that $\mathrm{cInd} = \mathrm{cInd}^*$ and the $\mathbb{G}_{p_1}$ components of all corresponding elements in $\boldsymbol{C}$ and $\boldsymbol{C}^*$ are equal. Together with the consistency check from (4.3) this implies that the $\mathbb{G}_{p_1}$ components of C'' and C''^* are equal, too. Furthermore, by the consistency checks in (4.4), the group elements of CT do not contain $\mathbb{G}_{p_3}$ components. The group elements of CT^* do not contain $\mathbb{G}_{p_3}$ components by construction. Hence, there is an index $l \in [w_1 + 1]$ such that $C_l \neq C_l^*$, the $\mathbb{G}_{p_1}$ components of both elements are equal and the $\mathbb{G}_{p_3}$ components are not present. We deduce that if $t = t^* \pmod N$ and the query is made in Phase II, $\mathcal{B}$ computes a generator $G_1 \in \mathbb{G}_{p_2}$ and can solve the own challenge with success probability 1.

Next, we consider the computation of $\mathcal{B}$ in Line 32 of Algorithm 12. In this case $t \neq t^* \pmod N$, which implies

$$\mathrm{HInput}(\mathrm{CT}) \neq \mathrm{HInput}(\mathrm{CT}^*) \ .$$

Hence, applying Lemma 5.12 it holds $t \neq t^* \pmod{p_2}$. Let g_2 be an arbitrary but fixed generator of $\mathbb{G}_{p_2}$. By the consistency checks in (4.4), the group elements of CT do not contain $\mathbb{G}_{p_3}$ components. Hence, we can denote $C_1 = g_1^s \cdot g_2^{\kappa_1}$, where $s \in \mathbb{Z}_{p_1}$ and $\kappa_1 \in \mathbb{Z}_{p_2}$. Furthermore, by the consistency checks in (4.3) it holds $C'' = g_1^{s \cdot (u \cdot t + v)} \cdot g_2^{\kappa''}$, where $\kappa'' \in \mathbb{Z}_{p_2}$. We deduce that $\mathcal{B}$ computes

$$G_2 = \frac{C''}{C_1^{u \cdot t + v}} = g_2^{\kappa'' - \kappa_1 \cdot (u \cdot t + v)} \in \mathbb{G}_{p_2} \ .$$

Hence, if $G_2 \neq 1_{\mathbb{G}}$ (which is additionally checked in the algorithm), $\mathcal{B}$ again solves the own challenge with success probability 1.

In summary, if $\mathcal{B}$ does not output the guess bit ν' (chosen uniformly and independently of any other random variables), and rather outputs 0 or 1 directly, the output is correct. In particular, the probability that $\mathcal{B}$ outputs 1 if $Z = Z_1$ is at least $\frac{1}{2}$, whereas the probability that $\mathcal{B}$ outputs 1 if $Z = Z_0$ is at most $\frac{1}{2}$. Hence, for every $\mathrm{des} \in \Omega$ and every ppt algorithm $\mathcal{A}$ there exists a ppt algorithm $\mathcal{B}' = \mathcal{B}_{\mathcal{A}}(\cdot, \cdot, \mathrm{des})$ such that for every security parameter λ it holds

$$\begin{aligned} \mathrm{Adv}_{\mathcal{B}'}^{\mathrm{SD2}}(\lambda) &= \left| \Pr\left[\mathcal{B}'(D, Z_0) = 1\right] - \Pr\left[\mathcal{B}'(D, Z_1) = 1\right] \right| \\ &= \Pr\left[\mathcal{B}'(D, Z_1) = 1\right] - \Pr\left[\mathcal{B}'(D, Z_0) = 1\right] \ . \end{aligned}$$

For our analysis, we can even neglect the advantage of $\mathcal{B}'$ in the case that the event CipherAbort does not occur. It holds

$$\Pr\left[\mathcal{B}'(D, Z_1) = 1 \;\middle|\; \overline{\mathrm{CipherAbort}}\right] - \Pr\left[\mathcal{B}'(D, Z_0) = 1 \;\middle|\; \overline{\mathrm{CipherAbort}}\right] \geq 0 \ ,$$

since also for this conditional probability distribution it holds

$$\begin{aligned} \Pr\left[\mathcal{B}'(D, Z_1) = 1 \;\middle|\; \overline{\mathrm{CipherAbort}}\right] &\geq \frac{1}{2} \ , \\ \Pr\left[\mathcal{B}'(D, Z_0) = 1 \;\middle|\; \overline{\mathrm{CipherAbort}}\right] &\leq \frac{1}{2} \ . \end{aligned}$$

Hence, we continue the analysis:

$$\begin{aligned}
\mathrm{Adv}^{\mathrm{SD2}}_{\mathcal{B}'}(\lambda) &= \Pr\left[\mathcal{B}'(D, Z_1) = 1\right] - \Pr\left[\mathcal{B}'(D, Z_0) = 1\right] \\
&= \Pr\left[\text{CipherAbort}\right] \cdot \big(\Pr\left[\mathcal{B}'(D, Z_1) = 1 \;\middle|\; \text{CipherAbort}\right] \\
&\qquad - \Pr\left[\mathcal{B}'(D, Z_0) = 1 \;\middle|\; \text{CipherAbort}\right]\big) \\
&\quad + \Pr\left[\,\overline{\text{CipherAbort}}\,\right] \cdot \big(\Pr\left[\mathcal{B}'(D, Z_1) = 1 \;\middle|\; \overline{\text{CipherAbort}}\,\right] \\
&\qquad - \Pr\left[\mathcal{B}'(D, Z_0) = 1 \;\middle|\; \overline{\text{CipherAbort}}\,\right]\big) \\
&\geq \Pr\left[\text{CipherAbort}\right] \cdot \big(\Pr\left[\mathcal{B}'(D, Z_1) = 1 \;\middle|\; \text{CipherAbort}\right] \\
&\qquad - \Pr\left[\mathcal{B}'(D, Z_0) = 1 \;\middle|\; \text{CipherAbort}\right]\big) \quad .
\end{aligned}$$

Next, we consider the advantage of $\mathcal{B}$ under the condition that the event CipherAbort occurs and the event Abort defined above does not occur. We claim that under these conditions $\mathcal{B}$ either outputs a correct bit using G_1 or $\mathcal{B}$ founds $G_2 \in \mathbb{G}_{p_2}$ except for a negligible probability and solve the own challenge. Namely, if a query with $t = t^* \pmod N$ is made in the second phase $\mathcal{B}$ will found G_1 and solve the own challenge as already explained above. The second part of our claim is a bit more complex.

By construction of $\mathcal{B}$, the adversary $\mathcal{A}$ gets information about $\hat{u}, \hat{v} \in \mathbb{Z}_{p_2}$ only from the value $\hat{u} \cdot t^* + \hat{v} \pmod{p_2}$, which is included in the exponent of C''^* in the semi-functional challenge encapsulation. Furthermore, $t \neq t^* \pmod N$ implies $t \neq t^* \pmod{p_2}$ as explained above. However, if $t \neq t^* \pmod{p_2}$, the values $\hat{u} \cdot t^* + \hat{v} \pmod{p_2}$ and $\hat{u} \cdot t + \hat{v} \pmod{p_2}$ are independent. Hence, $\mathcal{A}$ gets no information about $\hat{u} \cdot t + \hat{v} \pmod{p_2}$. However, if event CipherAbort occurs, there is an encapsulation CT such that $C_1 = g_1^s \cdot g_2^{\kappa_1}$ for some $s \in \mathbb{Z}_{p_1}$ and $\kappa_1 \in \mathbb{Z}^*_{p_2}$. For this encapsulation CT, by the analysis from above, the corresponding elements C'' and G_2 are equal to $g_1^{s \cdot (u \cdot t + v)} \cdot g_2^{\kappa''}$ and $g_2^{\kappa'' - \kappa_1 \cdot (u \cdot t + v)}$, respectively. Since $\kappa_1 \neq 0 \pmod{p_2}$ we deduce that the probability for $\kappa'' - \kappa_1 \cdot (u \cdot t + v) = 0 \pmod{p_2}$ is negligible (namely $\frac{1}{p_2}$) over the random choice of $\hat{u}, \hat{v} \pmod{p_2}$. Hence, except for negligible probability $\frac{1}{p_2}$, $\mathcal{B}$ computes a generator $G_2 \in \mathbb{G}_{p_2}$ and outputs a correct bit in Line 27 of Algorithm 12.

We deduce that it holds

$$\begin{aligned}
\Pr\left[\mathcal{B}'(D, Z_1) = 1 \;\middle|\; \text{CipherAbort} \wedge \overline{\text{Abort}}\right] &\geq 1 - \frac{1}{p_2} \\
\Pr\left[\mathcal{B}'(D, Z_0) = 1 \;\middle|\; \text{CipherAbort} \wedge \overline{\text{Abort}}\right] &\leq \frac{1}{p_2} \quad .
\end{aligned}$$

Now, we can continue the analysis from above

$$\begin{aligned}
\mathrm{Adv}^{\mathrm{SD2}}_{\mathcal{B}'}(\lambda) &\geq \Pr\left[\text{CipherAbort}\right] \cdot \big(\Pr\left[\mathcal{B}'(D, Z_1) = 1 \;\middle|\; \text{CipherAbort}\right] \\
&\qquad - \Pr\left[\mathcal{B}'(D, Z_0) = 1 \;\middle|\; \text{CipherAbort}\right]\big) \\
&\overset{(\star)}{=} \Pr\left[\text{CipherAbort}\right] \cdot \Pr\left[\,\overline{\text{Abort}} \;\middle|\; \text{CipherAbort}\right] \cdot \\
&\qquad \Big(\Pr\left[\mathcal{B}'(D, Z_1) = 1 \;\middle|\; \text{CipherAbort} \wedge \overline{\text{Abort}}\right] \\
&\qquad\quad - \Pr\left[\mathcal{B}'(D, Z_0) = 1 \;\middle|\; \text{CipherAbort} \wedge \overline{\text{Abort}}\right]\Big) \\
&\geq \Pr\left[\text{CipherAbort}\right] \cdot (1 - \Pr\left[\text{Abort} \mid \text{CipherAbort}\right]) \cdot \left(1 - \frac{2}{p_2}\right) \\
&\geq \Pr\left[\text{CipherAbort}\right] - \frac{2}{p_2} - \Pr[\text{Abort} \wedge \text{CipherAbort}] \\
&\geq \Pr\left[\text{CipherAbort}\right] - \frac{2}{p_2} - \frac{q_{\mathrm{dec1}}}{p_1} \quad .
\end{aligned}$$

Equation $(\star)$ holds since conditionally on the event Abort, $\mathcal{B}'$ outputs 1 with probability $\frac{1}{2}$ independently of Z, and hence

$$\Pr\left[\mathcal{B}'(D, Z_1) = 1 \;\middle|\; \text{CipherAbort} \wedge \text{Abort}\right] - \Pr\left[\mathcal{B}'(D, Z_0) = 1 \;\middle|\; \text{CipherAbort} \wedge \text{Abort}\right] = 0 \ .$$

In the following step we used the previous estimations. Finally, in the last two inequalities we used only simple estimations and $\Pr[\text{Abort}] \leq \frac{q_{\text{dec1}}}{p_1}$, explained above. This proves the lemma, since the prime numbers have a length of λ by our convention. □

From G'_{q_1+3} to $\mathrm{G}_{\text{Final}}$

$\mathrm{G}_{\text{Final}}$ is as G'_{q_1+3}, but the key K^* is chosen uniformly at random independetly of b.

Changes in $\mathrm{G}_{\text{Final}}$ in comparison to G'_{q_1+3}:
Exchange $\langle 6 \rangle$ for

- Set $\mathrm{K}^* \leftarrow \mathbb{G}_{\mathrm{T}}$.

Lemma 5.24. *(cf. Lemma 35 in [Att14b]) For every* $\text{des} \in \Omega$ *and every ppt algorithm* $\mathcal{A}$ *there exists a ppt algorithm* $\mathcal{B}$ *such that for every security parameter* λ *it holds*

$$\left|\mathrm{Adv}^{\mathrm{G}'_{q_1+3}}_{\Pi,\mathcal{A}}(\lambda, \text{des}) - \mathrm{Adv}^{\mathrm{G}_{\text{Final}}}_{\Pi,\mathcal{A}}(\lambda, \text{des})\right| \leq \mathrm{Adv}^{\text{SD3}}_{\mathcal{B}}(\lambda) + \frac{2}{2^\lambda} \ .$$

The running time of $\mathcal{B}$ *is essentially the same as the running time of* $\mathcal{A}$.

Proof. Given a ppt adversary $\mathcal{A}$ that can distinguish between both games, we construct a ppt algorithm $\mathcal{B}$ which breaks Assumption SD3 with the same advantage. Algorithm $\mathcal{B}$ is essentially the same as in the original reduction of Attrapadung. We additionally have to show how to answer the decapsulation queries. $\mathcal{B}$ against SD3 gets an input that includes $g_1^\alpha X_2$ and implicitly sets $\text{msk} := \alpha$. Furthermore, $\mathcal{A}$ gets no information about X_2. Hence, we can use $g_1^\alpha X_2$ in order to answer the decapsulation queries as defined in both games. Let $\text{des} \in \Omega$ be arbitrary but fixed. $\mathcal{B}$ is given $\text{des} \in \Omega$ in addition to its input (D, Z) from experiment SD3 as explained in Remark 5.9 and is presented in Algorithm 13. $\mathcal{B}$ is a ppt algorithm with respect to λ by construction. In particular, random elements from $\mathbb{G}_{p_3}$ can be chosen using generator $g_3 \in \mathbb{G}_{p_3}$, and all exponents can be computed explicitly. Next, we analyze the view of $\mathcal{A}$ and the success probability of $\mathcal{B}$.

By construction of $\mathcal{B}$ it holds $Y = \mathrm{e}(g_1, g_1)^\alpha$. Hence, the master secret key msk is implicitly set to α, which is properly distributed by the definition of Experiment SD3. The public parameters are correctly generated by construction. The semi-functional generator of $\mathbb{G}_{p_2}$ is set to g_2. Also this generator is properly distributed by the definition of Experiment SD3. Furthermore, the semi-functional elements $\hat{\boldsymbol{h}}$, $\hat{u}$ and $\hat{v}$ will be implicitly set to $\boldsymbol{h}' \pmod{p_2}$, $u \pmod{p_2}$, and $v \pmod{p_2}$ respectively. These elements are properly distributed by construction. In particular, these values are independent of the corresponding values modulo p_1 by the Chinese Remainder Theorem. Furthermore, we recall that by the definition of Experiment SD3, there exist $x_2, y_2 \in \mathbb{Z}^*_{p_2}$ such that $X_2 = g_2^{x_2}$ and $Y_2 = g_2^{y_2}$.

Now, consider keys generated in Phase I. By construction of $\mathcal{B}$ it holds:

$$\begin{aligned}
\boldsymbol{K} &= (g_1^\alpha X_2)^{\boldsymbol{k}(1,\mathbf{0},\mathbf{0})} \cdot g_1^{\boldsymbol{k}(0,\boldsymbol{r},\boldsymbol{h}')} \cdot g_2^{\boldsymbol{k}\left(\hat{\alpha}'_j,\mathbf{0},\mathbf{0}\right)} \cdot \boldsymbol{R}_3 \\
&= g_1^{\boldsymbol{k}(\text{msk},\mathbf{0},\mathbf{0})+\boldsymbol{k}(0,\boldsymbol{r},\boldsymbol{h}')} \cdot g_2^{\boldsymbol{k}(x_2,\mathbf{0},\mathbf{0})+\boldsymbol{k}\left(\hat{\alpha}'_j,\mathbf{0},\mathbf{0}\right)} \cdot \boldsymbol{R}_3 \\
&= g_1^{\boldsymbol{k}(\text{msk},\boldsymbol{r},\boldsymbol{h})} \cdot g_2^{\boldsymbol{k}\left(x_2+\hat{\alpha}'_j,\mathbf{0},\mathbf{0}\right)} \cdot \boldsymbol{R}_3 \ .
\end{aligned}$$

Algorithm 13: $\mathcal{B}$ against Assumption SD3

Input : (D, Z, des).
Require: $D = (\mathbb{GD}_N, g_1, g_1^{\alpha} X_2, g^s Y_2, g_2, g_3)$, $Z \in \mathbb{G}_{\mathrm{T}}$, $\mathrm{des} \in \Omega$.

Setup
- Set $\kappa := (\mathrm{des}, N)$ and compute $n := \mathrm{Param}(\kappa)$. Set $Y := \mathrm{e}(g_1, g_1^{\alpha} X_2)$.
- Pick $\boldsymbol{h'} \leftarrow \mathbb{Z}_N^n$ and $u, v \leftarrow \mathbb{Z}_N$. Compute $g_1^{\boldsymbol{h'}}$, $U_1 := g_1^u$ and $V_1 := g_1^v$.
- Choose a hash function $\mathrm{H} \leftarrow \mathcal{H}_{\kappa}$.
- Define $\mathrm{pp}_{\kappa} := \left(\mathrm{des}, \mathbb{GD}_N, g_1, g_1^{\boldsymbol{h'}}, U_1, V_1, g_3, Y, \mathrm{H}\right)$. Set $j := 0$
- Simulate $\mathcal{A}$ on input pp_{κ}.

Phase I
- **CKGen** (kInd_i) with $\mathrm{kInd}_i \in \mathbb{X}_{\kappa}$:
  - Store (i, kInd_i).
- **Open** (i)**:**
  - Set $j := j + 1$
  - Compute $(\boldsymbol{k}, m_2) := \mathrm{Enc1}(\mathrm{kInd})$. Let $m_1 := |\boldsymbol{k}|$.
  - Pick $\hat{\alpha}'_j \leftarrow \mathbb{Z}_N$, $\boldsymbol{r} \leftarrow \mathbb{Z}_N^{m_2}$, $\boldsymbol{R}_3 \leftarrow \mathbb{G}_{p_3}^{m_1}$ and compute
  $$\boldsymbol{K} := (g_1^{\alpha} X_2)^{\boldsymbol{k}(1,\boldsymbol{0},\boldsymbol{0})} \cdot g_1^{\boldsymbol{k}(0,\boldsymbol{r},\boldsymbol{h'})} \cdot g_2^{\boldsymbol{k}\left(\hat{\alpha}'_j,\boldsymbol{0},\boldsymbol{0}\right)} \cdot \boldsymbol{R}_3 \ .$$
  - Return $\mathrm{sk}_i = (\mathrm{kInd}_i, \boldsymbol{K})$
- **Decaps** (CT, i)**:**
  - **if** *all restrictions are satisfied and* CT *pass the consistency checks* **then** return $\mathrm{K} = \mathrm{e}(g_1^{\alpha} X_2, C_1)$;

Challenge (given cInd^* from $\mathcal{A}$)
- Compute $(\boldsymbol{c}, w_2) := \mathrm{Enc2}(\kappa, \mathrm{cInd}^*)$. Let $|\boldsymbol{c}| = w_1$.
- Pick $\boldsymbol{s'} \leftarrow \mathbb{Z}_N^{w_1}$ and compute
  $$\boldsymbol{C}^* := (g_1^s Y_2)^{\boldsymbol{c}(1,\boldsymbol{s'},\boldsymbol{h'})}$$
- Compute the hash value $t^* = \mathrm{HInput}(\mathrm{cInd}^*, \boldsymbol{C}^*, \_)$ and
  $$C''^* := (g_1^s Y_2)^{u \cdot t^* + v}$$
- Return $\mathrm{K}^* := Z$ and $\mathrm{CT}^* := (\mathrm{cInd}^*, \boldsymbol{C}^*, C''^*)$

Phase II
- As Phase I, but use $\hat{\alpha}' \leftarrow \mathbb{Z}_N$ instead of $\hat{\alpha}'_j$ for all keys.

Guess
- Output 1 if and only if $\mathcal{A}$ wins according to the definitions of the experiments.

This is a correctly generated semi-functional key of Type 3. The normal components are properly distributed due to the choice of $\boldsymbol{r}$ and $\boldsymbol{R}_3$. The semi-functional component $\hat{\alpha}$ is set to $\hat{\alpha}_j = x_2 + \hat{\alpha}'_j \pmod{p_2}$, which is correctly distributed due to the choice of $\hat{\alpha}'_j$. In Phase II the keys are generated in the same way, but with $\hat{\alpha} = x_2 + \hat{\alpha}'$ for all keys, which is properly distributed due to the choice of $\hat{\alpha}' \pmod{p_2}$. It is important to notice that the values $\hat{\alpha}_j$ and $\hat{\alpha}$ are independent of the value x_2.

Next, consider the challenge encapsulation. By construction of $\mathcal{B}$ it holds:

$$\begin{aligned} \boldsymbol{C}^* &= (g_1^s Y_2)^{\boldsymbol{c}(1,\boldsymbol{s}',\boldsymbol{h}')} & C''^* &= (g_1^s Y_2)^{u\cdot t^*+v} \\ &= g_1^{s\cdot\boldsymbol{c}(1,\boldsymbol{s}',\boldsymbol{h}')} \cdot g_2^{y_2\cdot\boldsymbol{c}\left(1,\boldsymbol{s}',\hat{\boldsymbol{h}}\right)} & &= g_1^{s\cdot(u\cdot t^*+v)} \cdot g_2^{y_2\cdot(u\cdot t^*+v)} \\ &= g_1^{\boldsymbol{c}(s,s\cdot\boldsymbol{s}',\boldsymbol{h})} \cdot g_2^{\boldsymbol{c}\left(y_2,y_2\cdot\boldsymbol{s}',\hat{\boldsymbol{h}}\right)}, & &= \left(U_1^{t^*}\cdot V_1\right)^s \cdot \left(g_2^{\hat{u}\cdot t^*+\hat{v}}\right)^{y_2} \quad . \end{aligned}$$

We claim that CT^* is a properly distributed semi-functional encapsulation except with negligible probability. The random values s and $\boldsymbol{s}$ of the normal components are set to $s \pmod{p_1}$ and $s\cdot\boldsymbol{s}'$ respectively. The value s is properly distributed due to the choice of $s \pmod{p_1}$. Vector $\boldsymbol{s}$ is properly distributed due to the choice of $\boldsymbol{s}' \pmod{p_1}$ as long as $s \neq 0 \pmod{p_2}$. The opposite happens with negligible probability $\frac{1}{p_2}$. The random values of the semi-functional components are set to $\hat{s} = y_2$ and $\hat{\boldsymbol{s}} = y_2\cdot\boldsymbol{s}'$. The value $\hat{s}$ is properly distributed due to the choice of $y_2 \pmod{p_2}$. The value $\hat{\boldsymbol{s}}$ is properly distributed due to the choice of $\boldsymbol{s}'$, since $y_2 \neq 0 \pmod{p_2}$. Hence, CT^* is a semi-functional encapsulation of key $\mathrm{K} = Y^s = Z_1$ except with probability $\frac{1}{p_2}$.

If $Z = Z_0$ the key K is chosen uniformly and independently at random from $\mathbb{G}_\mathrm{T}$ as required in $\mathrm{G}_\mathrm{Final}$. The simulation of CT^* is properly distributed except for negligible probability $\frac{1}{p_2}$.

The decapsulation queries are answered as defined in the experiment using X_2, properly distributed by the definition of Experiment SD3. In particular, all generated keys are independent of X_2.

In summary, for every $\mathrm{des} \in \Omega$ and every $\mathcal{A}$ there exists a ppt algorithm $\mathcal{B}' = \mathcal{B}_\mathcal{A}(\cdot,\cdot,\mathrm{des})$ such that for every security parameter λ it holds

$$\begin{aligned} \mathrm{Adv}^{\mathrm{SD2}}_{\mathcal{B}'}(\lambda) &= \left|\Pr\left[\mathcal{B}'(D,Z_0)=1\right] - \Pr\left[\mathcal{B}'(D,Z_1)=1\right]\right| \\ &\geq \left|\frac{1}{2} + \mathrm{Adv}^{\mathrm{G}_\mathrm{Final}}_{\Pi,\mathcal{A}}(\lambda,\mathrm{des}) - \left(\frac{1}{2} + \mathrm{Adv}^{\mathrm{G}_{q_1+3}}_{\Pi,\mathcal{A}}(\lambda,\mathrm{des})\right)\right| - \frac{2}{p_2} \quad . \\ &= \left|\mathrm{Adv}^{\mathrm{G}_\mathrm{Final}}_{\Pi,\mathcal{A}}(\lambda,\mathrm{des}) - \mathrm{Adv}^{\mathrm{G}_{q_1+3}}_{\Pi,\mathcal{A}}(\lambda,\mathrm{des})\right| - \frac{2}{p_2} \quad . \end{aligned}$$

The second equation holds since $\mathcal{B}$ almost perfectly simulates $\mathrm{G}_\mathrm{Final}$ and G_{q_1+3} if $Z = Z_0$ and $Z = Z_1$ respectively. Furthermore, $\mathcal{B}$ outputs 1 if and only if $\mathcal{A}$ wins the corresponding experiment. This proves the lemma, since the prime numbers have a length of λ by our convention. □

$\mathrm{G}_\mathrm{Final}$ and the Final Analysis

In the last game $\mathrm{G}_\mathrm{Final}$ the adversary gets no information about the challenge bit, and hence its advantage is equal to zero.

Lemma 5.25. *For every algorithm $\mathcal{A}$ and every* $\mathrm{des} \in \Omega$ *it holds*

$$\mathrm{Adv}^{\mathrm{G}_\mathrm{Final}}_{\Pi,\mathcal{A}}(\lambda,\mathrm{des}) = 0 \quad .$$

Summing up all factors from Lemma 5.10 to Lemma 5.25 we get

$$\begin{aligned}
\text{Adv-aP-KEM}^{\text{CCA2}}_{\Pi,\mathcal{A}}(\lambda,\text{des}) &= \text{Adv}^{\text{G}_{\text{Real}}}_{\Pi,\mathcal{A}}(\lambda,\text{des}) - \text{Adv}^{\text{G}_{\text{resH}}}_{\Pi,\mathcal{A}}(\lambda,\text{des}) \\
&\quad + \text{Adv}^{\text{G}_{\text{resH}}}_{\Pi,\mathcal{A}}(\lambda,\text{des}) - \text{Adv}^{\text{G}_{\text{resQ}}}_{\Pi,\mathcal{A}}(\lambda,\text{des}) \\
&\quad + \dots \\
&\quad + \text{Adv}^{\text{G}'_{q_1+3}}_{\Pi,\mathcal{A}}(\lambda,\text{des}) - \text{Adv}^{\text{G}_{\text{Final}}}_{\Pi,\mathcal{A}}(\lambda,\text{des}) \\
&\quad + \text{Adv}^{\text{G}_{\text{Final}}}_{\Pi,\mathcal{A}}(\lambda,\text{des}) \\
&\leq \text{Adv}^{\text{CR}}_{\mathcal{H},\mathcal{B}_1}(\lambda) + \text{Adv}^{\text{SD1}}_{\mathcal{B}_2}(\lambda) + \text{Adv}^{\text{SD3}}_{\mathcal{B}_4}(\lambda) \\
&\quad + (2q_1+4)\cdot\text{Adv}^{\text{SD2}}_{\mathcal{B}_3}(\lambda) + \text{Adv}^{\text{SMH}}_{\text{P},\mathcal{B}_6}(\lambda,\text{des}) \\
&\quad + q_1\cdot\text{Adv}^{\text{CMH}}_{\text{P},\mathcal{B}_5}(\lambda,\text{des}) + \frac{q_{\text{dec1}}}{2^\lambda} + \frac{4}{2^\lambda}\ .
\end{aligned}$$

This finally proves Theorem 5.1.

5.3. Verifiability for Regular Pair Encoding Schemes

In this section we prove that regular pair encoding schemes are verifiable. Recall Definition 1.11 of regular pair encodings and Definition 1.12 of set I. Before proving Theorem 5.2, we present the following lemma, which leads to simple verification algorithms for most known pair encoding schemes.

Lemma 5.26. *Suppose $\mathcal{R}_{\Omega,\Sigma}$ is a domain-transferable predicate family,* P *is a pair encoding scheme for $\mathcal{R}_{\Omega,\Sigma}$, and $\mathcal{G}$ is a composite-order group generator. If for every $\kappa\in\Omega\times\Sigma$, every* $\text{cInd}\in\mathbb{Y}_\kappa$ *with* $(\boldsymbol{c},w_2)=\text{Enc1}(\text{cInd})$, $w_1=|\boldsymbol{c}|$, *it holds $I=[w_2]_0$ – that is, for every $i\in[w_2]_0$ there exists an index $\tau_i\in[w_1]$ such that $c_{\tau_i}=\text{X}_{s_i}$, then* P *is verifiable with respect to $\mathcal{G}$ according to Definition 4.3.*

Proof. Let $\mathcal{R}_{\Omega,\Sigma}$ be an arbitrary but fixed domain-transferable predicate family and P be an arbitrary but fixed pair encoding scheme for $\mathcal{R}_{\Omega,\Sigma}$, which satisfies the property from the lemma. P is a normal encoding by definition, since among others, there exists $\tau_0\in[w_1]$ such that $c_{\tau_0}=\text{X}_{s_0}$. In order to prove the verifiability property of P, we construct an appropriate verification algorithm Vrfy for P. We assume w.l.o.g. that ciphertext encodings of P do not contain any polynomials multiple times, since these are redundant. Otherwise the corresponding group element in $\boldsymbol{C}$ must be checked for equality.

By Definition 4.3, Vrfy is given $\left(\text{des},\mathbb{GD}_N,g_1,g_1^{\boldsymbol{h}},\text{kInd},\text{cInd},\boldsymbol{E},\boldsymbol{C}\right)$ as input. The elements are as follows: λ is a security parameter, $\text{des}\in\Omega$, $\mathbb{GD}\in\left[\mathcal{G}\left(1^\lambda\right)\right]$ and $\mathbb{GD}_N$ is the corresponding restricted group description, $\kappa=(\text{des},N)\in\Omega\times\Sigma$, $g_1\in\mathbb{G}_{p_1}$, $g_1^{\boldsymbol{h}}\in\mathbb{G}_{p_1}^n$, where $n=\text{Param}(\kappa)$, $\text{kInd}\in\mathbb{X}_\kappa$ and $\text{cInd}\in\mathbb{Y}_\kappa$ with $\text{R}_\kappa(\text{kInd},\text{cInd})=1$, $\boldsymbol{E}\in[\text{Pair}(\kappa,\text{kInd},\text{cInd})]$, $\boldsymbol{C}\in\mathbb{G}^{w_1}$, where $(\boldsymbol{c},w_2)=\text{Enc1}(\kappa,\text{cInd})$, $w_1=|\boldsymbol{c}|$.

We make use of the observation from Lemma 4.4 and construct an algorithm Vrfy which checks if there exist $s_0\in\mathbb{Z}_{p_1}$ and $\boldsymbol{s}\in\mathbb{Z}_{p_1}^{w_2}$ such that the $\mathbb{G}_{p_1}$ components of $\boldsymbol{C}$ are equal to $g_1^{\boldsymbol{c}(s_0,\boldsymbol{s},\boldsymbol{h})}$. By the definition of pair encoding schemes, for every $\tau\in[w_1]$ the polynomial $c_\tau\in\boldsymbol{c}$ has the form

$$c_\tau=\sum_{i\in[w_2]_0}\left(b_{\tau,i}\cdot\text{X}_{s_i}+\sum_{j\in[n]}b_{\tau,i,j}\cdot\text{X}_{h_j}\text{X}_{s_i}\right)\in\mathbb{Z}_N\left[\text{X}_{s_0},\mathbf{X}_{\boldsymbol{s}},\mathbf{X}_{\boldsymbol{h}}\right]\ . \tag{5.1}$$

Due to the property of P from the lemma we can assume w.l.o.g. that it holds $c_1 = \mathrm{X}_{s_0}$, $c_2 = \mathrm{X}_{s_1}, \ldots, c_{w_2+1} = \mathrm{X}_{s_{w_2}}$. Hence, the $\mathbb{G}_{p_1}$ components of the corresponding group elements $C_1, \ldots, C_{w_2+1} \in \boldsymbol{C}$ particularly determine elements s_0 and $\boldsymbol{s} = (s_1, \ldots, s_{w_2})$ modulo p_1. Namely, for every $i \in [w_2]_0$ there exists unique $s_i \in \mathbb{Z}_{p_1}$ such that the $\mathbb{G}_{p_1}$ component of C_{i+1} is equal to $g_1^{s_i} = g_1^{c_{i+1}(s_0, \boldsymbol{s}, \boldsymbol{h})}$. Correctness of remaining elements $C_\tau \in \boldsymbol{C}$ for $w_2 + 1 < \tau \leq w_1$ can be checked as follows:

$$\mathrm{e}\left(C_\tau, g_1\right) \stackrel{?}{=} \prod_{i \in [w_2]_0} \mathrm{e}\left(C_{i+1}, g_1^{b_{\tau,i}} \cdot \prod_{j \in [n]} \left(g_1^{h_j}\right)^{b_{\tau,i,j}}\right) \quad . \tag{5.2}$$

The checks are constructed directly from (5.1). We deduce that for every $\tau \in [w_1] \setminus [w_2 + 1]$ the check of C_τ is satisfied if and only if the $\mathbb{G}_{p_1}$ components of C_τ is equal to $g_1^{c_\tau(s_0, \boldsymbol{s}, \boldsymbol{h})}$, since it holds

$$\begin{aligned}
& \prod_{i \in [w_2]_0} \mathrm{e}\left(C_{i+1}, g_1^{b_{\tau,i}} \cdot \prod_{j \in [n]} \left(g_1^{h_j}\right)^{b_{\tau,i,j}}\right) \\
= & \prod_{i \in [w_2]_0} \mathrm{e}\left(g_1^{s_i}, g_1^{b_{\tau,i} + \sum_{j \in [n]} (b_{\tau,i,j} \cdot h_j)}\right) \\
= & \ \mathrm{e}\left(g_1, g_1\right)^{\sum_{i \in [w_2]_0} \left(b_{\tau,i} \cdot s_i + \sum_{j \in [n]} (b_{\tau,i,j} \cdot s_i \cdot h_j)\right)} \\
= & \ \mathrm{e}\left(g_1^{c_\tau(s_0, \boldsymbol{s}, \boldsymbol{h})}, g_1\right) \quad .
\end{aligned}$$

Hence, Vrfy performs the checks in (5.2) for every $\tau \in [w_1] \setminus [w_2 + 1]$ and outputs 1, if and only if these checks are satisfied. The lemma follows by Lemma 4.4. □

The proof shows only the main idea. The algorithms constructed directly from the proof of Lemma 5.26 can be optimized for concrete schemes in different ways. On the one hand, one should look for reducing the number of pairing computations in (5.2) using usual techniques. Note furthermore that for the concrete schemes the number of coefficients unequal zero is small. On the other hand, the constructed algorithm does not use kInd and $\boldsymbol{E}$ given as input. In several predicate encryption schemes (e.g. attribute-based schemes) only few elements from $\boldsymbol{C}$ may be relevant for the decapsulation – that is, some columns of $\boldsymbol{E} \in \mathbb{Z}_N^{m_1 \times w_1}$ contain only zeros. In this case, the verification algorithm does not have to check the corresponding elements from $\boldsymbol{C} \in \mathbb{G}^{w_1}$ in order to ensure the soundness property. This results in more efficient verification algorithms. Formally, we have to perform the check in 5.2 for $k \in [w_1] \setminus [w_2 + 1]$ if and only if the column k in $\boldsymbol{E}$ is unequal $\boldsymbol{0}$. This still ensures the soundness property, since the values in $\boldsymbol{C}$ which are not checked do not affect the result of $\mathrm{e}\left(g_1^{\boldsymbol{k}(\alpha, \boldsymbol{r}, \boldsymbol{h}) \cdot \boldsymbol{E}}, \boldsymbol{C}\right)$.

Now we extend this result to regular pair encoding schemes.

Proof of Theorem 5.2. Let $\mathcal{R}_{\Omega,\Sigma}$ be an arbitrary but fixed domain-transferable predicate family and P be an arbitrary regular pair encoding scheme for $\mathcal{R}_{\Omega,\Sigma}$. P is a normal encoding by definition, due to the first property of regular pair encoding schemes. In order to prove the verifiability property of P, we construct an appropriate verification algorithm Vrfy for P.

By Definition 4.3, Vrfy is given $\left(\mathrm{des}, \mathbb{GD}_N, g_1, g_1^{\boldsymbol{h}}, \mathrm{kInd}, \mathrm{cInd}, \boldsymbol{E}, \boldsymbol{C}\right)$ as input. The elements are as follows: λ is a security parameter, $\mathrm{des} \in \Omega$, $\mathbb{GD} \in \left[\mathcal{G}\left(1^\lambda\right)\right]$ and $\mathbb{GD}_N$ is the corresponding restricted group description, $\kappa = (\mathrm{des}, N) \in \Omega \times \Sigma$, $g_1 \in \mathbb{G}_{p_1}$, $g_1^{\boldsymbol{h}} \in \mathbb{G}_{p_1}^n$, where $n = \mathrm{Param}\left(\kappa\right)$, $\mathrm{kInd} \in \mathbb{X}_\kappa$ and $\mathrm{cInd} \in \mathbb{Y}_\kappa$ with $\mathrm{R}_\kappa\left(\mathrm{kInd}, \mathrm{cInd}\right) = 1$, $\boldsymbol{E} \in \left[\mathrm{Pair}\left(\kappa, \mathrm{kInd}, \mathrm{cInd}\right)\right]$, $\boldsymbol{C} \in \mathbb{G}^{w_1}$, where $(\boldsymbol{c}, w_2) = \mathrm{Enc1}\left(\kappa, \mathrm{cInd}\right)$, $w_1 = |\boldsymbol{c}|$.

We make use of the observation from Lemma 4.4 and construct an algorithm Vrfy which checks if there exist $s_0 \in \mathbb{Z}_{p_1}$ and $\boldsymbol{s} \in \mathbb{Z}_{p_1}^{w_2}$ such that the $\mathbb{G}_{p_1}$ components of $\boldsymbol{C}$ are equal to $g_1^{\boldsymbol{c}(s_0, \boldsymbol{s}, \boldsymbol{h})}$.

The case $I = [w_2]_0$ is covered by Lemma 5.26. Hence, we only consider the case $I \subset [w_2]_0$. Note that $c_1 = \mathrm{X}_{s_0}$ by the first property of regular encodings, and hence $0 \in I$. Let $l := |I| - 1$. Assume w.l.o.g. that $I = [l]_0$ which implies $\overline{I} = \{l+1, \ldots, w_2\}$. Furthermore, assume w.l.o.g. that for every $i \in I$ it holds $c_{i+1} = \mathrm{X}_{s_i}$. Hence, the $\mathbb{G}_{p_1}$ components of the elements $C_1, \ldots, C_{l+1} \in \boldsymbol{C}$ particularly determine elements $s_0, s_1, \ldots, s_l \pmod{p_1}$. Namely, for every $i \in I$ there exists unique $s_i \in \mathbb{Z}_{p_1}$ such that the $\mathbb{G}_{p_1}$ component of C_{i+1} is equal to $g_1^{s_i} = g_1^{c_{i+1}(s_0, \boldsymbol{s}, \boldsymbol{h})}$, where only the first l elements of $\boldsymbol{s} = (s_1, \ldots, s_l, \_, \ldots, \_)$ are relevant.

We have to check if there exist $s_{l+1}, \ldots, s_{w_2} \in \mathbb{Z}_{p_1}$ such that the $\mathbb{G}_{p_1}$ components of remaining group elements $C_{l+2}, \ldots, C_{w_1} \in \boldsymbol{C}$ are consistent with all values $s_0, \ldots, s_{w_2}$.

Next, recall that the polynomials of regular pair encodings can be written as

$$c_\tau = \sum_{i \in \overline{I}} b_{\tau,i} \cdot \mathrm{X}_{s_i} + \sum_{i \in I} \left(b_{\tau,i} \cdot \mathrm{X}_{s_i} + \sum_{j \in [n]} b_{\tau,i,j} \cdot \mathrm{X}_{h_j} \cdot \mathrm{X}_{s_i} \right) \quad . \tag{5.3}$$

Furthermore, for $\tau \in \Gamma$ it holds by Definition 1.13

$$c_\tau = \sum_{i \in I} \left(b_{\tau,i} \cdot \mathrm{X}_{s_i} + \sum_{j \in [n]} b_{\tau,i,j} \cdot \mathrm{X}_{h_j} \cdot \mathrm{X}_{s_i} \right) \quad .$$

and we can perform the check for C_τ similar to the checks in (5.2) in the proof of Lemma 5.26:

$$\mathrm{e}\left(C_\tau, g_1\right) \stackrel{?}{=} \prod_{i \in I} \mathrm{e}\left(C_{i+1}, g_1^{b_{\tau,i}} \cdot \prod_{j \in [n]} \left(g_1^{h_j}\right)^{b_{\tau,i,j}} \right) \quad . \tag{5.4}$$

The checks are satisfied if and only if the $\mathbb{G}_{p_1}$ components of C_τ for every τ as above are equal to $g_1^{c_\tau(s_0, \boldsymbol{s}, \boldsymbol{h})}$, where only the first l elements of $\boldsymbol{s} = (s_1, \ldots, s_l, \_, \ldots, \_)$ are relevant. Namely, it holds

$$\begin{aligned} \prod_{i \in I} \mathrm{e}\left(C_{i+1}, g_1^{b_{\tau,i}} \cdot \prod_{j \in [n]} \left(g_1^{h_j}\right)^{b_{\tau,i,j}} \right) &= \prod_{i \in I} \mathrm{e}\left(g_1^{s_i}, g_1^{b_{\tau,i} + \sum_{j \in [n]} (b_{\tau,i,j} \cdot h_j)} \right) \\ &= \mathrm{e}\left(g_1, g_1\right)^{\sum_{i \in I} \left(b_{\tau,i} \cdot s_i + \sum_{j \in [n]} (b_{\tau,i,j} \cdot s_i \cdot h_j) \right)} \quad . \end{aligned}$$

Vrfy outputs 0, if one of these checks is not satisfied.

Let us assume w.l.o.g that t elements from $\boldsymbol{C}$, namely $C_{l+2}, \ldots, C_{l+t+1}$, can be checked using this kind of checks. It remains to check the elements $C_{l+t+2}, \ldots, C_{w_1} \in \boldsymbol{C}$ which correspond to the set $\overline{\Gamma}$ from Definition 1.13. By construction, for all $\tau \in [w_1] \setminus [l+t+1]$ the first summand of c_τ in (5.3) is unequal to zero. We first eliminate from every C_τ those parts which correspond to the correctly evaluated second summand from (5.3). Namely, Vrfy computes for every $\tau \in [w_1] \setminus [l+t+1]$:

$$Y_\tau := \mathrm{e}\left(C_\tau, g_1\right) \cdot \left(\prod_{i \in I} \mathrm{e}\left(C_{i+1}, g_1^{b_{\tau,i}} \cdot \prod_{j \in [n]} \left(g_1^{h_j}\right)^{b_{\tau,i,j}} \right) \right)^{-1} \quad . \tag{5.5}$$

Hence, it holds

$$Y_\tau = \mathrm{e}\left(C_k \cdot {g_1}^{-\sum_{i \in I} \left(b_{\tau,i} \cdot s_i + \sum_{j \in [n]} (b_{\tau,i,j} \cdot s_i \cdot h_j) \right)}, g_1 \right) \quad .$$

Note that all Y_τ's are element of the order p_1 subgroup of $\mathbb{G}_\mathrm{T}$ and $\mathrm{e}\left(g_1, g_1\right)$ is a generator of this subgroup. Hence, for every $\tau \in [w_1] \setminus [l+t+1]$ there exists unique $y_\tau \in \mathbb{Z}_{p_1}$ such that $Y_\tau = \mathrm{e}\left(g_1, g_1\right)^{y_\tau}$.

Next, we have to check if there exist $s_{l+1}, \ldots, s_{w_2} \in \mathbb{Z}_{p_1}$ such that for every $\tau \in [w_1] \backslash [l+t+1]$ it holds

$$Y_\tau = \mathrm{e}(g_1, g_1)^{\sum_{i \in \overline{I}} b_{\tau,i} \cdot s_i} \quad ,$$

or rather

$$y_\tau = \sum_{i \in \overline{I}} b_{\tau,i} \cdot s_i \pmod{p_1} \ . \tag{5.6}$$

This can be verified as follows.

Let us consider (5.6) more abstractly. Recall that $\overline{I} = \{l+1, \ldots, w_2\}$. Let $\varsigma := w_1 - (l+t+1)$ and $\varrho := w_2 - l$. These are the number of group elements $Y_\tau \in \mathbb{G}_\mathrm{T}$ which have to be checked, as well as the size of $\overline{I}$ respectively. Let $\boldsymbol{M} = (m_{i,j})_{i \in [\varsigma], j \in [\varrho]} \in \mathbb{Z}_N^{\varsigma \times \varrho}$ be the matrix of coefficients $b_{\tau,i}$ corresponding to (5.6). Hence, we have to check if there exist $\boldsymbol{s} \in \mathbb{Z}_{p_1}^{\varrho}$ such that

$$\boldsymbol{y} = \boldsymbol{M} \cdot \boldsymbol{s} \pmod{p_1} \ ,$$

where $\boldsymbol{y} = (y_\tau)_{\tau \in [w_1] \backslash [l+t+1]} \in \mathbb{Z}_{p_1}^{\varsigma}$ and $\boldsymbol{s} = (s_i)_{i \in \overline{I}} \in \mathbb{Z}_{p_1}^{\varrho}$ are vectors of the corresponding elements y_τ and s_i.

Using the Gaussian elimination algorithm (over $\mathbb{Z}_N$) on $\boldsymbol{M}$, Vrfy derives an invertible matrix $\boldsymbol{T} \in \mathbb{Z}_N^{\varsigma \times \varsigma}$ such that $\boldsymbol{T} \cdot \boldsymbol{M}$ is in reduced row echelon form. Under the assumption that the factorization of N is a difficult task, we can ignore the case that a zero divisor is hit during the computation of the algorithm. Furthermore, since $\boldsymbol{T}$ is invertible over $\mathbb{Z}_N$, it will be also invertible over $\mathbb{Z}_{p_1}$ and it holds

$$\begin{aligned}
& \exists_{\boldsymbol{s} \in \mathbb{Z}_{p_1}^{\varrho}} \ : \ \boldsymbol{M} \cdot \boldsymbol{s} = \boldsymbol{y} \pmod{p_1} \\
\Leftrightarrow \ & \exists_{\boldsymbol{s} \in \mathbb{Z}_{p_1}^{\varrho}} \ : \ \boldsymbol{T} \cdot \boldsymbol{M} \cdot \boldsymbol{s} = \boldsymbol{T} \cdot \boldsymbol{y} \pmod{p_1} \\
\Leftrightarrow \ & \exists_{\boldsymbol{s} \in \mathbb{Z}_{p_1}^{\varrho}} \ : \ \mathrm{e}(g_1, g_1)^{\boldsymbol{T} \cdot \boldsymbol{M} \cdot \boldsymbol{s}} = \mathrm{e}(g_1, g_1)^{\boldsymbol{T} \cdot \boldsymbol{y}} \quad .
\end{aligned}$$

Vrfy checks the last of these statements. It computes all elements of

$$\boldsymbol{x} = \mathrm{e}(g_1, g_1)^{\boldsymbol{T} \cdot \boldsymbol{y}} \in \mathbb{G}_\mathrm{T}^{\varsigma} \tag{5.7}$$

using $\boldsymbol{T}$ and $\{Y_\tau = \mathrm{e}(g_1, g_1)^{y_\tau}\}_{\tau \in [w_1] \backslash [l+t+1]}$.

We claim that there exist $\boldsymbol{s} \in \mathbb{Z}_{p_1}^{\varrho}$ such that $\mathrm{e}(g_1, g_1)^{\boldsymbol{T} \cdot \boldsymbol{M} \cdot \boldsymbol{s}} = \mathrm{e}(g_1, g_1)^{\boldsymbol{T} \cdot \boldsymbol{y}}$ if and only if for every zero row τ in $\boldsymbol{T} \cdot \boldsymbol{M} \in \mathbb{Z}_N^{\varsigma \times \varrho}$ it holds $x_\tau = \mathbf{1}_{\mathbb{G}_\mathrm{T}}$. This is true since $\boldsymbol{T} \cdot \boldsymbol{M}$ is in reduced row echelon form. If this is not the case, Vrfy outputs 0. Otherwise, Vrfy outputs 1.

In summary, we deduce that Vrfy outputs 1 if and only if there exist $s_0 \in \mathbb{Z}_{p_1}$ and $\boldsymbol{s} \in \mathbb{Z}_N^{w_2}$ such that the $\mathbb{G}_{p_1}$ components of $\boldsymbol{C}$ are equal to $g_1^{\boldsymbol{c}(s, \boldsymbol{s}, \boldsymbol{h})}$. Hence, according to Lemma 4.4 P is a verifiable pair encoding scheme. □

We note that the Vrfy algorithm from the proof might seam to be quite complex. Indeed, the most known predicate-based schemes are covered by much simpler algorithm from the proof of Lemma 5.26.

5.4. Simplified Construction

Our framework requires additional computational overhead during the computation of the hash value. Namely, a pairing is computed for every group element in the ciphertext. We can avoid this computation by hashing the original ciphertext. Formally, we only change the definition of the function HInput defined in (4.2). Then our last reduction must be adapted in order to prove the security for this variant. The other reductions require only minor modifications. We decided to present the given less efficient construction in order to explicitly show which parts

of the ciphertext are important for the consistency checks when the dual system encryption methodology is used to achieve CCA-secure schemes.

In this section we consider the modification for the more efficient construction. Indeed it is not difficult to see that it is sufficient to hash the original encapsulation. We already presented the security proof for our construction in Section 5.2 mostly independently of the concrete realization of the function HInput. This function defines which parts of the ciphertext are hashed (cf. (4.2)). Hence, in this section we do not present the complete formal proof, but indicate those parts of our original proof which depend on the definition of HInput.

The function HInput for our alternative construction is defined as follows:

$$\mathrm{HInput}\left(\mathrm{CT}\right) := \mathrm{cInd}, C_1, \ldots, C_{w_1} \quad .$$

instead of $\mathrm{HInput}\left(\mathrm{CT}\right) := \mathrm{cInd}, \mathrm{e}\left(g_1, C_1\right), \ldots, \mathrm{e}\left(g_1, C_{w_1}\right)$ defined in (4.2). That is, the hash value for the encapsulation $\mathrm{CT} = (\mathrm{cInd}, \boldsymbol{C} = (C_1, \ldots, C_{w_1}), C'')$ is computed by

$$t := \mathrm{H}\left(\mathrm{cInd}, C_1, \ldots, C_{w_1}\right) \quad , \tag{5.8}$$

both in the encapsulation and in the decapsulation algorithms. Note that t is independent of the last group element C'', and hence the encapsulation algorithm is well defined. Furthermore, the semi-functional encapsulation algorithm SFEncaps is already defined independently of the concrete realization of HInput.

Due to the modification of HInput the domain of computation for the family of collision resistant hash functions has to be changed. Namely, the hash family must be as follows:

$$\left\{\mathrm{H} : \mathbb{Y}_\kappa \times \left(\mathbb{G}\right)^{\leq m_{\mathrm{des},N}} \mapsto \mathbb{Z}_N\right\} \quad ,$$

where $m_{\mathrm{des},N}$ is the maximal number of polynomials in the pair encodings for $\mathrm{cInd} \in \mathbb{Y}_\kappa$. This completes the description of the structural modifications of our framework.

Next, let us consider the relevant parts of our security proof which require modifications due to the new definition of HInput. Note that Lemma 4.8 is formalized and proved independently of the definition of HInput. The statement of the Lemma 5.4 is also independent of HInput, since the proof of this lemma only requires that HInput is deterministic. Lemma 5.6 holds, since Algorithm 2 is defined independently of the concrete realization of HInput. Lemma 5.10 and Lemma 5.12 hold, since the event HashAbort is defined independently of the concrete realization of HInput.

Lemma 5.14, Lemma 5.18, Lemma 5.21, and Lemma 5.24 hold since Algorithm 5, Algorithm 7, Algorithm 10 and Algorithm 13 are defined independently of concrete realization of HInput, respectively. All other lemmata of the original proof do not depend on HInput except for the indistinguishability proof for the experiments G_{q_1+3} and G'_{q_1+3}. The proof for this step requires relatively simple modifications, which are considered next.

First of all notice that the first part of the proof for Lemma 5.23 still holds – that is,

$$\left|\mathrm{Adv}^{\mathrm{G}_{q_1+3}}_{\Pi,\mathcal{A}}\left(\lambda, \mathrm{des}\right) - \mathrm{Adv}^{\mathrm{G}'_{q_1+3}}_{\Pi,\mathcal{A}}\left(\lambda, \mathrm{des}\right)\right| \leq \Pr\left[\mathrm{CipherAbort}\right] \quad .$$

The analysis of event CipherAbort is almost the same. Only the element G_1 in Line 27 of the Algorithm 12 is computed as

$$G_1 := \frac{C''}{C''^*} \quad .$$

In this case a query with $t = t^* \pmod N$ is made in the second phase and consequently it holds $\mathrm{HInput}\left(\mathrm{CT}\right) = \mathrm{HInput}\left(\mathrm{CT}^*\right)$. However, in Phase II the adversary is not allowed to query the decapsulation of CT^* – that is, $\mathrm{CT} \neq \mathrm{CT}^*$. We deduce that $\mathrm{cInd} = \mathrm{cInd}^*$ and $\boldsymbol{C} = \boldsymbol{C}^*$ due to the new definition of HInput. Together with the consistency check from (4.3) this implies that

the $\mathbb{G}_{p_1}$ components of C'' and C''^* are equal, too. Furthermore, by the consistency checks in (4.4), the group elements of CT do not contain $\mathbb{G}_{p_3}$ components. The group elements of CT^* do not contain $\mathbb{G}_{p_3}$ components by construction. Hence, C'' and C''^* must differ in the $\mathbb{G}_{p_2}$ components and G_1 is a generator of $\mathbb{G}_{p_2}$. We deduce that if $t = t^* \pmod N$ and the query is made in Phase II, $\mathcal{B}$ computes a generator $G_1 \in \mathbb{G}_{p_2}$ and can solve the own challenge with success probability 1, which is similar to the original proof.

All other parts of the corresponding proof still holds independently of the definition of HInput.

5.5. Further Proofs

In this section we prove that under Assumption SD2, it is computationally infeasible to compute a non-trivial factor of N.

5.5.1. Hardness of Factorization Under Subgroup Decision Assumptions

The following lemma was implicitly proved in [LW10] (cf. Lemma 5 [LW09]).

Proof of Lemma 4.1. We prove that the following Algorithm 14 satisfies the required properties.

Algorithm 14: $\mathcal{B}$ against Assumption SD2 given a nontrivial factor of N

Input : (D, Z, F)
Require: $D = (\mathbb{GD}_N, g_1, X_1X_2, Y_2Y_3, g_3)$, $Z \in \mathbb{G}$, $F \in \mathbb{N}$, $1 < F < N$, and $F \mid N$.

1 Set $a := \min\left(F, \frac{N}{F}\right)$ and $b := \max\left(F, \frac{N}{F}\right)$;
2 **if** $(Y_2Y_3)^b = 1_{\mathbb{G}}$ **then**
3 | **if** $\mathrm{e}(Z^a, X_1X_2) = 1_{\mathbb{G}_\mathrm{T}}$ **then** output 0;
4 | **else** output 1;
5 **if** $(X_1X_2)^b = 1_{\mathbb{G}}$ **then**
6 | **if** $\mathrm{e}(Z^a, Y_2Y_3) = 1_{\mathbb{G}_\mathrm{T}}$ **then** output 0;
7 | **else** output 1;
8 **if** $Z^b = 1_{\mathbb{G}}$ **then** output 0;
9 **else** output 1;

$\mathcal{B}$ is a ppt algorithm with respect to λ by construction. Next, we analyze the success probability of the algorithm. Let $\hat{g}_1$, $\hat{g}_2$, $\hat{g}_3$ be arbitrary but fixed generators of $\mathbb{G}_{p_1}$, $\mathbb{G}_{p_2}$ and $\mathbb{G}_{p_3}$ respectively. By the definition of Experiment SD2 it holds $X_1 = \hat{g}_1^{x_1}$, $X_2 = \hat{g}_2^{x_2}$, $Y_2 = \hat{g}_2^{y_2}$, $Y_3 = \hat{g}_3^{y_3}$, and $Z = \hat{g}_1^{z_1} \cdot \hat{g}_2^{z_2} \cdot \hat{g}_3^{z_3}$. Therefore, x_1 and z_1 are uniformly distributed in $\mathbb{Z}_{p_1}^*$, x_2 and y_2 are uniformly distributed in $\mathbb{Z}_{p_2}^*$, y_3 and z_3 are uniformly distributed in $\mathbb{Z}_{p_3}^*$. Furthermore, $z_2 = 0$ if $Z = Z_0$, whereas z_2 is uniformly distributed in $\mathbb{Z}_{p_2}^*$ if $Z = Z_1$.

Given a non-trivial factor F of N we have three possible cases:

$$1)\ a = p_1 \wedge b = p_2p_3, \quad 2)\ a = p_2 \wedge b = p_1p_3, \quad 3)\ a = p_3 \wedge b = p_1p_2\ .$$

The condition $(Y_2Y_3)^b = 1_{\mathbb{G}}$ from Line 2 is satisfied if and only if $b = p_2p_3$. In this case $a = p_1$ and the $\mathbb{G}_{p_1}$ component of Z disappears in Z^a. Hence, $Z = Z_0 \in \mathbb{G}_{p_1p_3}$ if and only if $\mathrm{e}(Z^a, X_1X_2) = 1_{\mathbb{G}_\mathrm{T}}$. Analogously, the condition $(X_1X_2)^b = 1_{\mathbb{G}}$ from Line 5 is satisfied if and only if $b = p_1p_2$. In this case $a = p_3$ and the $\mathbb{G}_{p_3}$ component of Z disappears in Z^a. Hence, $Z = Z_0 \in \mathbb{G}_{p_1p_3}$ if and only if $\mathrm{e}(Z^a, Y_2Y_3) = 1_{\mathbb{G}_\mathrm{T}}$. Due to the observations from above, Line 5 is executed if and only if $b = p_1p_3$. In this case the $\mathbb{G}_{p_1}$ and the $\mathbb{G}_{p_3}$ components of Z disappears in Z^b. Hence, $Z = Z_0 \in \mathbb{G}_{p_1p_3}$ if and only if $Z^b = 1_{\mathbb{G}}$.

In summary, under the assumption from above, $\mathcal{B}$ outputs 1 if and only if $Z = Z_1$. Hence, it holds $\Pr[\mathcal{B}(D, Z_1, F) = 1] = 1$ and $\Pr[\mathcal{B}(D, Z_0, F) = 1] = 0$. Consequently,

$$|\Pr[\mathcal{B}(D, Z_0, F) = 1] - \Pr[\mathcal{B}(D, Z_1, F) = 1]| = 1$$

This proves the lemma. □

Part III.

Fully CCA-Secure Framework in Prime-Order Groups

Introduction and Main Contribution

As already discussed in the introduction of the second part of this thesis in the context of IBE, researchers have spent much effort to achieve efficient CCA-secure constructions [BCHK07, KG09, KV08]. However, IBE is a very special kind of PE and there is no generic method to apply similar techniques to more sophisticated predicates. In this part of the thesis we focus on PE scheme which are fully CCA-secure without the Random Oracle model and especially on constructions in prime-order groups. These constructions are significantly more efficient than constructions in composite-order groups [Gui13] which we considered in the previous part in order to develop techniques to achieve CCA security for sophisticated predicates.

We recall that under the fully secure predicate encryption schemes constructed in prime-order groups and considered in [YAHK11, NP15] only the ABE schemes from [OT10a] and the inner-product encryption schemes from [OT11] satisfy the requirements of the generic transformation from delegatability property. Furthermore, only for two predicate encryption schemes developed in prime-order groups a method to construct consistency checks, appropriate to apply the generic transformation from verifiability, was suggested in [YAS+12]. As already discussed in the introduction for the second part of this thesis, the corresponding approach does not seem to be sufficient for other predicate encryption schemes. These results and observations raise the question why the techniques to construct appropriate consistency checks in composite-order groups do not work for prime-order groups. In our opinion, the main challenge in the development of consistency checks for schemes constructed using dual system encryption techniques in prime-order groups results from the fact that the so-called normal components and the semi-functional components are not strictly separated as is the case in composite-order groups. Namely, in groups of composite order the normal and the semi-functional components are realized in different subgroups and consequently do not interact among each other; whereas in prime-order groups normal and semi-functional components are in the same group. Our consistency checks in composite-order groups simply ignore the semi-functional components of the ciphertext, which is in principle not possible in prime-order groups.

Main Contribution

In this last part of the thesis we present a CCA-secure variant of the prime-order pair encoding framework from [Att16]. Our construction is inspired by the direct chosen-ciphertext techniques [BMW05, KG09] and especially by our CCA-secure pair encoding framework in composite-order groups from the second part of the thesis. Differently from the CHK-like transformations from [YAHK11, YAS+12, NP15], our construction is not generic. That is, we cannot transform any given CPA-secure scheme into a CCA-secure scheme, but we benefit from the abstraction of pair encodings such that our framework yields a variety of new CCA-secure PE schemes. We again exploit a single collision-resistant hash function, do not use any additional security assumptions, and this time do not even make any additional demands on the pair encodings. That is, for every pair encoding scheme which leads to a CPA-secure scheme via the framework of [Att16] a CCA-secure scheme can be constructed using our framework. Hence, according to the instantiations presented in [Att16] we achieve CCA-secure PE schemes in prime-order groups for various key-policy ABE, ciphertext-policy ABE, dual-policy ABE, schemes for branching programs and regular languages, to name just a few. We refer to [Att16] for an excellent overview about known pair encoding schemes and the resulting PE schemes (over 25 schemes are presented). The research on pair encodings is ongoing [AC17b] and new

computationally secure regular pair encodings will directly lead to CCA-secure schemes from our framework.

The structural extensions of the CPA-secure schemes in our framework are indeed similar to our framework in composite-order groups. We add only four *group elements* (in terms of [Att16]) to the public parameters and one group element to the ciphertext. We use a more efficient asymmetric bilinear map, and hence the public parameters are extended by four instead of two additional group elements. The user secret keys are not changed at all. Our main extensions concern the decryption algorithm, which verifies to a certain extent the structure of the ciphertext before the actual decryption using our newly developed consistency checks. The efficiency of these checks depends on the predicate or rather on the structure of the ciphertext pair encodings. Regarding the computational efficiency of the decryption algorithm, we usually achieve more efficient constructions for key-policy schemes than for ciphertext-policy schemes. Last but not least, the reduction costs of our framework are almost the same as the reduction costs of the underlying CPA-secure framework [Att16] due to our proof techniques introduced in the second part of this thesis.

New techniques to construct consistency checks in prime-order groups. The original CPA-secure framework of Attrapadung [Att16] as well as our framework are based on the dual system encryption methodology. As already discussed in the introduction of the second part of this thesis, one cannot hope to achieve public verifiability for schemes constructed using these techniques. Nevertheless, we introduce a kind of consistency check for schemes in prime-order groups that is sufficient to achieve CCA-security using our framework. On the one hand, our techniques are generic and lead to appropriate consistency checks for all schemes constructed using the prime-order pair encoding framework from [Att16]. On the other hand our checks do not satisfy the demands of the generic transformations from verifiability property [YAHK11, YAS$^+$12]. Indeed, the guarantees of our checks are weaker than the guarantees of the verification checks constructed for our framework in composite-order groups. This is due to the fact that the normal and the semi-functional components are not strictly separated in the schemes from prime-order groups as explained above.

In the security proof of our framework we apply the proof strategy introduced in the second part of this thesis to deal with decryption queries within the dual system encryption proof techniques. This not only leads to security guarantees comparable to the security guarantees of the underlying CPA-secure framework but also leaves most parts of the CPA-secure proof unchanged. Nevertheless, due to the weaker guarantees of our consistency checks in comparison to the guarantees of the checks in our composite-order framework, more sophisticated proof techniques are required for the security proof. Particularly, one of the main reductions steps in the proof for our framework in composite-order groups is based on the verifiability property and does not even require any security assumptions. The corresponding reduction for the framework in this part of the thesis is based on a Matrix Decisional Diffie-Hellman security assumption used in the original framework anyway.

Finally, we remark that our consistency checks exploit the structure of the prime-order Dual-System Groups introduced in [CW13]. This variant of the Dual-Pairing Vector Space approach presented in [OT08, OT09] was utilized in the original CPA-secure pair encoding framework from Attrapadung [Att16]. Hence, we believe that our consistency checks can also be applied to achieve CCA-security for schemes constructed using the prime-order Dual-System Groups even if these schemes do not fit into the pair encoding framework.

Organization. In Chapter 6 we introduce our CCA-secure framework and explain the construction of our verification checks and the intuition behind them. In Chapter 7 we present the security proof for the framework.

We remark that the results presented in this part are submitted to the IACR conference Eurocrypt 2018.

6. Background, Construction and Intuition

This chapter is organized as follows. First, in Section 6.1 we define domain of computation and computational rules, prove general lemmata used in the security proof and recall the security assumptions. Then, in Section 6.2 we present our CCA-secure framework and the semi-functional algorithms essential for the proof. We also prove some properties that are used in the security proofs. In particular, in Subsection 6.2.6 we not only prove the correctness of the framework, but also the guarantees of our verification checks.

6.1. Preliminaries

Let $n, m \in \mathbb{N}$ be arbitrary integers and $\boldsymbol{M}$ be a matrix over domain $\mathcal{D}$ with n rows and m columns – that is, $\boldsymbol{M} \in \mathcal{D}^{n \times m}$. We denote by $m_{i,j} \in \mathcal{D}$ the element of $\boldsymbol{M}$ in the i'th row and the j'th column and by $\left(m_{i,j}\right)_{n,m}$ we denote the whole matrix $\boldsymbol{M}$. If n and m are obvious from context, we will sometimes simply write $\boldsymbol{M} = \left(m_{i,j}\right)$.

In this chapter we work with field $\mathbb{F}_p$ for prime p denoted by $\mathbb{Z}_p$. The matrices over $\mathbb{Z}_p$ are denoted by bold capital letters e.g. $\boldsymbol{M}$ and vectors are denoted by bold lowercase letters, e.g. $\boldsymbol{v}$. Furthermore, we also consider matrices of group elements, which are also denoted by bold capital letters but in roman font, e.g. $\mathbf{G}$. Notice that all vectors in this chapter are column vectors.

For prime p and integer d we denote by $\mathbb{GL}_{p,d}$ the general linear group of degree d according to the ordinary matrix multiplication. That is, $\mathbb{GL}_{p,d}$ is the set of $d \times d$ invertible matrices over $\mathbb{Z}_p$ together with the operation of ordinary matrix multiplication. By $\boldsymbol{I}_d = \left(e_{i,j}\right)_{d,d}$ we denote the neutral element of $\mathbb{GL}_{p,d}$ – that is, $e_{i,j} = 1$ if $i = j$ and $e_{i,j} = 0$ otherwise.

6.1.1. Matrices of Group Elements and Computation Rules

In this subsection we first of all define notational conventions used in the whole chapter and consider computational rules which are used in the following framework and in the proof. We also present the running times for the mentioned computations in order to give the reader indication for complexity of the corresponding operation. For the whole subsection let $l, m, n \in \mathbb{N}$ be arbitrary integers, $(\mathbb{G}, \cdot)$ be an Abelian group of prime order p, $g \in \mathbb{G}$ and $a, b \in \mathbb{Z}_p$ be also arbitrary. The neutral element of $\mathbb{G}$ is denoted by $1_{\mathbb{G}}$.

Let $\mathbf{G} = \left(g_{i,j}\right)_{n,m}, \mathbf{H} = \left(h_{i,j}\right)_{n,m} \in \mathbb{G}^{n \times m}$ be $n \times m$ matrices of group elements. Define

$$\mathbf{G} \cdot \mathbf{H} := \left(f_{i,j}\right)_{n,m} \in \mathbb{G}^{n \times m} \text{ , where} \tag{6.1}$$

$$\forall_{i \in [n]} \forall_{j \in [m]} : f_{i,j} = g_{i,j} \cdot h_{i,j} \ .$$

Matrices from $\mathbb{G}^{n \times m}$ with this operation form an Abelian group of order $p^{n \cdot m}$ with neutral element $\left(1_{\mathbb{G}}\right)_{n,m}$. This group is isomorphic to the finite product group $\mathbb{G}^{n \cdot m}$. Furthermore, every element from this group (except the neutral element) generates a subgroup of order p. Hence, we use a shortcut for multiple application of the group operation and define

$$\mathbf{G}^a := \left(f_{i,j}\right)_{n,m} \in \mathbb{G}^{n \times m} \text{ , where} \tag{6.2}$$

$$\forall_{i \in [n]} \forall_{j \in [m]} : f_{i,j} = g_{i,j}^a \ .$$

We get the following running times for these operations:

Input 1	Input 2	Output	Running time
$\mathbf{G} \in \mathbb{G}^{n \times m}$	$\mathbf{H} \in \mathbb{G}^{n \times m}$	$\mathbf{G} \cdot \mathbf{H} \in \mathbb{G}^{n \times m}$	$n \cdot m \cdot \mathrm{mul}_{\mathbb{G}}$
$\mathbf{G} \in \mathbb{G}^{n \times m}$	$a \in \mathbb{Z}_p$	$\mathbf{G}^a \in \mathbb{G}^{n \times m}$	$n \cdot m \cdot \exp_{\mathbb{G}}$

The following computational rules can be deduced from the properties of Abelian groups:

$$\begin{aligned} \mathbf{G} \cdot \mathbf{H} &= \mathbf{H} \cdot \mathbf{G} \ , \\ \mathbf{G}^{a+b} &= \mathbf{G}^a \cdot \mathbf{G}^b \ , \\ (\mathbf{G} \cdot \mathbf{H})^a &= \mathbf{G}^a \cdot \mathbf{H}^a \ . \end{aligned}$$

Next, we define a shortcut that has been previously used in many papers. For matrix $\boldsymbol{A} = \left(a_{i,j}\right)_{n,m} \in \mathbb{Z}_p^{n \times m}$ we define:

$$g^{\boldsymbol{A}} := \mathbf{F} = \left(f_{i,j}\right)_{n,m} \in \mathbb{G}^{n \times m} \ , \text{ where} \tag{6.3}$$

$$\forall_{i \in [n]} \forall_{j \in [m]} : f_{i,j} = g^{a_{i,j}} \ .$$

Given $g \in \mathbb{G}$, $\boldsymbol{A} \in \mathbb{Z}_p^{n \times m}$ we can compute $g^{\boldsymbol{A}} \in \mathbb{G}^{n \times m}$. The running time for this operation is:

Input 1	Input 2	Output	Running time
$g \in \mathbb{G}$	$\boldsymbol{A} \in \mathbb{Z}_p^{n \times m}$	$g^{\boldsymbol{A}} \in \mathbb{G}^{n \times m}$	$n \cdot m \cdot \exp_{\mathbb{G}}$

Notice that for every generator $g \in \mathbb{G}$ and every $\mathbf{G} = \left(g_{i,j}\right)_{n,m} \in \mathbb{G}^{n \times m}$ there exists unique $\boldsymbol{M} = \left(m_{i,j}\right)_{n,m} \in \mathbb{Z}_p^{n \times m}$ such that for every $i \in [n]$, $j \in [m]$ it holds $g_{i,j} = g^{m_{i,j}}$. Let $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{Z}_p^{n \times m}$ and $\mathbf{G}, \mathbf{H} \in \mathbb{G}^{n \times m}$ be such that $\mathbf{G} = g^{\boldsymbol{A}}$ and $\mathbf{H} = g^{\boldsymbol{B}}$. Then, from (6.1) and (6.2) we deduce that it holds

$$\begin{aligned} \mathbf{G}^a &= \left(g^{\boldsymbol{A}}\right)^a \\ &= g^{a \cdot \boldsymbol{A}} \ , \end{aligned} \tag{6.4}$$

where $a \cdot \boldsymbol{A}$ denotes ordinary scalar multiplication of matrices over $\mathbb{Z}_p$. Furthermore, from (6.4) and (6.1) we deduce that it holds

$$\begin{aligned} \mathbf{G}^a \cdot \mathbf{H}^b &= g^{a \cdot \boldsymbol{A}} \cdot g^{b \cdot \boldsymbol{B}} \\ &= g^{a \cdot \boldsymbol{A} + b \cdot \boldsymbol{B}} \ , \end{aligned} \tag{6.5}$$

where $a \cdot \boldsymbol{A} + b \cdot \boldsymbol{B}$ denotes the ordinary matrix addition over $\mathbb{Z}_p$. In particular, $\mathbf{G} \cdot \mathbf{H} = g^{\boldsymbol{A} + \boldsymbol{B}}$.

Let $\mathbf{G} = g^{\boldsymbol{A}} \in \mathbb{G}^{n \times d}$ and $\boldsymbol{B} \in \mathbb{Z}_p^{d \times l}$ we define

$$\mathbf{G}^{\boldsymbol{B}} := g^{\boldsymbol{A} \cdot \boldsymbol{B}} \in \mathbb{G}^{n \times l} \ , \tag{6.6}$$

where $\boldsymbol{A} \cdot \boldsymbol{B}$ denotes the ordinary matrix multiplication over $\mathbb{Z}_p$. Next, we consider different ways to compute $\mathbf{G}^{\boldsymbol{B}}$. Let $\boldsymbol{C} = \left(c_{i,j}\right)_{n,l} = \boldsymbol{A} \cdot \boldsymbol{B}$. Then, by the definition of matrix multiplication it holds $\forall_{i \in [n]} \forall_{j \in [l]} : c_{i,j} = \sum_{k=1}^{d} a_{i,k} \cdot b_{k,j}$. Hence, it holds

$$\begin{aligned} \forall_{i \in [n]} \forall_{j \in [l]} : g^{c_{i,j}} &= g^{\sum_{k=1}^{d} a_{i,k} \cdot b_{k,j}} \\ &= \prod_{k=1}^{d} g^{a_{i,k} \cdot b_{k,j}} \\ &= \prod_{k=1}^{d} \left(g^{a_{i,k}}\right)^{b_{k,j}} \qquad (6.7) \\ &= \prod_{k=1}^{d} \left(g^{b_{k,j}}\right)^{a_{i,k}} \ . \qquad (6.8) \end{aligned}$$

Using the expression from (6.7) we can compute $g^{\boldsymbol{A}\cdot\boldsymbol{B}}$ component by component given $g^{\boldsymbol{A}}$ and $\boldsymbol{B}$. Analogously, from (6.8) we can compute $g^{\boldsymbol{A}\cdot\boldsymbol{B}}$ given $\boldsymbol{A}$ and $g^{\boldsymbol{B}}$. In our work we often explicitly use the fact that $g^{\boldsymbol{A}\cdot\boldsymbol{B}} \in \mathbb{G}^{n\times l}$ can be computed from $\boldsymbol{A} \in \mathbb{Z}_p^{n\times d}$ and $\mathbf{H} = g^{\boldsymbol{B}} \in \mathbb{G}^{d\times l}$. Hence, we establish the following notation

$$ {}^{\boldsymbol{A}}\mathbf{H} = {}^{\boldsymbol{A}}\left(g^{\boldsymbol{B}}\right) := g^{\boldsymbol{A}\cdot\boldsymbol{B}} \quad . \tag{6.9} $$

The running times are summarized in the following table (dominating computations).

Input 1	Input 2	Input 3	Output	Running time
$g \in \mathbb{G}$	$\boldsymbol{A} \in \mathbb{Z}_p^{n\times d}$	$\boldsymbol{B} \in \mathbb{Z}_p^{d\times l}$	$g^{\boldsymbol{A}\cdot\boldsymbol{B}} \in \mathbb{G}^{n\times l}$	$\mathrm{Mat\_mul}_{\mathbb{Z}_p} + n\cdot l\cdot \exp_{\mathbb{G}}$
$\mathbf{G} = g^{\boldsymbol{A}} \in \mathbb{G}^{n\times d}$	$\boldsymbol{B} \in \mathbb{Z}_p^{d\times l}$		$\mathbf{G}^{\boldsymbol{B}} \in \mathbb{G}^{n\times l}$	$n\cdot l\cdot d\cdot \exp_{\mathbb{G}}$ by (6.7)
$\boldsymbol{A} \in \mathbb{Z}_p^{n\times d}$	$\mathbf{H} = g^{\boldsymbol{B}} \in \mathbb{G}^{d\times l}$		${}^{\boldsymbol{A}}\mathbf{H} \in \mathbb{G}^{n\times l}$	$n\cdot l\cdot d\cdot \exp_{\mathbb{G}}$ by (6.8)

Next let $\mathbb{GD} = (p, (g_1, \mathbb{G}_1), (g_2, \mathbb{G}_2), \mathbb{G}_\mathrm{T}, \mathrm{e})$ be asymmetric bilinear groups of prime order p with bilinear map $\mathrm{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_\mathrm{T}$. Furthermore, let $d_1, d_2 \in \mathbb{N}$ be arbitrary integers, $\boldsymbol{A} = \left(a_{k,j}\right)_{n,d_1} \in \mathbb{Z}_p^{n\times d_1}$ and $\boldsymbol{B} = \left(b_{k,i}\right)_{n,d_2} \in \mathbb{Z}_p^{n\times d_2}$. Define the following shortcut (established in [Att16])

$$ \mathrm{e}\left(g_1^{\boldsymbol{A}}, g_2^{\boldsymbol{B}}\right) := \mathrm{e}\left(g_1, g_2\right)^{\boldsymbol{B}^\top\cdot\boldsymbol{A}} \in {\mathbb{G}_\mathrm{T}}^{d_2\times d_1} \quad . \tag{6.10} $$

For $\boldsymbol{C} = \left(c_{i,j}\right)_{d_2,d_1} = \boldsymbol{B}^\top \cdot \boldsymbol{A}$ it holds $c_{i,j} = \sum_{k=1}^{n} b_{k,i}\cdot a_{k,j}$, and hence

$$ \begin{aligned} \forall_{i\in[d_2]}\forall_{j\in[d_1]} : \mathrm{e}\left(g_1, g_2\right)^{c_{i,j}} &= \mathrm{e}\left(g_1, g_2\right)^{\sum_{k=1}^{n} b_{k,i}\cdot a_{k,j}} && (6.11)\\ &= \prod_{k=1}^{n} \mathrm{e}\left(g_1, g_2\right)^{b_{k,i}\cdot a_{k,j}} \\ &= \mathrm{e}\left(g_1, \prod_{k=1}^{n}\left(g_2^{b_{k,i}}\right)^{a_{k,j}}\right) && (6.12)\\ &= \mathrm{e}\left(\prod_{k=1}^{n}\left(g_1^{a_{k,j}}\right)^{b_{k,i}}, g_2\right) && (6.13)\\ &= \prod_{k=1}^{n} \mathrm{e}\left(g_1^{a_{k,j}}, g_2^{b_{k,i}}\right) \quad . && (6.14) \end{aligned} $$

Let $\mathbf{G}_1 := g_1^{\boldsymbol{A}}$ and $\mathbf{G}_2 := g_2^{\boldsymbol{B}}$. We can compute the elements of $\mathrm{e}\left(\mathbf{G}_1, \mathbf{G}_2\right) \in {\mathbb{G}_\mathrm{T}}^{d_2\times d_1}$ component by component depending on the given elements. The running times are summarized in the following table (dominating computations).

Input 1	Input 2	Input 3	Running time
$\mathrm{e}\left(g_1, g_2\right) \in \mathbb{G}_\mathrm{T}$	$\boldsymbol{A} \in \mathbb{Z}_p^{n\times d_1}$	$\boldsymbol{B} \in \mathbb{Z}_p^{n\times d_2}$	$\mathrm{Mat\_mul}_{\mathbb{Z}_p} + d_1\cdot d_2\cdot \exp_{\mathbb{G}_\mathrm{T}}$ by (6.11)
$g_1 \in \mathbb{G}_1$	$\boldsymbol{A} \in \mathbb{Z}_p^{n\times d_1}$	$\mathbf{G}_2 \in \mathbb{G}_2^{n\times d_2}$	$d_1\cdot d_2\cdot\left(n\cdot \exp_{\mathbb{G}_2} + \mathrm{pair}\right)$ by (6.12)
$\mathbf{G}_1 \in \mathbb{G}_1^{n\times d_1}$	$g_2 \in \mathbb{G}_2$	$\boldsymbol{B} \in \mathbb{Z}_p^{n\times d_2}$	$d_1\cdot d_2\cdot\left(n\cdot \exp_{\mathbb{G}_1} + \mathrm{pair}\right)$ by (6.13)
$\mathbf{G}_1 \in \mathbb{G}_1^{n\times d_1}$	$\mathbf{G}_2 \in \mathbb{G}_2^{n\times d_2}$		$d_1\cdot d_2\cdot n\cdot \mathrm{pair}$ by (6.14)

Let $g_1 \in \mathbb{G}_1$, $g_2 \in \mathbb{G}_2$ be generators and $\mathbf{G}_1 \in \mathbb{G}_1^{n\times d_1}$ and $\mathbf{G}_2 \in \mathbb{G}_2^{n\times d_2}$ be arbitrary. Then, there exist unique $\boldsymbol{A} \in \mathbb{Z}_p^{n\times d_1}$ and $\boldsymbol{B} \in \mathbb{Z}_p^{n\times d_2}$ such that $\mathbf{G}_1 = g_1^{\boldsymbol{A}}$ and $\mathbf{G}_2 = g_2^{\boldsymbol{B}}$. We derive the following calculation rules for $a, b \in \mathbb{Z}_p$:

$$ \mathrm{e}\left(\mathbf{G}_1^a, \mathbf{G}_2^b\right) \overset{(6.4)}{=} \mathrm{e}\left(g_1^{a\cdot\boldsymbol{A}}, g_2^{b\cdot\boldsymbol{B}}\right) \overset{(6.10)}{=} \mathrm{e}\left(g_1, g_2\right)^{a\cdot b\cdot\boldsymbol{B}^\top\cdot\boldsymbol{A}} \overset{(6.10)}{=} \mathrm{e}\left(\mathbf{G}_1, \mathbf{G}_2\right)^{a\cdot b} \quad . $$

Hence, we can work with element in $\mathbb{Z}_p$ in the exponents as usual in the context of bilinear maps. Next, let $\boldsymbol{C} \in \mathbb{Z}_p^{d_1 \times d_1'}$, and $\boldsymbol{D} \in \mathbb{Z}_p^{d_2 \times d_2'}$. Then, it holds:

$$\mathrm{e}\left(\mathbf{G}_1^{\boldsymbol{C}}, \mathbf{G}_2^{\boldsymbol{D}}\right) \overset{(6.6)}{=} \mathrm{e}\left(g_1^{\boldsymbol{A}\cdot\boldsymbol{C}}, g_1^{\boldsymbol{B}\cdot\boldsymbol{D}}\right) \overset{(6.10)}{=} \mathrm{e}\left(g_1, g_1\right)^{\boldsymbol{D}^\top\cdot\boldsymbol{B}^\top\cdot\boldsymbol{A}\cdot\boldsymbol{C}} \overset{(6.9),(6.10)}{=} \mathrm{e}\left(\mathbf{G}_1, \mathbf{G}_2\right)^{\boldsymbol{D}^\top\boldsymbol{C}} \ .$$

Furthermore, let $\boldsymbol{H} \in \mathbb{Z}_p^{n\times n}$ be arbitrary, then it holds

$$\begin{aligned} \mathrm{e}\left(\mathbf{G}_1, {}^{\boldsymbol{H}^\top}\mathbf{G}_2\right) &\overset{(6.9)}{=} \mathrm{e}\left(g_1^{\boldsymbol{A}}, g_2^{\boldsymbol{H}^\top\cdot\boldsymbol{B}}\right) \overset{(6.10)}{=} \mathrm{e}\left(g_1, g_2\right)^{\boldsymbol{B}^\top\cdot\boldsymbol{H}\cdot\boldsymbol{A}} \\ &\overset{(6.10)}{=} \mathrm{e}\left(g_1^{\boldsymbol{H}\cdot\boldsymbol{A}}, g_2^{\boldsymbol{B}}\right) \overset{(6.9)}{=} \mathrm{e}\left({}^{\boldsymbol{H}}\mathbf{G}_1, \mathbf{G}_2\right) \ . \end{aligned} \tag{6.15}$$

Hence, we can also perform transformation with matrices in the exponents, but have to take care of these unusual rules. In particular, we explicitly notice that for $\boldsymbol{A} \in \mathbb{Z}_p^{n\times d_1}$, $\boldsymbol{B} \in \mathbb{Z}_p^{n\times d_2}$ it holds

$$\begin{aligned} \mathrm{e}\left(g_1^{\boldsymbol{A}}, g_2^{\boldsymbol{B}}\right) &= \mathrm{e}\left(g_1^{\boldsymbol{A}}, g_2^{\boldsymbol{B}\cdot\boldsymbol{I}_{d_2}}\right) \\ &\overset{(6.15)}{=} \mathrm{e}\left(g_1^{\boldsymbol{B}^\top\cdot\boldsymbol{A}}, g_2^{\boldsymbol{I}_{d_2}}\right) \\ &= \mathrm{e}\left(g_1^{\boldsymbol{I}_{d_1}}, g_2^{\boldsymbol{A}^\top\cdot\boldsymbol{B}}\right) \in {\mathbb{G}_\mathrm{T}}^{d_2\times d_1} \ . \end{aligned}$$

Finally, let $\mathbf{G}, \mathbf{H} \in \mathbb{G}_1^{n\times d_1}$ and $\mathbf{F} \in \mathbb{G}_2^{n\times d_2}$ be such that $\mathbf{G} = g_1^{\boldsymbol{G}}$, $\mathbf{H} = g_1^{\boldsymbol{H}}$ and $\mathbf{F} = g_2^{\boldsymbol{F}}$. Then, using previously defined computational rules we deduce that it holds

$$\begin{aligned} \mathrm{e}\left(\mathbf{G}\cdot\mathbf{H}, \mathbf{F}\right) &\overset{(6.5)}{=} \mathrm{e}\left(g_1^{\boldsymbol{G}+\boldsymbol{H}}, g_2^{\boldsymbol{F}}\right) \\ &\overset{(6.10)}{=} \mathrm{e}\left(g_1, g_2\right)^{\boldsymbol{F}^\top\cdot(\boldsymbol{G}+\boldsymbol{H})} \\ &\overset{(6.5)}{=} \mathrm{e}\left(g_1, g_2\right)^{\boldsymbol{F}^\top\cdot\boldsymbol{G}} \cdot \mathrm{e}\left(g_1, g_2\right)^{\boldsymbol{F}^\top\cdot\boldsymbol{H}} \\ &\overset{(6.10)}{=} \mathrm{e}\left(\mathbf{G}, \mathbf{F}\right) \cdot \mathrm{e}\left(\mathbf{H}, \mathbf{F}\right) \ . \end{aligned}$$

Analogously, for $\mathbf{U} \in \mathbb{G}_2^{n\times d_2}$ it holds $\mathrm{e}\left(\mathbf{G}, \mathbf{F}\cdot\mathbf{U}\right) = \mathrm{e}\left(\mathbf{G}, \mathbf{F}\right) \cdot \mathrm{e}\left(\mathbf{G}, \mathbf{U}\right)$.

6.1.2. General Lemmata

In this subsection we state lemmata which are quite specific for our framework. Our construction and the security proof are based on these lemmata. The first lemma is very basic but using this lemma we explain the form of group elements in the ciphertexts corresponding to set Γ_0 of the related ciphertext encoding. This lemma is also used in Lemma 6.13, which states the main properties of public parameters of our framework. Then we consider lemmata which are essential for our verification checks and the so-called parameter hiding lemma introduced in [Att16] and based on parameter hiding property of Dual System Groups [CW13]. We extended the parameter hiding lemma in order to explain our extensions of the ciphertext which lead to the CCA-security.

Lemma 6.1. *Let $(\mathbb{G}, \cdot)$ be a cyclic group of prime order p and $g \in \mathbb{G}$ be a generator. Furthermore, let $d \in \mathbb{N}$, $\mathbf{H} \in \mathbb{G}^{d+1}$ and $\boldsymbol{B} = (\boldsymbol{B}_d \ \boldsymbol{b}_{d+1}) \in \mathbb{GL}_{p,d+1}$ be arbitrary. Then, there exist unique $\boldsymbol{v} \in \mathbb{Z}_p^d$ and $\hat{v} \in \mathbb{Z}_p$ such that*

$$\mathbf{G}^{\boldsymbol{v}} \cdot \widehat{\mathbf{G}}^{\hat{v}} = \mathbf{H} \ ,$$

where $\mathbf{G} := g^{\boldsymbol{B}_d} \in \mathbb{G}^{(d+1)\times d}$ and $\widehat{\mathbf{G}} := g^{\boldsymbol{b}_{d+1}} \in \mathbb{G}^{d+1}$.

In particular, there exists unique $\boldsymbol{h} \in \mathbb{Z}_p^{d+1}$ such that $\mathbf{H} = g^{\boldsymbol{h}}$ and it holds $\left(\begin{smallmatrix}\boldsymbol{v}\\ \hat{v}\end{smallmatrix}\right) = \boldsymbol{B}^{-1}\cdot\boldsymbol{h}$.

Proof. First of all, g is a generator of $\mathbb{G}$, and hence for every $\mathbf{H} \in \mathbb{G}^{d+1}$ there exists unique $\boldsymbol{h} \in \mathbb{Z}_p^{d+1}$ such that $\mathbf{H} = g^{\boldsymbol{h}}$.

Existence: The $d+1$ column vectors of $\boldsymbol{B}$ form a basis of $\mathbb{Z}_p^{d+1}$, since $\boldsymbol{B} \in \mathbb{GL}_{p,d+1}$. Hence, there exist a unique vector $\boldsymbol{w} = (w_1, \ldots, w_{d+1})^\top \in \mathbb{Z}_p^{d+1}$ such that $\boldsymbol{B} \cdot \boldsymbol{w} = \boldsymbol{h}$, namely $\boldsymbol{w} = \boldsymbol{B}^{-1} \cdot \boldsymbol{h}$. By the definition of $\boldsymbol{B}_d$ and $\boldsymbol{b}_{d+1}$ it holds

$$\begin{aligned} \boldsymbol{h} &= \boldsymbol{B} \cdot \boldsymbol{w} \\ &= \boldsymbol{B}_d \cdot \begin{pmatrix} w_1 \\ \vdots \\ w_d \end{pmatrix} + \boldsymbol{b}_{d+1} \cdot w_{d+1} \quad . \end{aligned}$$

Hence, it holds

$$\begin{aligned} \mathbf{H} &= g^{\boldsymbol{h}} \\ &= g^{\boldsymbol{B}_d \cdot (w_1, \ldots, w_d)^\top + \boldsymbol{b}_{d+1} \cdot w_{d+1}} \\ &= \left(g^{\boldsymbol{B}_d}\right)^{(w_1, \ldots, w_d)^\top} \cdot \left(g^{\boldsymbol{b}_{d+1}}\right)^{w_{d+1}} \\ &= \mathbf{G}^{(w_1, \ldots, w_d)^\top} \cdot \widehat{\mathbf{G}}^{w_{d+1}} \quad , \end{aligned}$$

and the equation $\mathbf{G}^{\boldsymbol{v}} \cdot \widehat{\mathbf{G}}^{\hat{v}} = \mathbf{H}$ holds for $\boldsymbol{v} = (w_1, \ldots, w_d)^\top$ and $\hat{v} = w_{d+1}$. In particular, it holds $\left(\begin{smallmatrix} \boldsymbol{v} \\ \hat{v} \end{smallmatrix}\right) = \boldsymbol{w} = \boldsymbol{B}^{-1} \cdot \boldsymbol{h}$.

Uniqueness: Assume that there exist $\boldsymbol{v}, \boldsymbol{u} \in \mathbb{Z}_p^d$ and $\hat{v}, \hat{u} \in \mathbb{Z}_p$ such that

$$\mathbf{G}^{\boldsymbol{v}} \cdot \widehat{\mathbf{G}}^{\hat{v}} = \mathbf{H} = \mathbf{G}^{\boldsymbol{u}} \cdot \widehat{\mathbf{G}}^{\hat{u}} \quad .$$

Then, it holds

$$\begin{aligned} g^{\boldsymbol{B}_d \cdot \boldsymbol{v}} \cdot g^{\boldsymbol{b}_{d+1} \cdot \hat{v}} = g^{\boldsymbol{B}_d \cdot \boldsymbol{u}} \cdot g^{\boldsymbol{b}_{d+1} \cdot \hat{u}} \Leftrightarrow \\ \boldsymbol{B}_d \cdot \boldsymbol{v} + \boldsymbol{b}_{d+1} \cdot \hat{v} = \boldsymbol{B}_d \cdot \boldsymbol{u} + \boldsymbol{b}_{d+1} \cdot \hat{u} \Leftrightarrow \\ \boldsymbol{B} \cdot \begin{pmatrix} \boldsymbol{v} \\ \hat{v} \end{pmatrix} = \boldsymbol{B} \cdot \begin{pmatrix} \boldsymbol{u} \\ \hat{u} \end{pmatrix} \Leftrightarrow \\ \begin{pmatrix} \boldsymbol{v} - \boldsymbol{u} \\ \hat{v} - \hat{u} \end{pmatrix} \in \ker(\boldsymbol{B}) \end{aligned}$$

We deduce that it holds $\boldsymbol{v} = \boldsymbol{u}$ and $\hat{v} = \hat{u}$, since $\boldsymbol{B} \in \mathbb{GD}_{p,d+1}$. □

Notice that ciphertext elements $\mathbf{C}_\tau$ corresponding to index $\tau \in \Gamma_0$ of the related ciphertext encoding will have exactly the form of $\mathbf{H}$ from Lemma 6.1, let say $\mathbf{C}_\tau = \mathbf{G}^{\boldsymbol{s}} \cdot \widehat{\mathbf{G}}^{\hat{s}}$. Recall that by definition of Γ_0 every $\tau \in \Gamma_0$ is related to an index $i \in I$. Elements $\boldsymbol{s}$ and $\hat{s}$ will be the normal and the semi-functional random elements corresponding to the ciphertext-specific variable X_{s_i}, respectively. That is, elements $\{\mathbf{C}_\tau\}_{\tau \in \Gamma_0}$ uniquely define normal $\{\boldsymbol{s}_i\}_{i \in I}$ and semi-functional $\{\hat{s}_i\}_{i \in I}$ random elements of the ciphertext corresponding to set I.

Lemmata for Verification Checks

The lemmata from this subsection are used for verification checks. Lemma 6.2 allows us to verify to a certain extent the form of group elements in the ciphertext corresponding to set Γ. In turn, the algorithm from Lemma 6.4 is used to achieve similar guarantees for group elements corresponding to set $\overline{\Gamma}$. The guarantees for our verification checks are formally stated in Subsection 6.2.4.

Lemma 6.2. *Let* $\mathbb{GD} = (p, (g_1, \mathbb{G}_1), (g_2, \mathbb{G}_2), \mathbb{G}_T, e)$ *be bilinear groups of prime order* p*. Furthermore, let* $l, d \in \mathbb{N}$*,* $l \leq d$ *and* $\boldsymbol{B} \in \mathbb{Z}_p^{d \times l}$ *be a matrix with rank* l*. Suppose* $\boldsymbol{a}, \boldsymbol{c} \in \mathbb{Z}_p^d$ *satisfy*

$$e\left(g_1^{\boldsymbol{a}}, g_2^{\boldsymbol{B}}\right) = e\left(g_1^{\boldsymbol{c}}, g_2^{\boldsymbol{B}}\right) \in {\mathbb{G}_T}^l \ .$$

Then, it holds $\boldsymbol{a} - \boldsymbol{c} \in \ker\left(\boldsymbol{B}^\top\right)$*.*

Proof. We prove the statement directly

$$\begin{aligned} e\left(g_1^{\boldsymbol{a}}, g_2^{\boldsymbol{B}}\right) &= e\left(g_1^{\boldsymbol{c}}, g_2^{\boldsymbol{B}}\right) \Leftrightarrow \\ e\left(g_1, g_2\right)^{\boldsymbol{B}^\top \cdot \boldsymbol{a}} &= e\left(g_1, g_2\right)^{\boldsymbol{B}^\top \cdot \boldsymbol{c}} \Leftrightarrow \\ \boldsymbol{B}^\top \cdot \boldsymbol{a} &= \boldsymbol{B}^\top \cdot \boldsymbol{c} \Leftrightarrow \\ \boldsymbol{a} - \boldsymbol{c} &\in \ker\left(\boldsymbol{B}^\top\right) \ , \end{aligned}$$

where in the second step we used the fact that $e\left(g_1, g_2\right)$ is a generator of $\mathbb{G}_T$. □

The statement in the following corollary is a generalization of the statement in Lemma 6.2.

Corollary 6.3. Let $\mathbb{GD} = (p, (g_1, \mathbb{G}_1), (g_2, \mathbb{G}_2), \mathbb{G}_T, e)$ be bilinear groups of prime order p. Furthermore, let $l, l', d \in \mathbb{N}$, $l \leq d$ and $\boldsymbol{B} \in \mathbb{Z}_p^{d \times l}$ be a matrix with rank l. Suppose $\boldsymbol{A} = (\boldsymbol{a}_1, \ldots, \boldsymbol{a}_{l'})$, $\boldsymbol{C} = (\boldsymbol{c}_1, \ldots, \boldsymbol{c}_{l'}) \in \mathbb{Z}_p^{d \times l'}$ satisfy

$$e\left(g_1^{\boldsymbol{A}}, g_2^{\boldsymbol{B}}\right) = e\left(g_1^{\boldsymbol{C}}, g_2^{\boldsymbol{B}}\right) \in {\mathbb{G}_T}^{l \times l'} \ .$$

Then, for every $i \in [l']$ it holds $\boldsymbol{a}_i - \boldsymbol{c}_i \in \ker\left(\boldsymbol{B}^\top\right)$.

Recall that encoding polynomials c_τ, $\tau \in \Gamma$ contain only ciphertext-specific variables corresponding to set I. Using Corollary 6.3 we construct appropriate verification checks for ciphertext elements $\mathbf{C}_\tau$ with $\tau \in \Gamma$. Namely, we check if the randomness of these elements correlates with the randomness of group elements $\{\mathbf{C}_\tau\}_{\tau \in \Gamma_0}$. However, the guarantees of Corollary 6.3 are not perfect since we cannot check the elements for equality. We show how to deal with it in Subsection 6.2.4.

Consistency checks for ciphertext elements corresponding to encoding polynomials c_τ, $\tau \in \overline{\Gamma}$ are more complicated, since we have to check if the random elements corresponding to set $\overline{I}$ are the same in all $\{\mathbf{C}_\tau\}_{\tau \in \overline{\Gamma}}$. The algorithm in Lemma 6.4 is used to construct an appropriate consistency check for these elements. We refer to Subsection 6.2.4 for details.

Lemma 6.4. *Let* $(\mathbb{G}, \cdot)$ *be a group of prime order* p*, and* $g \in \mathbb{G}$ *be a generator. Furthermore, let* $d, r, t \in \mathbb{N}$*,* $\boldsymbol{M} \in \mathbb{Z}_p^{r \times t}$*, and* $\mathbf{Y} \in \mathbb{G}^{d \times t}$ *be arbitrary. Then, there is a deterministic polynomial time algorithm* ExtGauss*, which given the group description* $\mathbb{GD} = (p, (g, \mathbb{G}))$*,* $\mathbf{Y}$*, and* $\boldsymbol{M}$ *outputs 1 if and only if there exist* $\boldsymbol{S} \in \mathbb{Z}_p^{d \times r}$ *such that*

$$\mathbf{Y} = g^{\boldsymbol{S} \cdot \boldsymbol{M}} \ .$$

Proof. We prove that the following algorithm ExtGauss satisfies the requirements of the lemma.

Algorithm 15: ExtGauss - adapted Gaussian algorithm

Input : $(\mathbb{GD}, \mathbf{Y}, \boldsymbol{M})$
Require : $\mathbb{GD} = (p, (g, \mathbb{G}))$, $\mathbf{Y} \in \mathbb{G}^{d \times t}$, $\boldsymbol{M} \in \mathbb{Z}_p^{r \times t}$

1 Compute using Gaussian elimination algorithm (over $\mathbb{Z}_p$) on input $\boldsymbol{M}^\top$ an invertible matrix $\boldsymbol{T} \in \mathbb{GL}_{p,t} \subset \mathbb{Z}_p^{t \times t}$ such that $\boldsymbol{T} \cdot \boldsymbol{M}^\top$ is in the reduced row echelon form. Let $J \subseteq [t]$ be the index set of all zero rows of $\boldsymbol{T} \cdot \boldsymbol{M}^\top \in \mathbb{Z}_p^{t \times r}$.

2 Compute $\mathbf{L} = \left(l_{i,j}\right)_{d,t} := \mathbf{Y}^{\boldsymbol{T}^\top} \in \mathbb{G}^{d \times t}$ (or rather the j's columns of $\mathbf{L}$ for all $j \in J$).

Output : Output 0 if there exist $i \in [d]$ and $j \in J$ such that $l_{i,j} \neq 1_{\mathbb{G}}$. Otherwise output 1.

Notice that ExtGauss has to compute only the j'th columns of $\mathbf{L}$ in the second step, since only these elements are relevant for the output.

Group element $g \in \mathbb{G}$ is a generator and $\mathbf{Y} \in \mathbb{G}^{d \times t}$. Hence, there exist unique matrix $\boldsymbol{Y} \in \mathbb{Z}_p^{d \times t}$ such that $\mathbf{Y} = g^{\boldsymbol{Y}}$. We deduce that it holds

$$\begin{aligned}
\exists_{\boldsymbol{S} \in \mathbb{Z}_p^{d \times r}} : \mathbf{Y} = g^{\boldsymbol{S} \cdot \boldsymbol{M}} \Leftrightarrow \\
\exists_{\boldsymbol{S} \in \mathbb{Z}_p^{d \times r}} : g^{\boldsymbol{Y}} = g^{\boldsymbol{S} \cdot \boldsymbol{M}} \Leftrightarrow \\
\exists_{\boldsymbol{S} \in \mathbb{Z}_p^{d \times r}} : \boldsymbol{Y} = \boldsymbol{S} \cdot \boldsymbol{M} \Leftrightarrow \\
\exists_{\boldsymbol{S} \in \mathbb{Z}_p^{d \times r}} : \boldsymbol{M}^\top \cdot \boldsymbol{S}^\top = \boldsymbol{Y}^\top \Leftrightarrow \\
\exists_{\boldsymbol{X} \in \mathbb{Z}_p^{r \times d}} : \boldsymbol{M}^\top \cdot \boldsymbol{X} = \boldsymbol{Y}^\top \ .
\end{aligned}$$

In the first step, using the Gaussian elimination algorithm on $\boldsymbol{M}^\top$, ExtGauss derives an invertible matrix $\boldsymbol{T} \in \mathbb{GL}_{p,t}$ such that $\boldsymbol{T} \cdot \boldsymbol{M}^\top$ is in reduced row echelon form. Since $\boldsymbol{T}$ is invertible, it holds

$$\begin{aligned}
\exists_{\boldsymbol{X} \in \mathbb{Z}_p^{r \times d}} : \boldsymbol{M}^\top \cdot \boldsymbol{X} = \boldsymbol{Y}^\top \Leftrightarrow \\
\exists_{\boldsymbol{X} \in \mathbb{Z}_p^{r \times d}} : \boldsymbol{T} \cdot \boldsymbol{M}^\top \cdot \boldsymbol{X} = \boldsymbol{T} \cdot \boldsymbol{Y}^\top \ .
\end{aligned}$$

Let $J \subseteq [t]$ be the index set as defined in the algorithm. Since $\boldsymbol{T} \cdot \boldsymbol{M}^\top$ is in the reduced row echelon form, for every $\boldsymbol{X} \in \mathbb{Z}_p^{r \times d}$ and every $j \in J$ the j'th row of $\boldsymbol{T} \cdot \boldsymbol{M}^\top \cdot \boldsymbol{X}$ is equal to $\mathbf{0}^\top$. Furthermore, the rows of $\boldsymbol{T} \cdot \boldsymbol{M}^\top$ with indices $[t] \setminus J$ are linearly independent, and hence the rank of the matrix is $t - |J|$. Hence, there exists $\boldsymbol{X} \in \mathbb{Z}_p^{r \times d}$ such that $\boldsymbol{T} \cdot \boldsymbol{M}^\top \cdot \boldsymbol{X} = \boldsymbol{T} \cdot \boldsymbol{Y}^\top$ if and only if for every $j \in J$ the j'th row of $\boldsymbol{T} \cdot \boldsymbol{Y}^\top$ is equal to $\mathbf{0}^\top$. Formally, it holds:

$$\begin{aligned}
\exists_{\boldsymbol{X} \in \mathbb{Z}_p^{r \times d}} : \boldsymbol{T} \cdot \boldsymbol{M}^\top \cdot \boldsymbol{X} = \boldsymbol{T} \cdot \boldsymbol{Y}^\top \Leftrightarrow \\
\forall_{j \in J} : \boldsymbol{e}_j^\top \cdot \boldsymbol{T} \cdot \boldsymbol{Y}^\top = \mathbf{0}^\top \in \mathbb{Z}_p^{1 \times t} \ .
\end{aligned}$$

We deduce that an appropriate matrix $\boldsymbol{S}$ exists if and only if for every $j \in J$ the j'th row of $\boldsymbol{T} \cdot \boldsymbol{Y}^\top$ is equal to $\mathbf{0}^\top$. Furthermore, it holds

$$\begin{aligned}
\forall_{j \in J} : \boldsymbol{e}_j^\top \cdot \boldsymbol{T} \cdot \boldsymbol{Y}^\top = \mathbf{0}^\top \in \mathbb{Z}_p^{1 \times t} \Leftrightarrow \\
\forall_{j \in J} : \boldsymbol{Y} \cdot \boldsymbol{T}^\top \cdot \boldsymbol{e}_j = \mathbf{0} \in \mathbb{Z}_p^t \Leftrightarrow \\
\forall_{j \in J} : g^{\boldsymbol{Y} \cdot \boldsymbol{T}^\top \cdot \boldsymbol{e}_j} = \mathbf{1}_{\mathbb{G}} \in \mathbb{G}^t \ .
\end{aligned}$$

That is, an appropriate matrix $\boldsymbol{S}$ exists if and only if for every $j \in J$ the j'th *column* of $\boldsymbol{Y} \cdot \boldsymbol{T}^\top$ is equal to $\mathbf{0}$, respectively for every $j \in J$ the vector $g^{\boldsymbol{Y} \cdot \boldsymbol{T}^\top \cdot \boldsymbol{e}_j} \in \mathbb{G}^t$ contains only neutral elements of $\mathbb{G}$.

In the second step algorithm ExtGauss computes

$$\begin{aligned}\mathbf{L} &= \mathbf{Y}^{\boldsymbol{T}^\top} \\ &= g^{\boldsymbol{Y}\cdot\boldsymbol{T}^\top} \in \mathbb{G}^{d\times t}\end{aligned}$$

and outputs 1 if and only if for every $j \in J$ the column j of $\mathbf{L}$ contains only neutral elements of $\mathbb{G}$ – that is, if and only if there exists an appropriate matrix $\boldsymbol{S} \in \mathbb{Z}_p^{d\times r}$ (such that $\mathbf{Y} = g^{\boldsymbol{S}\cdot\boldsymbol{M}}$). □

Parameter Hiding

The following lemma was presented in [Att16] and is used to argue that normal public parameter information-theoretically hide some chosen random elements (corresponding to the semi-functional parameter). For our framework we have to extend this lemma. All matrices considered in the following lemmata have corresponding elements in the public parameters of the scheme. For convenience we use the corresponding names already here.

Lemma 6.5 (Parameter Hiding Lemma (cf. Lemma 1 in [Att16]))**.** *Let d be an integer and p be a prime number. Let $\boldsymbol{B} \in \mathbb{GL}_{p,d+1}$ and $\tilde{\boldsymbol{D}} \in \mathbb{GL}_{p,d}$ be arbitrary, $\boldsymbol{D} := \left(\begin{smallmatrix}\tilde{\boldsymbol{D}} & \mathbf{0}\\ \mathbf{0} & 1\end{smallmatrix}\right)$, $\boldsymbol{Z} := \boldsymbol{B}^{-\top}\cdot\boldsymbol{D}$. Furthermore, let $\boldsymbol{H} \in \mathbb{Z}_p^{(d+1)\times(d+1)}$ be arbitrary and $\boldsymbol{M} = \left(\begin{smallmatrix}\boldsymbol{M}_1 & \boldsymbol{m}_2\\ \boldsymbol{m}_3^\top & \delta\end{smallmatrix}\right) := \boldsymbol{B}^{-1}\cdot\boldsymbol{H}\cdot\boldsymbol{B}$. Then, given $\boldsymbol{B}$, $\boldsymbol{Z}$ as well as $\boldsymbol{H}\cdot\boldsymbol{B}\cdot\left(\begin{smallmatrix}\boldsymbol{I}_d\\ \mathbf{0}\end{smallmatrix}\right)$ and $\boldsymbol{H}^\top\cdot\boldsymbol{Z}\cdot\left(\begin{smallmatrix}\boldsymbol{I}_d\\ \mathbf{0}\end{smallmatrix}\right)$ the quantity of the entry δ is information-theoretically hidden.*

Proof. Let us denote the given tuple of matrices by $D = (D_1, D_2, D_3, D_4)$. Hence, $D_1, D_2 \in \mathbb{GL}_{p,d+1}$ and $D_3, D_4 \in \mathbb{Z}_p^{(d+1)\times d}$. By construction D_1 and D_2 directly determine $\boldsymbol{B}$ and $\boldsymbol{Z}$. Furthermore, since these matrices are invertible, $\tilde{\boldsymbol{D}}$ is also uniquely defined and can be reconstructed.

Using the fact that $\boldsymbol{B} \in \mathbb{GL}_{p,d+1}$ we deduce from the definition of $\boldsymbol{M}$:

$$\boldsymbol{H}\cdot\boldsymbol{B} = \boldsymbol{B}\cdot\boldsymbol{M}\ , \tag{6.16}$$

$$\boldsymbol{B}^{-1}\cdot\boldsymbol{H} = \boldsymbol{M}\cdot\boldsymbol{B}^{-1}\ . \tag{6.17}$$

Next, let us explain D_3 in terms of $\boldsymbol{M}$:

$$\begin{aligned}\boldsymbol{D}_3 = \boldsymbol{H}\cdot\boldsymbol{B}\cdot\begin{pmatrix}\boldsymbol{I}_d\\ \mathbf{0}\end{pmatrix} &\overset{(6.16)}{=} \boldsymbol{B}\cdot\boldsymbol{M}\cdot\begin{pmatrix}\boldsymbol{I}_d\\ \mathbf{0}\end{pmatrix}\\ &= \boldsymbol{B}\cdot\begin{pmatrix}\boldsymbol{M}_1 & \boldsymbol{m}_2\\ \boldsymbol{m}_3^\top & \delta\end{pmatrix}\cdot\begin{pmatrix}\boldsymbol{I}_d\\ \mathbf{0}\end{pmatrix}\\ &= \boldsymbol{B}\cdot\begin{pmatrix}\boldsymbol{M}_1\\ \boldsymbol{m}_3^\top\end{pmatrix}\ .\end{aligned} \tag{6.18}$$

We can analogously explain D_4 in terms of $\boldsymbol{M}$:

$$\begin{aligned}\boldsymbol{D}_4 = \boldsymbol{H}^\top\cdot\boldsymbol{Z}\cdot\begin{pmatrix}\boldsymbol{I}_d\\ \mathbf{0}\end{pmatrix} &= \boldsymbol{H}^\top\cdot\boldsymbol{B}^{-\top}\cdot\boldsymbol{D}\cdot\begin{pmatrix}\boldsymbol{I}_d\\ \mathbf{0}\end{pmatrix}\\ &= \left(\boldsymbol{B}^{-1}\cdot\boldsymbol{H}\right)^\top\cdot\begin{pmatrix}\tilde{\boldsymbol{D}} & \mathbf{0}\\ \mathbf{0} & 1\end{pmatrix}\cdot\begin{pmatrix}\boldsymbol{I}_d\\ \mathbf{0}\end{pmatrix}\\ &\overset{(6.17)}{=} \left(\boldsymbol{M}\cdot\boldsymbol{B}^{-1}\right)^\top\cdot\begin{pmatrix}\tilde{\boldsymbol{D}}\\ \mathbf{0}\end{pmatrix}\end{aligned}$$

$$= \quad \boldsymbol{B}^{-\top} \cdot \begin{pmatrix} \boldsymbol{M}_1^\top & \boldsymbol{m}_3 \\ \boldsymbol{m}_2^\top & \delta \end{pmatrix} \cdot \begin{pmatrix} \tilde{\boldsymbol{D}} \\ \boldsymbol{0} \end{pmatrix}$$
$$= \quad \boldsymbol{B}^{-\top} \cdot \begin{pmatrix} \boldsymbol{M}_1^\top \\ \boldsymbol{m}_2^\top \end{pmatrix} \cdot \tilde{\boldsymbol{D}} \ . \tag{6.19}$$

We conclude that in addition to D_1 and D_2 matrices D_3 and D_4 determine $\boldsymbol{M}_1$, $\boldsymbol{m}_2$, and $\boldsymbol{m}_3$, which can be reconstructed from (6.18) and (6.19). The value δ of $\boldsymbol{M}$ remains information-theoretically hidden. Namely, for every $\sigma \in \mathbb{Z}_p$ there exists unique $\boldsymbol{H}$ that satisfies $\boldsymbol{H} = \boldsymbol{B} \cdot \left(\begin{smallmatrix} \boldsymbol{M}'_1 & \boldsymbol{m}'_2 \\ \boldsymbol{m}'^\top_3 & \sigma \end{smallmatrix}\right) \cdot \boldsymbol{B}^{-1}$, where $\boldsymbol{M}'_1$, $\boldsymbol{m}'_2$, and $\boldsymbol{m}'^\top_3$ are the determined entries of $\boldsymbol{M}$. □

The following lemma is used to argue that the normal public parameter corresponding to $\boldsymbol{H} \in \mathbb{Z}_p^{(d+1)\times(d+1)}$ remain the same if $\boldsymbol{H}$ is changed to $\boldsymbol{H} + \boldsymbol{B} \cdot (\begin{smallmatrix} \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \sigma \end{smallmatrix}) \cdot \boldsymbol{B}^{-1}$. That is, if we fix the normal parameter, the value σ is hidden and can be chosen later. This lemma is a direct consequence of the previous Parameter Hiding Lemma.

Lemma 6.6. *Let $d \in \mathbb{N}$ be an integer and p be a prime number. Let $\boldsymbol{B} \in \mathbb{GL}_{p,d+1}$ and $\tilde{\boldsymbol{D}} \in \mathbb{GL}_{p,d}$ be arbitrary, $\boldsymbol{D} := \left(\begin{smallmatrix} \tilde{\boldsymbol{D}} & \boldsymbol{0} \\ \boldsymbol{0} & 1 \end{smallmatrix}\right)$, $\boldsymbol{Z} := \boldsymbol{B}^{-\top} \cdot \boldsymbol{D}$. Furthermore, let $\boldsymbol{H} \in \mathbb{Z}_p^{(d+1)\times(d+1)}$ and $\sigma \in \mathbb{Z}_p$ be arbitrary, $\boldsymbol{H}' := \boldsymbol{H} + \boldsymbol{B} \cdot (\begin{smallmatrix} \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \sigma \end{smallmatrix}) \cdot \boldsymbol{B}^{-1}$. Then, it holds $\boldsymbol{H} \cdot \boldsymbol{B} \cdot \left(\begin{smallmatrix} \boldsymbol{I}_d \\ \boldsymbol{0} \end{smallmatrix}\right) = \boldsymbol{H}' \cdot \boldsymbol{B} \cdot \left(\begin{smallmatrix} \boldsymbol{I}_d \\ \boldsymbol{0} \end{smallmatrix}\right)$ and $\boldsymbol{H}^\top \cdot \boldsymbol{Z} \cdot \left(\begin{smallmatrix} \boldsymbol{I}_d \\ \boldsymbol{0} \end{smallmatrix}\right) = \boldsymbol{H}'^\top \cdot \boldsymbol{Z} \cdot \left(\begin{smallmatrix} \boldsymbol{I}_d \\ \boldsymbol{0} \end{smallmatrix}\right)$.*

Proof. We present a direct proof of both statements:

$$\begin{aligned} \boldsymbol{H}' \cdot \boldsymbol{B} \cdot \left(\begin{smallmatrix} \boldsymbol{I}_d \\ \boldsymbol{0} \end{smallmatrix}\right) &= \boldsymbol{H} \cdot \boldsymbol{B} \cdot \left(\begin{smallmatrix} \boldsymbol{I}_d \\ \boldsymbol{0} \end{smallmatrix}\right) + \boldsymbol{B} \cdot (\begin{smallmatrix} \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \sigma \end{smallmatrix}) \cdot \boldsymbol{B}^{-1} \cdot \boldsymbol{B} \cdot \left(\begin{smallmatrix} \boldsymbol{I}_d \\ \boldsymbol{0} \end{smallmatrix}\right) \\ &= \boldsymbol{H} \cdot \boldsymbol{B} \cdot \left(\begin{smallmatrix} \boldsymbol{I}_d \\ \boldsymbol{0} \end{smallmatrix}\right) \ , \\ \boldsymbol{H}'^\top \cdot \boldsymbol{Z} \cdot \left(\begin{smallmatrix} \boldsymbol{I}_d \\ \boldsymbol{0} \end{smallmatrix}\right) &= \boldsymbol{H}^\top \cdot \boldsymbol{Z} \cdot \left(\begin{smallmatrix} \boldsymbol{I}_d \\ \boldsymbol{0} \end{smallmatrix}\right) + \boldsymbol{B}^{-\top} \cdot (\begin{smallmatrix} \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \sigma \end{smallmatrix}) \cdot \boldsymbol{B}^\top \cdot \boldsymbol{B}^{-\top} \cdot \left(\begin{smallmatrix} \tilde{\boldsymbol{D}} & \boldsymbol{0} \\ \boldsymbol{0} & 1 \end{smallmatrix}\right) \cdot \left(\begin{smallmatrix} \boldsymbol{I}_d \\ \boldsymbol{0} \end{smallmatrix}\right) \\ &= \boldsymbol{H}^\top \cdot \boldsymbol{Z} \cdot \left(\begin{smallmatrix} \boldsymbol{I}_d \\ \boldsymbol{0} \end{smallmatrix}\right) \ . \end{aligned}$$

This finalizes the proof. □

The following lemma is used to argue about the distribution of semi-functional parameters corresponding to $\boldsymbol{H}$ when the matrix is changed to $\boldsymbol{H} + \boldsymbol{B} \cdot (\begin{smallmatrix} \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \sigma \end{smallmatrix}) \cdot \boldsymbol{B}^{-1}$ (cf. Lemma 7.14).

Lemma 6.7 (Part of Lemma 6 in [Att15])**.** *Let $d \in \mathbb{N}$ be an integer and $\mathbb{GD} = (p, (g_1, \mathbb{G}_1), (g_2, \mathbb{G}_2), \mathbb{G}_\mathrm{T}, \mathrm{e})$ be bilinear groups of prime order p, $\boldsymbol{B} \in \mathbb{GL}_{p,d+1}$, and $\tilde{\boldsymbol{D}} \in \mathbb{GL}_{p,d}$ be arbitrary. Furthermore, let $\boldsymbol{D} := \left(\begin{smallmatrix} \tilde{\boldsymbol{D}} & \boldsymbol{0} \\ \boldsymbol{0} & 1 \end{smallmatrix}\right)$, $\boldsymbol{Z} := \boldsymbol{B}^{-\top} \cdot \boldsymbol{D}$, and $D := \left(\mathbb{GD}, g_1^{\boldsymbol{B}}, g_2^{\boldsymbol{Z}}\right)$. Consider a probability experiment defined by $\boldsymbol{H} \leftarrow \mathbb{Z}_p^{(d+1)\times(d+1)}$ and $\sigma \leftarrow \mathbb{Z}_p$. Let $\boldsymbol{H}' := \boldsymbol{H} + \boldsymbol{B} \cdot (\begin{smallmatrix} \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \sigma \end{smallmatrix}) \cdot \boldsymbol{B}^{-1}$. Then, the distributions of*

$$\left(D, g_1^{\boldsymbol{H}\cdot\boldsymbol{B}\cdot\left(\begin{smallmatrix} \boldsymbol{I}_d \\ \boldsymbol{0} \end{smallmatrix}\right)}, g_1^{\boldsymbol{H}\cdot\boldsymbol{B}\cdot\left(\begin{smallmatrix} \boldsymbol{0} \\ 1 \end{smallmatrix}\right)}, g_2^{\boldsymbol{H}^\top\cdot\boldsymbol{Z}\cdot\left(\begin{smallmatrix} \boldsymbol{I}_d \\ \boldsymbol{0} \end{smallmatrix}\right)}, g_2^{\boldsymbol{H}^\top\cdot\boldsymbol{Z}\cdot\left(\begin{smallmatrix} \boldsymbol{0} \\ 1 \end{smallmatrix}\right)}\right)$$

and

$$\left(D, g_1^{\boldsymbol{H}'\cdot\boldsymbol{B}\cdot\left(\begin{smallmatrix} \boldsymbol{I}_d \\ \boldsymbol{0} \end{smallmatrix}\right)}, g_1^{\boldsymbol{H}'\cdot\boldsymbol{B}\cdot\left(\begin{smallmatrix} \boldsymbol{0} \\ 1 \end{smallmatrix}\right)}, g_2^{\boldsymbol{H}'^\top\cdot\boldsymbol{Z}\cdot\left(\begin{smallmatrix} \boldsymbol{I}_d \\ \boldsymbol{0} \end{smallmatrix}\right)}, g_2^{\boldsymbol{H}'^\top\cdot\boldsymbol{Z}\cdot\left(\begin{smallmatrix} \boldsymbol{0} \\ 1 \end{smallmatrix}\right)}\right)$$

are equal and it holds

$$g_1^{\boldsymbol{H}'\cdot\boldsymbol{B}\cdot\binom{\mathbf{0}}{1}} = g_1^{\boldsymbol{H}\cdot\boldsymbol{B}\cdot\binom{\mathbf{0}}{1}} \cdot \left(g_1^{\boldsymbol{B}\cdot\binom{\mathbf{0}}{1}}\right)^{\sigma} \quad ,$$
$$g_2^{\boldsymbol{H}'^{\top}\cdot\boldsymbol{Z}\cdot\binom{\mathbf{0}}{1}} = g_2^{\boldsymbol{H}^{\top}\cdot\boldsymbol{Z}\cdot\binom{\mathbf{0}}{1}} \cdot \left(g_2^{\boldsymbol{B}^{-\top}\cdot\binom{\mathbf{0}}{1}}\right)^{\sigma} \quad .$$

Proof. Matrix $\boldsymbol{B}$ is invertible, and hence the random choice of $\boldsymbol{H}$ can be seen as the random choice of $\boldsymbol{M} = \left(\begin{smallmatrix} \boldsymbol{M}_1 & \boldsymbol{m}_2 \\ \boldsymbol{m}_3^{\top} & \delta \end{smallmatrix}\right) \leftarrow \mathbb{Z}_p^{(d+1)\times(d+1)}$, $\boldsymbol{H} := \boldsymbol{B}\cdot\boldsymbol{M}\cdot\boldsymbol{B}^{-1}$. However, by definition of $\boldsymbol{H}'$ it holds

$$\begin{aligned}\boldsymbol{B}^{-1}\cdot\boldsymbol{H}'\cdot\boldsymbol{B} &= \left(\begin{smallmatrix} \boldsymbol{M}_1 & \boldsymbol{m}_2 \\ \boldsymbol{m}_3^{\top} & \delta \end{smallmatrix}\right) + \left(\begin{smallmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \sigma \end{smallmatrix}\right) \\ &= \left(\begin{smallmatrix} \boldsymbol{M}_1 & \boldsymbol{m}_2 \\ \boldsymbol{m}_3^{\top} & \delta+\sigma \end{smallmatrix}\right) \quad .\end{aligned}$$

Hence, the distributions of $\boldsymbol{H}$ and $\boldsymbol{H}'$ are equal and we deduce that both distributions defined in the lemma are equal.

Next, let us prove the last statement in the lemma. Due to the prime order of the groups it is sufficient to consider the exponents. It holds

$$\begin{aligned}\boldsymbol{H}'\cdot\boldsymbol{B}\cdot\begin{pmatrix}\mathbf{0}\\1\end{pmatrix} &= \left(\boldsymbol{H} + \boldsymbol{B}\cdot\begin{pmatrix}\mathbf{0} & \mathbf{0}\\ \mathbf{0} & \sigma\end{pmatrix}\cdot\boldsymbol{B}^{-1}\right)\cdot\boldsymbol{B}\cdot\begin{pmatrix}\mathbf{0}\\1\end{pmatrix} \\ &= \boldsymbol{H}\cdot\boldsymbol{B}\cdot\begin{pmatrix}\mathbf{0}\\1\end{pmatrix} + \sigma\cdot\boldsymbol{B}\cdot\begin{pmatrix}\mathbf{0}\\1\end{pmatrix} \quad ,\end{aligned}$$

and

$$\begin{aligned}\boldsymbol{H}'^{\top}\cdot\boldsymbol{Z}\cdot\begin{pmatrix}\mathbf{0}\\1\end{pmatrix} &= \boldsymbol{H}^{\top}\cdot\boldsymbol{Z}\cdot\begin{pmatrix}\mathbf{0}\\1\end{pmatrix} + \left(\boldsymbol{B}\cdot\begin{pmatrix}\mathbf{0} & \mathbf{0}\\ \mathbf{0} & \sigma\end{pmatrix}\cdot\boldsymbol{B}^{-1}\right)^{\top}\cdot\boldsymbol{B}^{-\top}\cdot\boldsymbol{D}\cdot\begin{pmatrix}\mathbf{0}\\1\end{pmatrix} \\ &= \boldsymbol{H}^{\top}\cdot\boldsymbol{Z}\cdot\begin{pmatrix}\mathbf{0}\\1\end{pmatrix} + \boldsymbol{B}^{-\top}\cdot\begin{pmatrix}\mathbf{0} & \mathbf{0}\\ \mathbf{0} & \sigma\end{pmatrix}\cdot\begin{pmatrix}\tilde{\boldsymbol{D}} & \mathbf{0}\\ \mathbf{0} & 1\end{pmatrix}\cdot\begin{pmatrix}\mathbf{0}\\1\end{pmatrix} \\ &= \boldsymbol{H}^{\top}\cdot\boldsymbol{Z}\cdot\begin{pmatrix}\mathbf{0}\\1\end{pmatrix} + \sigma\cdot\boldsymbol{B}^{-\top}\cdot\begin{pmatrix}\mathbf{0}\\1\end{pmatrix} \quad .\end{aligned}$$

This finally proves the lemma. □

The following lemma is essential for our CCA-secure framework and is based on the parameter hiding lemma.

Lemma 6.8 (Parameter hiding extended)**.** *Let $d \in \mathbb{N}$ be an integer and p be a prime number. Let $\boldsymbol{B} = (\boldsymbol{B}_d\ \boldsymbol{b}_{d+1}) \in \mathbb{GL}_{p,d+1}$ and $\tilde{\boldsymbol{D}} \in \mathbb{GL}_{p,d}$ be arbitrary, $\boldsymbol{D} := \left(\begin{smallmatrix}\tilde{\boldsymbol{D}} & \mathbf{0}\\ \mathbf{0} & 1\end{smallmatrix}\right)$, $\boldsymbol{Z} := \boldsymbol{B}^{-\top}\cdot\boldsymbol{D}$. Furthermore, let $t, t^* \in \mathbb{Z}_p$, $t \neq t^*$. Consider a probability experiment defined by $\boldsymbol{U}, \boldsymbol{V} \leftarrow \mathbb{Z}_p^{(d+1)\times(d+1)}$. Then, given $\boldsymbol{B}$, $\boldsymbol{Z}$, $\boldsymbol{U}\cdot\boldsymbol{B}\cdot\left(\begin{smallmatrix}\boldsymbol{I}_d\\ \mathbf{0}\end{smallmatrix}\right)$, $\boldsymbol{V}\cdot\boldsymbol{B}\cdot\left(\begin{smallmatrix}\boldsymbol{I}_d\\ \mathbf{0}\end{smallmatrix}\right)$, $\boldsymbol{U}^{\top}\cdot\boldsymbol{Z}\cdot\left(\begin{smallmatrix}\boldsymbol{I}_d\\ \mathbf{0}\end{smallmatrix}\right)$, $\boldsymbol{V}^{\top}\cdot\boldsymbol{Z}\cdot\left(\begin{smallmatrix}\boldsymbol{I}_d\\ \mathbf{0}\end{smallmatrix}\right)$, t^*, t, and $t^*\cdot\boldsymbol{U}+\boldsymbol{V}$ there are p possible values of $t\cdot\boldsymbol{U}+\boldsymbol{V}$, and all these values are equally probable.*

Proof. Let us denote the given tuple by $D = (\boldsymbol{B}, \boldsymbol{Z}, D_1, D_2, D_3, D_4, t^*, t, T)$, where D_i's are all from $\mathbb{Z}_p^{(d+1)\times d}$ and $T \in \mathbb{Z}_p^{(d+1)\times(d+1)}$. Furthermore, let us denote $\boldsymbol{B}^{-1} := \begin{pmatrix}\boldsymbol{W}_d\\ \boldsymbol{w}_{d+1}^{\top}\end{pmatrix}$. Then, $\boldsymbol{B}\cdot\begin{pmatrix}\mathbf{0} & \mathbf{0}\\ \mathbf{0} & 1\end{pmatrix}\cdot\boldsymbol{B}^{-1} = \boldsymbol{b}_{d+1}\cdot\boldsymbol{w}_{d+1}^{\top} \neq \mathbf{0} \in \mathbb{Z}_p^{(d+1)\times(d+1)}$.

By Lemma 6.6 and by Parameter Hiding Lemma 6.5 the choice of $\boldsymbol{U}$ and $\boldsymbol{V}$ can be seen as the random choice of $\boldsymbol{U}', \boldsymbol{V}' \leftarrow \mathbb{Z}_p^{(d+1)\times(d+1)}$ and $\sigma_u, \sigma_v \leftarrow \mathbb{Z}_p$ and the assignments

$$\begin{aligned} \boldsymbol{U} &:= \boldsymbol{U}' + \boldsymbol{B} \cdot \begin{pmatrix} \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \sigma_u \end{pmatrix} \cdot \boldsymbol{B}^{-1} \\ &= \boldsymbol{U}' + \sigma_u \cdot \boldsymbol{b}_{d+1} \cdot \boldsymbol{w}_{d+1}^\top \end{aligned}$$

and

$$\begin{aligned} \boldsymbol{V} &:= \boldsymbol{V}' + \boldsymbol{B} \cdot \begin{pmatrix} \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \sigma_v \end{pmatrix} \cdot \boldsymbol{B}^{-1} \\ &= \boldsymbol{V}' + \sigma_v \cdot \boldsymbol{b}_{d+1} \cdot \boldsymbol{w}_{d+1}^\top \ . \end{aligned}$$

Thereby, by Parameter Hiding Lemma 6.5 the values σ_u and σ_v remain information-theoretically hidden when given $\boldsymbol{B}$, $\boldsymbol{Z}$, $\boldsymbol{U} \cdot \boldsymbol{B} \cdot \left(\begin{smallmatrix} \boldsymbol{I}_d \\ \boldsymbol{0} \end{smallmatrix}\right)$, $\boldsymbol{V} \cdot \boldsymbol{B} \cdot \left(\begin{smallmatrix} \boldsymbol{I}_d \\ \boldsymbol{0} \end{smallmatrix}\right)$, $\boldsymbol{U}^\top \cdot \boldsymbol{Z} \cdot \left(\begin{smallmatrix} \boldsymbol{I}_d \\ \boldsymbol{0} \end{smallmatrix}\right)$, and $\boldsymbol{V}^\top \cdot \boldsymbol{Z} \cdot \left(\begin{smallmatrix} \boldsymbol{I}_d \\ \boldsymbol{0} \end{smallmatrix}\right)$.

Consider the value $T = t^* \cdot \boldsymbol{U} + \boldsymbol{V}$ given in addition:

$$\begin{aligned} T = t^* \cdot \boldsymbol{U} + \boldsymbol{V} &= \left(t^* \cdot \boldsymbol{U}' + \boldsymbol{V}'\right) + \boldsymbol{B} \cdot \begin{pmatrix} \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & t^* \cdot \sigma_u + \sigma_v \end{pmatrix} \cdot \boldsymbol{B}^{-1} \\ &= \left(t^* \cdot \boldsymbol{U}' + \boldsymbol{V}'\right) + \left(t^* \cdot \sigma_u + \sigma_v\right) \cdot \boldsymbol{b}_{d+1} \cdot \boldsymbol{w}_{d+1}^\top \ . \end{aligned}$$

For every t^* the value of σ_u remains hidden when given all values of D due to the random choice of σ_v. Next, consider the value $t \cdot \boldsymbol{U} + \boldsymbol{V}$:

$$\begin{aligned} \mathbb{Z}_p^{(d+1)\times(d+1)} \ni t \cdot \boldsymbol{U} + \boldsymbol{V} &= \left(t \cdot \boldsymbol{U}' + \boldsymbol{V}'\right) + \boldsymbol{B} \cdot \begin{pmatrix} \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & t \cdot \sigma_u + \sigma_v \end{pmatrix} \cdot \boldsymbol{B}^{-1} \\ &= \left(t \cdot \boldsymbol{U}' + \boldsymbol{V}'\right) + \left(t \cdot \sigma_u + \sigma_v\right) \cdot \boldsymbol{b}_{d+1} \cdot \boldsymbol{w}_{d+1}^\top \\ &= T + \sigma_u \cdot (t - t^*) \cdot \boldsymbol{b}_{d+1} \cdot \boldsymbol{w}_{d+1}^\top \ . \end{aligned}$$

Matrix T, the value $t - t^* \neq 0$, and the matrix $\boldsymbol{b}_{d+1} \cdot \boldsymbol{w}_{d+1}^\top \neq \boldsymbol{0} \in \mathbb{Z}_p^{(d+1)\times(d+1)}$ are known. However, as shown above, value σ_u is information-theoretically hidden when given D, and hence the matrix $t \cdot \boldsymbol{U} + \boldsymbol{V}$ can take all p possible values, one for each $\sigma_u \in \mathbb{Z}_p$. □

6.1.3. Matrix-DDH Security Assumption

In this subsection we recall the security assumption used in [Att16] for the underlying CPA-secure framework. We also use the same security assumption for our CCA-secure framework. We refer to [EHK+15] for an extensive study on Matrix Diffie-Hellman Assumptions.

In this subsection we consider asymmetric bilinear groups $\mathbb{GD} = (p, (g_1, \mathbb{G}_1), (g_2, \mathbb{G}_2), \mathbb{G}_\mathrm{T}, \mathrm{e})$ of prime order p. Let $d \in \mathbb{N}$ be an integer and $\mathcal{D}_d$ be a distribution on matrices in $\mathbb{GL}_{p,d+1} \subset \mathbb{Z}_p^{(d+1)\times(d+1)}$ of the form

$$\begin{array}{c} \\ d \\ 1 \end{array} \begin{array}{c} \begin{array}{cc} d & 1 \end{array} \\ \begin{pmatrix} \boldsymbol{M} & \boldsymbol{0} \\ \boldsymbol{c}^\top & 1 \end{pmatrix} \end{array} ,$$

where $\boldsymbol{M} \in \mathbb{GL}_{p,d}$ and $\boldsymbol{c} \in \mathbb{Z}_p^d$.

Assumption 6.1. **$\mathcal{D}_d$-Matrix Decisional Diffie-Hellman Assumption** holds relatively to group generator $\mathcal{G}$ if the following function is negligible in λ:

$$\mathrm{Adv}_{\mathcal{A}}^{\mathcal{D}_d\text{-mDH}}(\lambda) := \left| \Pr\left[\mathcal{A}\left(\mathbb{GD}, g_1^{\boldsymbol{A}}, g_1^{\boldsymbol{A}\cdot\left(\begin{smallmatrix} \boldsymbol{y} \\ 0 \end{smallmatrix}\right)}\right) = 1\right] - \Pr\left[\mathcal{A}\left(\mathbb{GD}, g_1^{\boldsymbol{A}}, g_1^{\boldsymbol{A}\cdot\left(\begin{smallmatrix} \boldsymbol{y} \\ \hat{y} \end{smallmatrix}\right)}\right) = 1\right] \right| ,$$

where the probabilities are over $\mathbb{GD} \leftarrow \mathcal{G}\left(1^\lambda\right)$, $\boldsymbol{A} \leftarrow \mathcal{D}_d$, $\boldsymbol{y} \leftarrow \mathbb{Z}_p^d$, $\hat{y} \leftarrow \mathbb{Z}_p^*$, and over the coin tosses of $\mathcal{A}$.

Notice that we slightly changed the usual definition of the assumption. Namely, $\hat{y}$ is chosen from $\mathbb{Z}_p^*$ instead of $\mathbb{Z}_p$. This modification does not change the assumption, since the corresponding distributions are statistically close, but the arguments in the proofs are more precise.

Intuition. Next, we shortly explain the Matrix Decisional Diffie-Hellman Assumption in more detail in order to provide the intuition behind our construction and present an important and well-known representative. Denote the column vectors of $\boldsymbol{A}$ by $(\,\boldsymbol{a}_1 \cdots \boldsymbol{a}_{d+1}\,)$. Distinguisher $\mathcal{A}$ gets by $g_1^{\boldsymbol{A}}$ the matrix $\boldsymbol{A}$ in the exponent of g_1. By the elements of $g_1^{\boldsymbol{A}\cdot\binom{\boldsymbol{y}}{\hat{y}}} \in \mathbb{G}_1^{d+1}$ the adversary also gets a vector which is a linear combination of vectors $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_{d+1}$ in the exponent of g_1. Hence, $\mathcal{A}$ has to decide if the last vector $\boldsymbol{a}_{d+1}$ appears in this linear combination. Due to the special form of the matrices in $\mathcal{D}_d$ it is even more specific:

$$\begin{aligned}\boldsymbol{A} \cdot \binom{\boldsymbol{y}}{\hat{y}} &= \begin{pmatrix} \boldsymbol{M} & \boldsymbol{0} \\ \boldsymbol{c}^\top & 1 \end{pmatrix} \cdot \binom{\boldsymbol{y}}{\hat{y}} \\ &= y_1 \cdot \binom{\boldsymbol{m}_1}{c_1} + y_2 \cdot \binom{\boldsymbol{m}_2}{c_2} + \cdots + y_d \cdot \binom{\boldsymbol{m}_d}{c_d} + \hat{y} \cdot \binom{\boldsymbol{0}}{1} \\ &= \begin{pmatrix} \boldsymbol{M} \cdot \boldsymbol{y} \\ \boldsymbol{c}^\top \cdot \boldsymbol{y} + \hat{y} \end{pmatrix} \quad ,\end{aligned}$$

where $\boldsymbol{M} = (\,\boldsymbol{m}_1 \cdots \boldsymbol{m}_d\,)$ denotes the column vectors of $\boldsymbol{M}$. Information-theoretically $\mathcal{A}$ gets the complete information about $\boldsymbol{A}$ through $g_1^{\boldsymbol{A}}$ (that is, about the corresponding $\boldsymbol{M} \in \mathbb{GL}_{p,d}$ and $\boldsymbol{c}^\top \in \mathbb{Z}_p^d$). Furthermore, the first d elements of $g_1^{\boldsymbol{A}\cdot\binom{\boldsymbol{y}}{\hat{y}}}$ uniquely define $\boldsymbol{y}$, since these are equal to $g_1^{\boldsymbol{M}\cdot\boldsymbol{y}}$ and $\boldsymbol{M}$ is invertible. The assumption says that given all these elements $\mathcal{A}$ cannot distinguish between $g_1^{\boldsymbol{y}\cdot\boldsymbol{c}}$ and $g_1^{\boldsymbol{y}\cdot\boldsymbol{c}+\hat{y}}$. Given $\boldsymbol{y}$ in plain, this would be easy since one can compute $g_1^{\boldsymbol{y}\cdot\boldsymbol{c}} = (g_1^{\boldsymbol{c}})^{\boldsymbol{y}}$ from $g_1^{\boldsymbol{c}}$ included in $g_1^{\boldsymbol{A}}$.

Similarly to [Att16] our framework does not require any specific $\mathcal{D}_d$-Matrix Decisional Differ-Hellman Assumption. A special case of this assumption is the well-known Decisional d-Linear Assumption, where $\boldsymbol{M}$ is a random diagonal matrix and $\boldsymbol{c} = \boldsymbol{1}$. That is, the adversary has to distinguish between $g_1^{\sum_{k=1}^d y_k}$ and $g_1^{\hat{y}+\sum_{k=1}^d y_k}$ given $g_1^{\boldsymbol{m}}$ and $\left(g_1^{m_1\cdot y_1}, \ldots, g_1^{m_d\cdot y_d}\right)$, where $\boldsymbol{m}, \boldsymbol{y} \leftarrow \mathbb{Z}_p^d$, $\hat{y} \leftarrow \mathbb{Z}_p$. We notice that due to the bilinear groups used for our construction we can use $d \geq 2$ and there is a trade-off between efficiency (small values of d) and weaker assumptions (big values of d) [EHK+15].

Assumption 6.2. Decisional d-Linear Assumption holds relatively to group generator $\mathcal{G}$ if the following function is negligible in λ:

$$\mathrm{Adv}_{\mathcal{A}}^{d\text{-linDH}}(\lambda) := \left|\Pr\left[\mathcal{A}\left(\mathbb{GD}, g_1^{\boldsymbol{m}}, g_1^{\boldsymbol{v}}, Z_0\right) = 1\right] - \Pr\left[\mathcal{A}\left(\mathbb{GD}, g_1^{\boldsymbol{m}}, g_1^{\boldsymbol{v}}, Z_1\right) = 1\right]\right| \quad .$$

where the probabilities are over $\mathbb{GD} \leftarrow \mathcal{G}\left(1^\lambda\right)$, $\boldsymbol{m}, \boldsymbol{y} \leftarrow \mathbb{Z}_p^d$, $\hat{y} \leftarrow \mathbb{Z}_p^*$, and over the coin tosses of $\mathcal{A}$. Furthermore, $g_1^{\boldsymbol{v}} = g_1^{m_1\cdot y_1}, \ldots, g_1^{m_d\cdot y_d}$, $Z_0 = g_1^{\sum_{k=1}^d y_k}$ or $Z_1 = g_1^{\hat{y}+\sum_{k=1}^d y_k}$.

Random Self Reducibility for $\mathcal{D}_d$-Matrix Decisional Diffie-Hellman Problem

For the sake of completeness in this section we prove the random self reducibility of the $\mathcal{D}_d$-MDDH problem instances (see also [Att16, EHK+15]). Let d be an arbitrary integer. The rerandomization algorithm from the constructive proof of the following lemma is used in several reduction algorithms of our proof.

Lemma 6.9. *There exists a ppt algorithm* ReRand *such that for every security parameter* $\lambda \in \mathbb{N}$*, every* $\mathbb{GD} \in \left[\mathcal{G}\left(1^\lambda\right)\right]$*, every* $\boldsymbol{A} \in [\mathcal{D}_d] \subseteq \mathbb{GL}_{p,d+1}$*, every* $\boldsymbol{y} \in \mathbb{Z}_p^d$*, every* $k \in \mathbb{N}$ *and every* $\mathbf{X} \in \mathbb{G}_1^{(d+1)\times k}$ *it holds*

$$\Pr_{\boldsymbol{Y}\leftarrow\mathbb{Z}_p^{d\times k}}\left[\mathbf{X} = g_1^{\boldsymbol{A}\cdot\binom{\boldsymbol{Y}}{\mathbf{0}^\top}}\right] = \Pr\left[\mathbf{X} = \mathrm{ReRand}\left(\mathbb{GD}, g_1^{\boldsymbol{A}}, g_1^{\boldsymbol{A}\cdot\binom{\boldsymbol{y}}{0}}, k\right)\right] \ .$$

Furthermore, for every $\hat{y} \in \mathbb{Z}_p^*$ *it holds*

$$\Pr_{\boldsymbol{Y}\leftarrow\mathbb{Z}_p^{d\times k},\hat{\boldsymbol{y}}\leftarrow\mathbb{Z}_p^k}\left[\mathbf{X} = g_1^{\boldsymbol{A}\cdot\binom{\boldsymbol{Y}}{\hat{\boldsymbol{y}}^\top}}\right] = \Pr\left[\mathbf{X} = \mathrm{ReRand}\left(\mathbb{GD}, g_1^{\boldsymbol{A}}, g_1^{\boldsymbol{A}\cdot\binom{\boldsymbol{y}}{\hat{y}}}, k\right)\right] \ .$$

Proof. The algorithm ReRand is presented in Algorithm 16. Consider the output probability

Algorithm 16: ReRand - Self reducibility for $\mathcal{D}_d$-MDDH problem

Input : $\mathbb{GD}, \mathbf{A}, \mathbf{Z}, k$

Require: $\mathbb{GD} = (p, (g_1, \mathbb{G}_1), (g_2, \mathbb{G}_2), \mathbb{G}_\mathrm{T}, \mathrm{e})$, $\mathbf{A} \in \mathbb{G}^{(d+1)\times(d+1)}$, $\mathbf{Z} \in \mathbb{G}^d$, $k \in \mathbb{N}$

1 **foreach** $i \in [k]$ **do**

2 Choose $\boldsymbol{\delta}_i \leftarrow \mathbb{Z}_p^d$, $\hat{\delta}_i \leftarrow \mathbb{Z}_p$ and compute

$$\mathbf{Y}_i := \mathbf{A}^{\hat{\delta}_i} \cdot \mathbf{Z}^{\binom{\boldsymbol{\delta}_i}{0}} \ .$$

3 **end**

Output : $\begin{pmatrix}\mathbf{Y}_1 & \mathbf{Y}_2 & \cdots & \mathbf{Y}_k\end{pmatrix} \in \mathbb{G}^{(d+1)\times k}$.

distribution of ReRand. For every $i \in [k]$ the algorithm chooses $\boldsymbol{\delta}_i \leftarrow \mathbb{Z}_p^d$ and $\hat{\delta}_i \leftarrow \mathbb{Z}_p$ and computes from $\mathbf{A} = g_1^{\boldsymbol{A}}$ and $\mathbf{Z} = g_1^{\boldsymbol{A}\cdot\binom{\boldsymbol{y}}{\hat{y}}}$:

$$\begin{aligned}
\mathbf{Y}_i &= \mathbf{A}^{\hat{\delta}_i} \cdot \mathbf{Z}^{\binom{\boldsymbol{\delta}_i}{0}} \\
&= \left(g_1^{\boldsymbol{A}\cdot\binom{\boldsymbol{y}}{\hat{y}}}\right)^{\hat{\delta}_i} \cdot \left(g_1^{\boldsymbol{A}}\right)^{\binom{\boldsymbol{\delta}_i}{0}} \\
&= g_1^{\boldsymbol{A}\cdot\binom{\hat{\delta}_i\cdot\boldsymbol{y}}{\hat{\delta}_i\cdot\hat{y}}+\boldsymbol{A}\cdot\binom{\boldsymbol{\delta}_i}{0}} \\
&= g_1^{\boldsymbol{A}\cdot\binom{\hat{\delta}_i\cdot\boldsymbol{y}+\boldsymbol{\delta}_i}{\hat{\delta}_i\cdot\hat{y}}} \\
&= g_1^{\boldsymbol{A}\cdot\binom{\boldsymbol{y}'_i}{\hat{y}'_i}} \ .
\end{aligned}$$

Hence, by construction of the algorithm it outputs the matrix $g_1^{\boldsymbol{A}\cdot\left(\begin{smallmatrix}\boldsymbol{y}'_1 & \boldsymbol{y}'_2 & \cdots & \boldsymbol{y}'_k\\ \hat{y}'_1 & \hat{y}'_2 & \cdots & \hat{y}'_k\end{smallmatrix}\right)}$ of group elements, where for every $i \in [k]$ it holds

$$\begin{aligned}\boldsymbol{y}'_i &= \hat{\delta}_i \cdot \boldsymbol{y} + \boldsymbol{\delta}_i \ ,\\ \hat{y}'_i &= \hat{\delta}_i \cdot \hat{y} \ .\end{aligned}$$

We conclude that if $\hat{y} = 0$, then all $\hat{y}'_i$'s are equal zero, whereas $\boldsymbol{y}'_i$'s are uniformly distributed and mutually independent by the choice of corresponding $\boldsymbol{\delta}_i$'s. Alternatively speaking in this case the distribution on $\left(\begin{smallmatrix}\boldsymbol{y}'_1 & \boldsymbol{y}'_2 & \cdots & \boldsymbol{y}'_k\\ \hat{y}'_1 & \hat{y}'_2 & \cdots & \hat{y}'_k\end{smallmatrix}\right)$ is the same as the distribution on $\left(\begin{smallmatrix}\boldsymbol{Y}\\ \boldsymbol{0}^\top\end{smallmatrix}\right)$, where $\boldsymbol{Y} \leftarrow \mathbb{Z}_p^{d\times k}$.

Analogously, if $\hat{y} \neq 0$ then $\boldsymbol{y}'_i$'s and $\hat{y}'_i$'s are uniformly distributed and mutually independent by the choice of corresponding $\boldsymbol{\delta}_i$'s and $\hat{\delta}_i$'s. Hence, the distribution on $\left(\begin{smallmatrix}\boldsymbol{y}'_1 & \boldsymbol{y}'_2 & \cdots & \boldsymbol{y}'_k\\ \hat{y}'_1 & \hat{y}'_2 & \cdots & \hat{y}'_k\end{smallmatrix}\right)$ is the same as the distribution on $\left(\begin{smallmatrix}\boldsymbol{Y}\\ \hat{\boldsymbol{y}}^\top\end{smallmatrix}\right)$, where $\boldsymbol{Y} \leftarrow \mathbb{Z}_p^{d\times k}$, $\hat{\boldsymbol{y}} \leftarrow \mathbb{Z}_p^k$. □

Assumption 6.3. $(k, \mathcal{D}_d)$-Matrix Decisional Diffie-Hellman Assumption holds relatively to group generator $\mathcal{G}$ if the following function is negligible in λ:

$$\mathrm{Adv}^{k,\mathcal{D}_d\text{-mDH}}_{\mathcal{A}}(\lambda) := \left|\Pr\left[\mathcal{A}\left(\mathbb{GD}, g_1^{\boldsymbol{A}}, g_1^{\boldsymbol{A}\cdot\left(\begin{smallmatrix}\boldsymbol{Y}\\ \boldsymbol{0}\end{smallmatrix}\right)}\right) = 1\right] - \Pr\left[\mathcal{A}\left(\mathbb{GD}, g_1^{\boldsymbol{A}}, g_1^{\boldsymbol{A}\cdot\left(\begin{smallmatrix}\boldsymbol{Y}\\ \hat{\boldsymbol{y}}\end{smallmatrix}\right)}\right) = 1\right]\right| \ ,$$

where the probabilities are over $\mathbb{GD} \leftarrow \mathcal{G}\left(1^\lambda\right)$, $\boldsymbol{A} \leftarrow \mathcal{D}_d$, $\boldsymbol{Y} \leftarrow \mathbb{Z}_p^{d\times k}$, $\hat{\boldsymbol{y}} \leftarrow \mathbb{Z}_p^k$, and over the coin tosses of $\mathcal{A}$.

The following corollary follows directly from Lemma 6.9.

Corollary 6.10. For every integer $k \in \mathbb{N}$ and every ppt algorithm $\mathcal{A}$ there exists a ppt algorithm $\mathcal{A}'$ such that

$$\mathrm{Adv}^{k,\mathcal{D}_d\text{-mDH}}_{\mathcal{A}}(\lambda) = \mathrm{Adv}^{\mathcal{D}_d\text{-mDH}}_{\mathcal{A}'}(\lambda) \ .$$

6.1.4. CCA Security Definition for PE

Similarly to Subsection 4.1.3, here we present the security experiment that is used in the formal security proof for our framework. In the proof we consider slightly modified distributions. Those parts of the experiment which will be changed later are framed and numbered.

Let Π be a PE for predicate family $\mathcal{R}_{\Omega,\Sigma}$ and message space $\mathcal{M}$. The full (adaptive) security experiment $\mathrm{aPE}^{\mathrm{CCA2}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})$ against adaptive chosen-ciphertext attacks between challenger $\mathcal{C}$ and adversary $\mathcal{A}$ is defined in Fig. 6.1. In this experiment, index i denotes the number of a covered key generation query and kInd_i denotes the key index used in the query with number i. W.l.o.g. we assume that $\mathcal{A}$ uses index i in the oracle queries only after the i'th query to the covered key generation oracle. Furthermore, due to the analysis from Chapter 2 we always assume that all inputs of oracles are syntactically correct and for every CT submitted to the decryption oracle it holds $\mathrm{CT} \in \mathbb{C}_{\mathrm{cInd}} \subseteq \mathbb{C}_{\mathrm{pp}_\kappa}$ and $\mathrm{R}_\kappa(\mathrm{kInd}_i, \mathrm{cInd}) = 1$, where i is the index specified for the decryption query. W.l.o.g. we can also assume that $\mathcal{A}$ never asks for the decryption of CT^* in the second query phase. That is, we use the SE-model as defined in Chapter 2. In this chapter we denote the set of ppt adversaries against Π by $\mathbf{A}_\Pi$.

$\mathbf{aPE}^{\mathrm{CCA2}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})$:

Setup : $\mathcal{C}$ generates $^{\langle 12\rangle}$ $\boxed{(\mathrm{msk}, \mathrm{pp}_\kappa) \leftarrow \mathrm{Setup}\left(1^\lambda, \mathrm{des}\right)}$ and starts $\mathcal{A}$ on input $\left(1^\lambda, \mathrm{pp}_\kappa\right)$.

Phase I : $\mathcal{A}$ has access to the following oracles:

- **CKGen** (kInd_i) for $\mathrm{kInd}_i \in \mathbb{X}_\kappa$: Challenger $\mathcal{C}$ generates a secret key for kInd_i $^{\langle 13\rangle}$ $\boxed{\mathrm{sk}_i \leftarrow \mathrm{KeyGen}(\mathrm{msk}, \mathrm{kInd}_i)}$, stores sk_i, and returns nothing.
- **Open** (i) for $i \in \mathbb{N}$: $\mathcal{C}$ returns $^{\langle 14\rangle}$ $\boxed{\mathrm{sk}_i}$. We call the corresponding key index kInd_i of sk_i a corrupted key index.
- **Dec** (CT, i) with $\mathrm{CT} \in \mathbb{C}_{\mathrm{cInd}} \subseteq \mathbb{C}_{\mathrm{pp}_\kappa}$ and $i \in \mathbb{N}$: Challenger $\mathcal{C}$ returns the decryption $^{\langle 15\rangle}$ $\boxed{\mathrm{Dec}(\mathrm{sk}_i, \mathrm{CT})}$.

Challenge : $\mathcal{A}$ submits a target ciphertext index $\mathrm{cInd}^* \in \mathbb{Y}_\kappa$, messages $m_0, m_1 \in \mathcal{M}$, $|m_0| = |m_1|$ such that for every corrupted key index kInd it holds $\mathrm{R}_\kappa(\mathrm{kInd}, \mathrm{cInd}^*) = 0$. $\mathcal{C}$ flips a bit $b \leftarrow \{0,1\}$, computes $^{\langle 16\rangle}$ $\boxed{\mathrm{CT}^* \leftarrow \mathrm{Enc}(\mathrm{cInd}^*, m_b)}$, and returns the challenge ciphertext CT^*.

Phase II : $\mathcal{A}$ has access to the following oracles:

- **CKGen** (kInd_i) for $\mathrm{kInd}_i \in \mathbb{X}_\kappa$: As before, challenger $\mathcal{C}$ generates a secret key $^{\langle 17\rangle}$ $\boxed{\mathrm{sk}_i \leftarrow \mathrm{KeyGen}(\mathrm{msk}, \mathrm{kInd}_i)}$, stores sk_i, and returns nothing.
- **Open** (i) for $i \in \mathbb{N}$ such that $\mathrm{R}_\kappa(\mathrm{kInd}_i, \mathrm{cInd}^*) = 0$: $\mathcal{C}$ returns $^{\langle 18\rangle}$ $\boxed{\mathrm{sk}_i}$.
- **Dec** (CT, i) with $\mathrm{CT} \in \mathbb{C}_{\mathrm{cInd}} \subseteq \mathbb{C}_{\mathrm{pp}_\kappa}$, $\mathrm{CT} \neq \mathrm{CT}^*$ and $i \in \mathbb{N}$: $\mathcal{C}$ returns the decryption $^{\langle 19\rangle}$ $\boxed{\mathrm{Dec}(\mathrm{sk}_i, \mathrm{CT})}$.

Guess : $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$. $^{\langle 20\rangle}$ $\boxed{\text{The output of the experiment is 1 iff } b' = b}$.

Figure 6.1.: CCA-secure experiment for the framework in prime-order groups.

The advantage of $\mathcal{A}$ in security experiment $\mathrm{aPE}^{\mathrm{CCA2}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})$ is defined as

$$\text{Adv-aPE}^{\mathrm{CCA2}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) := \Pr\left[\mathrm{aPE}^{\mathrm{CCA2}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) = 1\right] - \frac{1}{2} \ .$$

Definition 6.11. A predicate encryption Π for predicate family $\mathcal{R}_{\Omega,\Sigma}$ is called **fully (or adaptively) secure against adaptively chosen-ciphertext attacks (or CCA2-secure)** if for every $\mathrm{des} \in \Omega$ and every ppt adversary $\mathcal{A} \in \mathbf{A}_\Pi$ the function $\text{Adv-aPE}^{\mathrm{CCA2}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})$ is negligible in λ.

6.2. Fully CCA-secure Framework

In this section we present our framework to construct CCA-secure PE schemes from pair encodings. We also explain the guarantees of our consistency-checks, which lead to the CCA-security together with our structural extensions of the public parameters and the ciphertexts.

6.2.1. Normal Algorithms

In this subsection we present our framework for constructing fully CCA-secure predicate encryption schemes from pair encoding schemes in prime-order bilinear groups. The framework is based on the CPA-secure framework from [Att16]. We use different notation and explicitly denote all our extensions and modifications. We refer to [Att16] for an excellent discussion about the relation between composite-order framework and the prime-order framework as well as about the techniques used in order to replicate properties from composite-order groups in prime-order setting. In this thesis we focus on CCA-security and the techniques to achieve this stronger security notion.

Let $\mathrm{P} = (\mathrm{Param}, \mathrm{Enc1}, \mathrm{Enc2}, \mathrm{Pair})$ be a regular pair encoding scheme for predicate family $\mathcal{R}_{\Omega,\Sigma}$. Let $\mathcal{G}$ be a prime-order group generator, and $\mathcal{H}$ be a family of collision-resistant hash functions as defined in Subsection 1.1.3. We require that a hash function from $\mathcal{H}$ chosen by $h \leftarrow \mathcal{H}\left(1^\lambda\right)$ works on domain $\mathbb{Y}_\kappa \times \mathbb{G}_\mathrm{T} \times \mathbb{G}_1^{\mathrm{poly}(\lambda)}$ and has range $\mathbb{Z}_p$, where p is of length λ. Formally, we require a collision-resistant hash function with range $\mathbb{Z}_p$ and a canonical binary representation of elements $\mathbb{Y}_\kappa \times \mathbb{G}^{\mathrm{poly}(\lambda)}$. We abstract from these technicalities. Furthermore, let $d \in \mathbb{N}$, $d \geq 2$ be the fixed parameter of $\mathcal{D}_d$-Matrix DDH Assumption.

The predicate encryption scheme $\Pi = (\mathrm{Setup}, \mathrm{KeyGen}, \mathrm{Enc}, \mathrm{Dec})$ for $\mathcal{R}_{\Omega,\Sigma}$ is as follows:

Setup $\left(1^\lambda, \mathrm{des}\right)$:

- Choose $\mathbb{GD} = \left(p, \left(g_1, \mathbb{G}_1\right), \left(g_2, \mathbb{G}_2\right), \mathbb{G}_\mathrm{T}, \mathrm{e}\right) \leftarrow \mathcal{G}\left(1^\lambda\right)$ and set $\kappa := (\mathrm{des}, p)$.
- Pick $\boldsymbol{B} \leftarrow \mathbb{GL}_{p,d+1}$ and $\tilde{\boldsymbol{D}} \leftarrow \mathbb{GL}_{p,d}$. Compute $\boldsymbol{D} := \left(\begin{smallmatrix} \tilde{\boldsymbol{D}} & \mathbf{0} \\ \mathbf{0} & 1 \end{smallmatrix}\right) \in \mathbb{GL}_{p,d+1}$, $\boldsymbol{Z} := \boldsymbol{B}^{-\top} \cdot \boldsymbol{D} \in \mathbb{GL}_{p,d+1}$, and
$$\mathbf{G}_1 := g_1^{\boldsymbol{B}\cdot\left(\begin{smallmatrix} \boldsymbol{I}_d \\ \mathbf{0} \end{smallmatrix}\right)} \quad , \qquad \mathbf{G}_2 := g_2^{\boldsymbol{Z}\cdot\left(\begin{smallmatrix} \boldsymbol{I}_d \\ \mathbf{0} \end{smallmatrix}\right)} \quad .$$
By construction $\mathbf{G}_1 \in \mathbb{G}_1^{(d+1)\times d}$ and $\mathbf{G}_2 \in \mathbb{G}_2^{(d+1)\times d}$.
- Compute $n := \mathrm{Param}\left(\kappa\right)$, pick $\boldsymbol{H}_1, \ldots, \boldsymbol{H}_n, \boldsymbol{U}, \boldsymbol{V} \leftarrow \mathbb{Z}_p^{(d+1)\times(d+1)}$, and compute:
$$\forall_{j\in[n]} : \mathbf{H}_j := \mathbf{G}_1^{\boldsymbol{H}_j} \quad , \qquad \forall_{j\in[n]} : \overline{\mathbf{H}}_j := \mathbf{G}_2^{\boldsymbol{H}_j^\top} \quad , \tag{6.20}$$
$$\mathbf{U} := \mathbf{G}_1^{\boldsymbol{U}} \quad , \qquad \overline{\mathbf{U}} := \mathbf{G}_2^{\boldsymbol{U}^\top} \quad , \tag{6.21}$$
$$\mathbf{V} := \mathbf{G}_1^{\boldsymbol{V}} \quad , \qquad \overline{\mathbf{V}} := \mathbf{G}_2^{\boldsymbol{V}^\top} \quad . \tag{6.22}$$
By construction, $\mathbf{H}_1, \ldots \mathbf{H}_n, \mathbf{U}, \mathbf{V} \in \mathbb{G}_1^{(d+1)\times d}$, $\overline{\mathbf{H}}_1, \ldots, \overline{\mathbf{H}}_n, \overline{\mathbf{U}}, \overline{\mathbf{V}} \in \mathbb{G}_2^{(d+1)\times d}$. We denote $\mathbb{H} := (\boldsymbol{H}_1, \ldots, \boldsymbol{H}_n)$ and $\overline{\mathbb{H}} := \left(\boldsymbol{H}_1^\top, \ldots, \boldsymbol{H}_n^\top\right)$.
- Pick $\boldsymbol{\alpha} \leftarrow \mathbb{Z}_p^{d+1}$ and compute
$$\mathbf{Y} := \mathrm{e}\left(\mathbf{G}_1, g_2^{\boldsymbol{\alpha}}\right) \quad .$$
By construction, $\mathbf{Y} \in {\mathbb{G}_\mathrm{T}}^{1\times d}$.
- Pick a hash function $\mathrm{H} \leftarrow \mathcal{H}\left(1^\lambda\right)$ and output the master secret key $\mathrm{msk} := g_2^{\boldsymbol{\alpha}}$ and the public parameters $\mathrm{pp}_\kappa := \left(\mathrm{des}, \mathbb{GD}, \mathrm{H}, \mathbf{G}_1, \{\mathbf{H}_j\}_{j\in[n]}, \mathbf{U}, \mathbf{V}, \mathbf{Y}, \mathbf{G}_2, \left\{\overline{\mathbf{H}}_j\right\}_{j\in[n]}, \overline{\mathbf{U}}, \overline{\mathbf{V}}\right)$.

The elements $\mathbf{G}_2, \left\{\overline{\mathbf{H}}_j\right\}_{j\in[n]}$ were part of the master secret key in the original CPA-secure framework (required only for computation of keys there). We need these elements for the verification checks. Notice that it is not obvious that these elements can be made public, but our security proof deals with these circumstances. The hash function and the elements $\mathbf{U}, \mathbf{V}, \overline{\mathbf{U}}, \overline{\mathbf{V}}$ are new. The message space $\mathcal{M}$ depends on pp_κ and is equal to $\mathbb{G}_\mathrm{T}$ specified by $\mathbb{GD} \in \mathrm{pp}_\kappa$.

KeyGen $\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{kInd}\right)$:

- Parse $\mathrm{pp}_\kappa = \left(\mathrm{des}, \mathbb{GD}, \mathrm{H}, \mathbf{G}_1, \{\mathbf{H}_j\}_{j\in[n]}, \mathbf{U}, \mathbf{V}, \mathbf{Y}, \mathbf{G}_2, \{\overline{\mathbf{H}}_j\}_{j\in[n]}, \overline{\mathbf{U}}, \overline{\mathbf{V}}\right)$. If $\mathrm{kInd} \notin \mathbb{X}_\kappa$, output $\perp$.
- Compute $(\boldsymbol{k}, m_2) := \mathrm{Enc1}(\kappa, \mathrm{kInd})$, let $m_1 = |\boldsymbol{k}|$. Pick $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_{m_2} \leftarrow \mathbb{Z}_p^d$ and compute for all $\tau \in [m_1]$:

$$\mathbf{K}_\tau := \mathrm{msk}^{a_\tau} \cdot \mathbf{G}_2^{\sum_{i\in[m_2]} a_{\tau,i}\cdot \boldsymbol{r}_i} \cdot \prod_{j\in[n]} \overline{\mathbf{H}}_j^{\sum_{i\in[m_2]} a_{\tau,i,j}\cdot \boldsymbol{r}_i} \in \mathbb{G}_2^{d+1} \ , \tag{6.23}$$

  where $k_\tau = a_\tau \cdot \mathrm{X}_\alpha + \sum_{i\in[m_2]} \left(a_{\tau,i}\cdot \mathrm{X}_{r_i} + \sum_{j\in[n]} a_{\tau,i,j}\cdot \mathrm{X}_{h_j}\cdot \mathrm{X}_{r_i}\right)$ is the τ'th polynomial of $\boldsymbol{k}$. Set $\overrightarrow{\mathbf{K}} := (\mathbf{K}_1, \ldots, \mathbf{K}_{m_1}) \in \left(\mathbb{G}_2^{d+1}\right)^{m_1}$.
- Output $\mathrm{sk} := \left(\mathrm{kInd}, \overrightarrow{\mathbf{K}}\right)$.

The key generation algorithm is exactly the same as in [Att16].

Enc $\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{cInd}, m\right)$ with $\mathrm{cInd} \in \mathbb{Y}_\kappa$ and $m \in \mathbb{G}_\mathrm{T} = \mathcal{M}$:

- Parse $\mathrm{pp}_\kappa = \left(\mathrm{des}, \mathbb{GD}, \mathrm{H}, \mathbf{G}_1, \{\mathbf{H}_j\}_{j\in[n]}, \mathbf{U}, \mathbf{V}, \mathbf{Y}, \mathbf{G}_2, \{\overline{\mathbf{H}}_j\}_{j\in[n]}, \overline{\mathbf{U}}, \overline{\mathbf{V}}\right)$.
- Compute $(\boldsymbol{c}, w_2) := \mathrm{Enc2}(\kappa, \mathrm{cInd})$, let $w_1 = |\boldsymbol{c}|$. Pick $\boldsymbol{s}_0, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_{w_2} \leftarrow \mathbb{Z}_p^d$ and compute for all $\tau \in [w_1]$ (recall definition of $I \subseteq [w_2]_0$ from Subsection 1.3.2):

$$\mathbf{C}_\tau := \mathbf{G}_1^{\sum_{i\in[w_2]_0} b_{\tau,i}\cdot \boldsymbol{s}_i} \cdot \prod_{j\in[n]} \mathbf{H}_j^{\sum_{i\in I} b_{\tau,i,j}\cdot \boldsymbol{s}_i} \in \mathbb{G}_1^{d+1} \ , \tag{6.24}$$

  where $c_\tau = \sum_{i\in[w_2]_0} b_{\tau,i}\cdot \mathrm{X}_{s_i} + \sum_{i\in I}\sum_{j\in[n]} b_{\tau,i,j}\cdot \mathrm{X}_{h_j}\cdot \mathrm{X}_{s_i}$ is the τ'th polynomial of $\boldsymbol{c}$. Set $\overrightarrow{\mathbf{C}} := (\mathbf{C}_1, \ldots, \mathbf{C}_{w_1}) \in \left(\mathbb{G}_1^{d+1}\right)^{w_1}$.
- Compute

$$C_0 := m \cdot \mathbf{Y}^{\boldsymbol{s}_0} \in \mathbb{G}_\mathrm{T} \ . \tag{6.25}$$

- Compute the hash value $t := \mathrm{H}\left(\mathrm{cInd}, C_0, \overrightarrow{\mathbf{C}}\right) \in \mathbb{Z}_p$ and set

$$\mathbf{C}'' := \left(\mathbf{U}^t \cdot \mathbf{V}\right)^{\boldsymbol{s}_0} \in \mathbb{G}_1^{d+1} \ . \tag{6.26}$$

- Output

$$\mathrm{CT} := \left(\mathrm{cInd}, C_0, \overrightarrow{\mathbf{C}}, \mathbf{C}''\right) \ .$$

  For every $\mathrm{cInd} \in \mathbb{Y}_\kappa$ we define the ciphertext space $\mathbb{C}_\mathrm{cInd} := \{\mathrm{cInd}\} \times \mathbb{G}_\mathrm{T} \times \left(\mathbb{G}_1^{d+1}\right)^{w_1+1}$, where w_1 is defined by $\mathrm{Enc2}(\kappa, \mathrm{cInd})$ as above. Hence, ciphertext spaces for different indices are disjoint and (using appropriate binary representation) for every $\mathrm{CT} \in \{0,1\}^*$ we can easily check if there exists $\mathrm{cInd} \in \mathbb{Y}_\kappa$ such that $\mathrm{CT} \in \mathbb{C}_\mathrm{cInd} \subseteq \mathbb{C}_{\mathrm{pp}_\kappa}$. For every $\mathrm{CT} \in \mathbb{C}_{\mathrm{pp}_\kappa}$ we explicitly define the function $\mathrm{HInput}(\mathrm{CT}) := \mathrm{HInput}\left(\left(\mathrm{cInd}, C_0, \overrightarrow{\mathbf{C}}, \mathbf{C}''\right)\right) \mapsto \left(\mathrm{cInd}, C_0, \overrightarrow{\mathbf{C}}\right)$. In comparison to [Att16] we only add the element $\mathbf{C}''$ to the ciphertext.

Dec $\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{sk}, \mathrm{CT}\right)$:

- Parse $\mathrm{pp}_\kappa = \left(\mathrm{des}, \mathbb{GD}, \mathrm{H}, \mathbf{G}_1, \{\mathbf{H}_j\}_{j\in[n]}, \mathbf{U}, \mathbf{V}, \mathbf{Y}, \mathbf{G}_2, \{\overline{\mathbf{H}}_j\}_{j\in[n]}, \overline{\mathbf{U}}, \overline{\mathbf{V}}\right)$ and the secret key $\mathrm{sk} = \left(\mathrm{kInd}, \overrightarrow{\mathbf{K}}\right)$. Denote $\overrightarrow{\mathbf{K}} = (\mathbf{K}_1, \ldots, \mathbf{K}_{m_1}) \in \left(\mathbb{G}_2^{d+1}\right)^{m_1}$

- (Syntactical checks:) Output $\perp$ if $\mathrm{CT} \notin \mathbb{C}_{\mathrm{cInd}} \subseteq \mathbb{C}_{\mathrm{cInd}}$. Otherwise parse $\mathrm{CT} \in \mathbb{C}_{\mathrm{cInd}}$ by $\mathrm{CT} = \left(\mathrm{cInd}, C_0, \overrightarrow{\mathbf{C}}, \mathbf{C}''\right)$. Output $\perp$ if $\mathrm{R}_\kappa(\mathrm{kInd}, \mathrm{cInd}) \neq 1$. Denote $\overrightarrow{\mathbf{C}} := (\mathbf{C}_1, \ldots, \mathbf{C}_{w_1}) \in \left(\mathbb{G}_1^{d+1}\right)^{w_1}$.
- Choose $\boldsymbol{E} \leftarrow \mathrm{Pair}(\kappa, \mathrm{kInd}, \mathrm{cInd})$. Let $\boldsymbol{E} = \left(e_{\tau,\tau'}\right)_{m_1,w_1} \in \mathbb{Z}_p^{m_1 \times w_1}$, where m_1 and w_1 are defined by $\mathrm{Enc1}(\kappa, \mathrm{kInd})$ and $\mathrm{Enc2}(\kappa, \mathrm{cInd})$, respectively.
- **Verification checks according to $\boldsymbol{E}$:** Recall the definitions of Γ_0 (respectively I), Γ and $\overline{\Gamma}$ (respectively $\overline{I}$) from Subsection 1.3.2 on page 14. For every $\tau \in \Gamma_0$ there is an index $i \in I$. Hence, we denote $\tau \in \Gamma_0$ by τ_i.

  Compute $(\boldsymbol{c}, w_2) := \mathrm{Enc2}(\kappa, \mathrm{cInd})$. For every $\tau \in [w_1]$ denote the polynomial c_τ by

$$c_\tau = \sum_{i \in \overline{I}} b_{\tau,i} \cdot \mathrm{X}_{s_i} + \sum_{i \in I} \left(b_{\tau,i} \cdot \mathrm{X}_{s_i} + \sum_{j \in [n]} b_{\tau,i,j} \cdot \mathrm{X}_{h_j} \cdot \mathrm{X}_{s_i} \right) \ .$$

  If one of the following three checks fails, return $\perp$:

  1. (Check for $\mathbf{C}''$) Compute the hash value $t := \mathrm{H}\left(\mathrm{cInd}, C_0, \overrightarrow{\mathbf{C}}\right) \in \mathbb{Z}_p$ and check the equation:

$$\mathrm{e}\left(\mathbf{C}'', \mathbf{G}_2\right) \stackrel{?}{=} \mathrm{e}\left(\mathbf{C}_1, \overline{\mathbf{U}}^t \cdot \overline{\mathbf{V}}\right) \ . \tag{6.27}$$

  2. (Check for the elements $\{\mathbf{C}_\tau\}_{\tau \in \Gamma}$) For all $\tau \in \Gamma$ (such that $\exists_{\tau' \in [m_1]} : e_{\tau',\tau} \neq 0$) verify $\mathbf{C}_\tau$ as follows:

$$\mathrm{e}\left(\mathbf{C}_\tau, \mathbf{G}_2\right) \stackrel{?}{=} \mathrm{e}\left(\prod_{i \in I} \left(\mathbf{C}_{\tau_i}\right)^{b_{\tau,i}}, \mathbf{G}_2\right) \cdot \prod_{j \in [n]} \mathrm{e}\left(\prod_{i \in I} \left(\mathbf{C}_{\tau_i}\right)^{b_{\tau,i,j}}, \overline{\mathbf{H}}_j\right) \ . \tag{6.28}$$

  3. (Check for the elements $\{\mathbf{C}_\tau\}_{\tau \in \overline{\Gamma}}$) Let $l := \left|\overline{\Gamma}\right|$. Define any order on the elements in $\overline{\Gamma}$, let $\overline{\Gamma} = (\tau_1, \ldots, \tau_l)$ and compute for all $k \in [l]$:

$$\mathbf{Y}_k := \mathrm{e}\left(\mathbf{C}_{\tau_k}, \mathbf{G}_2\right) \cdot \left(\mathrm{e}\left(\prod_{i \in I} \left(\mathbf{C}_{\tau_i}\right)^{b_{\tau_k,i}}, \mathbf{G}_2\right) \cdot \prod_{j \in [n]} \mathrm{e}\left(\prod_{i \in I} \left(\mathbf{C}_{\tau_i}\right)^{b_{\tau_k,i,j}}, \overline{\mathbf{H}}_j\right) \right)^{-1} \ .$$

  By construction it holds $\mathbf{Y}_k \in {\mathbb{G}_\mathrm{T}}^d$. Set the target matrix of group elements $\mathbf{Y}_T := (\mathbf{Y}_1, \ldots, \mathbf{Y}_l) \in {\mathbb{G}_\mathrm{T}}^{d \times l}$. Let $r := \left|\overline{I}\right|$. Define any order on the coefficients in $\overline{I}$, let $\overline{I} = \left(\bar{i}_1, \ldots, \bar{i}_r\right)$. Set the matrix $\boldsymbol{M} = \left(m_{s,k}\right)_{s \in [r], k \in [l]}$ of coefficients such that $\forall_{s \in [r]}, \forall_{k \in [l]} : m_{s,k} := b_{\tau_k, \bar{i}_s}$ – that is,

$$\boldsymbol{M} := \begin{pmatrix} b_{\tau_1,\bar{i}_1} & b_{\tau_2,\bar{i}_1} & \cdots & b_{\tau_l,\bar{i}_1} \\ b_{\tau_1,\bar{i}_2} & b_{\tau_2,\bar{i}_2} & \cdots & b_{\tau_l,\bar{i}_2} \\ \vdots & \vdots & \ddots & \vdots \\ b_{\tau_1,\bar{i}_r} & b_{\tau_2,\bar{i}_r} & \cdots & b_{\tau_l,\bar{i}_r} \end{pmatrix} \in \mathbb{Z}_p^{r \times l} \ .$$

  Let $\overline{\mathbb{GD}} := (p, (\mathrm{e}(g_1, g_2), \mathbb{G}_\mathrm{T}))$ - this is the group description of $\mathbb{G}_\mathrm{T}$. Check if

$$\mathrm{ExtGauss}\left(\overline{\mathbb{GD}}, \mathbf{Y}_T, \boldsymbol{M}\right) \stackrel{?}{=} 1 \ , \tag{6.29}$$

  where ExtGauss is the algorithm from Lemma 6.4 presented on page 137.

- If all consistency checks in (6.27), in (6.28), and in (6.29) are satisfied, compute

$$X := \prod_{\tau \in [m_1]} \prod_{\tau' \in [w_1]} \mathrm{e}\left(\mathbf{C}_{\tau'}, \mathbf{K}_{\tau}\right)^{e_{\tau,\tau'}} \ ,$$

  and return $C_0 \cdot X^{-1}$.

The decryption in our framework works in the same way as in the CPA-secure framework except for syntactical checks and verification checks which are new. We refer to the following subsections for the guarantees of these checks.

At this point notice that in the decryption algorithm we do not have to check elements in $\overrightarrow{\mathbf{C}}$ corresponding to the index set Γ_0 – that is, for all $i \in I$ the elements $\mathbf{C}_{\tau_i}$ are not checked. Similar to the composite-order framework the elements in $\overrightarrow{\mathbf{C}}$ corresponding to Γ_0 fix the random elements $\boldsymbol{s}_i$ for all $i \in I$. Hence, these elements are used to check all other elements. The elements in Γ are checked one by one directly. The elements in $\overline{\Gamma}$ require a more involved *single* check, since these elements contain additional random elements corresponding to the set $\overline{I}$. On a very high abstract level the last check ensures that for every $i \in \overline{I}$ there exists $\boldsymbol{s}_i$ such that all elements in $\{\mathbf{C}_\tau\}_{\tau \in \overline{\Gamma}}$ are consistent with these elements. For most known schemes the set $\overline{\Gamma}$ is empty, but encodings for ciphertext-policy attributed-based Encryption schemes [Wat11, LW12] require this kind of polynomials.

We also notice that similarly to the composite-order framework we cannot verify if a ciphertext is a correctly generated ciphertext. Rather the verification checks ensure that a ppt adversary is not able to construct a ciphertext which pass the consistency checks but cannot be correctly decrypted except for negligible probability. This makes the decryption oracle useless for such an adversary. The proof of this statement is the main part of the security proof of our framework. In order to make it comprehensible we explicitly state and prove the formal guaranties of our verification in Subsection 6.2.4. However, before we can start with it we have to consider the framework itself.

6.2.2. Semi-Functional Algorithms

In this subsection we define the so-called semi-functional algorithms, that are essential for the proof. The semi-functional public parameters from the original CPA-secure framework [Att16] are extended by the elements corresponding to $\mathbf{U}$ and $\mathbf{V}$. The keys are not changed at all and for the ciphertexts we modify C'' added to the ciphertext in our construction. Hence, all the extensions are very similar to those which we made in the composite-order framework.

Semi-Functional Setup Algorithm

In this subsection we consider the semi-functional setup algorithm.
$\mathbf{SFSetup}\left(1^\lambda, \mathrm{des}\right) \to \left(\mathrm{pp}_\kappa, \mathrm{msk}, \widehat{\mathrm{pp}}_\kappa, \widehat{\mathrm{msk}}\right)$:

- Compute $(\mathrm{pp}_\kappa, \mathrm{msk}) \leftarrow \mathrm{Setup}\left(1^\lambda, \mathrm{des}\right)$. Let $\boldsymbol{B}, \boldsymbol{Z} \in \mathbb{GL}_{p,d+1}$, and $\boldsymbol{H}_1, \ldots, \boldsymbol{H}_n, \boldsymbol{U}, \boldsymbol{V} \in \mathbb{Z}_p^{(d+1)\times(d+1)}$ be as defined in this algorithm.

- Compute additionally

$$\widehat{\mathbf{G}}_1 := g_1^{\boldsymbol{B}\cdot\binom{\mathbf{0}}{1}} \ , \qquad \widehat{\mathbf{G}}_2 := g_2^{\boldsymbol{Z}\cdot\binom{\mathbf{0}}{1}} \ ,$$

  Hence, it holds $\widehat{\mathbf{G}}_1 \in \mathbb{G}_1^{d+1}$ and $\widehat{\mathbf{G}}_2 \in \mathbb{G}_2^{d+1}$.

- Compute:

$$\begin{aligned} \forall_{j\in[n]} : \widehat{\mathbf{H}}_j &:= {}^{\boldsymbol{H}_j}\widehat{\mathbf{G}}_1 \ , & \forall_{j\in[n]} : \widehat{\overline{\mathbf{H}}}_j &:= {}^{\boldsymbol{H}_j^\top}\widehat{\mathbf{G}}_2 \ , \\ \widehat{\mathbf{U}} &:= {}^{\boldsymbol{U}}\widehat{\mathbf{G}}_1 \ , & \widehat{\overline{\mathbf{U}}} &:= {}^{\boldsymbol{U}^\top}\widehat{\mathbf{G}}_2 \ , \\ \widehat{\mathbf{V}} &:= {}^{\boldsymbol{V}}\widehat{\mathbf{G}}_1 \ , & \widehat{\overline{\mathbf{V}}} &:= {}^{\boldsymbol{V}^\top}\widehat{\mathbf{G}}_2 \ . \end{aligned}$$

  By construction $\widehat{\mathbf{H}}_1, \ldots, \widehat{\mathbf{H}}_n, \widehat{\mathbf{U}}, \widehat{\mathbf{V}} \in \mathbb{G}_1^{d+1}$ and $\widehat{\overline{\mathbf{H}}}_1, \ldots, \widehat{\overline{\mathbf{H}}}_n, \widehat{\overline{\mathbf{U}}}, \widehat{\overline{\mathbf{V}}} \in \mathbb{G}_2^{d+1}$.

- Compute $\widehat{\mathbf{Y}} := \mathrm{e}\left(\widehat{\mathbf{G}}_1, g_2^{\boldsymbol{\alpha}}\right) \in \mathbb{G}_\mathrm{T}$.

- Output pp_κ, msk as well as

$$\widehat{\mathrm{pp}_\kappa} := \left(\widehat{\mathbf{Y}}, \widehat{\mathbf{G}}_1, \left\{\widehat{\mathbf{H}}_j\right\}_{j\in[n]}, \widehat{\mathbf{U}}, \widehat{\mathbf{V}}\right) \ ,$$

$$\widehat{\mathrm{msk}} := \left(\widehat{\mathbf{G}}_2, \left\{\widehat{\overline{\mathbf{H}}}_j\right\}_{j\in[n]}\right) \ , \text{ and}$$

$$\widehat{\mathrm{aux}} := \left(\widehat{\overline{\mathbf{U}}}, \widehat{\overline{\mathbf{V}}}\right) \ .$$

The semi-function public parameters $\widehat{\mathrm{pp}_\kappa}$ are used for the definition of semi-functional encryption, $\widehat{\mathrm{msk}}$ is required to define semi-functional keys whereas $\widehat{\mathrm{aux}}$ is not used at all. We remark that all semi-functional elements are used only in the proof, and hence $\widehat{\mathrm{pp}_\kappa}$ and $\widehat{\mathrm{msk}}$ are just the names for the corresponding tuples.

Semi-Functional Keys

Next we recall the semi-functional key generation algorithm and three types of secret keys presented by Attrapadung [Att16] in our notation. We do not have to modify this algorithm and the keys at all.

SFKeyGen $\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{msk}, \widehat{\mathrm{msk}}, \mathrm{kInd}, \mathrm{type}\right)$ with $\mathrm{kInd} \in \mathbb{X}_\kappa$, $\mathrm{type} \in [3]$:

- Parse $\mathrm{pp}_\kappa = \left(\mathrm{des}, \mathbb{GD}, \mathrm{H}, \mathbf{G}_1, \{\mathbf{H}_j\}_{j\in[n]}, \mathbf{U}, \mathbf{V}, \mathbf{Y}, \mathbf{G}_2, \left\{\overline{\mathbf{H}}_j\right\}_{j\in[n]}, \overline{\mathbf{U}}, \overline{\mathbf{V}}\right)$ and the semi-functional master secret key $\widehat{\mathrm{msk}} = \left(\widehat{\mathbf{G}}_2, \left\{\widehat{\overline{\mathbf{H}}}_j\right\}_{j\in[n]}\right)$.

- Compute $\mathrm{sk} := \left(\mathrm{kInd}, \vec{\mathbf{K}}\right) \leftarrow \mathrm{KeyGen}\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{kInd}\right)$. Let $(\boldsymbol{k}, m_2) := \mathrm{Enc1}\left(\kappa, \mathrm{kInd}\right)$, $m_1 := |\boldsymbol{k}|$.

- Choose $\beta, \hat{r}_1, \ldots, \hat{r}_{m_2} \leftarrow \mathbb{Z}_p$, and compute for every $\tau \in [m_1]$ depending on type:

$$\widehat{\mathbf{K}}_\tau := \begin{cases} \widehat{\mathbf{G}}_2^{\sum_{i\in[m_2]} a_{\tau,i}\cdot\hat{r}_i} \cdot \prod_{j\in[n]} \widehat{\overline{\mathbf{H}}}_j^{\sum_{i\in[m_2]} a_{\tau,i,j}\cdot\hat{r}_i} & \text{if type} = 1 \\ \widehat{\mathbf{G}}_2^{a_\tau\cdot\beta+\sum_{i\in[m_2]} a_{\tau,i}\cdot\hat{r}_i} \cdot \prod_{j\in[n]} \widehat{\overline{\mathbf{H}}}_j^{\sum_{i\in[m_2]} a_{\tau,i,j}\cdot\hat{r}_i} & \text{if type} = 2 \\ \widehat{\mathbf{G}}_2^{a_\tau\cdot\beta} & \text{if type} = 3 \ , \end{cases} \tag{6.30}$$

  where $k_\tau = a_\tau \cdot \mathrm{X}_\alpha + \sum_{i\in[m_2]} \left(a_{\tau,i} \cdot \mathrm{X}_{r_i} + \sum_{j\in[n]} a_{\tau,i,j} \cdot \mathrm{X}_{h_j} \cdot \mathrm{X}_{r_i}\right)$ is the τ'th polynomial of $\boldsymbol{k}$.

- Set $\vec{\mathbf{K}}' = \left(\mathbf{K}_1 \cdot \widehat{\mathbf{K}}_1, \ldots, \mathbf{K}_{m_1} \cdot \widehat{\mathbf{K}}_{m_1}\right)$ and output $\widehat{\mathrm{sk}} := \left(\mathrm{kInd}, \vec{\mathbf{K}}'\right)$.

Note that as required it holds $\widehat{\mathrm{sk}} \in \mathbb{SK}_{\mathrm{kInd}}$.

We notice that depending on the type of the key different inputs of the algorithm are used. In order to make this circumstance explicit, we establish the following notation:

$$\begin{cases} \mathrm{sk} \leftarrow \mathrm{SFKeyGen}\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{msk}, \widehat{\mathrm{msk}}, \mathrm{kInd}, 1\right) & \text{if type} = 1 \\ \mathrm{sk} \leftarrow \mathrm{SFKeyGen}\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{msk}, \widehat{\mathrm{msk}}, \mathrm{kInd}, 2; \beta\right) & \text{if type} = 2 \\ \mathrm{sk} \leftarrow \mathrm{SFKeyGen}\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{msk}, \left(\widehat{\mathbf{G}}_2, \_\right), \mathrm{kInd}, 3; \beta\right) & \text{if type} = 3 \end{cases} .$$

That is, we explicitly denote the random element β as input if this element is used. Furthermore, the notation for type 3 keys makes explicit the fact that values $\left\{\widehat{\overline{\mathbf{H}}}_j\right\}_{j\in[n]} \in \widehat{\mathrm{msk}}$ are not required for the computation of these keys. We will use this notation in formal statements about the user secret keys of the corresponding types.

Semi-Functional Ciphertexts

Next we show the extended semi-functional encryption algorithm.
$\mathbf{SFEnc}\left(1^\lambda, \mathrm{pp}_\kappa, \widehat{\mathrm{pp}}_\kappa, \mathrm{cInd}, m\right)$ with $\mathrm{cInd} \in \mathbb{Y}_\kappa$ and $m \in \mathcal{M} = \mathbb{G}_\mathrm{T}$:

- Parse $\mathrm{pp}_\kappa = \left(\mathrm{des}, \mathbb{GD}, \mathrm{H}, \mathbf{G}_1, \{\mathbf{H}_j\}_{j\in[n]}, \mathbf{U}, \mathbf{V}, \mathbf{Y}, \mathbf{G}_2, \left\{\overline{\mathbf{H}}_j\right\}_{j\in[n]}, \overline{\mathbf{U}}, \overline{\mathbf{V}}\right)$ and the semi-functional public parameters $\widehat{\mathrm{pp}}_\kappa = \left(\widehat{\mathbf{Y}}, \widehat{\mathbf{G}}_1, \left\{\widehat{\mathbf{H}}_j\right\}_{j\in[n]}, \widehat{\mathbf{U}}, \widehat{\mathbf{V}}\right)$.
- Choose $\boldsymbol{s}_0 \leftarrow \mathbb{Z}_p^d$ and compute $\mathrm{CT} := \left(\mathrm{cInd}, C_0, \overrightarrow{\mathbf{C}}, \_\right) \leftarrow \mathrm{Enc}\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{cInd}, m; \boldsymbol{s}_0\right)$. Let $(\boldsymbol{c}, w_2) := \mathrm{Enc2}\left(\kappa, \mathrm{cInd}\right)$, $w_1 := |\boldsymbol{c}|$.
- Choose additionally $\hat{s}_0, \ldots, \hat{s}_{w_2} \leftarrow \mathbb{Z}_p$ and compute for all $\tau \in [w_1]$:

$$\widehat{\mathbf{C}}_\tau := \widehat{\mathbf{G}}_1^{\sum_{i\in[w_2]_0} b_{\tau,i}\cdot\hat{s}_i} \cdot \prod_{j\in[n]} \widehat{\mathbf{H}}_j^{\sum_{i\in I} b_{\tau,i,j}\cdot\hat{s}_i} \in \mathbb{G}_1^{d+1} \quad , \tag{6.31}$$

  where $c_\tau = \sum_{i\in[w_2]_0} b_{\tau,i}\cdot \mathrm{X}_{s_i} + \sum_{i\in I}\sum_{j\in[n]} b_{\tau,i,j}\cdot \mathrm{X}_{h_j}\cdot \mathrm{X}_{s_i}$ is the τ'th polynomial of $\boldsymbol{c}$. Furthermore, compute

$$\widehat{C}_0 := \widehat{\mathbf{Y}}^{\hat{s}_0} \in \mathbb{G}_\mathrm{T} \quad . \tag{6.32}$$

  Set $C_0' := C_0 \cdot \widehat{C}_0$, and $\overrightarrow{\mathbf{C}}' := \left(\mathbf{C}_1 \cdot \widehat{\mathbf{C}}_1, \ldots, \mathbf{C}_{w_1} \cdot \widehat{\mathbf{C}}_{w_1}\right)$.
- Compute the hash value $t := \mathrm{H}\left(\mathrm{cInd}, C_0', \overrightarrow{\mathbf{C}}'\right) \in \mathbb{Z}_p$ and then (using $\boldsymbol{s}_0, \hat{s}_0$ from above)

$$\mathbf{C}'' := \left(\mathbf{U}^t \cdot \mathbf{V}\right)^{\boldsymbol{s}_0} \cdot \left(\widehat{\mathbf{U}}^t \cdot \widehat{\mathbf{V}}\right)^{\hat{s}_0} \in \mathbb{G}_1^{d+1} \quad . \tag{6.33}$$

- Output $\widehat{\mathrm{CT}} = \left(\mathrm{cInd}, C_0', \overrightarrow{\mathbf{C}}', \mathbf{C}''\right)$.

Notice that it holds $\widehat{\mathrm{CT}} \in \mathbb{C}_{\mathrm{cInd}}$.

6.2.3. Properties of Public Parameters, User Secret Keys, and Ciphertexts

In this subsection we state and prove certain properties of the scheme. Most of these properties were already used in the CPA-secure framework, but these properties were not explicitly proved or even stated. We expand the corresponding statements according to our extensions and present formal statements. We require this formal treatment of the properties, since our CCA-secure framework is more sophisticated in comparison to the CPA-secure framework and our verification checks exploit the structural properties of the ciphertext.

Public Parameters

Let us consider the normal and the semi-functional public parameters of the scheme. By the definition of algorithm SFSetup it uses the normal setup algorithm Setup as a subroutine. The following lemma states the properties for both algorithms.

Lemma 6.12. *Let* $\mathbb{GD}$, $\tilde{\boldsymbol{D}}$, $\mathbf{G}_1$, $\mathbf{G}_2$, $\left\{\mathbf{H}_j, \overline{\mathbf{H}}_j\right\}_{j\in[n]}$ *be as defined in* Setup *and* $\widehat{\mathbf{G}}_1$, $\widehat{\mathbf{G}}_2$, $\left\{\widehat{\mathbf{H}}_j, \widehat{\overline{\mathbf{H}}}_j\right\}_{j\in[n]}$ *be as additionally defined in* SFSetup. *Then, the following properties are satisfied:*

1. *Normal components:*
   a) $\mathrm{e}\left(\mathbf{G}_1, \mathbf{G}_2\right) = \mathrm{e}\left(g_1, g_2\right)^{\tilde{\boldsymbol{D}}^\top} \in {\mathbb{G}_\mathrm{T}}^{d\times d}$.
   b) $\mathrm{e}\left(\mathbf{G}_1, \overline{\mathbf{H}}_j\right) = \mathrm{e}\left(\mathbf{H}_j, \mathbf{G}_2\right) \in {\mathbb{G}_\mathrm{T}}^{d\times d}$.
2. *Normal components as the first parameter and semi-functional components as the second parameter:*
   a) $\mathrm{e}\left(\mathbf{G}_1, \widehat{\mathbf{G}}_2\right) = \left(1_{\mathbb{G}_\mathrm{T}}, \ldots, 1_{\mathbb{G}_\mathrm{T}}\right) \in {\mathbb{G}_\mathrm{T}}^{1\times d}$.
   b) $\mathrm{e}\left(\mathbf{G}_1, \widehat{\overline{\mathbf{H}}}_j\right) = \mathrm{e}\left(\mathbf{H}_j, \widehat{\mathbf{G}}_2\right) \in {\mathbb{G}_\mathrm{T}}^{1\times d}$.
3. *Semi-functional components as the first parameter and normal components as the second parameter:*
   a) $\mathrm{e}\left(\widehat{\mathbf{G}}_1, \mathbf{G}_2\right) = \left(1_{\mathbb{G}_\mathrm{T}}, \ldots, 1_{\mathbb{G}_\mathrm{T}}\right)^\top \in {\mathbb{G}_\mathrm{T}}^{d}$.
   b) $\mathrm{e}\left(\widehat{\mathbf{G}}_1, \overline{\mathbf{H}}_j\right) = \mathrm{e}\left(\widehat{\mathbf{H}}_j, \mathbf{G}_2\right) \in {\mathbb{G}_\mathrm{T}}^{d}$.
4. *Semi-functional components:*
   a) $\mathrm{e}\left(\widehat{\mathbf{G}}_1, \widehat{\mathbf{G}}_2\right) = \mathrm{e}\left(g_1, g_2\right) \in \mathbb{G}_\mathrm{T}$.
   b) $\mathrm{e}\left(\widehat{\mathbf{G}}_1, \widehat{\overline{\mathbf{H}}}_j\right) = \mathrm{e}\left(\widehat{\mathbf{H}}_j, \widehat{\mathbf{G}}_2\right) \in \mathbb{G}_\mathrm{T}$.

Proof. All equations hold by the definition of the corresponding elements and the proof is presented on page 168. □

Next consider some further properties of the group elements contained in the public parameters. The following lemma can be proved using previously stated general results from Subsection 6.1.2.

Lemma 6.13. *Let* $\mathbb{GD}$, $\boldsymbol{B}$, $\boldsymbol{Z}$, $\mathbf{G}_1$, *and* $\mathbf{G}_2$ *be as defined in* Setup *and* $\widehat{\mathbf{G}}_1$, $\widehat{\mathbf{G}}_2$ *be as defined in* SFSetup. *Then, the following properties are satisfied:*

1. *For the generators* $g_1 \in \mathbb{G}_1$ *and* $g_2 \in \mathbb{G}_2$ *defined in* $\mathbb{GD}$ *it holds*

$$g_1^{\boldsymbol{B}} = \begin{pmatrix} \mathbf{G}_1 & \widehat{\mathbf{G}}_1 \end{pmatrix} \in \mathbb{G}_1^{(d+1)\times(d+1)} \ ,$$

$$g_2^{\boldsymbol{Z}} = \begin{pmatrix} \mathbf{G}_2 & \widehat{\mathbf{G}}_2 \end{pmatrix} \in \mathbb{G}_2^{(d+1)\times(d+1)} \ .$$

2. For every $\mathbf{W}_1 \in \mathbb{G}_1^{d+1}$ *there exist unique* $\boldsymbol{w}_1 \in \mathbb{Z}_p^d$ *and* $\hat{w}_1 \in \mathbb{Z}_p$ *such that*

$$\mathbf{G}_1^{\boldsymbol{w}_1} \cdot \widehat{\mathbf{G}}_1^{\hat{w}_1} = \mathbf{W}_1 \ ,$$

3. For every $\mathbf{W}_2 \in \mathbb{G}_2^{d+1}$ *there exist unique* $\boldsymbol{w}_2 \in \mathbb{Z}_p^d$ *and* $\hat{w}_2 \in \mathbb{Z}_p$ *such that*

$$\mathbf{G}_2^{\boldsymbol{w}_2} \cdot \widehat{\mathbf{G}}_2^{\hat{w}_2} = \mathbf{W}_2 \ .$$

Proof. The first statement holds by construction of the scheme. The second and the third statements holds by Lemma 6.1 using the first statement, since $\boldsymbol{B}, \boldsymbol{Z} \in \mathbb{GL}_{d,p+1}$. □

Normal and Semi-Functional User Secret Keys

In the following lemma we prove that our definitions of the key generation algorithm corresponds to the definitions of the corresponding algorithms by Attrapadung. This lemma will be used to argue about the correctness of the scheme in Subsection 6.2.6.

At first, let us consider the normal keys.

Lemma 6.14. *Let* $\lambda \in \mathbb{N}$, $\mathrm{des} \in \Omega$, $(\mathrm{pp}_\kappa, \mathrm{msk}) \in \left[\mathrm{Setup}\left(1^\lambda, \mathrm{des}\right)\right]$, *and* $\mathrm{kInd} \in \mathbb{X}_\kappa$ *be arbitrary but fixed,* $(\boldsymbol{k}, m_2) := \mathrm{Enc1}\left(\kappa, \mathrm{kInd}\right)$. *Furthermore, let* $\boldsymbol{\alpha}$, $\overline{\mathbb{H}}$, *and* $\boldsymbol{Z}$ *be as defined by the computation of* $(\mathrm{pp}_\kappa, \mathrm{msk})$. *Then, for every* $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_{m_2} \in \mathbb{Z}_p^d$ *and for* $\left(\mathrm{kInd}, \overrightarrow{\mathbf{K}}\right) := \mathrm{KeyGen}\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{kInd}; \boldsymbol{r}_1, \ldots, \boldsymbol{r}_{m_2}\right)$ *it holds*

$$\overrightarrow{\mathbf{K}} = g_2^{\boldsymbol{k}\left(\boldsymbol{\alpha}, \boldsymbol{R}, \overline{\mathbb{H}}\right)} \ ,$$

where

$$\boldsymbol{R} := \left(\boldsymbol{Z} \cdot \begin{pmatrix} \boldsymbol{r}_1 \\ 0 \end{pmatrix}, \ldots, \boldsymbol{Z} \cdot \begin{pmatrix} \boldsymbol{r}_{m_2} \\ 0 \end{pmatrix}\right) \ .$$

Proof. Consider the setting defined in the lemma. By definition of pair encodings, every polynomial $k_\tau \in \boldsymbol{k}$ has the form:

$$k_\tau = a_\tau \cdot \mathrm{X}_\alpha + \sum_{i \in [m_2]} \left(a_{\tau,i} \cdot \mathrm{X}_{r_i} + \sum_{j \in [n]} a_{\tau,i,j} \cdot \mathrm{X}_{h_j} \cdot \mathrm{X}_{r_i} \right) \ ,$$

where the order of variables in monomials $\mathrm{X}_{h_j} \cdot \mathrm{X}_{r_i}$ is important. Let $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_{m_2} \in \mathbb{Z}_p^d$ be arbitrary but fixed. Then, for every $\tau \in [m_1]$ it holds

$$k_\tau\left(\boldsymbol{\alpha}, \boldsymbol{R}, \overline{\mathbb{H}}\right) = a_\tau \cdot \boldsymbol{\alpha} + \sum_{i \in [m_2]} \left(a_{\tau,i} \cdot \boldsymbol{Z} \cdot \begin{pmatrix} \boldsymbol{r}_i \\ 0 \end{pmatrix} + \sum_{j \in [n]} a_{\tau,i,j} \cdot \boldsymbol{H}_j^\top \cdot \boldsymbol{Z} \cdot \begin{pmatrix} \boldsymbol{r}_i \\ 0 \end{pmatrix} \right) \ .$$

And we deduce

$$g_2^{k_\tau\left(\boldsymbol{\alpha}, \boldsymbol{R}, \overline{\mathbb{H}}\right)} = g_2^{a_\tau \cdot \boldsymbol{\alpha} + \sum_{i \in [m_2]} \left(a_{\tau,i} \cdot \boldsymbol{Z} \cdot \left(\begin{smallmatrix} \boldsymbol{r}_i \\ 0 \end{smallmatrix}\right) + \sum_{j \in [n]} a_{\tau,i,j} \cdot \boldsymbol{H}_j^\top \cdot \boldsymbol{Z} \cdot \left(\begin{smallmatrix} \boldsymbol{r}_i \\ 0 \end{smallmatrix}\right) \right)}$$

$$
\begin{aligned}
&= g_2^{a_\tau \cdot \boldsymbol{\alpha}} \cdot g_2^{\sum_{i\in[m_2]} a_{\tau,i} \cdot \boldsymbol{Z} \cdot \left(\begin{smallmatrix}\boldsymbol{I}_d\\ \boldsymbol{0}\end{smallmatrix}\right) \cdot \boldsymbol{r}_i} \cdot g_2^{\sum_{i\in[m_2]} \sum_{j\in[n]} a_{\tau,i,j} \cdot \boldsymbol{H}_j^\top \cdot \boldsymbol{Z} \cdot \left(\begin{smallmatrix}\boldsymbol{I}_d\\ \boldsymbol{0}\end{smallmatrix}\right) \cdot \boldsymbol{r}_i} \\
&= (g_2^{\boldsymbol{\alpha}})^{a_\tau} \cdot \left(g_2^{\boldsymbol{Z} \cdot \left(\begin{smallmatrix}\boldsymbol{I}_d\\ \boldsymbol{0}\end{smallmatrix}\right)}\right)^{\sum_{i\in[m_2]} a_{\tau,i} \cdot \boldsymbol{r}_i} \cdot \prod_{j\in[n]} \left(g_2^{\boldsymbol{H}_j^\top \cdot \boldsymbol{Z} \cdot \left(\begin{smallmatrix}\boldsymbol{I}_d\\ \boldsymbol{0}\end{smallmatrix}\right)}\right)^{\sum_{i\in[m_2]} a_{\tau,i,j} \cdot \boldsymbol{r}_i} \\
&= (g_2^{\boldsymbol{\alpha}})^{a_\tau} \cdot \mathbf{G}_2^{\sum_{i\in[m_2]} a_{\tau,i} \cdot \boldsymbol{r}_i} \cdot \prod_{j\in[n]} \overline{\mathbf{H}}_j^{\sum_{i\in[m_2]} a_{\tau,i,j} \cdot \boldsymbol{r}_i} \\
&\overset{(6.23)}{=} \mathbf{K}_\tau \in \mathbb{G}_2^{d+1} \quad .
\end{aligned}
$$

Since τ was arbitrary, this proves the lemma. □

Next, let us consider the semi-functional components of the user secret keys.

Lemma 6.15. *Let* $\lambda \in \mathbb{N}$, $\mathrm{des} \in \Omega$, $\left(\mathrm{pp}_\kappa, \mathrm{msk}, \widehat{\mathrm{pp}}_\kappa, \widehat{\mathrm{msk}}\right) \in \left[\mathrm{SFSetup}\left(1^\lambda, \mathrm{des}\right)\right]$, $\mathrm{kInd} \in \mathbb{X}_\kappa$ *be arbitrary but fixed,* $(\boldsymbol{k}, m_2) := \mathrm{Enc1}\left(\kappa, \mathrm{kInd}\right)$. *Furthermore, let* $\overline{\mathbb{H}}$, *and* $\boldsymbol{Z}$ *be as defined by the computation of* $(\mathrm{pp}_\kappa, \mathrm{msk})$. *Then, for every* $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_{m_2} \in \mathbb{Z}_p^d$, $\beta, \hat{r}_1, \ldots, \hat{r}_{m_2} \in \mathbb{Z}_p$, *and for* $\left(\mathrm{kInd}, \overrightarrow{\mathbf{K}'}\right) := \mathrm{SFKeyGen}\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{msk}, \widehat{\mathrm{msk}}, \mathrm{kInd}, \mathit{type}; \boldsymbol{r}_1, \ldots, \boldsymbol{r}_{m_2}, \beta, \hat{r}_1, \ldots, \hat{r}_{m_2}\right)$ *it holds for the semi-functional components generated in* SFKeyGen*:*

$$
\left(\widehat{\mathbf{K}}_1, \ldots, \widehat{\mathbf{K}}_{m_1}\right)^\top = \begin{cases} g_2^{\boldsymbol{k}\left(\boldsymbol{0}, \widehat{\boldsymbol{R}}, \overline{\mathbb{H}}\right)} & \text{if } \mathit{type} = 1 \\ g_2^{\boldsymbol{k}\left(\boldsymbol{Z} \cdot \left(\begin{smallmatrix}\boldsymbol{0}\\ \beta\end{smallmatrix}\right), \widehat{\boldsymbol{R}}, \overline{\mathbb{H}}\right)} & \text{if } \mathit{type} = 2 \\ g_2^{\boldsymbol{k}\left(\boldsymbol{Z} \cdot \left(\begin{smallmatrix}\boldsymbol{0}\\ \beta\end{smallmatrix}\right), \boldsymbol{0}, \boldsymbol{0}\right)} & \text{if } \mathit{type} = 3 \end{cases} ,
$$

where

$$
\widehat{\boldsymbol{R}} = \left(\boldsymbol{Z} \cdot \begin{pmatrix}\boldsymbol{0}\\ \hat{r}_1\end{pmatrix}, \ldots, \boldsymbol{Z} \cdot \begin{pmatrix}\boldsymbol{0}\\ \hat{r}_{m_2}\end{pmatrix}\right) \in \left(\mathbb{Z}_p^{d+1}\right)^{m_2} \quad .
$$

Proof. The proof is similar to the proof of Lemma 6.14 and is presented on page 169. □

Corollary 6.16. Let $\lambda \in \mathbb{N}$, $\mathrm{des} \in \Omega$, $\left(\mathrm{pp}_\kappa, \mathrm{msk}, \widehat{\mathrm{pp}}_\kappa, \widehat{\mathrm{msk}}\right) \in \left[\mathrm{SFSetup}\left(1^\lambda, \mathrm{des}\right)\right]$, $\mathrm{kInd} \in \mathbb{X}_\kappa$ be arbitrary but fixed, $(\boldsymbol{k}, m_2) := \mathrm{Enc1}\left(\kappa, \mathrm{kInd}\right)$. Furthermore, let $\boldsymbol{\alpha}$, $\overline{\mathbb{H}}$, and $\boldsymbol{Z}$ be as defined by the computation of $(\mathrm{pp}_\kappa, \mathrm{msk})$. Then, for every $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_{m_2} \in \mathbb{Z}_p^d$, $\beta, \hat{r}_1, \ldots, \hat{r}_{m_2} \in \mathbb{Z}_p$, and for $\left(\mathrm{kInd}, \overrightarrow{\mathbf{K}'}\right) := \mathrm{SFKeyGen}\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{msk}, \widehat{\mathrm{msk}}, \mathrm{kInd}, \mathrm{type}; \boldsymbol{r}_1, \ldots, \boldsymbol{r}_{m_2}, \beta, \hat{r}_1, \ldots, \hat{r}_{m_2}\right)$ it holds

$$
\overrightarrow{\mathbf{K}'} = \begin{cases} g_2^{\boldsymbol{k}\left(\boldsymbol{\alpha}, \boldsymbol{R}', \overline{\mathbb{H}}\right)} = g_2^{\boldsymbol{k}\left(\boldsymbol{\alpha}, \boldsymbol{R}, \overline{\mathbb{H}}\right) + \boldsymbol{k}\left(\boldsymbol{0}, \widehat{\boldsymbol{R}}, \overline{\mathbb{H}}\right)} & \text{if type} = 1 \\ g_2^{\boldsymbol{k}\left(\boldsymbol{\alpha} + \boldsymbol{Z} \cdot \left(\begin{smallmatrix}\boldsymbol{0}\\ \beta\end{smallmatrix}\right), \boldsymbol{R}', \overline{\mathbb{H}}\right)} = g_2^{\boldsymbol{k}\left(\boldsymbol{\alpha}, \boldsymbol{R}, \overline{\mathbb{H}}\right) + \boldsymbol{k}\left(\boldsymbol{Z} \cdot \left(\begin{smallmatrix}\boldsymbol{0}\\ \beta\end{smallmatrix}\right), \widehat{\boldsymbol{R}}, \overline{\mathbb{H}}\right)} & \\ \qquad = g_2^{\boldsymbol{k}\left(\boldsymbol{\alpha}, \boldsymbol{R}, \overline{\mathbb{H}}\right) + \boldsymbol{k}\left(\boldsymbol{0}, \widehat{\boldsymbol{R}}, \overline{\mathbb{H}}\right) + \boldsymbol{k}\left(\boldsymbol{Z} \cdot \left(\begin{smallmatrix}\boldsymbol{0}\\ \beta\end{smallmatrix}\right), \boldsymbol{0}, \boldsymbol{0}\right)} & \text{if type} = 2 \\ g_2^{\boldsymbol{k}\left(\boldsymbol{\alpha} + \boldsymbol{Z} \cdot \left(\begin{smallmatrix}\boldsymbol{0}\\ \beta\end{smallmatrix}\right), \boldsymbol{R}, \overline{\mathbb{H}}\right)} = g_2^{\boldsymbol{k}\left(\boldsymbol{\alpha}, \boldsymbol{R}, \overline{\mathbb{H}}\right) + \boldsymbol{k}\left(\boldsymbol{Z} \cdot \left(\begin{smallmatrix}\boldsymbol{0}\\ \beta\end{smallmatrix}\right), \boldsymbol{0}, \boldsymbol{0}\right)} & \text{if type} = 3 \end{cases} ,
$$

where

$$
\boldsymbol{R} = \left(\boldsymbol{Z} \cdot \begin{pmatrix}\boldsymbol{r}_1\\ 0\end{pmatrix}, \ldots, \boldsymbol{Z} \cdot \begin{pmatrix}\boldsymbol{r}_{m_2}\\ 0\end{pmatrix}\right) \in \left(\mathbb{Z}_p^{d+1}\right)^{m_2} \quad ,
$$

$$\widehat{\boldsymbol{R}} = \left(\boldsymbol{Z}\cdot\begin{pmatrix}\mathbf{0}\\ \hat{r}_1\end{pmatrix},\ldots,\boldsymbol{Z}\cdot\begin{pmatrix}\mathbf{0}\\ \hat{r}_{m_2}\end{pmatrix}\right)\in\left(\mathbb{Z}_p^{d+1}\right)^{m_2} \quad ,$$

and

$$\boldsymbol{R}' = \boldsymbol{R}+\widehat{\boldsymbol{R}} = \left(\boldsymbol{Z}\cdot\begin{pmatrix}\boldsymbol{r}_1\\ \hat{r}_1\end{pmatrix},\ldots,\boldsymbol{Z}\cdot\begin{pmatrix}\boldsymbol{r}_{m_2}\\ \hat{r}_{m_2}\end{pmatrix}\right)\in\left(\mathbb{Z}_p^{d+1}\right)^{m_2} \quad .$$

Proof. The corollary holds by Lemma 6.14, Lemma 6.15 and the linearity of pair encodings (see Lemma 1.10). □

Alternative (Semi-Functional) KeyGen Algorithm

In this subsection we present an alternative key-generation algorithm that can be used to compute normal keys and semi-functional keys of type 1. This algorithm is used in the proof when keys are computed from the elements contained in the security assumption. The construction of this algorithm goes back to Lemma 5 in [Att15], where this algorithm was implicitly used.

From Corollary 6.16 we deduce that every normal key is a special semi-functional key with $\hat{r}_1 = \cdots = \hat{r}_{m_2} = 0$. Hence, every statement for the semi-functional key can be applied to the normal key if the semi-functional random elements are all set to zero. We present the more general statement for semi-functional keys of type 1.

Lemma 6.17 (Alternative KeyGen algorithm for normal/type 1 keys)**.** *Let* $\lambda\in\mathbb{N}$, $\mathrm{des}\in\Omega$, $\left(\mathrm{pp}_\kappa,\mathrm{msk},\widehat{\mathrm{pp}}_\kappa,\widehat{\mathrm{msk}}\right)\in\left[\mathrm{SFSetup}\left(1^\lambda,\mathrm{des}\right)\right]$, $\mathrm{kInd}\in\mathbb{X}_\kappa$ *be arbitrary but fixed,* $(\boldsymbol{k},m_2) := \mathrm{Enc1}\left(\kappa,\mathrm{kInd}\right)$. *Furthermore, let* $\mathbf{G}_2$, $\widehat{\mathbf{G}}_2$, $\overline{\mathbb{H}}$ *be as defined by the computation of* SFSetup. *Then, there exists a deterministic polynomial time algorithm* KeyGenAlt *which given* $\boldsymbol{k}$, msk, $\overline{\mathbb{H}}$ *and* $\left\{\mathbf{R}_i=\mathbf{G}_2^{\boldsymbol{r}_i}\cdot\widehat{\mathbf{G}}_2^{\hat{r}_i}\right\}_{i\in[m_2]}$ *outputs* $\mathrm{sk}=\left(\mathrm{kInd},\overrightarrow{\mathbf{K}}\right)$ *such that*

$$\mathrm{sk}=\mathrm{SFKeyGen}\left(1^\lambda,\mathrm{pp}_\kappa,\mathrm{msk},\widehat{\mathrm{msk}},\mathrm{kInd},1;\boldsymbol{r}_1,\ldots,\boldsymbol{r}_{m_2},\hat{r}_1,\ldots,\hat{r}_{m_2}\right) \quad ,$$

where $(\boldsymbol{k},m_w)=\mathrm{Enc1}\left(\kappa,\mathrm{kInd}\right)$.

We denote the computation of this algorithm by $\mathrm{sk}:=\mathrm{KeyGenAlt}\left(\boldsymbol{k},\mathrm{msk},\overline{\mathbb{H}},\mathbf{R}_1,\ldots,\mathbf{R}_{m_2}\right)$. $\boldsymbol{k}$ *can be alternatively replaced by* κ *and* kInd.

Proof. By definition of KeyGen and SFKeyGen a semi-functional key of type 1 with randomness $\boldsymbol{r}_1,\ldots,\boldsymbol{r}_{m_2},\hat{r}_1,\ldots,\hat{r}_{m_2}$ can be computed component by component given the elements as defined in the lemma as follows (recall $\overline{\mathbb{H}}:=\left(\boldsymbol{H}_1^\top,\ldots,\boldsymbol{H}_n^\top\right)$, $\overline{\mathbf{H}}_j={\boldsymbol{H}_j^\top}\mathbf{G}_2$, and $\widehat{\overline{\mathbf{H}}}_j={\boldsymbol{H}_j^\top}\widehat{\mathbf{G}}_2$):

$$\begin{aligned}
\mathbf{K}_\tau \overset{\text{by def.}}{=}\ & \mathrm{msk}^{a_\tau}\cdot\mathbf{G}_2^{\sum_{i\in[m_2]}a_{\tau,i}\cdot\boldsymbol{r}_i}\cdot\prod_{j\in[n]}\overline{\mathbf{H}}_j^{\sum_{i\in[m_2]}a_{\tau,i,j}\cdot\boldsymbol{r}_i}\\
&\cdot\widehat{\mathbf{G}}_2^{\sum_{i\in[m_2]}a_{\tau,i}\cdot\hat{r}_i}\cdot\prod_{j\in[n]}\widehat{\overline{\mathbf{H}}}_j^{\sum_{i\in[m_2]}a_{\tau,i,j}\cdot\hat{r}_i}\\
=\ & \mathrm{msk}^{a_\tau}\cdot\prod_{i\in[m_2]}(\mathbf{G}_2^{\boldsymbol{r}_i})^{a_{\tau,i}}\cdot\prod_{j\in[n]}\overline{\mathbf{H}}_j^{\sum_{i\in[m_2]}a_{\tau,i,j}\cdot\boldsymbol{r}_i}\\
&\cdot\prod_{i\in[m_2]}\left(\widehat{\mathbf{G}}_2^{\hat{r}_i}\right)^{a_{\tau,i}}\cdot\prod_{j\in[n]}\widehat{\overline{\mathbf{H}}}_j^{\sum_{i\in[m_2]}a_{\tau,i,j}\cdot\hat{r}_i}
\end{aligned}$$

$$
\begin{aligned}
&= \quad \mathrm{msk}^{a_\tau} \cdot \prod_{i\in[m_2]} \left(\mathbf{G}_2^{\boldsymbol{r}_i} \cdot \widehat{\mathbf{G}}_2^{\hat{r}_i}\right)^{a_{\tau,i}} \cdot \prod_{j\in[n]} \prod_{i\in[m_2]} \left(\boldsymbol{H}_j^\top \mathbf{G}_2^{\boldsymbol{r}_i} \cdot {}^{\boldsymbol{H}_j^\top}\widehat{\mathbf{G}}_2^{\hat{r}_i}\right)^{a_{\tau,i,j}} \\
&= \quad \mathrm{msk}^{a_\tau} \cdot \prod_{i\in[m_2]} \mathbf{R}_i^{a_{\tau,i}} \cdot \prod_{j\in[n]} \prod_{i\in[m_2]} {}^{\boldsymbol{H}_j^\top}\mathbf{R}_i^{a_{\tau,i,j}} \quad ,
\end{aligned}
$$

where $k_\tau = a_\tau \cdot \mathrm{X}_\alpha + \sum_{i\in[m_2]} \left(a_{\tau,i} \cdot \mathrm{X}_{r_i} + \sum_{j\in[n]} a_{\tau,i,j} \cdot \mathrm{X}_{h_j} \cdot \mathrm{X}_{r_i}\right)$ is the τ'th polynomial of $\boldsymbol{k}$.

When κ, kInd are given instead of $\boldsymbol{k}$, the polynomials can be computed by $(\boldsymbol{k}, m_2) := \mathrm{Enc1}(\kappa, \mathrm{kInd})$. □

Normal and Semi-Functional Ciphertexts

Similarly to the previous subsection, we will prove the properties of the ciphertext in this subsection.

Lemma 6.18. *Let* $\lambda \in \mathbb{N}$, $\mathrm{des} \in \Omega$, $(\mathrm{pp}_\kappa, \mathrm{msk}) \in \left[\mathrm{Setup}\left(1^\lambda, \mathrm{des}\right)\right]$, *and* $\mathrm{cInd} \in \mathbb{Y}_\kappa$, $m \in \mathcal{M}$ *be arbitrary but fixed,* $(\boldsymbol{c}, w_2) = \mathrm{Enc2}(\kappa, \mathrm{cInd})$. *Furthermore, let* $\mathbb{H}$ *and* $\boldsymbol{B}$ *be as defined by the computation of* $(\mathrm{pp}_\kappa, \mathrm{msk})$. *Then, for every* $\boldsymbol{s}_0, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_{w_2} \in \mathbb{Z}_p^d$ *and for* $\left(\mathrm{cInd}, C_0', \overrightarrow{\mathbf{C}}', \mathbf{C}''\right) := \mathrm{Enc}\left(1^\lambda, \mathrm{pp}_\kappa, \widehat{\mathrm{pp}}_\kappa, \mathrm{cInd}, m; \boldsymbol{s}_0, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_{w_2}\right)$ *it holds*

$$\overrightarrow{\mathbf{C}} = g_1^{\boldsymbol{c}(\boldsymbol{S},\mathbb{H})} \quad ,$$

where

$$\boldsymbol{S} := \left(\boldsymbol{B} \cdot \begin{pmatrix}\boldsymbol{s}_0\\0\end{pmatrix}, \boldsymbol{B} \cdot \begin{pmatrix}\boldsymbol{s}_1\\0\end{pmatrix}, \ldots, \boldsymbol{B} \cdot \begin{pmatrix}\boldsymbol{s}_{w_2}\\0\end{pmatrix}\right) \quad .$$

Proof. The proof is analogous to the proof of Lemma 6.14. Consider the setting defined in the lemma. By the definition of regular pair encoding schemes, every polynomial $c_\tau \in \boldsymbol{c}$ has the form

$$c_\tau = \sum_{i\in[w_2]_0} b_{\tau,i} \cdot \mathrm{X}_{s_i} + \sum_{i\in I} \sum_{j\in[n]} b_{\tau,i,j} \cdot \mathrm{X}_{h_j} \cdot \mathrm{X}_{s_i} \quad ,$$

where the order of variables in monomials $\mathrm{X}_{h_j} \cdot \mathrm{X}_{s_i}$ is important. Let $\boldsymbol{s}_0, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_{w_2} \in \mathbb{Z}_p^d$ be arbitrary but fixed. Let additionally $\mathbf{G}_1$, $\{\boldsymbol{H}_j, \mathbf{H}_j\}_{j\in[n]}$ be as defined by the computation of $(\mathrm{pp}_\kappa, \mathrm{msk})$. Then, for every $\tau \in [w_1]$ it holds (recall $I \subseteq [w_2]_0$)

$$c_\tau(\boldsymbol{S}, \mathbb{H}) = \sum_{i\in[w_2]_0} b_{\tau,i} \cdot \boldsymbol{B} \cdot \begin{pmatrix}\boldsymbol{s}_i\\0\end{pmatrix} + \sum_{i\in I} \sum_{j\in[n]} b_{\tau,i,j} \cdot \boldsymbol{H}_j \cdot \boldsymbol{B} \cdot \begin{pmatrix}\boldsymbol{s}_i\\0\end{pmatrix} \quad .$$

Hence, we deduce

$$
\begin{aligned}
g_1^{c_\tau(\boldsymbol{S},\mathbb{H})} &= g_1^{\sum_{i\in[w_2]_0} b_{\tau,i}\cdot\boldsymbol{B}\cdot\left(\begin{smallmatrix}\boldsymbol{s}_i\\0\end{smallmatrix}\right) + \sum_{i\in I}\sum_{j\in[n]} b_{\tau,i,j}\cdot\boldsymbol{H}_j\cdot\boldsymbol{B}\cdot\left(\begin{smallmatrix}\boldsymbol{s}_i\\0\end{smallmatrix}\right)} \\
&= g_1^{\sum_{i\in[w_2]_0} b_{\tau,i}\cdot\boldsymbol{B}\cdot\left(\begin{smallmatrix}\boldsymbol{I}_d\\\mathbf{0}\end{smallmatrix}\right)\cdot\boldsymbol{s}_i} \cdot g_1^{\sum_{i\in I}\sum_{j\in[n]} b_{\tau,i,j}\cdot\boldsymbol{H}_j\cdot\boldsymbol{B}\cdot\left(\begin{smallmatrix}\boldsymbol{I}_d\\\mathbf{0}\end{smallmatrix}\right)\cdot\boldsymbol{s}_i} \\
&= \left(g_1^{\boldsymbol{B}\cdot\left(\begin{smallmatrix}\boldsymbol{I}_d\\\mathbf{0}\end{smallmatrix}\right)}\right)^{\sum_{i\in[w_2]_0} b_{\tau,i}\cdot\boldsymbol{s}_i} \cdot \prod_{j\in[n]} \left(g_1^{\boldsymbol{H}_j\cdot\boldsymbol{B}\cdot\left(\begin{smallmatrix}\boldsymbol{I}_d\\\mathbf{0}\end{smallmatrix}\right)}\right)^{\sum_{i\in I} b_{\tau,i,j}\cdot\boldsymbol{s}_i} \\
&= \mathbf{G}_1^{\sum_{i\in[w_2]_0} b_{\tau,i}\cdot\boldsymbol{s}_i} \cdot \prod_{j\in[n]} \mathbf{H}_j^{\sum_{i\in I} b_{\tau,i,j}\cdot\boldsymbol{s}_i} \\
&\overset{(6.24)}{=} \mathbf{C}_\tau \in \mathbb{G}_1^{d+1} \quad .
\end{aligned}
$$

Since τ was arbitrary, the second claim is correct. □

Next, consider the semi-functional ciphertexts.

Lemma 6.19. *Let* $\lambda \in \mathbb{N}$, $\mathrm{des} \in \Omega$, $\left(\mathrm{pp}_\kappa, \mathrm{msk}, \widehat{\mathrm{pp}}_\kappa, \widehat{\mathrm{msk}}\right) \in \left[\mathrm{SFSetup}\left(1^\lambda, \mathrm{des}\right)\right]$, *ciphertext index* $\mathrm{cInd} \in \mathbb{Y}_\kappa$, *and message* $m \in \mathcal{M}$ *be arbitrary but fixed,* $(\boldsymbol{c}, w_2) := \mathrm{Enc2}\left(\kappa, \mathrm{cInd}\right)$, $w_1 := |\boldsymbol{c}|$. *Furthermore, let* $\mathbb{H}$ *and* $\boldsymbol{B}$ *be as defined by the computation of* $(\mathrm{pp}_\kappa, \mathrm{msk})$. *Then, for every vector* $\boldsymbol{s}_0, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_{w_2} \in \mathbb{Z}_p^d$, *every* $\hat{s}_0, \hat{s}_1, \ldots, \hat{s}_{w_2} \in \mathbb{Z}_p$ *and for ciphertext* $\left(\mathrm{cInd}, C_0, \overrightarrow{\mathbf{C}}, \mathbf{C}''\right) := \mathrm{SFEnc}\left(1^\lambda, \mathrm{pp}_\kappa, \widehat{\mathrm{pp}}_\kappa, \mathrm{cInd}, m; \boldsymbol{s}_0, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_{w_2}, \hat{s}_0, \hat{s}_1, \ldots, \hat{s}_{w_2}\right)$ *it holds for the semi-functional components generated in* SFEnc*:*

$$\left(\widehat{\mathbf{C}}_1, \ldots, \widehat{\mathbf{C}}_{w_1}\right)^\top = g_1^{\boldsymbol{c}\left(\widehat{\boldsymbol{S}}, \mathbb{H}\right)} \ ,$$

where

$$\widehat{\boldsymbol{S}} = \left(\boldsymbol{B} \cdot \begin{pmatrix} \mathbf{0} \\ \hat{s}_0 \end{pmatrix}, \boldsymbol{B} \cdot \begin{pmatrix} \mathbf{0} \\ \hat{s}_1 \end{pmatrix}, \ldots, \boldsymbol{B} \cdot \begin{pmatrix} \mathbf{0} \\ \hat{s}_{w_2} \end{pmatrix}\right) \in \left(\mathbb{Z}_p^{d+1}\right)^{w_2+1} \ .$$

Proof. The proof is similar to the proof of Lemma 6.18 and is presented on page 170. □

Corollary 6.20. Let $\lambda \in \mathbb{N}$, $\mathrm{des} \in \Omega$, $\left(\mathrm{pp}_\kappa, \mathrm{msk}, \widehat{\mathrm{pp}}_\kappa, \widehat{\mathrm{msk}}\right) \in \left[\mathrm{SFSetup}\left(1^\lambda, \mathrm{des}\right)\right]$, ciphertext index $\mathrm{cInd} \in \mathbb{Y}_\kappa$, and message $m \in \mathcal{M}$ be arbitrary but fixed, $(\boldsymbol{c}, w_2) := \mathrm{Enc2}\left(\kappa, \mathrm{cInd}\right)$, $w_1 := |\boldsymbol{c}|$. Furthermore, let $\mathbb{H}$ and $\boldsymbol{B}$ be as defined by the computation of $(\mathrm{pp}_\kappa, \mathrm{msk})$. Then, for every $\boldsymbol{s}_0, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_{w_2} \in \mathbb{Z}_p^d$ and every $\hat{s}_0, \hat{s}_1, \ldots, \hat{s}_{w_2} \in \mathbb{Z}_p$ and for $\left(\mathrm{cInd}, C_0, \overrightarrow{\mathbf{C}}, \mathbf{C}''\right) := \mathrm{SFEnc}\left(1^\lambda, \mathrm{pp}_\kappa, \widehat{\mathrm{pp}}_\kappa, \mathrm{cInd}, m; \boldsymbol{s}_0, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_{w_2}, \hat{s}_0, \hat{s}_1, \ldots, \hat{s}_{w_2}\right)$ it holds

$$\overrightarrow{\mathbf{C}}' = g_1^{\boldsymbol{c}\left(\boldsymbol{S}', \mathbb{H}\right)} = g_1^{\boldsymbol{c}\left(\boldsymbol{S}+\widehat{\boldsymbol{S}}, \mathbb{H}\right)} \ ,$$

where

$$\boldsymbol{S} := \left(\boldsymbol{B} \cdot \begin{pmatrix} \boldsymbol{s}_0 \\ 0 \end{pmatrix}, \boldsymbol{B} \cdot \begin{pmatrix} \boldsymbol{s}_1 \\ 0 \end{pmatrix}, \ldots, \boldsymbol{B} \cdot \begin{pmatrix} \boldsymbol{s}_{w_2} \\ 0 \end{pmatrix}\right) \in \left(\mathbb{Z}_p^{d+1}\right)^{w_2+1} \ ,$$

$$\widehat{\boldsymbol{S}} = \left(\boldsymbol{B} \cdot \begin{pmatrix} \mathbf{0} \\ \hat{s}_0 \end{pmatrix}, \boldsymbol{B} \cdot \begin{pmatrix} \mathbf{0} \\ \hat{s}_1 \end{pmatrix}, \ldots, \boldsymbol{B} \cdot \begin{pmatrix} \mathbf{0} \\ \hat{s}_{w_2} \end{pmatrix}\right) \in \left(\mathbb{Z}_p^{d+1}\right)^{w_2+1} \ ,$$

and

$$\boldsymbol{S}' = \boldsymbol{S} + \widehat{\boldsymbol{S}} = \left(\boldsymbol{B} \cdot \begin{pmatrix} \boldsymbol{s}_0 \\ \hat{s}_0 \end{pmatrix}, \boldsymbol{B} \cdot \begin{pmatrix} \boldsymbol{s}_1 \\ \hat{s}_1 \end{pmatrix}, \ldots, \boldsymbol{B} \cdot \begin{pmatrix} \boldsymbol{s}_{w_2} \\ \hat{s}_{w_2} \end{pmatrix}\right) \ .$$

Proof. The corollary holds by Lemma 6.18, Lemma 6.19 and the linearity of pair encodings (cf. Lemma 1.10). □

Alternative (Semi-Functional) Encryption Algorithm

In this subsection we present an alternative encryption algorithm which can be used to compute normal and semi-functional ciphertexts. This algorithm is used in the proof when keys are computed from the elements contained in the security assumption. The construction of this algorithm goes back to Lemma 4 in [Att15], where this algorithm was implicitly used.

From Corollary 6.20 we deduce that every normal ciphertext is a special semi-functional ciphertext with $\hat{s}_0 = \hat{s}_1 = \cdots = \hat{s}_{w_2} = 0$. Hence, every statement for the semi-functional ciphertexts can be applied to the normal ciphertext if the semi-functional random elements are all set to zero. The following lemma first of all shows how the indirect and the shared elements of a ciphertext (which corresponds to Γ and $\overline{\Gamma}$, respectively) can be expressed as extensions of the direct elements (the elements corresponding to Γ_0). This lemma is also used to explain our verification checks and was not considered in [Att16].

Lemma 6.21 (Alternative view at ciphertexts.)**.** *Let* $\lambda \in \mathbb{N}$, $\mathrm{des} \in \Omega$, $\left(\mathrm{pp}_\kappa, \mathrm{msk}, \widehat{\mathrm{pp}}_\kappa, \widehat{\mathrm{msk}}\right) \in \left[\mathrm{SFSetup}\left(1^\lambda, \mathrm{des}\right)\right]$, $\mathrm{cInd} \in \mathbb{Y}_\kappa$ *and* $m \in \mathcal{M}$ *be arbitrary. Furthermore, let* $\mathbf{G}_1$, $\widehat{\mathbf{G}}_1$, $\boldsymbol{H}_j$, $\boldsymbol{U}$ *and* $\boldsymbol{V}$ *be as defined by* SFSetup*. Let* $(\boldsymbol{c}, w_2) := \mathrm{Enc2}\left(\kappa, \mathrm{cInd}\right)$, $w_1 := |\boldsymbol{c}|$ *and for every* $\tau \in [w_1]$ *the polynomial* $c_\tau \in \boldsymbol{c}$ *be as follows*

$$c_\tau = \sum_{i \in [w_2]_0} \left(b_{\tau,i} \cdot \mathrm{X}_{s_i} + \sum_{j \in [n]} b_{\tau,i,j} \cdot \mathrm{X}_{h_j} \cdot \mathrm{X}_{s_i} \right) \quad .$$

Then, for every $\boldsymbol{s}_0, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_{w_2} \in \mathbb{Z}_p^d$ *and* $\hat{s}_0, \hat{s}_1, \ldots, \hat{s}_{w_2} \in \mathbb{Z}_p$ *the ciphertext* $\left(\mathrm{cInd}, C_0, \overrightarrow{\mathbf{C}}, \mathbf{C}''\right) := \mathrm{SFEnc}\left(1^\lambda, \mathrm{pp}_\kappa, \widehat{\mathrm{pp}}_\kappa, \mathrm{cInd}, m; \boldsymbol{s}_0, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_{w_2} \hat{s}_0, \hat{s}_1, \ldots, \hat{s}_{w_2}\right)$ *satisfies the following statements:*

1. *For every* $\tau \in \Gamma_0$ *with corresponding index* $i \in I$ *it holds (denote such an index* τ *by* τ_i*)*

$$\mathbf{C}_{\tau_i} = \mathbf{G}_1^{\boldsymbol{s}_i} \cdot \widehat{\mathbf{G}}_1^{\hat{s}_i} \quad .$$

   In particular $\mathbf{C}_1 = \mathbf{G}_1^{\boldsymbol{s}_0} \cdot \widehat{\mathbf{G}}_1^{\hat{s}_0}$*.*

2. *For every* $\tau \in \Gamma$ *it holds*

$$\mathbf{C}_\tau = \prod_{i \in I} \left(\mathbf{C}_{\tau_i}\right)^{b_{\tau,i}} \cdot \prod_{j \in [n]} \prod_{i \in I} {\boldsymbol{H}_j^\top} \left(\mathbf{C}_{\tau_i}\right)^{b_{\tau,i,j}} \quad .$$

3. *For every* $\tau \in \overline{\Gamma}$ *it holds*

$$\mathbf{C}_\tau = \widehat{\mathbf{G}}_1^{\sum_{i \in \overline{I}} b_{\tau,i} \cdot \hat{s}_i} \cdot \prod_{i \in \overline{I}} \left(\mathbf{G}_1^{\boldsymbol{s}_i}\right)^{b_{\tau,i}} \cdot \prod_{i \in I} \left(\mathbf{C}_{\tau_i}\right)^{b_{\tau,i}} \cdot \prod_{j \in [n]} \prod_{i \in I} {\boldsymbol{H}_j^\top} \left(\mathbf{C}_{\tau_i}\right)^{b_{\tau,i,j}} \quad .$$

4. *For* C_0 *it holds*

$$C_0 = m \cdot \mathrm{e}\left(\mathbf{C}_1, \mathrm{msk}\right) \quad .$$

5. *For* $\mathbf{C}''$ *it holds*

$$\mathbf{C}'' = {}^{t \cdot \boldsymbol{U} + \boldsymbol{V}} \mathbf{C}_1 \quad ,$$

   where $t := \mathrm{H}\left(\mathrm{cInd}, C_0, \overrightarrow{\mathbf{C}}\right)$*.*

Proof. See the proof on page 170. □

Lemma 6.22 (Algorithm EncAlt given msk)**.** *Let* $\lambda \in \mathbb{N}$, $\mathrm{des} \in \Omega$, $\left(\mathrm{pp}_\kappa, \mathrm{msk}, \widehat{\mathrm{pp}}_\kappa, \widehat{\mathrm{msk}}\right) \in \left[\mathrm{SFSetup}\left(1^\lambda, \mathrm{des}\right)\right]$, $\mathrm{cInd} \in \mathbb{Y}_\kappa$ *and* $m \in \mathcal{M}$ *be arbitrary. Furthermore, let* $\mathbf{G}_1$, $\widehat{\mathbf{G}}_1$, $\mathbb{H}$, $\boldsymbol{U}$, $\boldsymbol{V}$, *and* H *be as defined by the computation of* $\left(\mathrm{pp}_\kappa, \mathrm{msk}, \widehat{\mathrm{pp}}_\kappa, \widehat{\mathrm{msk}}\right)$. *Let* $\mathrm{cInd} \in \mathbb{Y}_\kappa$, $(\boldsymbol{c}, w_2) := \mathrm{Enc2}\left(\kappa, \mathrm{cInd}\right)$. *Then, there exists a deterministic polynomial time algorithm* EncAlt *which given* $\boldsymbol{c}$, msk, cInd, m, $\mathbb{H}$, $\boldsymbol{U}$, $\boldsymbol{V}$, H, *and* $\left\{\mathbf{S}_i = \mathbf{G}_1^{\boldsymbol{s}_i} \cdot \widehat{\mathbf{G}}_1^{\hat{s}_i}\right\}_{i \in [w_2]_0}$ *outputs* $\mathrm{CT} = \left(\mathrm{cInd}, C_0, \overrightarrow{\mathbf{C}}, \mathbf{C}''\right)$ *such that*

$$\mathrm{CT} = \mathrm{SFEnc}\left(1^\lambda, \mathrm{pp}_\kappa, \widehat{\mathrm{pp}}_\kappa, \mathrm{cInd}, m; \boldsymbol{s}_0, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_{w_2} \hat{s}_0, \hat{s}_1, \ldots, \hat{s}_{w_2}\right) \ .$$

We denote by $\left(\mathrm{cInd}, C_0, \overrightarrow{\mathbf{C}}, \mathbf{C}''\right) := \mathrm{EncAlt}\left(\boldsymbol{c}, \mathrm{msk}, m, \mathbb{H}, \boldsymbol{U}, \boldsymbol{V}, \mathrm{H}, \mathbf{S}_0, \ldots, \mathbf{S}_{w_2}\right)$ *the computation of this algorithm. Encoding* $\boldsymbol{c}$ *can be alternatively replaced by* κ *and* cInd.

Proof. Denote for every $\tau \in [w_1]$ the polynomial $c_\tau \in \boldsymbol{c}$ by

$$c_\tau = \sum_{i \in [w_2]_0} \left(b_{\tau,i} \cdot \mathrm{X}_{s_i} + \sum_{j \in [n]} b_{\tau,i,j} \cdot \mathrm{X}_{h_j} \cdot \mathrm{X}_{s_i} \right) \ .$$

Furthermore, recall that $\mathbb{H} := (\boldsymbol{H}_1, \ldots, \boldsymbol{H}_n)$.

$\mathbf{C}_{\tau_i}$ for $\tau_i \in \Gamma_0$ can be computed according to Lemma 6.21

$$\begin{aligned} \forall_{\tau_i \in \Gamma_0} : \mathbf{C}_{\tau_i} &= \mathbf{G}_1^{\boldsymbol{s}_i} \cdot \widehat{\mathbf{G}}_1^{\hat{s}_i} \\ &= \mathbf{S}_i \ . \end{aligned}$$

Next, the elements $\mathbf{C}_\tau$ for $\tau \in \Gamma$ can be computed from these elements due to Lemma 6.21:

$$\forall_{\tau \in \Gamma} : \mathbf{C}_\tau = \prod_{i \in I} \left(\mathbf{C}_{\tau_i}\right)^{b_{\tau,i}} \cdot \prod_{j \in [n]} \prod_{i \in I} {\boldsymbol{H}_j^\top} \left(\mathbf{C}_{\tau_i}\right)^{b_{\tau,i,j}} \ .$$

For $\tau \in \overline{\Gamma}$ we have we deduce from Lemma 6.21:

$$\begin{aligned} \mathbf{C}_\tau &= \widehat{\mathbf{G}}_1^{\sum_{i \in \overline{I}} b_{\tau,i} \cdot \hat{s}_i} \cdot \prod_{i \in \overline{I}} \left(\mathbf{G}_1^{\boldsymbol{s}_i}\right)^{b_{\tau,i}} \cdot \prod_{i \in I} \left(\mathbf{C}_{\tau_i}\right)^{b_{\tau,i}} \cdot \prod_{j \in [n]} \prod_{i \in I} {\boldsymbol{H}_j^\top} \left(\mathbf{C}_{\tau_i}\right)^{b_{\tau,i,j}} \\ &= \prod_{i \in \overline{I}} \left(\mathbf{G}_1^{\boldsymbol{s}_i} \cdot \widehat{\mathbf{G}}_1^{\hat{s}_i}\right)^{b_{\tau,i}} \cdot \prod_{i \in I} \mathbf{S}_i^{b_{\tau,i}} \cdot \prod_{j \in [n]} \prod_{i \in I} {\boldsymbol{H}_j^\top} \mathbf{S}_i^{b_{\tau,i,j}} \\ &= \prod_{i \in [w_2]_0} \mathbf{S}_i^{b_{\tau,i}} \cdot \prod_{j \in [n]} \prod_{i \in I} {\boldsymbol{H}_j^\top} \mathbf{S}_i^{b_{\tau,i,j}} \ . \end{aligned}$$

Finally, C_0 and $\mathbf{C}''$ can be computed from $\mathbf{C}_1 = \mathbf{S}_0$ according to Lemma 6.21. □

6.2.4. Guarantees of the Verification Checks

In this subsection we present formal statements that explain our consistency checks. First of all the following lemma applies Corollary 6.3 to the settings of the scheme. Based on this lemma we directly explain the guaranties of the verification checks for $\mathbf{C}''$ and $\{\mathbf{C}_\tau\}_{\tau \in \Gamma}$ in the following two lemmata. In Lemma 6.26 we explain our last verification check for $\{\mathbf{C}_\tau\}_{\tau \in \overline{\Gamma}}$ based on Lemma 6.4.

Lemma 6.23. *Let d be an integer and $\mathbb{GD} = (p, (g_1, \mathbb{G}_1), (g_2, \mathbb{G}_2), \mathbb{G}_T, \mathrm{e})$ be asymmetric bilinear groups of prime order p. Let $\boldsymbol{B} \in \mathbb{GL}_{p,d+1}$ and $\tilde{\boldsymbol{D}} \in \mathbb{GL}_{p,d}$ be arbitrary but fixed, $\widehat{\mathbf{G}}_1 := g_1^{\boldsymbol{B}\cdot\binom{\mathbf{0}}{1}} \in \mathbb{G}_1^{d+1}$ and $\mathbf{G}_2 := g_2^{\boldsymbol{B}^{-\top}\cdot\binom{\tilde{\boldsymbol{D}}}{\mathbf{0}}} \in \mathbb{G}_2^{(d+1)\times d}$. Suppose that $\mathbf{X} = g_1^{\boldsymbol{x}} \in \mathbb{G}_1^{d+1}$ and $\mathbf{Y} = g_1^{\boldsymbol{y}} \in \mathbb{G}_1^{d+1}$ satisfy $\mathbf{X} \neq \mathbf{Y}$ and*

$$\mathrm{e}(\mathbf{X}, \mathbf{G}_2) = \mathrm{e}(\mathbf{Y}, \mathbf{G}_2) \in {\mathbb{G}_T}^{d\times 1} \ .$$

Then, there exist $\hat{u} \in \mathbb{Z}_p^*$ such that

$$\mathbf{X} \cdot \mathbf{Y}^{-1} = \widehat{\mathbf{G}}_1^{\hat{u}} \ .$$

Consequently, $\mathbf{X} = g_1^{\boldsymbol{x}} \in \mathbb{G}_1^{d+1}$ and $\mathbf{Y} = g_1^{\boldsymbol{y}} \in \mathbb{G}_1^{d+1}$ satisfy $\mathrm{e}(\mathbf{X}, \mathbf{G}_2) = \mathrm{e}(\mathbf{Y}, \mathbf{G}_2)$ if and only if there is $\xi \in \mathbb{Z}_p$ such that $\mathbf{Y} = \mathbf{X} \cdot \widehat{\mathbf{G}}_1^{\xi}$.

Proof. The first observation is that the d columns of $\boldsymbol{B}^{-\top} \cdot \binom{\tilde{\boldsymbol{D}}}{\mathbf{0}} \in \mathbb{Z}_p^{(d+1)\times d}$ are linearly independent since $\boldsymbol{B} \in \mathbb{GL}_{p,d+1}$ and $\tilde{\boldsymbol{D}} \in \mathbb{GL}_{p,d}$. Hence, by Lemma 6.2 and since $\mathbf{X} \neq \mathbf{Y}$ we deduce from the given equation in the lemma:

$$\mathbf{0} \neq \boldsymbol{u} := \boldsymbol{x} - \boldsymbol{y} \in \ker\left(\left(\tilde{\boldsymbol{D}}^{\top} \quad \mathbf{0}\right) \cdot \boldsymbol{B}^{-1}\right) \ .$$

Hence, $\boldsymbol{B}^{-1} \cdot \boldsymbol{u} \in \ker\left(\left(\tilde{\boldsymbol{D}}^{\top} \ \mathbf{0}\right)\right)$ and furthermore, $\boldsymbol{B}^{-1} \cdot \boldsymbol{u} \neq \mathbf{0}$ (since $\boldsymbol{u} \neq \mathbf{0}$ and $\boldsymbol{B}^{-1} \in \mathbb{GL}_{p,d+1}$). However, the kernel of $\left(\tilde{\boldsymbol{D}}^{\top} \ \mathbf{0}\right)$ has dimension one, namely $\ker\left(\left(\tilde{\boldsymbol{D}}^{\top} \ \mathbf{0}\right)\right) = \left\langle (0, \cdots, 0, 1)^{\top} \right\rangle$. We deduce that there exist $\hat{u} \in \mathbb{Z}_p^*$ such that

$$\begin{aligned} \boldsymbol{B}^{-1} \cdot (\boldsymbol{x} - \boldsymbol{y}) &= \boldsymbol{B}^{-1} \cdot \boldsymbol{u} \\ &= (0, \cdots, 0, \hat{u})^{\top} \ . \end{aligned}$$

Hence, $\boldsymbol{x} - \boldsymbol{y} = \boldsymbol{B} \cdot (0, \cdots, 0, \hat{u})^{\top} = \hat{u} \cdot \boldsymbol{B} \cdot (0, \cdots, 0, 1)^{\top}$ and it holds:

$$\begin{aligned} \mathbf{X} \cdot \mathbf{Y}^{-1} &= g_1^{\boldsymbol{x}-\boldsymbol{y}} \\ &= \left(g_1^{\boldsymbol{B}\cdot\binom{\mathbf{0}}{1}}\right)^{\hat{u}} \\ &= \widehat{\mathbf{G}}_1^{\hat{u}} \ . \end{aligned}$$

This proves the lemma. □

The following two lemmata are direct consequences of Lemma 6.23.

Lemma 6.24. *Let $\lambda \in \mathbb{N}$, $\mathrm{des} \in \Omega$, $(\mathrm{pp}_\kappa, \mathrm{msk}) \in \left[\mathrm{Setup}\left(1^\lambda, \mathrm{des}\right)\right]$, $\mathrm{cInd} \in \mathbb{Y}_\kappa$, and $\mathrm{CT} = \left(\mathrm{cInd}, C_0, \overrightarrow{\mathbf{C}}, \mathbf{C}''\right) \in \mathbb{C}_{\mathrm{cInd}}$ be arbitrary but fixed. Furthermore, let $\boldsymbol{B} = (\boldsymbol{B}_d \ \boldsymbol{b}_{d+1}) \in \mathbb{GL}_{p,d+1}$ be as defined by the computation of $(\mathrm{pp}_\kappa, \mathrm{msk})$. Then, CT satisfies the verification check in (6.27) if and only if there exists $\xi \in \mathbb{Z}_p$ such that*

$$\mathbf{C}'' = {}^{t\cdot\boldsymbol{U}+\boldsymbol{V}}\mathbf{C}_1 \cdot \widehat{\mathbf{G}}_1^{\xi} \ ,$$

where $t = \mathrm{H}\left(\mathrm{cInd}, C_0, \overrightarrow{\mathbf{C}}\right)$ is the corresponding hash value and $\widehat{\mathbf{G}}_1 = g_1^{\boldsymbol{B}\cdot\binom{\mathbf{0}}{1}} = g_1^{\boldsymbol{b}_{d+1}}$.

Proof. If the check in (6.27) for $\mathbf{C}''$ is satisfied, than it holds

$$\begin{aligned} \mathrm{e}\left(\mathbf{C}'', \mathbf{G}_2\right) &\stackrel{!}{=} \mathrm{e}\left(\mathbf{C}_1, \overline{\mathbf{U}}^t \cdot \overline{\mathbf{V}}\right) \\ &= \mathrm{e}\left(\mathbf{C}_1, {}^{t\cdot\boldsymbol{U}^\top+\boldsymbol{V}^\top}\mathbf{G}_2\right) \\ &= \mathrm{e}\left({}^{t\cdot\boldsymbol{U}+\boldsymbol{V}}\mathbf{C}_1, \mathbf{G}_2\right) \quad . \end{aligned}$$

By Lemma 6.23 the check is satisfied if and only if there exists $\xi \in \mathbb{Z}_p$ such that

$$\mathbf{C}'' = {}^{t\cdot\boldsymbol{U}+\boldsymbol{V}}\mathbf{C}_1 \cdot \widehat{\mathbf{G}}_1^{\xi} \quad .$$

This proves the lemma. □

Lemma 6.25. *Let* $\lambda \in \mathbb{N}$, $\mathrm{des} \in \Omega$, $(\mathrm{pp}_\kappa, \mathrm{msk}) \in \left[\mathrm{Setup}\left(1^\lambda, \mathrm{des}\right)\right]$, $\mathrm{cInd} \in \mathbb{Y}_\kappa$, *and* $\mathrm{CT} = \left(\mathrm{cInd}, C_0, \overrightarrow{\mathbf{C}}, \mathbf{C}''\right) \in \mathbb{C}_{\mathrm{cInd}}$ *be arbitrary but fixed. Furthermore, let* $\boldsymbol{B} = (\,\boldsymbol{B}_d\ \boldsymbol{b}_{d+1}\,) \in \mathbb{GL}_{p,d+1}$ *be as defined by the computation of* $(\mathrm{pp}_\kappa, \mathrm{msk})$. *Then,* CT *satisfies the verification check in* (6.28) *if and only if for every* $\tau \in \Gamma$ *there exists* $\xi_\tau \in \mathbb{Z}_p$ *such that*

$$\mathbf{C}_\tau = \widehat{\mathbf{G}}_1^{\xi_\tau} \cdot \prod_{i\in I} (\mathbf{C}_{\tau_i})^{b_{\tau,i}} \cdot \prod_{j\in[n]} \prod_{i\in I} {}^{\boldsymbol{H}_j}(\mathbf{C}_{\tau_i})^{b_{\tau,i,j}} \quad ,$$

where $\widehat{\mathbf{G}}_1 = g_1^{\boldsymbol{B}\cdot\binom{\mathbf{0}}{1}} = g_1^{\boldsymbol{b}_{d+1}}$.

Proof. Let $\tau \in \Gamma$ be arbitrary but fixed. If the verification checks in (6.28) for τ is satisfied, then it holds:

$$\begin{aligned} \mathrm{e}\left(\mathbf{C}_\tau, \mathbf{G}_2\right) &\stackrel{!}{=} \mathrm{e}\left(\prod_{i\in I} (\mathbf{C}_{\tau_i})^{b_{\tau,i}}, \mathbf{G}_2\right) \cdot \prod_{j\in[n]} \mathrm{e}\left(\prod_{i\in I} (\mathbf{C}_{\tau_i})^{b_{\tau,i,j}}, \overline{\mathbf{H}}_j\right) \quad , \\ &= \mathrm{e}\left(\prod_{i\in I} (\mathbf{C}_{\tau_i})^{b_{\tau,i}}, \mathbf{G}_2\right) \cdot \mathrm{e}\left(\prod_{j\in[n]} \prod_{i\in I} {}^{\boldsymbol{H}_j}(\mathbf{C}_{\tau_i})^{b_{\tau,i,j}}, \mathbf{G}_2\right) \\ &= \mathrm{e}\left(\prod_{i\in I} (\mathbf{C}_{\tau_i})^{b_{\tau,i}} \cdot \prod_{j\in[n]} \prod_{i\in I} {}^{\boldsymbol{H}_j}(\mathbf{C}_{\tau_i})^{b_{\tau,i,j}}, \mathbf{G}_2\right) \quad . \end{aligned}$$

By Lemma 6.23 the check is satisfied if and only if there exists $\xi_\tau \in \mathbb{Z}_p$ such that

$$\mathbf{C}_\tau = \widehat{\mathbf{G}}_1^{\xi_\tau} \cdot \prod_{i\in I} (\mathbf{C}_{\tau_i})^{b_{\tau,i}} \cdot \prod_{j\in[n]} \prod_{i\in I} {}^{\boldsymbol{H}_j}(\mathbf{C}_{\tau_i})^{b_{\tau,i,j}} \quad .$$

This proves the lemma. □

In the following lemma we explain the last check based on Lemma 6.23 and on the properties of Algorithm ExtGauss from Lemma 6.4.

Lemma 6.26. *Let* $\lambda \in \mathbb{N}$, $\mathrm{des} \in \Omega$, $(\mathrm{pp}_\kappa, \mathrm{msk}) \in \left[\mathrm{Setup}\left(1^\lambda, \mathrm{des}\right)\right]$, $\mathrm{cInd} \in \mathbb{Y}_\kappa$, *and* $\mathrm{CT} = \left(\mathrm{cInd}, C_0, \overrightarrow{\mathbf{C}}, \mathbf{C}''\right) \in \mathbb{C}_{\mathrm{cInd}}$ *be arbitrary but fixed. Furthermore, let* $\boldsymbol{B} = (\, \boldsymbol{B}_d \; \boldsymbol{b}_{d+1}\,) \in \mathbb{GL}_{p,d+1}$ *be as defined by the computation of* $(\mathrm{pp}_\kappa, \mathrm{msk})$. *Then,* CT *satisfies the verification check in* (6.29) *if and only if for every* $i \in \overline{I}$ *there exists* $\boldsymbol{s}_i \in \mathbb{Z}_p^d$ *such that for every* $\tau \in \overline{\Gamma}$ *there exists* $\xi_\tau \in \mathbb{Z}_p$ *and it holds*

$$\mathbf{C}_\tau = \widehat{\mathbf{G}}_1^{\xi_\tau} \cdot \prod_{i \in \overline{I}} (\mathbf{G}_1^{\boldsymbol{s}_i})^{b_{\tau_k,i}} \cdot \prod_{i \in I} (\mathbf{C}_{\tau_i})^{b_{\tau,i}} \cdot \prod_{j \in [n]} \prod_{i \in I} {}^{\boldsymbol{H}_j}(\mathbf{C}_{\tau_i})^{b_{\tau,i,j}} \quad ,$$

where $\widehat{\mathbf{G}}_1 = g_1^{\boldsymbol{B} \cdot \binom{\mathbf{0}}{1}} = g_1^{\boldsymbol{b}_{d+1}}$.

Proof. In preparation for the last check in (6.29), for all $\tau_k \in \overline{\Gamma} = \{\tau_1, \ldots, \tau_l\}$ the decryption algorithm computes from $\mathbf{C}_{\tau_k}$ an element $\mathbf{Y}_k \in {\mathbb{G}_\mathrm{T}}^d$ as follows:

$$\begin{aligned}
\mathbf{Y}_k &= \mathrm{e}\left(\mathbf{C}_{\tau_k}, \mathbf{G}_2\right) \cdot \left(\mathrm{e}\left(\prod_{i \in I} (\mathbf{C}_{\tau_i})^{b_{\tau_k,i}}, \mathbf{G}_2 \right) \cdot \prod_{j \in [n]} \mathrm{e}\left(\prod_{i \in I} (\mathbf{C}_{\tau_i})^{b_{\tau_k,i,j}}, \overline{\mathbf{H}}_j \right) \right)^{-1} \\
&= \mathrm{e}\left(\mathbf{C}_{\tau_k}, \mathbf{G}_2\right) \cdot \mathrm{e}\left(\prod_{i \in I} (\mathbf{C}_{\tau_i})^{b_{\tau,i}} \cdot \prod_{j \in [n]} \prod_{i \in I} {}^{\boldsymbol{H}_j}(\mathbf{C}_{\tau_i})^{b_{\tau,i,j}}, \mathbf{G}_2 \right)^{-1} \\
&= \mathrm{e}\left(\mathbf{C}_{\tau_k} \cdot \prod_{i \in I} (\mathbf{C}_{\tau_i})^{-b_{\tau,i}} \cdot \prod_{j \in [n]} \prod_{i \in I} {}^{\boldsymbol{H}_j}(\mathbf{C}_{\tau_i})^{-b_{\tau,i,j}}, \mathbf{G}_2 \right) \quad .
\end{aligned}$$

Let us consider the last term and in particular the first input of the pairing, an element of $\mathbb{G}_1^{d+1}$ by construction. Recall that every $\mathbf{X} \in \mathbb{G}_1^{d+1}$ can be written as $\mathbf{G}_1^{\boldsymbol{x}} \cdot \widehat{\mathbf{G}}_1^{\hat{x}}$ for uniquely defined $\boldsymbol{x} \in \mathbb{Z}_p^d, \hat{x} \in \mathbb{Z}_p$ by Lemma 6.13. Hence, for every $k \in [l]$ there exist unique $\boldsymbol{y}_k \in \mathbb{Z}_p^d$ and $\hat{y}_k \in \mathbb{Z}_p$ such that

$$\mathbf{G}_1^{\boldsymbol{y}_k} \cdot \widehat{\mathbf{G}}_1^{\hat{y}_k} = \mathbf{C}_{\tau_k} \cdot \prod_{i \in I} (\mathbf{C}_{\tau_i})^{-b_{\tau,i}} \cdot \prod_{j \in [n]} \prod_{i \in I} {}^{\boldsymbol{H}_j}(\mathbf{C}_{\tau_i})^{-b_{\tau,i,j}} \quad ,$$

and we can write every $\mathbf{C}_{\tau_k}$ as

$$\mathbf{C}_{\tau_k} = \mathbf{G}_1^{\boldsymbol{y}_k} \cdot \widehat{\mathbf{G}}_1^{\hat{y}_k} \cdot \prod_{i \in I} (\mathbf{C}_{\tau_i})^{b_{\tau,i}} \cdot \prod_{j \in [n]} \prod_{i \in I} {}^{\boldsymbol{H}_j}(\mathbf{C}_{\tau_i})^{b_{\tau,i,j}} \quad .$$

We conclude

$$\begin{aligned}
\mathbf{Y}_k &= \mathrm{e}\left(\mathbf{C}_{\tau_k} \cdot \prod_{i \in I} (\mathbf{C}_{\tau_i})^{-b_{\tau,i}} \cdot \prod_{j \in [n]} \prod_{i \in I} {}^{\boldsymbol{H}_j}(\mathbf{C}_{\tau_i})^{-b_{\tau,i,j}}, \mathbf{G}_2 \right) \\
&= \mathrm{e}\left(\mathbf{G}_1^{\boldsymbol{y}_k} \cdot \widehat{\mathbf{G}}_1^{\hat{y}_k}, \mathbf{G}_2 \right) \\
&= \mathrm{e}\left(\mathbf{G}_1, \mathbf{G}_2\right)^{\boldsymbol{y}_k} \\
&= \mathrm{e}\left(g_1, g_2\right)^{\tilde{\boldsymbol{D}}^\top \cdot \boldsymbol{y}_k} \quad ,
\end{aligned}$$

where the last equation holds by Lemma 6.12.

In the next step the decryption algorithm executes ExtGauss on input $(\mathbb{G}_\mathrm{T}, \mathbf{Y}_T, \boldsymbol{M})$, where description of $\mathbb{G}_\mathrm{T}$ contains generator $\mathrm{e}\left(g_1, g_2\right)$, $\mathbf{Y}_T = (\mathbf{Y}_1, \ldots, \mathbf{Y}_t) \in {\mathbb{G}_\mathrm{T}}^{d \times t}$ and $\boldsymbol{M}$ contains the coefficients of the polynomials $c_{\tau_i} \in \boldsymbol{c}$ for $\tau_i \in \overline{I}$. Namely, for $r := \left|\overline{I}\right|$ an arbitrary order of

coefficients in $\overline{I}$ is considered, let say $\overline{I} = \left(\bar{i}_1, \ldots, \bar{i}_r\right)$. Matrix $\boldsymbol{M} = \left(m_{s,k}\right) \in \mathbb{Z}_p^{r \times l}$ of coefficients is then set to $\forall_{l \in [r]} \forall_{k \in [t]} : m_{l,k} := b_{\tau_k, \bar{i}_l}$ – that is,

$$\boldsymbol{M} = \begin{pmatrix} b_{\tau_1,\bar{i}_1} & b_{\tau_2,\bar{i}_1} & \cdots & b_{\tau_l,\bar{i}_1} \\ b_{\tau_1,\bar{i}_2} & b_{\tau_2,\bar{i}_2} & \cdots & b_{\tau_l,\bar{i}_2} \\ \vdots & \vdots & \ddots & \vdots \\ b_{\tau_1,\bar{i}_r} & b_{\tau_2,\bar{i}_r} & \cdots & b_{\tau_l,\bar{i}_r} \end{pmatrix} = \begin{pmatrix} \boldsymbol{m}_1 & \boldsymbol{m}_2 & \ldots & \boldsymbol{m}_l \end{pmatrix} .$$

By Lemma 6.4 the check in (6.29) is satisfied if and only if there exists $\boldsymbol{S'} \in \mathbb{Z}_p^{d \times r}$ such that

$$\begin{aligned} \mathbf{Y}_T = \mathrm{e}\left(g_1, g_2\right)^{\boldsymbol{S'} \cdot \boldsymbol{M}} &\Leftrightarrow \\ \mathrm{e}\left(g_1, g_2\right)^{\tilde{\boldsymbol{D}}^\top \cdot \left(\boldsymbol{y}_1 \ \boldsymbol{y}_2 \ \cdots \ \boldsymbol{y}_l\right)} = \mathrm{e}\left(g_1, g_2\right)^{\boldsymbol{S'} \cdot \boldsymbol{M}} &\Leftrightarrow \\ \tilde{\boldsymbol{D}}^\top \cdot \begin{pmatrix} \boldsymbol{y}_1 & \boldsymbol{y}_2 & \ldots & \boldsymbol{y}_l \end{pmatrix} = \boldsymbol{S'} \cdot \boldsymbol{M} & . \end{aligned}$$

$\tilde{\boldsymbol{D}} \in \mathbb{GL}_{p,d}$, and hence the check is satisfied if and only if there exists $\boldsymbol{S} = \tilde{\boldsymbol{D}}^{-\top} \cdot \boldsymbol{S'} = \left(\boldsymbol{s}_1 \ \boldsymbol{s}_2 \ \cdots \ \boldsymbol{s}_r\right) \in \mathbb{Z}_p^{d \times r}$ such that

$$\begin{aligned} \begin{pmatrix} \boldsymbol{y}_1 & \boldsymbol{y}_2 & \ldots & \boldsymbol{y}_l \end{pmatrix} = \boldsymbol{S} \cdot \boldsymbol{M} &\Leftrightarrow \\ \forall_{k \in [l]} : \boldsymbol{y}_k = \boldsymbol{S} \cdot \boldsymbol{m}_k = \sum_{i \in \overline{I}} b_{\tau_k, i} \cdot \boldsymbol{s}_i & . \end{aligned} \tag{6.34}$$

In summary, the last check is satisfied if and only if for every $i \in \overline{I}$ there exists $\boldsymbol{s}_i \in \mathbb{Z}_p^d$ such that for every $\tau_k \in \overline{\Gamma}$ it holds

$$\begin{aligned} \mathbf{C}_{\tau_k} &= \mathbf{G}_1^{\boldsymbol{y}_k} \cdot \widehat{\mathbf{G}}_1^{\hat{y}_k} \cdot \prod_{i \in I} \left(\mathbf{C}_{\tau_i}\right)^{b_{\tau,i}} \cdot \prod_{j \in [n]} \prod_{i \in I} {}^{\boldsymbol{H}_j} \left(\mathbf{C}_{\tau_i}\right)^{b_{\tau,i,j}} \\ &\overset{(6.34)}{=} \widehat{\mathbf{G}}_1^{\hat{y}_k} \cdot \prod_{i \in \overline{I}} \mathbf{G}_1^{b_{\tau_k,i} \cdot \boldsymbol{s}_i} \cdot \prod_{i \in I} \left(\mathbf{C}_{\tau_i}\right)^{b_{\tau,i}} \cdot \prod_{j \in [n]} \prod_{i \in I} {}^{\boldsymbol{H}_j} \left(\mathbf{C}_{\tau_i}\right)^{b_{\tau,i,j}} . \end{aligned}$$

This proves the lemma. □

Lemma 6.27. *Let* $\lambda \in \mathbb{N}$, $\mathrm{des} \in \Omega$, $\left(\mathrm{pp}_\kappa, \mathrm{msk}, \widehat{\mathrm{pp}}_\kappa, \widehat{\mathrm{msk}}\right) \in \left[\mathrm{SFSetup}\left(1^\lambda, \mathrm{des}\right)\right]$, $\mathrm{cInd} \in \mathbb{Y}_\kappa$, $m \in \mathcal{M}$, *and* $\mathrm{CT}' \in \left[\mathrm{SFEnc}\left(1^\lambda, \mathrm{pp}_\kappa, \widehat{\mathrm{pp}}_\kappa, \mathrm{cInd}, m\right)\right]$ *be arbitrary but fixed. Then,* CT *satisfies all verification checks. As a special case every* $\mathrm{CT} \in \left[\mathrm{Enc}\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{cInd}, m\right)\right]$ *satisfies all verification checks.*

Proof. The statement holds by Lemma 6.21. It is easy to verify that all group elements of the correctly generated (semi-functional) ciphertext fit the form ensured by Lemma 6.24, by Lemma 6.25, and by Lemma 6.26. □

6.2.5. Intuition Behind the Construction

Given the statement of Lemma 6.21 as well as statements of Lemma 6.24, Lemma 6.25, and Lemma 6.26 we explain the intuition behind our construction. Our explanation in this subsection leaves out many important details of the formal proof and can be used as a blueprint for the construction of consistency checks in other contexts.

From the result in the previous subsection we deduce that consistency checks ensure that every element $\mathbf{C}_i$ of the ciphertext (including $\mathbf{C}''$) has the same form as in a correctly generated

(semi-functional) ciphertext except for additional factor $\widehat{\mathbf{G}}_1^{\xi_i}$ for some $\xi_i \in \mathbb{Z}_p$. Assume for a moment that all ξ_i are equal zero. Then, the given ciphertext is indeed a correctly generated (semi-functional) ciphertext (with $\forall_{i \in \overline{I}} : \hat{s}_i = 0 \pmod p$). Decryption of such a ciphertext using a correctly generated normal user secret key results in the correct message (see the following subsection). This is the first useful fact that we use in the proof. Notice that this alone does not at all ensure the CCA-security, since the adversary still could learn something using the decryption oracle (see the details below). Rather, this is the reason for the reduction costs comparable to the reduction costs of the underlying CPA-secure framework (cf. Section 5.1).

Next, consider the case that for every $\mathrm{cInd} \in \mathbb{Y}_\kappa$ set $\overline{I}$ for the corresponding ciphertext encoding is empty (many pair encoding schemes satisfy this property). That is, the check in (6.29) is not required. We claim that in this case due to the Matrix Decisional Diffie-Hellman Assumption a ppt adversary is not able to compute (except for negligible probability) a ciphertext which passes all consistency checks such that there is an element $\mathbf{C}_i$ with $\xi_i \neq 0 \pmod p$. Hence, for this case we can drop our assumption about ξ_i's from above. Intuitively, if we look e.g. at the form of $\mathbf{C}'' = {}^{t \cdot \boldsymbol{U} + \boldsymbol{V}}\mathbf{C}_1 \cdot \widehat{\mathbf{G}}_1^{\xi}$ ensured by the consistency check in (6.27), we immediately deduce that if $\xi \neq 0 \pmod p$, we can reconstruct $\widehat{\mathbf{G}}_1^{\xi}$ from $\mathbf{C}$ and $\mathbf{C}''$ if $\boldsymbol{U}$ and $\boldsymbol{V}$ are known. Given such an element we can break the challenge of the Matrix Decisional Diffie-Hellman Assumption in the proof, since $\boldsymbol{U}$ and $\boldsymbol{V}$ are under our control. Similar statement holds also for consistency checks in (6.28) since all group elements of the ciphertext corresponding to $\tau \in \Gamma$ are checked one by one (here $\boldsymbol{H}_i$'s are required instead of $\boldsymbol{U}$ and $\boldsymbol{V}$ for the reconstruction). Recall that due to the Dual System Encryption Methodology [Wat09a] exploited in the pair encoding framework we cannot even hope to be able to distinguish between normal and semi-functional ciphertext. Hence, for the case $\overline{I} = \emptyset$ our checks allow negligible error but except for that achieve ideal guarantees.

Finally let us consider the general case $\overline{I} \neq \emptyset$. Schemes with reach structure of the ciphertexts, such as ciphertext-policy attribute-based encryption schemes, require these elements. We prove that due to the Matrix Decisional Diffie-Hellman Assumption a ppt adversary is not able to compute (except for negligible probability) a ciphertext that passes all consistency checks with respect to a reconstruction matrix $\boldsymbol{E}$ such that decryption using $\boldsymbol{E}$ may result in different messages if different normal user secret keys are used. The most challenging part of the proof relates exactly to this statement. The intuition behind this statement is as follows. Either the decryption of the considered ciphertext using reconstruction matrix $\boldsymbol{E}$ will work *as usual* – that is, $\xi_i \neq 0 \pmod p$ will not influence the result – or $\boldsymbol{E}$ reveals a linear combination of ciphertext elements which results in an element $\widehat{\mathbf{G}}_1^{\sigma}$, $\sigma \neq 0 \pmod p$. We conclude that the guarantees of the check in (6.29) are weaker then the guarantees of the other checks, but these guarantees are sufficient to prove CCA-security of our framework.

As noticed at the beginning of this subsection, all the statements from above do not yet ensure that a ppt adversary is not able to use decryption oracle to learn something about the challenge. Rather one can prove that if a ciphertext satisfies the consistency checks, the result of the decryption query is equal $m = C_0 \cdot \mathrm{e}\left(\mathbf{C}_1, \mathrm{msk}\right)^{-1}$ except for negligible probability. Recall that $\mathbf{C}_1 = \mathbf{G}_1^{\boldsymbol{s}_0} \cdot \widehat{\mathbf{G}}_1^{\hat{s}_0}$ due to the regularity of the underlying pair encodings (cf. Lemma 6.21). The last piece of the puzzle comes from our redundant element $\mathbf{C}''$ added to the ciphertext and containing the hash value t of all other elements. Namely, this element ensures that $\hat{s}_0 = 0 \pmod p$, except for negligible probability, and furthermore the adversary must know $\boldsymbol{s}_0$. Otherwise the adversary would not be able to compute a correct value $\mathbf{C}''$ for the ciphertext. But then, using $\boldsymbol{s}_0$ and $\mathbf{Y} \in \mathrm{pp}_\kappa$ the adversaries can compute $\mathbf{Y}^{\boldsymbol{s}_0} = \mathrm{e}\left(\mathbf{C}_1, \mathrm{msk}\right)$ for themselves, which makes the decryption oracle useless for them. The proof of this statement is the other challenging part of the formal proof.

We finally notice that the overall structure of our additional element $\mathbf{C}''$ is not new. Similar extensions (over various domains) have previously been used in the context of CCA-security,

e.g. in [CS03, KG09] for public key encryption schemes, identity-based encryption schemes and also in our composite-order framework. The intuition behind the construction of $\mathbf{C}''$ can be explained if we look at the exponent of this element, which has the form $(u \cdot t + v) \cdot s$ in all the schemes. Thereby, t is the hash value, u and v are unknown parameters from the public parameters (respectively public key) and s is the random element chosen during the encryption. Now, if adversary $\mathcal{A}$ tries to change a given ciphertext (for unknown s), it has to compute a valid $\mathbf{C}''$ with respect to the new hash value. Without knowledge of s, u and v this is a difficult task. The main observation exploited in all corresponding proofs is that for all $t_1 \neq t_2$ the values $u \cdot t_1 + v$ and $u \cdot t_2 + v$ are independent over the random choice of u and v. The collision resistance of the hash function ensures in our construction that adversary cannot find two different ciphertexts with the same hash value that would make it possible to reuse $\mathbf{C}''$. Notice that in [CS03, KG09] the so-called target collision-resistant hash functions were sufficient due to the simpler structure of the ciphertexts. Both our frameworks require collision-resistant hash functions. It is remarkable that despite the complex domain of computation in our prime-order framework we can reuse the overall structure of $\mathbf{C}''$. Indeed, in our case we cannot perfectly check the form of $\mathbf{C}''$ as already explained above, but the guarantees for this element stated in Lemma 6.24 are sufficient to prove the CCA-security of our construction.

6.2.6. Correctness

In this subsection we formally prove the correctness of the scheme. We have already proved that correctly generated ciphertext pass all our verification checks. Hence, the correctness of the schemes follows directly from the correctness of the CPA-secure framework (cf. Claim 15 in [Att15]). For the sake of completeness we present the formal proof.

Pair Encoding Reconstruction

Lemma 6.14 and Lemma 6.18 state that the group elements in the user secret keys for kInd are of the form $\overrightarrow{\mathbf{K}} = g_2^{\boldsymbol{k}(\boldsymbol{\alpha},\boldsymbol{R},\overline{\mathbb{H}})}$, whereas the group elements of the ciphertext under cInd are of the form $\overrightarrow{\mathbf{C}} = g_1^{\boldsymbol{c}(\boldsymbol{S},\mathbb{H})}$, where $\boldsymbol{R} \in \left(\mathbb{Z}_p^{d+1}\right)^{m_2}$ and $\boldsymbol{S} \in \left(\mathbb{Z}_p^{d+1}\right)^{w_2}$. The following lemma shows that even though we use $\overline{\mathbb{H}}$ in the evaluation of k_i's and $\mathbb{H}$ in the evaluation of c_i's, the reconstruction works as defined by pair encodings. We remark that by construction of the decryption algorithm the group elements of the keys are used as the second input of the pairing. Hence, by (6.10) the have to consider the transpose of the corresponding matrices.

Lemma 6.28. *Let* $\kappa \in \Omega \times \Sigma$, $\mathrm{kInd} \in \mathbb{X}_\kappa$, *and* $\mathrm{cInd} \in \mathbb{Y}_\kappa$ *be arbitrary. Let* $(\boldsymbol{k}, m_2) := \mathrm{Enc1}(\kappa, \mathrm{kInd})$, $m_1 := |\boldsymbol{k}|$, $(\boldsymbol{c}, w_2) := \mathrm{Enc2}(\kappa, \mathrm{cInd})$, *and* $w_1 = |\boldsymbol{c}|$. *Then, for all* $\boldsymbol{\alpha} \in \mathbb{Z}_p^{d+1}$, $\boldsymbol{H}_1, \ldots, \boldsymbol{H}_n \in \mathbb{Z}_p^{(d+1)\times(d+1)}$, $\boldsymbol{R} = (\boldsymbol{R}_1, \ldots, \boldsymbol{R}_{m_2}) \in \left(\mathbb{Z}_p^{d+1}\right)^{m_2}$ *and* $\boldsymbol{S} = (\boldsymbol{S}_0, \ldots, \boldsymbol{S}_{w_2+1}) \in \left(\mathbb{Z}_p^{d+1}\right)^{w_2+1}$ *it holds*

$$\sum_{\tau\in[m_1],\tau'\in[w_1]} e_{\tau,\tau'} \cdot k_\tau\left(\boldsymbol{\alpha}, \boldsymbol{R}, \overline{\mathbb{H}}\right)^\top \cdot c_{\tau'}\left(\boldsymbol{S}, \mathbb{H}\right) = \boldsymbol{\alpha}^\top \cdot \boldsymbol{S}_0 \ ,$$

where $\mathbb{H} := (\boldsymbol{H}_1, \ldots, \boldsymbol{H}_n)$ *and* $\overline{\mathbb{H}} := \left(\boldsymbol{H}_1^\top, \ldots, \boldsymbol{H}_n^\top\right)$.

Proof. Consider the settings of the lemma. By the evaluation of $k_\tau\left(\boldsymbol{\alpha}, \boldsymbol{R}, \overline{\mathbb{H}}\right)^\top$ we substitute

$$\begin{aligned} \mathrm{X}_\alpha &\hookrightarrow \boldsymbol{\alpha}^\top \ , \\ \forall_{i\in[m_2]} : \mathrm{X}_{r_i} &\hookrightarrow \boldsymbol{R}_i^\top \ , \\ \forall_{j\in[n]}\forall_{i\in[m_2]} : \mathrm{X}_{h_j}\cdot\mathrm{X}_{r_i} &\hookrightarrow \left(\overline{\mathbb{H}}_j \cdot \boldsymbol{R}_i\right)^\top \\ &= \boldsymbol{R}_i^\top \cdot \overline{\boldsymbol{H}}_j^\top \\ &= \boldsymbol{R}_i^\top \cdot \boldsymbol{H}_j \ . \end{aligned}$$

By the evaluation of $c\left(\boldsymbol{S}, \mathbb{H}\right)$ we substitute

$$\begin{aligned} \forall_{i\in[w_2]_0} : \mathrm{X}_{s_i} &\hookrightarrow \boldsymbol{S}_i \ , \\ \forall_{j\in[n]}\forall_{i\in[w_2]_0} : \mathrm{X}_{h_j}\cdot\mathrm{X}_{s_i} &\hookrightarrow \boldsymbol{H}_j \cdot \boldsymbol{S}_i \ . \end{aligned}$$

Independently of the substitutions, due to the regularity of P and by Lemma 1.15, all monomials disappear except those corresponding to $\mathrm{X}_\alpha\cdot\mathrm{X}_{s_0}$ and $\mathrm{X}_{r_i}\cdot\mathrm{X}_{h_j}\cdot\mathrm{X}_{s_{i'}}$. Furthermore, by Lemma 1.15 it holds

$$\forall_{i\in[m_2]}\forall_{i'\in[w_2]_0}\forall_{j\in[n]} : \sum_{\tau\in[m_1]}\sum_{\tau'\in[w_1]} e_{\tau,\tau'}\cdot\left(a_{\tau,i}\cdot b_{\tau',i',j} + a_{\tau,i,j}\cdot b_{\tau',i'}\right) = 0 \ .$$

Let $i \in [m_2]$, $i' \in [w_2]_0$ and $j \in [n]$ be arbitrary, but fixed. The coefficients by monomial $\mathrm{X}_{r_i}\cdot\mathrm{X}_{h_j}\cdot\mathrm{X}_{s_{i'}}$ come from the sum of coefficients by $\mathrm{X}_{r_i}\cdot\mathrm{X}_{h_{\hat{j}}}\cdot\mathrm{X}_{s_{i'}}$ and $\mathrm{X}_{h_{j'}}\cdot\mathrm{X}_{r_i}\cdot\mathrm{X}_{s_{i'}}$ with $\hat{j} = j = j'$. However,

$$\begin{aligned} \mathrm{X}_{r_i}\cdot\mathrm{X}_{h_{j'}}\cdot\mathrm{X}_{s_{i'}} &\hookrightarrow \left(\boldsymbol{R}_i^\top\right)\cdot\left(\boldsymbol{H}_j\cdot\boldsymbol{S}_{i'}\right) \\ &= \boldsymbol{R}_i^\top\cdot\boldsymbol{H}_j\cdot\boldsymbol{S}_{i'} \ , \\ \mathrm{X}_{h_{\hat{j}}}\cdot\mathrm{X}_{r_i}\cdot\mathrm{X}_{s_{i'}} &\hookrightarrow \left(\boldsymbol{R}_i^\top\cdot\boldsymbol{H}_j\right)\cdot\left(\boldsymbol{S}_{i'}\right) \\ &= \boldsymbol{R}_i^\top\cdot\boldsymbol{H}_j\cdot\boldsymbol{S}_{i'} \ . \end{aligned}$$

Thus, the substitutions of both parts are equal and thus due to Lemma 1.15 the coefficient by $\mathrm{X}_{r_i}\cdot\mathrm{X}_{h_j}\cdot\mathrm{X}_{s_{i'}}$ is also equal to zero.

Hence, the only one monomial which does not disappear is $\mathrm{X}_\alpha\cdot\mathrm{X}_{s_0}$, which substitution is equal to $\boldsymbol{\alpha}^\top\cdot\boldsymbol{S}_0$. The corresponding coefficient is 1 by Lemma 1.15. □

Correctness Proof

Correctness Proof. Let $\lambda \in \mathbb{N}$, $\mathrm{des} \in \Omega$, $(\mathrm{pp}_\kappa, \mathrm{msk}) \in \left[\mathrm{Setup}\left(1^\lambda, \mathrm{des}\right)\right]$ be arbitrary but fixed. Let $\mathrm{kInd} \in \mathbb{X}_\kappa$ and $\mathrm{cInd} \in \mathbb{Y}_\kappa$ be arbitrary but fixed such that $\mathrm{R}\left(\mathrm{kInd}, \mathrm{cInd}\right) = 1$. Furthermore, let $m \in \mathcal{M}$, $\mathrm{sk} = \left(\mathrm{kInd}, \overrightarrow{\mathbf{K}}\right) \in \left[\mathrm{KeyGen}\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{kInd}\right)\right]$, and $\mathrm{CT} = \left(\mathrm{cInd}, C_0, \overrightarrow{\mathbf{C}}, \mathbf{C}''\right) \in \left[\mathrm{Enc}\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{cInd}, m\right)\right]$ be arbitrary but fixed. Then, by Lemma 6.27 the consistency checks for CT are satisfied. It remains to prove that it holds

$$\Pr\left[\mathrm{Dec}\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{sk}, \mathrm{CT}\right) = m\right] = 1 \ .$$

Lemma 6.14 and Lemma 6.18 state that the group elements in the user secret key are of the form $\overrightarrow{\mathbf{K}} = g_2^{\boldsymbol{k}\left(\boldsymbol{\alpha}, \boldsymbol{R}, \overline{\mathbb{H}}\right)} \in \left(\mathbb{G}_2^{d+1}\right)^{m_1}$ and the ciphertext elements are of the form $\overrightarrow{\mathbf{C}} = g_1^{\boldsymbol{c}(\boldsymbol{S}, \mathbb{H})} \in \left(\mathbb{G}_1^{d+1}\right)^{w_1}$, where $\boldsymbol{R} \in \left(\mathbb{Z}_p^{d+1}\right)^{m_2}$ and $\boldsymbol{S} := \left(\boldsymbol{S}_0, \ldots, \boldsymbol{S}_{w_2}\right) \in \left(\mathbb{Z}_p^{d+1}\right)^{w_2+1}$ with $\boldsymbol{S}_0 = \boldsymbol{B}\cdot\begin{pmatrix}\boldsymbol{s}_0 \\ 0\end{pmatrix}$.

We deduce that the decryption algorithm computes:

$$\begin{aligned} X = \prod_{\tau\in[m_1],\tau'\in[w_1]} \mathrm{e}\left(\mathbf{C}_{\tau'},\mathbf{K}_{\tau}\right)^{e_{\tau,\tau'}} &= \prod_{\tau\in[m_1],\tau'\in[w_1]} \mathrm{e}\left(g_1^{c_{\tau'}(\boldsymbol{S},\mathbb{H})}, g_2^{k_{\tau}\left(\boldsymbol{\alpha},\boldsymbol{R},\overline{\mathbb{H}}\right)}\right)^{e_{\tau,\tau'}} \\ &= \prod_{\tau\in[m_1],\tau'\in[w_1]} \mathrm{e}\left(g_1,g_2\right)^{e_{\tau,\tau'}k_{\tau}\left(\boldsymbol{\alpha},\boldsymbol{R},\overline{\mathbb{H}}\right)^{\top}\cdot c_{\tau'}(\boldsymbol{S},\mathbb{H})} \\ &= \mathrm{e}\left(g_1,g_2\right)^{\sum_{\tau\in[m_1],\tau'\in[w_1]} e_{\tau,\tau'}\cdot k_{\tau}\left(\boldsymbol{\alpha},\boldsymbol{R},\overline{\mathbb{H}}\right)^{\top}\cdot c_{\tau'}(\boldsymbol{S},\mathbb{H})} \\ &= \mathrm{e}\left(g_1,g_2\right)^{\boldsymbol{\alpha}^{\top}\cdot\boldsymbol{S}_0} \quad , \end{aligned}$$

where the last equation holds by Lemma 6.28. Hence, the result of this step is

$$\begin{aligned} X &= \mathrm{e}\left(g_1,g_2\right)^{\boldsymbol{\alpha}^{\top}\cdot\boldsymbol{S}_0} \\ &= \mathrm{e}\left(g_1,g_2\right)^{\boldsymbol{\alpha}^{\top}\cdot\boldsymbol{B}\cdot\binom{\boldsymbol{s}_0}{0}} \\ &= \mathrm{e}\left(g_1,g_2\right)^{\boldsymbol{\alpha}^{\top}\cdot\boldsymbol{B}\cdot\binom{\boldsymbol{I}_d}{\boldsymbol{0}}\cdot\boldsymbol{s}_0} \\ &= \mathrm{e}\left(\mathbf{G}_1,g_2^{\boldsymbol{\alpha}}\right)^{\boldsymbol{s}_0} \\ &= \boldsymbol{Y}^{\boldsymbol{s_0}} \quad . \end{aligned}$$

Hence, the decryption algorithm outputs $C_0 \cdot X^{-1} = m \cdot \boldsymbol{Y}^{\boldsymbol{s_0}} \cdot \boldsymbol{Y}^{-\boldsymbol{s_0}} = m$. This proves the correctness of the scheme. □

Decryption of Semi-Functional Ciphertext Using Normal User Secret Key

In this section we additionally prove that the decryption of any correctly generated semi-functional ciphertext using a normal key works as usual. This lemma will be used in one of our new reduction steps.

Lemma 6.29. *Let* $\lambda \in \mathbb{N}$, $\mathrm{des} \in \Omega$, $\left(\mathrm{pp}_{\kappa}, \mathrm{msk}, \widehat{\mathrm{pp}}_{\kappa}, \widehat{\mathrm{msk}}\right) \in \left[\mathrm{SFSetup}\left(1^{\lambda}, \mathrm{des}\right)\right]$ *be arbitrary but fixed. Let* $\mathrm{kInd} \in \mathbb{X}_{\kappa}$ *and* $\mathrm{cInd} \in \mathbb{Y}_{\kappa}$ *be arbitrary but fixed such that* $\mathrm{R}\left(\mathrm{kInd}, \mathrm{cInd}\right) = 1$. *Furthermore, let* $m \in \mathcal{M}$, $\mathrm{sk} = \left(\mathrm{kInd}, \overrightarrow{\mathbf{K}}\right) \in \left[\mathrm{KeyGen}\left(1^{\lambda}, \mathrm{pp}_{\kappa}, \mathrm{msk}, \mathrm{kInd}\right)\right]$, *and* $\mathrm{CT} = \left(\mathrm{cInd}, C_0, \overrightarrow{\mathbf{C}}, \mathbf{C}''\right) \in \left[\mathrm{SFEnc}\left(1^{\lambda}, \mathrm{pp}_{\kappa}, \widehat{\mathrm{pp}}_{\kappa}, \mathrm{cInd}, m\right)\right]$ *be arbitrary but fixed. Then,*

$$\mathrm{Dec}\left(1^{\lambda}, \mathrm{pp}_{\kappa}, \mathrm{sk}, \mathrm{CT}\right) = m \quad .$$

Proof. By Lemma 6.27 the consistency checks for CT are satisfied. It remains to prove that it holds

$$\Pr\left[\mathrm{Dec}\left(1^{\lambda}, \mathrm{pp}_{\kappa}, \mathrm{sk}, \mathrm{CT}\right) = m\right] = 1 \quad .$$

Lemma 6.14 and Corollary 6.20 state that the user secret key group elements are of the form $\overrightarrow{\mathbf{K}} = g_2^{\boldsymbol{k}\left(\boldsymbol{\alpha},\boldsymbol{R},\overline{\mathbb{H}}\right)} \in \left(\mathbb{G}_2^{(d+1)\times 1}\right)^{1\times m_1}$ and the ciphertext elements are of the form $\overrightarrow{\mathbf{C}} = g_1^{\boldsymbol{c}(\boldsymbol{S},\mathbb{H})} \in \left(\mathbb{G}_1^{(d+1)\times 1}\right)^{1\times w_1}$, where $\boldsymbol{R} \in \left(\mathbb{Z}_p^{d+1}\right)^{m_2}$ and $\boldsymbol{S} := (\boldsymbol{S}_0, \ldots, \boldsymbol{S}_{w_2}) \in \left(\mathbb{Z}_p^{d+1}\right)^{w_2+1}$ with $\boldsymbol{S}_0 = \boldsymbol{B} \cdot$

$\begin{pmatrix} \boldsymbol{s}_0 \\ \hat{s}_0 \end{pmatrix}$. We deduce that the decryption algorithm computes:

$$\begin{aligned} X = \prod_{\tau\in[m_1],\tau'\in[w_1]} \mathrm{e}\left(\mathbf{C}_{\tau'},\mathbf{K}_{\tau}\right)^{e_{\tau,\tau'}} &= \prod_{\tau\in[m_1],\tau'\in[w_1]} \mathrm{e}\left(g_1^{c_{\tau'}(\boldsymbol{S},\mathbb{H})}, g_2^{k_{\tau}\left(\boldsymbol{\alpha},\boldsymbol{R},\overline{\mathbb{H}}\right)}\right)^{e_{\tau,\tau'}} \\ &= \prod_{\tau\in[m_1],\tau'\in[w_1]} \mathrm{e}\left(g_1,g_2\right)^{e_{\tau,\tau'}k_{\tau}\left(\boldsymbol{\alpha},\boldsymbol{R},\overline{\mathbb{H}}\right)^{\top}\cdot c_{\tau'}(\boldsymbol{S},\mathbb{H})} \\ &= \mathrm{e}\left(g_1,g_2\right)^{\sum_{\tau\in[m_1],\tau'\in[w_1]} e_{\tau,\tau'}\cdot k_{\tau}\left(\boldsymbol{\alpha},\boldsymbol{R},\overline{\mathbb{H}}\right)^{\top}\cdot c_{\tau'}(\boldsymbol{S},\mathbb{H})} \\ &= \mathrm{e}\left(g_1,g_2\right)^{\boldsymbol{\alpha}^{\top}\cdot\boldsymbol{S}_0} \quad , \end{aligned}$$

where the last equation holds by Lemma 6.28. Hence, the result of this step is

$$\begin{aligned} X &= \mathrm{e}\left(g_1,g_2\right)^{\boldsymbol{\alpha}^{\top}\cdot\boldsymbol{S}_0} \\ &= \mathrm{e}\left(g_1,g_2\right)^{\boldsymbol{\alpha}^{\top}\cdot\boldsymbol{B}\cdot\left(\begin{smallmatrix}\boldsymbol{s}_0\\ \hat{s}_0\end{smallmatrix}\right)} \\ &= \mathrm{e}\left(g_1,g_2\right)^{\boldsymbol{\alpha}^{\top}\cdot\boldsymbol{B}\cdot\left(\begin{smallmatrix}\boldsymbol{I}_d\\ \boldsymbol{0}\end{smallmatrix}\right)\cdot\boldsymbol{s}_0+\boldsymbol{\alpha}^{\top}\cdot\boldsymbol{B}\cdot\left(\begin{smallmatrix}\boldsymbol{0}\\ 1\end{smallmatrix}\right)\cdot\hat{s}_0} \\ &= \mathrm{e}\left(\mathbf{G}_1,g_2^{\boldsymbol{\alpha}}\right)^{\boldsymbol{s}_0}\cdot\mathrm{e}\left(\widehat{\mathbf{G}}_1,g_2^{\boldsymbol{\alpha}}\right)^{\hat{s}_0} \\ &= \mathbf{Y}^{\boldsymbol{s}_0}\cdot\widehat{\mathbf{Y}}^{\hat{s}_0} \quad . \end{aligned}$$

Hence, the decryption algorithm outputs $C_0\cdot X^{-1} = m\cdot\mathbf{Y}^{\boldsymbol{s}_0}\cdot\widehat{\mathbf{Y}}^{\hat{s}_0}\cdot\mathbf{Y}^{-\boldsymbol{s}_0}\cdot\widehat{\mathbf{Y}}^{-\hat{s}_0} = m$. Hence, decryption of the semi-functional ciphertext using normal secret key results in the correct message. □

6.2.7. Further Proofs

Proof of Lemma 6.12. We prove all equations directly:

1. The first statement:

$$\begin{aligned} \mathrm{e}\left(\mathbf{G}_1,\mathbf{G}_2\right) &\overset{\text{by def.}}{=} \mathrm{e}\left(g_1^{\boldsymbol{B}\cdot\left(\begin{smallmatrix}\boldsymbol{I}_d\\ \boldsymbol{0}\end{smallmatrix}\right)}, g_2^{\boldsymbol{B}^{-\top}\cdot\boldsymbol{D}\cdot\left(\begin{smallmatrix}\boldsymbol{I}_d\\ \boldsymbol{0}\end{smallmatrix}\right)}\right) \\ &= \mathrm{e}\left(g_1,g_2\right)^{\left(\begin{smallmatrix}\tilde{\boldsymbol{D}}^{\top} & \boldsymbol{0}\end{smallmatrix}\right)\cdot\boldsymbol{B}^{-1}\cdot\boldsymbol{B}\cdot\left(\begin{smallmatrix}\boldsymbol{I}_d\\ \boldsymbol{0}\end{smallmatrix}\right)} \\ &= \mathrm{e}\left(g_1,g_2\right)^{\tilde{\boldsymbol{D}}^{\top}} \quad , \end{aligned}$$

$$\begin{aligned} \mathrm{e}\left(\mathbf{G}_1,\overline{\mathbf{H}}_j\right) &\overset{\text{by def.}}{=} \mathrm{e}\left(\mathbf{G}_1,\boldsymbol{H}_j^{\top}\mathbf{G}_2\right) \\ &= \mathrm{e}\left(\boldsymbol{H}_j\mathbf{G}_1,\mathbf{G}_2\right) \\ &= \mathrm{e}\left(\mathbf{H}_j,\mathbf{G}_2\right) \quad . \end{aligned}$$

2. The second statement:

$$\begin{aligned} \mathrm{e}\left(\mathbf{G}_1,\widehat{\mathbf{G}}_2\right) &\overset{\text{by def.}}{=} \mathrm{e}\left(g_1^{\boldsymbol{B}\cdot\left(\begin{smallmatrix}\boldsymbol{I}_d\\ \boldsymbol{0}\end{smallmatrix}\right)}, g_2^{\boldsymbol{B}^{-\top}\cdot\left(\begin{smallmatrix}\boldsymbol{0}\\ 1\end{smallmatrix}\right)}\right) \\ &= \mathrm{e}\left(g_1,g_2\right)^{\left(\begin{smallmatrix}\boldsymbol{0} & 1\end{smallmatrix}\right)\cdot\left(\begin{smallmatrix}\boldsymbol{I}_d\\ \boldsymbol{0}\end{smallmatrix}\right)} \\ &= \left(1_{\mathbb{G}_{\mathrm{T}}},\ldots,1_{\mathbb{G}_{\mathrm{T}}}\right) \quad , \end{aligned}$$

$$\begin{aligned} \mathrm{e}\left(\mathbf{G}_1, \widehat{\overline{\mathbf{H}}}_j\right) &\overset{\text{by def.}}{=} \mathrm{e}\left(\mathbf{G}_1, {}^{\boldsymbol{H}_j^\top}\widehat{\mathbf{G}}_2\right) \\ &= \mathrm{e}\left({}^{\boldsymbol{H}_j}\mathbf{G}_1, \widehat{\mathbf{G}}_2\right) \\ &= \mathrm{e}\left(\mathbf{H}_j, \widehat{\mathbf{G}}_2\right) \ . \end{aligned}$$

3. The third statement:

$$\begin{aligned} \mathrm{e}\left(\widehat{\mathbf{G}}_1, \mathbf{G}_2\right) &\overset{\text{by def.}}{=} \mathrm{e}\left(g_1^{\boldsymbol{B}\cdot\binom{\mathbf{0}}{1}}, g_2^{\boldsymbol{B}^{-\top}\cdot\binom{\tilde{\boldsymbol{D}}}{\mathbf{0}}}\right) \\ &= \mathrm{e}\left(g_1, g_2\right)^{(\tilde{\boldsymbol{D}}\ \mathbf{0})\cdot\binom{\mathbf{0}}{1}} \\ &= \left(1_{\mathbb{G}_\mathrm{T}}, \ldots, 1_{\mathbb{G}_\mathrm{T}}\right)^\top \ , \end{aligned}$$

$$\begin{aligned} \mathrm{e}\left(\widehat{\mathbf{G}}_1, \overline{\mathbf{H}}_j\right) &\overset{\text{by def.}}{=} \mathrm{e}\left(\widehat{\mathbf{G}}_1, {}^{\boldsymbol{H}_j^\top}\mathbf{G}_2\right) \\ &= \mathrm{e}\left({}^{\boldsymbol{H}_j}\widehat{\mathbf{G}}_1, \mathbf{G}_2\right) \\ &= \mathrm{e}\left(\widehat{\mathbf{H}}_j, \mathbf{G}_2\right) \ . \end{aligned}$$

4. The last statement:

$$\begin{aligned} \mathrm{e}\left(\widehat{\mathbf{G}}_1, \widehat{\mathbf{G}}_2\right) &\overset{\text{by def.}}{=} \mathrm{e}\left(g_1^{\boldsymbol{B}\cdot\binom{\mathbf{0}}{1}}, g_2^{\boldsymbol{B}^{-\top}\cdot\binom{\mathbf{0}}{1}}\right) \\ &= \mathrm{e}\left(g_1, g_2\right)^{(\mathbf{0}\ 1)\cdot\binom{\mathbf{0}}{1}} \\ &= \mathrm{e}\left(g_1, g_2\right) \ , \end{aligned}$$

and

$$\begin{aligned} \mathrm{e}\left(\widehat{\mathbf{G}}_1, \widehat{\overline{\mathbf{H}}}_j\right) &\overset{\text{by def.}}{=} \mathrm{e}\left(\widehat{\mathbf{G}}_1, {}^{\boldsymbol{H}_j^\top}\widehat{\mathbf{G}}_2\right) \\ &= \mathrm{e}\left({}^{\boldsymbol{H}_j}\widehat{\mathbf{G}}_1, \widehat{\mathbf{G}}_2\right) \\ &= \mathrm{e}\left(\widehat{\mathbf{H}}_j, \widehat{\mathbf{G}}_2\right) \ . \end{aligned}$$

This finalizes the proof. □

Proof of Lemma 6.15. For Type 1 keys we have for every $k_\tau \in \boldsymbol{k}$

$$\begin{aligned} g_2^{k_\tau\left(\mathbf{0}, \widehat{\boldsymbol{R}}, \widehat{\mathbb{H}}\right)} &= g_2^{\sum_{i\in[m_2]}\left(a_{\tau,i}\cdot\boldsymbol{Z}\cdot\binom{\mathbf{0}}{\hat{r}_i}+\sum_{j\in[n]} a_{\tau,i,j}\cdot\boldsymbol{H}_j^\top\cdot\boldsymbol{Z}\cdot\binom{\mathbf{0}}{\hat{r}_i}\right)} \\ &= \left(g_2^{\boldsymbol{Z}\cdot\binom{\mathbf{0}}{1}}\right)^{\sum_{i\in[m_2]} a_{\tau,i}\cdot\hat{r}_i} \cdot \prod_{j\in[n]} \left(g_2^{\boldsymbol{H}_j^\top\cdot\boldsymbol{Z}\cdot\binom{\mathbf{0}}{1}}\right)^{\sum_{i\in[m_2]} a_{\tau,i,j}\cdot\hat{r}_i} \\ &= \widehat{\mathbf{G}}_2^{\sum_{i\in[m_2]} a_{\tau,i}\cdot\hat{r}_i} \cdot \prod_{j\in[n]} \widehat{\overline{\mathbf{H}}}_j^{\sum_{i\in[m_2]} a_{\tau,i,j}\cdot\hat{r}_i} \\ &\overset{(6.30)}{=} \widehat{\mathbf{K}}_\tau \ . \end{aligned}$$

This is exactly how the semi-functional parts of the key are computed.

For Type 3 keys we have

$$\begin{aligned} g_2^{k_\tau\left(\boldsymbol{Z}\cdot\binom{\mathbf{0}}{\beta},\mathbf{0},\mathbf{0}\right)} &= g_2^{a_\tau\cdot\boldsymbol{Z}\cdot\binom{\mathbf{0}}{\beta}} \\ &= \left(g_2^{\boldsymbol{Z}\cdot\binom{\mathbf{0}}{1}}\right)^{a_\tau\cdot\beta} \\ &= \widehat{\mathbf{G}}_2^{a_\tau\cdot\beta} \\ &\overset{(6.30)}{=} \widehat{\mathbf{K}}_\tau \ . \end{aligned}$$

This is exactly how the semi-functional parts of the key are computed.

Together we get for Type 2 keys:

$$\begin{aligned} g_2^{k_\tau\left(\boldsymbol{Z}\cdot\binom{\mathbf{0}}{\beta},\widehat{\boldsymbol{R}},\overline{\mathbb{H}}\right)} &= g_2^{k_\tau\left(\boldsymbol{Z}\cdot\binom{\mathbf{0}}{\beta},\mathbf{0},\mathbf{0}\right)}\cdot g_2^{\boldsymbol{k}_\tau\left(\mathbf{0},\widehat{\boldsymbol{R}},\overline{\mathbb{H}}\right)} \\ &= \widehat{\mathbf{G}}_2^{a_\tau\cdot\beta+\sum_{i\in[m_2]}a_{\tau,i}\cdot\hat{r}_i}\cdot\prod_{j\in[n]}\widehat{\overline{\mathbf{H}}}_j^{\sum_{i\in[m_2]}a_{\tau,i,j}\cdot\hat{r}_i} \\ &\overset{(6.30)}{=} \widehat{\mathbf{K}}_\tau \ . \end{aligned}$$

This is exactly how the semi-functional parts of the key are computed. □

Proof of Lemma 6.19. We prove it directly for every $c_\tau \in \boldsymbol{c}$:

$$\begin{aligned} g_1^{c_\tau\left(\hat{\boldsymbol{S}},\mathbb{H}\right)} &= g_1^{\sum_{i\in[w_2]_0}\left(b_{\tau,i}\cdot\boldsymbol{B}\cdot\binom{\mathbf{0}}{\hat{s}_i}+\sum_{j\in[n]}b_{\tau,i,j}\cdot\boldsymbol{H}_j\cdot\boldsymbol{B}\cdot\binom{\mathbf{0}}{\hat{s}_i}\right)} \\ &= g_1^{\sum_{i\in[w_2]_0}b_{\tau,i}\cdot\boldsymbol{B}\cdot\binom{\mathbf{0}}{1}\cdot\hat{s}_i}\cdot g_1^{\sum_{i\in[w_2]_0}\sum_{j\in[n]}b_{\tau,i,j}\cdot\boldsymbol{H}_j\cdot\boldsymbol{B}\cdot\binom{\mathbf{0}}{1}\cdot\hat{s}_i} \\ &= \left(g_1^{\boldsymbol{B}\cdot\binom{\mathbf{0}}{1}}\right)^{\sum_{i\in[w_2]_0}b_{\tau,i}\cdot\hat{s}_i}\cdot\prod_{j\in[n]}\left(g_1^{\boldsymbol{H}_j\cdot\boldsymbol{B}\cdot\binom{\mathbf{0}}{1}}\right)^{\sum_{i\in[w_2]_0}b_{\tau,i,j}\cdot\hat{s}_i} \\ &= \widehat{\mathbf{G}}_1^{\sum_{i\in[w_2]_0}b_{\tau,i}\cdot\hat{s}_i}\cdot\prod_{j\in[n]}\widehat{\mathbf{H}}_j^{\sum_{i\in[w_2]_0}b_{\tau,i,j}\cdot\hat{s}_i} \\ &\overset{(6.31)}{=} \widehat{\mathbf{C}}_\tau \ . \end{aligned}$$

This is exactly how the ciphertexts are computed. □

Proof of Lemma 6.21. By the definition of the semi-functional encryption algorithm SFEnc it holds for every $\tau \in [m_1]$:

$$\begin{aligned} \mathbf{C}_\tau &\overset{(6.24),(6.31)}{=} \mathbf{G}_1^{\sum_{i\in[w_2]_0}b_{\tau,i}\cdot\boldsymbol{s}_i}\cdot\prod_{j\in[n]}\mathbf{H}_j^{\sum_{i\in I}b_{\tau,i,j}\cdot\boldsymbol{s}_i} \\ &\quad\cdot\widehat{\mathbf{G}}_1^{\sum_{i\in[w_2]_0}b_{\tau,i}\cdot\hat{s}_i}\cdot\prod_{j\in[n]}\widehat{\mathbf{H}}_j^{\sum_{i\in I}b_{\tau,i,j}\cdot\hat{s}_i} \\ &= \prod_{i\in[w_2]_0}\left(\mathbf{G}_1^{b_{\tau,i}\cdot\boldsymbol{s}_i}\cdot\widehat{\mathbf{G}}_1^{b_{\tau,i}\cdot\hat{s}_i}\right)\cdot\prod_{j\in[n]}\prod_{i\in I}\left(\boldsymbol{H}_j\mathbf{G}_1^{b_{\tau,i,j}\cdot\boldsymbol{s}_i}\cdot{}^{\boldsymbol{H}_j}\widehat{\mathbf{G}}_1^{b_{\tau,i,j}\cdot\hat{s}_i}\right) \qquad (6.35) \\ &= \prod_{i\in\overline{I}}\left(\mathbf{G}_1^{\boldsymbol{s}_i}\cdot\widehat{\mathbf{G}}_1^{\hat{s}_i}\right)^{b_{\tau,i}}\cdot\prod_{i\in I}\left(\mathbf{G}_1^{\boldsymbol{s}_i}\cdot\widehat{\mathbf{G}}_1^{\hat{s}_i}\right)^{b_{\tau,i}}\cdot\prod_{j\in[n]}\prod_{i\in I}{}^{\boldsymbol{H}_j}\left(\mathbf{G}_1^{\boldsymbol{s}_i}\cdot\widehat{\mathbf{G}}_1^{\hat{s}_i}\right)^{b_{\tau,i,j}} \qquad (6.36) \end{aligned}$$

Next consider the statements one by one

1. By the definition of Γ_0 it holds $\tau \in \Gamma_0 \Leftrightarrow \exists_{i\in I} : c_\tau = \mathrm{X}_{s_i}$. Such a τ will be denoted by τ_i here. Hence, for every $\tau \in \Gamma_0$ the only coefficient unequal zero is $b_{\tau,i} = 1$. Hence, it holds for every $\tau_i \in \Gamma_0$:

$$\begin{aligned}\mathbf{C}_{\tau_i} &\overset{(6.35)}{=} \prod_{i\in[w_2]_0} \left(\mathbf{G}_1^{b_{\tau,i}\cdot \boldsymbol{s}_i} \cdot \widehat{\mathbf{G}}_1^{b_{\tau,i}\cdot \hat{s}_i}\right) \cdot \prod_{j\in[n]} \prod_{i\in I} \left({}^{\boldsymbol{H}_j}\mathbf{G}_1^{b_{\tau,i,j}\cdot \boldsymbol{s}_i} \cdot {}^{\boldsymbol{H}_j}\widehat{\mathbf{G}}_1^{b_{\tau,i,j}\cdot \hat{s}_i}\right) \\ &= \mathbf{G}_1^{\boldsymbol{s}_i} \cdot \widehat{\mathbf{G}}_1^{\hat{s}_i} \ . \end{aligned} \tag{6.37}$$

   In particular, since $c_1 = \mathrm{X}_{s_0}$ it holds

$$\mathbf{C}_1 = \mathbf{G}_1^{\boldsymbol{s}_0} \cdot \widehat{\mathbf{G}}_1^{\hat{s}_0} \ . \tag{6.38}$$

   This proves the first statement.

2. By the definition of Γ it holds $c_\tau = \sum_{i\in I} \left(b_{\tau,i} \cdot \mathrm{X}_{s_i} + \sum_{j\in[n]} b_{\tau,i,j} \cdot \mathrm{X}_{h_j} \cdot \mathrm{X}_{s_i}\right)$ for every $\tau \in \Gamma$. Hence, it holds for every $\tau \in \Gamma$:

$$\begin{aligned}\mathbf{C}_\tau &\overset{(6.36)}{=} \prod_{i\in\overline{I}} \left(\mathbf{G}_1^{\boldsymbol{s}_i} \cdot \widehat{\mathbf{G}}_1^{\hat{s}_i}\right)^{b_{\tau,i}} \cdot \prod_{i\in I} \left(\mathbf{G}_1^{\boldsymbol{s}_i} \cdot \widehat{\mathbf{G}}_1^{\hat{s}_i}\right)^{b_{\tau,i}} \cdot \prod_{j\in[n]} \prod_{i\in I} {}^{\boldsymbol{H}_j}\left(\mathbf{G}_1^{\boldsymbol{s}_i} \cdot \widehat{\mathbf{G}}_1^{\hat{s}_i}\right)^{b_{\tau,i,j}} \\ &= \prod_{i\in I} \left(\mathbf{G}_1^{\boldsymbol{s}_i} \cdot \widehat{\mathbf{G}}_1^{\hat{s}_i}\right)^{b_{\tau,i}} \cdot \prod_{j\in[n]} \prod_{i\in I} {}^{\boldsymbol{H}_j}\left(\mathbf{G}_1^{\boldsymbol{s}_i} \cdot \widehat{\mathbf{G}}_1^{\hat{s}_i}\right)^{b_{\tau,i,j}} \\ &\overset{(6.37)}{=} \prod_{i\in I} (\mathbf{C}_{\tau_i})^{b_{\tau,i}} \cdot \prod_{j\in[n]} \prod_{i\in I} {}^{\boldsymbol{H}_j^\top}(\mathbf{C}_{\tau_i})^{b_{\tau,i,j}} \end{aligned}$$

   This proves the second statement.

3. By the definition of $\overline{\Gamma}$ for every $\tau \in \overline{\Gamma}$ it holds $c_\tau = \sum_{i\in\overline{I}} b_{\tau,i} \cdot \mathrm{X}_{s_i} + \sum_{i\in I} \left(b_{\tau,i} \cdot \mathrm{X}_{s_i} + \sum_{j\in[n]} b_{\tau,i,j} \cdot \mathrm{X}_{h_j} \cdot \mathrm{X}_{s_i}\right)$. Hence,

$$\begin{aligned}\mathbf{C}_\tau &\overset{(6.36)}{=} \prod_{i\in\overline{I}} \left(\mathbf{G}_1^{\boldsymbol{s}_i} \cdot \widehat{\mathbf{G}}_1^{\hat{s}_i}\right)^{b_{\tau,i}} \cdot \prod_{i\in I} \left(\mathbf{G}_1^{\boldsymbol{s}_i} \cdot \widehat{\mathbf{G}}_1^{\hat{s}_i}\right)^{b_{\tau,i}} \cdot \prod_{j\in[n]} \prod_{i\in I} {}^{\boldsymbol{H}_j}\left(\mathbf{G}_1^{\boldsymbol{s}_i} \cdot \widehat{\mathbf{G}}_1^{\hat{s}_i}\right)^{b_{\tau,i,j}} \\ &= \widehat{\mathbf{G}}_1^{\sum_{i\in\overline{I}} b_{\tau,i}\cdot \hat{s}_i} \cdot \prod_{i\in\overline{I}} (\mathbf{G}_1^{\boldsymbol{s}_i})^{b_{\tau,i}} \cdot \prod_{i\in I} (\mathbf{C}_{\tau_i})^{b_{\tau,i}} \cdot \prod_{j\in[n]} \prod_{i\in I} {}^{\boldsymbol{H}_j^\top}(\mathbf{C}_{\tau_i})^{b_{\tau,i,j}} \ . \end{aligned}$$

   This proves the third statement.

4. Next, let us consider C_0. It holds by construction of SFEnc and the previous result for $\mathbf{C}_1$:

$$\begin{aligned} C_0 &\overset{(6.25),(6.32)}{=} m \cdot \mathbf{Y}^{\boldsymbol{s}_0} \cdot \mathrm{e}\left(\widehat{\mathbf{G}}_1, g_2^{\boldsymbol{\alpha}}\right)^{\hat{s}_0} \\ &= m \cdot \mathrm{e}\left(\mathbf{G}_1, g_2^{\boldsymbol{\alpha}}\right)^{\boldsymbol{s}_0} \cdot \mathrm{e}\left(\widehat{\mathbf{G}}_1, g_2^{\boldsymbol{\alpha}}\right)^{\hat{s}_0} \\ &= m \cdot \mathrm{e}\left(\mathbf{G}_1^{\boldsymbol{s}_0} \cdot \widehat{\mathbf{G}}_1^{\hat{s}_0}, g_2^{\boldsymbol{\alpha}}\right) \\ &\overset{(6.38)}{=} m \cdot \mathrm{e}\left(\mathbf{C}_1, g_2^{\boldsymbol{\alpha}}\right) \ . \end{aligned}$$

5. Furthermore, by construction of SFEnc it holds

$$\begin{aligned}
\mathbf{C}'' &\overset{(6.33)}{=} \left(\mathbf{U}^t \cdot \mathbf{V}\right)^{\boldsymbol{s}_0} \cdot \left(\widehat{\mathbf{U}}^t \cdot \widehat{\mathbf{V}}\right)^{\hat{s}_0} \\
&= \left({}^{t\cdot\boldsymbol{U}}\mathbf{G}_1 \cdot {}^{\boldsymbol{V}}\mathbf{G}_1\right)^{\boldsymbol{s}_0} \cdot \left({}^{t\cdot\boldsymbol{U}}\widehat{\mathbf{G}}_1 \cdot {}^{\boldsymbol{V}}\widehat{\mathbf{G}}_1\right)^{\hat{s}_0} \\
&= {}^{t\cdot\boldsymbol{U}+\boldsymbol{V}}\left(\mathbf{G}_1^{\boldsymbol{s}_0} \cdot \widehat{\mathbf{G}}_1^{\hat{s}_0}\right) \\
&\overset{(6.38)}{=} {}^{t\cdot\boldsymbol{U}+\boldsymbol{V}}\mathbf{C}_1 \ ,
\end{aligned}$$

where $t := \mathrm{H}\left(\mathrm{cInd}, C_0, \overrightarrow{\mathbf{C}}\right)$.

This proves the lemma. □

7. Security of the Framework

In this chapter we present the formal proof of security for our CCA-secure framework in prime-order groups. We use many lemmata from the previous chapter in the corresponding reductions, which simplifies the actual proofs. In Section 7.1 we first of all state the main theorem, present an overview of the proof and explain our proof strategy. Then, in Section 7.2 we present the formal proof. Our main reduction steps are presented in Subsection 7.2.1. The proofs in Subsection 7.2.2 require only simple modification in comparison to the proofs for the original CPA-secure framework. The last reduction and the final analysis in Subsection 7.2.3 require simple but important modifications.

7.1. Main Theorem and an Overview of the Proof

In this section we present our main theorem and explain the main steps of the proof regarding the CCA-security.

Theorem 7.1. *Let* Π *be the predicate encryption scheme with public index for relation family* $\mathcal{R}_{\Omega,\Sigma}$ *from Section 4.2.2. Suppose that the* $\mathcal{D}_d$*-Matrix Decisional Diffie-Hellman Assumption from Subsection 6.1.3 is satisfied, the underlying pair encoding scheme* P *is selectively and co-selectively master key hiding, and the family of collision-resistant hash functions* $\mathcal{H}$ *is secure. Then,* Π *is fully CCA-secure with respect to Definition 6.11.*

In particular, for every $\mathrm{des} \in \Omega$ *and every ppt algorithm* $\mathcal{A}$*, there exist ppt algorithms* $\mathcal{B}_1, \ldots, \mathcal{B}_4$ *with essentially the same running time as* $\mathcal{A}$ *such that*

$$\begin{aligned} \text{Adv-aPE}^{\text{CCA2}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) \;\leq\; & \mathrm{Adv}^{\mathrm{CR}}_{\mathcal{H},\mathcal{B}_1}(\lambda) + (5 + 2 \cdot q_1) \cdot \mathrm{Adv}^{\mathcal{D}_d\text{-mDH}}_{\mathcal{B}_2}(\lambda) \\ & + q_1 \cdot \mathrm{Adv}^{\mathrm{CMH}}_{\mathcal{B}_3}(\lambda) + \mathrm{Adv}^{\mathrm{SMH}}_{\mathcal{B}_4}(\lambda) + \frac{q_{\mathrm{dec}} + 1}{2^{\lambda-1}} \;, \end{aligned}$$

where q_1 *is the number of keys that are corrupted in Phase 1 and* q_{dec} *is the upper bound for the number of decryption queries of* $\mathcal{A}$*.*

It is important to notice that the number of decryption queries q_{dec} only appears in the last term and decreases the security guarantees only negligibly. This is mainly due to our proof strategy proposed for the CCA-secure framework in composite-order groups. We used these techniques as a blueprint in our proof as explained below. Furthermore, we notice that in comparison to the CPA-secure framework of [Att16] we only lose the additional terms $\mathrm{Adv}^{\mathrm{CR}}_{\mathcal{H},\mathcal{B}_1}(\lambda)$ and $2 \cdot \mathrm{Adv}^{\mathcal{D}_d\text{-mDH}}_{\mathcal{B}_2}(\lambda)$ which is remarkable and is also due to our proof strategy.

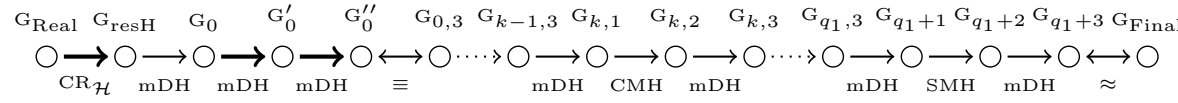


Figure 7.1.: Proof structure for the framework in prime-order groups.

The structure for the proof of Theorem 7.1 is presented in Fig. 7.1 (index k takes values in $[q_1]$). The nodes represent different probability experiments. In Table 7.1 the modifications between the probability experiments are defined. The first experiment $\mathrm{G}_{\mathrm{Real}}$ is the target security experiment $a\mathrm{PE}^{\mathrm{CCA2}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})$ from Fig. 6.1 and the last experiment $\mathrm{G}_{\mathrm{Final}}$ is constructed in such a way that the advantage of every adversary is zero. The edges represent reduction steps and their labels the underlying security assumptions except for the edge labeled with $\equiv$ representing equal experiment and expect for the edge labeled with $\approx$ representing statistically close distributions. In the proof we show that no ppt algorithm can distinguish between any pair of consecutive experiments. Here, we explain the structure of the proof and the intuition behind the main reduction steps with respect to the guarantees of our consistency checks considered in Subsection 6.2.4.

$\mathrm{G}_{\mathrm{resH}}$:	Modify $\langle 20 \rangle$	Output is 0 if there is a collision for H
G_0:	Modify $\langle 12 \rangle$ Modify $\langle 16 \rangle$	$\left(\mathrm{pp}_\kappa, \mathrm{msk}, \widehat{\mathrm{pp}_\kappa}, \widehat{\mathrm{msk}}\right) \leftarrow \mathrm{SFSetup}\left(1^\lambda, \mathrm{des}\right)$: $\mathrm{CT}^* \leftarrow \mathrm{SFEnc}\left(\mathrm{pp}_\kappa, \widehat{\mathrm{pp}_\kappa}, \mathrm{cInd}, m_b\right)$
G'_0:	Modify $\langle 15 \rangle$, $\langle 19 \rangle$ Modify $\langle 13 \rangle$, $\langle 17 \rangle$	$\mathrm{sk}'_i \leftarrow \mathrm{KeyGen}\left(\mathrm{msk}, \mathrm{kInd}_i\right)$, $\mathrm{Dec}\left(\mathrm{sk}'_i, \mathrm{CT}\right)$ Generate keys in oracle **Open**.
G''_0:	Insert Modify $\langle 15 \rangle$, $\langle 19 \rangle$	Choose $\sigma \leftarrow \mathbb{Z}_p$ in **Setup** $\mathbf{Dec}\left(\mathrm{CT}, i\right) := C_0 \cdot \mathrm{e}\left(\mathbf{C}_1, \mathrm{msk} \cdot \widehat{\mathbf{G}}_2^{\sigma}\right)^{-1}$ after verification checks.
$\mathrm{G}_{k,1}$:	Insert Insert Modify $\langle 14 \rangle$	Counter j for number of oracle queries on **Open** in **Phase I**. $\beta_j \leftarrow \mathbb{Z}_p$ for every oracle query **Open** in **Phase I**. $\mathrm{sk}_j \leftarrow \begin{cases} \mathrm{SFKeyGen}\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{msk}, \left(\widehat{\mathbf{G}}_2, \_\right), \mathrm{kInd}, 3; \beta_j\right) & \text{if } j < k \\ \mathrm{SFKeyGen}\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{msk}, \widehat{\mathrm{msk}}, \mathrm{kInd}, 1\right) & \text{if } j = k \\ \mathrm{KeyGen}\left(\mathrm{msk}, \mathrm{kInd}\right) & \text{if } j > k \end{cases}$
$\mathrm{G}_{k,2}$:	Modify $\langle 14 \rangle$	$\mathrm{sk}_j \leftarrow \begin{cases} \mathrm{SFKeyGen}\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{msk}, \left(\widehat{\mathbf{G}}_2, \_\right), \mathrm{kInd}, 3; \beta_j\right) & \text{if } j < k \\ \mathrm{SFKeyGen}\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{msk}, \widehat{\mathrm{msk}}, \mathrm{kInd}, 2; \beta\right) & \text{if } j = k \\ \mathrm{KeyGen}\left(\mathrm{msk}, \mathrm{kInd}\right) & \text{if } j > k \end{cases}$
$\mathrm{G}_{k,3}$:	Modify $\langle 14 \rangle$	$\mathrm{sk}_j \leftarrow \begin{cases} \mathrm{SFKeyGen}\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{msk}, \left(\widehat{\mathbf{G}}_2, \_\right), \mathrm{kInd}, 3; \beta_j\right) & \text{if } j \leq k \\ \mathrm{KeyGen}\left(\mathrm{msk}, \mathrm{kInd}\right) & \text{if } j > k \end{cases}$
G_{q_1+1}:	Modify $\langle 18 \rangle$	$\mathrm{SFKeyGen}\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{msk}, \widehat{\mathrm{msk}}, \mathrm{kInd}, 1\right)$
G_{q_1+2}:	Insert Modify $\langle 18 \rangle$	$\beta \leftarrow \mathbb{Z}_p$ at the beginning of **Phase II** $\mathrm{SFKeyGen}\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{msk}, \widehat{\mathrm{msk}}, \mathrm{kInd}, 2; \beta\right)$
G_{q_1+3}:	Modify $\langle 18 \rangle$	$\mathrm{SFKeyGen}\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{msk}, \left(\widehat{\mathbf{G}}_2, \_\right), \mathrm{kInd}, 3; \beta\right)$
$\mathrm{G}_{\mathrm{Final}}$:	Modify $\langle 16 \rangle$	$m \leftarrow \mathcal{M}$, $\mathrm{CT}^* \leftarrow \mathrm{Enc}\left(\mathrm{cInd}^*, m\right)$

Table 7.1.: The probability experiments for the framework in prime-order groups.

The structure of the proof for our CCA-secure framework is similar to the structure of the proof for the underlying CPA-secure framework of [Att16]. Probability experiments $\mathrm{G}_{\mathrm{resH}}$, G'_0, and G''_0 as well as the three reduction steps denoted by bold edges in Fig. 7.1 are new. The remaining experiments and reductions are from the original CPA-security proof from [Att14a] and require only relatively simple extensions.

Before we proceed, let us shortly recall the overall proof structure for schemes constructed using Dual System Encryption Techniques [Wat09a, LW10]. As in our case all proofs for such schemes consist of a sequence of indistinguishable probability experiments. At the beginning of the sequence the challenge ciphertext CT^* is changed from normal to semi-functional. In

the last probability experiment the challenge ciphertext is again changed. This time CT^* is a ciphertext of a uniformly at random chosen message. This ensures that adversary gets no information about the challenge bit and its advantage in the final game is equal zero. Our reduction steps $G_{resH} \rightarrow G_0$ and $G_{q_1+3} \rightarrow G_{Final}$ correspond to these general steps. In between of these two steps all user secret keys are changed one by one from normal to semi-functional, which basically enables the possibility for the last step. When Attrapadung introduced pair encodings and presented the pair encoding framework in composite-order groups [Att14a], he already refined these reduction steps. Namely, he differs between three types of semi-functional keys. Then, in the first query phase the keys are changed one by one, but in three steps: from normal to semi-functional of type 1, from semi-functional of type 1 to semi-functional of type 2, and finally from semi-functional of type 2 to the semi-functional of type 3. Our reduction steps $G_{k-1,3} \rightarrow G_{k,1} \rightarrow G_{k,2} \rightarrow G_{k,3}$ correspond to these steps (key number k is changed). In the second query phase all keys are simultaneously changed in the sequence $G_{q_1,3} \rightarrow G_{q_1+1} \rightarrow G_{q_1+2} \rightarrow G_{q_1+3}$. Only the steps from type 2 to type 3 semi-functional keys are based on security properties of the corresponding pair encodings and are specific for every predicate. The other two steps are alike for all predicates. As already stated, all these reduction steps require only slight modification, since we have to show how to compute the additional elements in public parameters as well as in the challenge ciphertext. Furthermore, as already noticed in the construction of algorithm Setup we made some of the elements from the master secret key of the underlying CPA-secure scheme public. Hence, we also prove that this does not influence the security. In the CPA-secure framework all these elements were part of the master secret key, since they are only required for generation of the user secret keys. The actual master secret key is the same as in our construction.

Finally, we explain our new reduction steps which deal with CCA-security. The first step from G_{Real} to G_{resH} is quite simple and is due to the collision resistance of the exploited hash function. Namely, the output of the experiment G_{resH} is defined to be zero if adversary submited a ciphertext CT such that the corresponding hash value is equal to the hash value of the challenge ciphertext but the inputs of the hash function differ for these ciphertexts. Hence, CT and CT^* immediately lead to a collision for the hash function.

Next, in order to deal with the decryption queries we use our proof strategy introduced for the framework in composite-order groups as a blueprint. In experiment G'_0 decryption queries are answered using separately generated normal user secret keys, which are never given to adversary. Only those keys which are corrupted by adversary are changed to semi-functional. This is the reason why security guarantees in Theorem 7.1 decrease only negligible in the number of decryption queries. Notice that according to our strong security model adversary can ask to reveal a secret key number i after several decryption queries for this key. We proved that a ppt adversary is not able to realize that the given key was not the key used in the previous decryption queries, as explained next.

The reduction step $G_0 \rightarrow G'_0$ is the most challenging part of the proof and significantly differs from the corresponding reduction step in the proof in the second part of the thesis. Namely, in composite-order groups we constructed consistency checks which ensure that the form of the normal components of the ciphertext is correct. Consequently, in this construction it does not matter which normal user secret key is used for decryption and the corresponding reduction step does not even require a security assumption. In prime-order groups, our consistency checks do not give such strong guarantees mainly due to the fact that the normal components as well as the semi-functional components are not strictly separated as this is the case in groups of composite order, where these components are in different subgroups.

The proof of the reduction step $G_0 \rightarrow G'_0$ is done in two steps. First of all we define a BadQuery event in the probability space of G_0 and prove that the probability for this event is negligible under the security assumption. The BadQuery event covers our intuition behind the consistency checks explained in Subsection 6.2.5. Namely, this event is defined in such a

way that it occurs if $\mathcal{A}$ submits a ciphertext CT, which pass all verification checks but there is a ciphertext element $\mathbf{C}_i$ such that it holds $\xi_i \neq 0 \pmod p$ for the additional factor $\widehat{\mathbf{G}}_1^{\xi_i}$ of $\mathbf{C}_i$. This is indeed not obvious from the definition of BadQuery and is proved in the actual reduction for $\mathrm{G}_0 \rightarrow \mathrm{G}_0'$. Then, the conclusion is simple, since $\xi_i = 0 \pmod p$ for all elements in the ciphertext ensures that CT has the form of a semi-functional ciphertext, and hence G_0 and G_0' cannot differ. As the additional statement of this proof we show that the result of the decryption query on CT is $m = C_0 \cdot \mathrm{e}\left(\mathbf{C}_1, \mathrm{msk}\right)^{-1}$, which is used in the following reduction.

The last step regarding CCA-security is the reduction $\mathrm{G}_0' \rightarrow \mathrm{G}_0''$. In the experiment G_0'' all decryption queries are answered without user secret keys by $m = C_0 \cdot \mathrm{e}\left(\mathbf{C}_1, \mathrm{msk} \cdot \widehat{\mathbf{G}}_2^{\sigma}\right)^{-1}$, where $\sigma \leftarrow \mathbb{Z}_p$ is chosen once in the setup phase. Notice that in composite-order groups a similar step was induced at the end of the reduction sequence. In prime-order groups we have to deal with this reduction before the keys are changed. The critical observation is that in all reduction steps except for the last step the master secret key is known, and hence m from above can be efficiently computed. In the last step the master secret key is not known, but we can compute an element corresponding to $\mathrm{msk} \cdot \widehat{\mathbf{G}}_2^{\sigma}$ and use it for all decryption queries. Notice that due to this circumstance we cannot simple output $m = C_0 \cdot \mathrm{e}\left(\mathbf{C}_1, \mathrm{msk}\right)^{-1}$ as expected in G_0'. Due to the applied technique all reductions in between regarding the user secret keys remain almost unchanged. Obviously, the experiments G_0' and G_0'' differ if and only if $\mathcal{A}$ submits a ciphertext CT which pass the consistency checks such that $\mathrm{e}\left(\mathbf{C}_1, \mathrm{msk}\right) \neq \mathrm{e}\left(\mathbf{C}_1, \mathrm{msk} \cdot \widehat{\mathbf{G}}_2^{\sigma}\right)$. In the proof we show that this cannot happen except for negligible probability due to the security assumption. In this reduction step our additional element $\mathbf{C}''$ plays the main role. We basically prove that such a ciphertext (with $\mathbf{C}''$ which pass the consistency check) cannot be computed by a ppt adversary except for negligible probability over the random choice of $\boldsymbol{U}$ and $\boldsymbol{V}$, which again covers the intuition from Subsection 6.2.5.

7.2. Security Proof for the CCA-Secure Pair Encoding Framework

In this section we present the formal proof for our framework in prime-order groups. It is important to notice that due to the special construction of the scheme, values of some random variables which determine the output of the setup algorithm are informational-theoretically hidden from adversary which is given the public parameters. That is, the public parameter do not contain information about these values. The same holds for the user secret keys and for the challenge ciphertext. Hence, in order to analyze the probability distributions we usually show how to define certain random values and argue that these are correctly distributed. We remark that not all these values will be known in the corresponding reduction algorithms. However, we prove that all the elements which must be given to adversary can be efficiently computed according to the defined random values.

7.2.1. Main Reduction Steps Regarding CCA-Security

In this subsection we prove the first four reduction steps, including our three main reduction steps which are new. The reduction steps $\mathrm{G}_0 \rightarrow \mathrm{G}_0'$ and $\mathrm{G}_0' \rightarrow \mathrm{G}_0''$ are quite complex such that we divided the corresponding proofs in several parts.

First Reduction $\mathrm{G}_{\mathrm{Real}} \rightarrow \mathrm{G}_{\mathrm{resH}}$

Probability experiment $\mathrm{G}_{\mathrm{Real},\Pi,\mathcal{A}}\left(\lambda, \mathrm{des}\right)$, which we simply denote by $\mathrm{G}_{\mathrm{Real}}$ is the original CCA-security experiment $\mathrm{aPE}_{\Pi,\mathcal{A}}^{\mathrm{CCA2}}\left(\lambda, \mathrm{des}\right)$ presented in Subsection 6.1.4. Probability experiment $\mathrm{G}_{\mathrm{resH},\Pi,\mathcal{A}}\left(\lambda, \mathrm{des}\right)$ (also denoted by $\mathrm{G}_{\mathrm{resH}}$) is defined over the same probability space as $\mathrm{G}_{\mathrm{Real}}$ and differs only in the definition of the output of the experiment, defined next.

First of all recall the definition of function HInput from algorithm Enc on page 147. We define event HashAbort in the probability space of $\mathrm{G}_{\mathrm{Real}}$ (respectively $\mathrm{G}_{\mathrm{resH}}$) as follows: $\mathcal{A}$ submitted at least one ciphertext CT to the decryption oracle (in **Phase I** or in **Phase II**) such that CT passed the consistency checks, $\mathrm{HInput}(\mathrm{CT}) \neq \mathrm{HInput}(\mathrm{CT}^*)$ but $t = \mathrm{H}(\mathrm{HInput}(\mathrm{CT})) = \mathrm{H}(\mathrm{HInput}(\mathrm{CT}^*)) = t^*$. According to the HashAbort event the output in experiment $\mathrm{G}_{\mathrm{resH}}$ is defined as follows:

- Output $\overline{\mathrm{HashAbort}} \wedge b' = b$.

That is, the output of the experiment is 0 if HashAbort event occurs and otherwise the output is the same as in the experiment $\mathrm{G}_{\mathrm{Real}}$.

Lemma 7.2. *For every ppt* $\mathcal{A} \in \mathbf{A}_{\Pi}$ *and every* $\mathrm{des} \in \Omega$ *there exists a ppt algorithm* $\mathcal{B}$ *such that it holds*

$$\left|\mathrm{Adv}^{\mathrm{G}_{\mathrm{Real}}}_{\Pi,\mathcal{A}}(\lambda,\mathrm{des}) - \mathrm{Adv}^{\mathrm{G}_{\mathrm{resH}}}_{\Pi,\mathcal{A}}(\lambda,\mathrm{des})\right| \leq \Pr[\mathrm{HashAbort}] = \mathrm{Adv}^{\mathrm{CR}}_{\mathcal{H},\mathcal{B}}(\lambda) \quad .$$

Proof. Let $\mathcal{A} \in \mathbf{A}_{\Pi}$ and $\mathrm{des} \in \Omega$ be arbitrary but fixed. By the definitions of $\mathrm{G}_{\mathrm{Real}}$ and $\mathrm{G}_{\mathrm{resH}}$ the outputs of these experiments can differ only if HashAbort event occurs. In particular it holds

$$\mathrm{G}_{\mathrm{Real},\Pi,\mathcal{A}}(\lambda,\mathrm{des}) = 1 \wedge \overline{\mathrm{HashAbort}} \quad \Leftrightarrow \quad \mathrm{G}_{\mathrm{resH},\Pi,\mathcal{A}}(\lambda,\mathrm{des}) = 1 \wedge \overline{\mathrm{HashAbort}} \ ,$$

and we can apply Difference Lemma 5.8. We deduce

$$\begin{aligned} & \left|\mathrm{Adv}^{\mathrm{G}_{\mathrm{Real}}}_{\Pi,\mathcal{A}}(\lambda,\mathrm{des}) - \mathrm{Adv}^{\mathrm{G}_{\mathrm{resH}}}_{\Pi,\mathcal{A}}(\lambda,\mathrm{des})\right| \\ \overset{\text{by def.}}{=} \ & \left|\Pr[\mathrm{G}_{\mathrm{Real},\Pi,\mathcal{A}}(\lambda,\mathrm{des}) = 1] - \Pr[\mathrm{G}_{\mathrm{resH},\Pi,\mathcal{A}}(\lambda,\mathrm{des}) = 1]\right| \\ \leq \ & \Pr[\mathrm{HashAbort}] \quad , \end{aligned}$$

where the inequality holds by Difference Lemma 5.8.

Next, due to the collision-resistance of $\mathcal{H}$ no ppt algorithm $\mathcal{A}$ can distinguish between the experiments. Namely, there is ppt algorithm $\mathcal{B}$ with the following advantage against $\mathcal{H}$:

$$\mathrm{Adv}^{\mathrm{CR}}_{\mathcal{H},\mathcal{B}}(\lambda,\mathrm{des}) = \Pr[\mathrm{HashAbort}] \quad .$$

$\mathcal{B}$ given the hash function H generated by $\mathrm{H} \leftarrow \mathcal{H}(1^{\lambda})$ simulates experiment $\mathrm{G}_{\mathrm{Real},\Pi,\mathcal{A}}(\lambda,\mathrm{des})$ for $\mathcal{A}$, ignores its output, and outputs collision $(\mathrm{HInput}(\mathrm{CT}), \mathrm{HInput}(\mathrm{CT}^*))$ if HashAbort event occurs. □

Notice that we cannot directly deduce that conditioned on $\overline{\mathrm{HashAbort}}$ it holds $t \neq t^*$ for every submitted ciphertext CT. Additional arguments are required and we deal with this in the following reduction steps.

Remark 7.3. It is important to notice that HashAbort event is recognizable in polynomial time. Hence, analyzing the advantage of adversary $\mathcal{A}$ in all following reduction steps we simply assume that $\mathcal{A}$ does not cause this event, since conditioned on HashAbort the output of all following experiments is always zero – that is, $\mathcal{A}$ loses.

Second Reduction $\mathrm{G}_{\mathrm{resH}} \to \mathrm{G}_0$

Probability experiment G_0 (that is $\mathrm{G}_{0,\mathcal{A},\Pi}(\lambda, \mathrm{des})$) is the same as $\mathrm{G}_{\mathrm{resH}}$ except for the challenge, which is semi-functional. That is, additionally to the public parameters the semi-functional parameters are generated in G_0. Furthermore, the semi-functional ciphertext in G_0 requires additional random elements $\hat{s}_0, \ldots, \hat{s}_{w_2}$ taking values in $\mathbb{Z}_p$. Despite these extensions both experiment can be seen as defined over the same probability space. On the one hand, SFSetup generates the public parameters using Setup (recall that SFSetup does not choose any additional random elements). Hence, also in $\mathrm{G}_{\mathrm{resH}}$ we can use SFSetup and simply ignore the semi-functional components. On the other hand, in $\mathrm{G}_{\mathrm{resH}}$ the semi-function random elements for the challenge are just ignored.

The following lemma is very similar to Lemma 4 in [Att15]. We only have to extend the proof due to our additional elements $\mathbf{U}$, $\mathbf{V}$, $\overline{\mathbf{U}}$, and $\overline{\mathbf{V}}$ in the public parameters and due to the HashAbort event. Notice that during the analysis we fall back to our lemmata from Section 6.2 which make the description of the reductions simpler and the formal arguments more explicit.

Remark 7.4. As already mentioned by the definition of our framework, elements $\mathbf{G}_2, \left\{\overline{\mathbf{H}}_j\right\}_{j\in[n]}$ generated by setup algorithm were part of the master secret key in the original CPA-secure framework, whereas in our framework these values are contained in the public parameter. Hence, in all reductions we have to take care of these elements in addition, but actually this is not difficult and is implicitly covered by the proof of the original framework [Att16], where these elements are known to the reduction algorithm and can be correctly generated. Therefore it is not our achievement to show how to generate these elements.

Lemma 7.5 (cf. Lemma 4 in [Att15])**.** *For every ppt* $\mathcal{A} \in \mathbf{A}_\Pi$ *and every* $\mathrm{des} \in \Omega$ *there exists a ppt algorithm* $\mathcal{B}$ *such that*

$$\left|\mathrm{Adv}^{\mathrm{G}_{\mathrm{resH}}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) - \mathrm{Adv}^{\mathrm{G}_0}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})\right| = \mathrm{Adv}^{\mathcal{D}_d\text{-mDH}}_{\mathcal{B}}(\lambda) \quad .$$

Proof. Let $\mathcal{A} \in \mathbf{A}_\Pi$ and $\mathrm{des} \in \Omega$ be arbitrary but fixed. We present an algorithm $\mathcal{B}$, which receives des as an additional input. According to Assumption 6.1 algorithm $\mathcal{B}$ is also given the group description $\mathbb{GD} = (p, (g_1, \mathbb{G}_1), (g_2, \mathbb{G}_2), \mathbb{G}_\mathrm{T}, \mathrm{e})$, $\mathbf{T} = g_1^{\boldsymbol{T}} \in \mathbb{G}^{(d+1)\times(d+1)}$ as well as the actual challenge $\mathbf{Z} = g_1^{\boldsymbol{T}\cdot\binom{\boldsymbol{y}}{\hat{y}}} \in \mathbb{G}_1^{d+1}$. The elements are chosen by $\mathbb{GD} \leftarrow \mathcal{G}(1^\lambda)$, $\boldsymbol{T} = \begin{pmatrix} \boldsymbol{M} & \boldsymbol{0} \\ \boldsymbol{c}^\top & 1 \end{pmatrix} \leftarrow \mathcal{D}_d$, $\boldsymbol{y} \leftarrow \mathbb{Z}_p^d$, where $\boldsymbol{M} \in \mathbb{GL}_{p,d}$ and $\boldsymbol{c} \in \mathbb{Z}_p^d$. Furthermore, $\hat{y}$ is either zero or uniformly distributed in $\mathbb{Z}_p^*$ and $\mathcal{B}$ has to distinguish between these both cases.

Let us first consider how to generate the public parameters. The first part of $\mathcal{B}$ is presented in Algorithm 17. First of all notice that $\mathcal{B}$ is a ppt algorithm, since it performs only simple computations.

We claim that $\mathcal{B}$ implicitly sets $\boldsymbol{B} := \tilde{\boldsymbol{B}} \cdot \boldsymbol{T}$. By the definition of Assumption 6.1 it holds $\boldsymbol{T} \in \mathbb{GL}_{p,d+1}$, whereas $\tilde{\boldsymbol{B}}$ is uniformly distributed over $\mathbb{GL}_{p,d+1}$ and is mutually independently of the other random values by construction of $\mathcal{B}$. Hence, $\boldsymbol{B}$ is uniformly distributed over $\mathbb{GL}_{p,d+1}$ by the choice of $\tilde{\boldsymbol{B}}$ as required by the setup algorithm. Analogously, $\mathcal{B}$ implicitly sets $\tilde{\boldsymbol{D}} := \boldsymbol{M}^\top \cdot \boldsymbol{J}$, which is correctly distributed by the choice of $\boldsymbol{J} \leftarrow \mathbb{GL}_{p,d}$ since $\boldsymbol{M} \in \mathbb{GL}_{p,d}$ by the definition of Assumption 6.1.

Next we prove that $\mathcal{B}$ computes $\mathbf{G}_1$, $\widehat{\mathbf{G}}_1$, and $\mathbf{G}_2$ according to choice of $\boldsymbol{B}$ and $\tilde{\boldsymbol{D}}$. Recall the definition of $\mathbf{G}_1$, $\widehat{\mathbf{G}}_1$ from algorithm Setup on page 146 and from algorithm SFSetup on

Algorithm 17: $\mathcal{B}$ against Assumption 6.1

Input : $(\mathrm{des}, \mathbb{GD}, \mathbf{T}, \mathbf{Z})$.

Require: $\mathrm{des} \in \Omega$, $\mathbb{GD} = (p, (g_1, \mathbb{G}_1), (g_2, \mathbb{G}_2), \mathbb{G}_\mathrm{T}, \mathrm{e}) \in \left[\mathcal{G}\left(1^\lambda\right)\right]$, $\mathbf{T} \in \mathbb{G}_1^{(d+1)\times(d+1)}$, $\mathbf{Z} \in \mathbb{G}_1^{d+1}$.

Setup

Set $\kappa := (p, \mathrm{des})$;

Choose $\tilde{\boldsymbol{B}} \leftarrow \mathbb{GL}_{p,d+1}$ and $\boldsymbol{J} \leftarrow \mathbb{GL}_{p,d}$;

Compute

$$\begin{pmatrix} \mathbf{G}_1 & \widehat{\mathbf{G}}_1 \end{pmatrix} := {}^{\tilde{\boldsymbol{B}}}\mathbf{T} \ ;$$

Compute

$$\mathbf{G}_2 := g_2^{\tilde{\boldsymbol{B}}^{-\top} \cdot \binom{\boldsymbol{J}}{\mathbf{0}^\top}} \ ;$$

Choose $\boldsymbol{H}_1, \ldots, \boldsymbol{H}_n, \boldsymbol{U}, \boldsymbol{V} \leftarrow \mathbb{Z}_p^{(d+1)\times(d+1)}$ and compute $\left\{\mathbf{H}_j, \overline{\mathbf{H}}_j\right\}_{j\in[n]}$, $\mathbf{U}$, $\mathbf{V}$, $\overline{\mathbf{U}}$, $\overline{\mathbf{V}}$ as defined in the scheme using $\mathbf{G}_1$ and $\mathbf{G}_2$;

Pick $\mathrm{H} \leftarrow \mathcal{H}\left(1^\lambda\right)$ and $\boldsymbol{\alpha} \leftarrow \mathbb{Z}_p^{d+1}$, compute msk and $\mathbf{Y}$ as defined in the scheme.

Simulate $\mathcal{A}$ with public parameters

$$\mathrm{pp}_\kappa := \left(\mathrm{des}, \mathbb{GD}, \mathrm{H}, \mathbf{G}_1, \{\mathbf{H}_j\}_{j\in[n]}, \mathbf{U}, \mathbf{V}, \mathbf{Y}, \mathbf{G}_2, \left\{\overline{\mathbf{H}}_j\right\}_{j\in[n]}, \overline{\mathbf{U}}, \overline{\mathbf{V}}\right) \ ;$$

page 149. We deduce that it holds:

$$\begin{aligned} \left(\mathbf{G}_1 \quad \widehat{\mathbf{G}}_1\right) &\overset{\text{by def.}}{=} g_1^{\boldsymbol{B}} \\ &= g_1^{\tilde{\boldsymbol{B}}\cdot\boldsymbol{T}} \\ &= {}^{\tilde{\boldsymbol{B}}}\mathbf{T} \ . \end{aligned} \tag{7.1}$$

We notice that $\widehat{\mathbf{G}}_1$ is known to $\mathcal{B}$, but is not used.

According to the definition of $\mathbf{G}_2$ from algorithm Setup on page 146 this element is also correctly computed:

$$\begin{aligned} \mathbf{G}_2 &\overset{\text{by def.}}{=} g_2^{\boldsymbol{Z}\cdot\binom{\boldsymbol{I}_d}{\mathbf{0}}} \\ &= g_2^{\boldsymbol{B}^{-\top}\cdot\left(\begin{smallmatrix}\tilde{\boldsymbol{D}} & \mathbf{0}\\ \mathbf{0} & 1\end{smallmatrix}\right)\cdot\binom{\boldsymbol{I}_d}{\mathbf{0}}} \\ &= g_2^{\left(\tilde{\boldsymbol{B}}\cdot\boldsymbol{T}\right)^{-\top}\cdot\binom{\tilde{\boldsymbol{D}}}{\mathbf{0}}} \\ &= g_2^{\tilde{\boldsymbol{B}}^{-\top}\cdot\boldsymbol{T}^{-\top}\cdot\binom{\boldsymbol{M}^\top\cdot\boldsymbol{J}}{\mathbf{0}^\top}} \\ &= g_2^{\tilde{\boldsymbol{B}}^{-\top}\cdot\binom{\boldsymbol{J}}{\mathbf{0}^\top}} \ , \end{aligned}$$

where in the last step we use the fact that the inverse matrix of $\boldsymbol{T}$ has the form $\boldsymbol{T}^{-1} = \left(\begin{smallmatrix}\boldsymbol{M}^{-1} & \mathbf{0}\\ \boldsymbol{v}^\top & 1\end{smallmatrix}\right)$ for vector $\boldsymbol{v} = -\boldsymbol{M}^{-\top}\cdot\boldsymbol{c}$, and hence $\boldsymbol{T}^{-\top}\cdot\binom{\boldsymbol{M}^\top\boldsymbol{J}}{\mathbf{0}^\top} = \left(\begin{smallmatrix}\boldsymbol{M}^{-\top} & \boldsymbol{v}\\ \mathbf{0}^\top & 1\end{smallmatrix}\right)\cdot\binom{\boldsymbol{M}^\top\cdot\boldsymbol{J}}{\mathbf{0}^\top} = \binom{\boldsymbol{J}}{\mathbf{0}^\top}$. All other elements of pp_κ, inclusive our additional elements $\mathbf{U}$, $\mathbf{V}$, $\overline{\mathbf{U}}$, and $\overline{\mathbf{V}}$ are computed as defined in the setup algorithm by construction. We deduce that $\mathcal{B}$ generates correctly generated public parameters for $\mathcal{A}$.

Algorithm 18: Continuation of Algorithm 17

1 **Phase I**
2 Realize the oracles using known msk as defined in the experiments.
3 **Challenge (given** cInd***,** $m_0, m_1 \in \mathcal{M} = \mathbb{G}_T$ **from** $\mathcal{A}$**)**
4 Choose a random bit $b \leftarrow \{0,1\}$.
5 Compute $\mathbf{S} = (\mathbf{S}_0, \ldots, \mathbf{S}_{w_2}) \in \mathbb{G}_1^{(d+1)\times(w_2+1)}$ by

$$\mathbf{S} \leftarrow \mathrm{ReRand}\left(\mathbb{G}_1, {}^{\tilde{\boldsymbol{B}}}\mathbf{T}, {}^{\tilde{\boldsymbol{B}}}\mathbf{Z}, w_2+1\right) \ ;$$

6 Return
$\mathrm{CT}^* := \left(\mathrm{cInd}^*, C_0, \overrightarrow{\mathbf{C}}, \mathbf{C}''\right) := \mathrm{EncAlt}\left(\kappa, \mathrm{msk}, \mathrm{cInd}^*, m_b, \mathbb{H}, \boldsymbol{U}, \boldsymbol{V}, \mathrm{H}, \mathbf{S}_0, \ldots, \mathbf{S}_{w_2}\right)$;
7 **Phase II**
8 Realize the oracles using known msk as defined in the experiments.
9 **Guess**
10 $\mathcal{A}$ outputs a bit b'. Output 1 if and only if $\overline{\mathrm{HashAbort}} \wedge b' = b$.

Next we consider the other phases of $\mathcal{B}$ (cf. Algorithm 18) and relate its success probability to the success probability of $\mathcal{A}$ in distinguishing both experiments. Using known master secret key both query phases can be perfectly simulated. Hence, let us consider the challenge ciphertext. First of all $\mathcal{B}$ randomizes the own challenge and computes

$$\mathbf{S} = g_1^{\tilde{\boldsymbol{B}}\cdot\boldsymbol{T}\cdot\left(\begin{smallmatrix}\boldsymbol{s}_0 & \boldsymbol{s}_1 & \cdots & \boldsymbol{s}_{w_2}\\ \hat{s}_0 & \hat{s}_1 & \cdots & \hat{s}_{w_2}\end{smallmatrix}\right)} \leftarrow \mathrm{ReRand}\left(\mathbb{G}_1, g_1^{\tilde{\boldsymbol{B}}\cdot\boldsymbol{T}}, g_1^{\tilde{\boldsymbol{B}}\cdot\boldsymbol{T}\cdot\binom{\boldsymbol{y}}{\hat{y}}}, w_2+1\right) \ .$$

By Lemma 6.9 $\boldsymbol{s}_0, \ldots, \boldsymbol{s}_{w_2} \in \mathbb{Z}_p^d$ are uniformly and independently distributeds. Furthermore, if $\hat{y} \neq 0$ (case $\hat{y} \in \mathbb{Z}_p^*$) the elements $\hat{s}_0, \ldots \hat{s}_{w_2} \in \mathbb{Z}_p$ are also uniformly and independently distributed. In turn, if $\hat{y} = 0$ it holds $\hat{s}_0 = \cdots = \hat{s}_{w_2} = 0$.

Next, from (7.1) we deduce that it holds

$$\mathbf{S} = \left(\mathbf{G}_1 \quad \widehat{\mathbf{G}}_1\right)^{\left(\begin{smallmatrix}\boldsymbol{s}_0 & \boldsymbol{s}_1 & \cdots & \boldsymbol{s}_{w_2}\\ \hat{s}_0 & \hat{s}_1 & \cdots & \hat{s}_{w_2}\end{smallmatrix}\right)} \ .$$

That is, for every $i \in [w_2]_0$ it holds $\mathbf{S}_i = \mathbf{G}_1^{\boldsymbol{s}_i} \cdot \widehat{\mathbf{G}}_1^{\hat{s}_i}$. Hence, by Lemma 6.22 in Line 6 $\mathcal{B}$ computes the challenge ciphertext

$$\left(\mathrm{cInd}, C_0, \overrightarrow{\mathbf{C}}, \mathbf{C}''\right) = \mathrm{SFEnc}\left(1^\lambda, \mathrm{pp}_\kappa, \widehat{\mathrm{pp}}_\kappa, \mathrm{cInd}, m; \boldsymbol{s}_0, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_{w_2}\hat{s}_0, \hat{s}_1, \ldots, \hat{s}_{w_2}\right) \ .$$

We deduce that in the case $\mathbf{Z} = g_1^{\boldsymbol{T}\cdot\binom{\boldsymbol{y}}{0}}$ (that is $\hat{y} = 0$) the generated challenge is a correctly distributed normal ciphertext of m_b and in the case of $\mathbf{Z} = g_1^{\boldsymbol{T}\cdot\binom{\boldsymbol{y}}{\hat{y}}}$ (that is $\hat{y} \in \mathbb{Z}_p^*$) the challenge ciphertext is a correctly distributed semi-functional ciphertext of m_b.

All together the view of $\mathcal{A}$ is as defined in game $\mathrm{G}_{\mathrm{Real}}$ if $\hat{y} = 0$ and it is as defined in game G_0 if $\hat{y} \neq 0$. In the guess phase $\mathcal{B}$ outputs 1 if and only if the event HashAbort did not occurred and $\mathcal{A}$ outputs the correct bit – that is, if and only if the output of the corresponding experiment is equal one. Hence, for every $\mathcal{A} \in \mathbf{A}_\Pi$ and every $\mathrm{des} \in \Omega$ there is $\mathcal{B} = \mathcal{B}_\mathcal{A}(\mathrm{des}, \cdot)$ as defined in

Algorithm 17 such that it holds

$$\begin{aligned}\mathrm{Adv}_{\mathcal{B}}^{\mathcal{D}_d-\mathrm{mDH}}(\lambda) &\overset{\text{by def.}}{=} \left|\Pr\left[\mathcal{B}\left(\mathbb{GD}, g_1^{\boldsymbol{T}}, g_1^{\boldsymbol{T}\cdot\binom{\boldsymbol{y}}{0}}\right)=1\right]-\Pr\left[\mathcal{A}\left(\mathbb{GD}, g_1^{\boldsymbol{T}}, g_1^{\boldsymbol{T}\cdot\binom{\boldsymbol{y}}{\hat{y}}}\right)=1\right]\right| \\ &\overset{\text{by const.}}{=} \left|\Pr\left[\mathrm{G}_{\mathrm{Real},\Pi,\mathcal{A}}(\lambda,\mathrm{des})=1\right]-\Pr\left[\mathrm{G}_{0,\Pi,\mathcal{A}}(\lambda,\mathrm{des})=1\right]\right| \\ &\overset{\text{by def.}}{=} \left|\mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{\mathrm{resH}}}(\lambda,\mathrm{des})-\mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_0}(\lambda,\mathrm{des})\right| \ .\end{aligned}$$

This finally proves the lemma. □

BadQuery Event

In this subsection we analyze the event BadQuery defined in the probability space of G_0 as follows: There is $\mathrm{CT} \in \mathbb{C}_{\mathrm{cInd}}$ submitted by $\mathcal{A}$ to the decryption oracle such that CT satisfies the verification checks and at least one of the following cases arises:

1. $\mathbf{C}'' \neq {}^{t\cdot\boldsymbol{U}+\boldsymbol{V}}\mathbf{C}_1$.
2. $\exists_{\tau\in\Gamma} : \mathbf{C}_\tau \neq \prod_{i\in I} {}^{b_{\tau,i}\cdot\boldsymbol{I}_{d+1}+\sum_{j\in[n]} b_{\tau,i,j}\cdot\boldsymbol{H}_j}\mathbf{C}_{\tau_i}$
3. For all $\tau_k \in \overline{\Gamma}$ let $\mathbf{Z}_k := \mathbf{C}_{\tau_k}\cdot\prod_{i\in I} {}^{-b_{\tau_k,i}\cdot\boldsymbol{I}_{d+1}-\sum_{j\in[n]} b_{\tau_k,i,j}\cdot\boldsymbol{H}_j}\mathbf{C}_{\tau_i} \in \mathbb{G}_1^{d+1}$. For all $\tau \in [m_1]$ let $\overline{\mathbf{Z}}_\tau := \prod_{\tau_k\in\overline{\Gamma}} \mathbf{Z}_k^{e_{\tau,\tau_k}} \in \mathbb{G}_1^{d+1}$, where $\boldsymbol{E} = \left(e_{i,j}\right)_{i\in[m_1],j\in[w_1]} \in \mathbb{Z}_p^{m_1\times w_1}$ is the reconstruction matrix used during the corresponding decryption.
   - $\prod_{\tau\in[m_1]} \overline{\mathbf{Z}}_\tau^{a_\tau} \neq \mathbf{1}_{\mathbb{G}_1^{d+1}}$.
   - $\exists_{i\in[m_2]} : \prod_{\tau\in[m_1]} \overline{\mathbf{Z}}_\tau^{a_{\tau,i}} \neq \mathbf{1}_{\mathbb{G}_1^{d+1}}$.
   - $\exists_{i\in[m_2]}\exists_{j\in[n]} : \prod_{\tau\in[m_1]} \overline{\mathbf{Z}}_\tau^{a_{\tau,i,j}} \neq \mathbf{1}_{\mathbb{G}_1^{d+1}}$.

Remark 7.6. Notice that the BadQuery event cannot be efficiently recognized given only the public parameters. Namely, the first check requires knowledge of $\boldsymbol{U}$, $\boldsymbol{V}$, the second and the third checks require knowledge of $\boldsymbol{H}_1, \ldots, \boldsymbol{H}_n$. Furthermore, the elements $\boldsymbol{H}_1, \ldots, \boldsymbol{H}_n$ will be not known in the reduction steps based on the security properties of pair encodings. Hence, we cannot deal with this event in similar way as with the event HashAbort (cf. Remark 7.3). Instead we prove in the following lemma that the probability $\Pr[\mathrm{BadQuery}]$ is negligible in G_0. Later we will also deduce that the probability for this event is negligible also in the following experiments and will use this in the analysis.

In the following sections we prove that if the event BadQuery does not occur, then the output of the decryption algorithm using any user secret key for the corresponding key index is the same. In turn, the following lemma states that the probability for BadQuery is negligible, and hence similar to the proof for composite-order framework we can use separately generated normal secret keys in order to answer decryption queries.

Lemma 7.7. *For every ppt $\mathcal{A} \in \mathbf{A}_\Pi$, and every $\mathrm{des} \in \Omega$ there exists a ppt algorithm $\mathcal{B}$ such that*

$$\mathrm{Adv}_{\mathcal{B}}^{\mathcal{D}_d-\mathrm{mDH}}(\lambda) = \Pr[\mathrm{BadQuery}] \ ,$$

where the probability space on the right side is defined by experiment $\mathrm{G}_{0,\mathcal{A},\Pi}(\lambda,\mathrm{des})$.

Proof. Let $\mathcal{A} \in \mathbf{A}_{\Pi}$ and $\mathrm{des} \in \Omega$ be arbitrary but fixed. We present an algorithm $\mathcal{B}$, which receives des as an additional input. According to Assumption 6.1 algorithm $\mathcal{B}$ is also given the group description $\mathbb{GD} = (p, (g_1, \mathbb{G}_1), (g_2, \mathbb{G}_2), \mathbb{G}_{\mathrm{T}}, \mathrm{e})$ and challenge $\mathbf{T} = g_2^{\boldsymbol{T}} \in \mathbb{G}^{(d+1)\times(d+1)}$ as well as $\mathbf{Z} = g_2^{\boldsymbol{T}\cdot\binom{\boldsymbol{y}}{\hat{y}}} \in \mathbb{G}_1^{d+1}$. The elements are chosen by $\mathbb{GD} \leftarrow \mathcal{G}(1^\lambda)$, $\boldsymbol{T} = \begin{pmatrix} \boldsymbol{M} & \boldsymbol{0} \\ \boldsymbol{c}^\top & 1 \end{pmatrix} \leftarrow \mathcal{D}_d$, $\boldsymbol{y} \leftarrow \mathbb{Z}_p^d$, where $\boldsymbol{M} \in \mathbb{GL}_{p,d}$ and $\boldsymbol{c} \in \mathbb{Z}_p^d$. Furthermore, $\hat{y}$ is either zero or uniformly distributed in $\mathbb{Z}_p^*$ and $\mathcal{B}$ has to distinguish between these both cases.

We present $\mathcal{B}$ in two steps. First we show how $\mathcal{B}$ simulates experiment G_0 for $\mathcal{A}$ (see Algorithm 19). Then, we show how $\mathcal{B}$ solves the own challenge in **Guess** phase if the BadQuery event occurs.

Algorithm 19: $\mathcal{B}$ against Assumption 6.1

Input : $(\mathrm{des}, \mathbb{GD}, \mathbf{T}, \mathbf{Z})$.

Require: $\mathrm{des} \in \Omega$, $\mathbb{GD} = (p, (g_1, \mathbb{G}_1), (g_2, \mathbb{G}_2), \mathbb{G}_{\mathrm{T}}, \mathrm{e}) \in \left[\mathcal{G}(1^\lambda)\right]$, $\mathbf{T} \in \mathbb{G}_2^{(d+1)\times(d+1)}$, $\mathbf{Z} \in \mathbb{G}_2^{d+1}$.

Setup

Set $\kappa := (p, \mathrm{des})$, compute $n := \mathrm{Pair}(\kappa)$;

Choose $\tilde{\boldsymbol{B}} \leftarrow \mathbb{GL}_{p,d+1}$ and compute

$$\mathbf{G}_1 := g_1^{\tilde{\boldsymbol{B}}\cdot\binom{\boldsymbol{I}_d}{\boldsymbol{0}}} ;$$

Choose $\boldsymbol{J} \leftarrow \mathbb{GL}_{p,d}$ and compute

$$\begin{pmatrix} \mathbf{G}_2 & \widehat{\mathbf{G}}_2 \end{pmatrix} := \mathbf{T}^{\tilde{\boldsymbol{B}}^{-\top}\binom{\boldsymbol{J}\ \boldsymbol{0}}{\boldsymbol{0}\ -1}} ;$$

Choose $\boldsymbol{H}_1, \ldots, \boldsymbol{H}_n, \boldsymbol{U}, \boldsymbol{V} \leftarrow \mathbb{Z}_p^{(d+1)\times(d+1)}$, $\mathrm{H} \leftarrow \mathcal{H}(1^\lambda)$, $\boldsymbol{\alpha} \leftarrow \mathbb{Z}_p^{d+1}$ and compute $\left\{\mathbf{H}_j, \overline{\mathbf{H}}_j\right\}_{j\in[n]}$, $\mathbf{U}$, $\mathbf{V}$, $\overline{\mathbf{U}}$, $\overline{\mathbf{V}}$, msk, $\mathbf{Y}$, and pp_κ as defined in the scheme using $\mathbf{G}_1$ and $\mathbf{G}_2$;

Simulate $\mathcal{A}(1^\lambda, \mathrm{pp}_\kappa)$;

Phase I

Realize the oracles using known msk as defined in the experiment G_0; Thereby store for every decryption query the chosen reconstruction matrix $\boldsymbol{E}$;

Challenge (given cInd^***,** $m_0, m_1 \in \mathcal{M} = \mathbb{G}_{\mathrm{T}}$ **from** $\mathcal{A}$**)**

Compute $(\boldsymbol{c}, w_2) := \mathrm{Enc2}(\kappa, \mathrm{cInd}^*)$, choose a random bit $b \leftarrow \{0, 1\}$;

For all $i \in [w_2]_0$ choose $\binom{\boldsymbol{s}_i'}{\hat{s}_i'} \leftarrow \mathbb{Z}_p^{d+1}$ and compute

$$\mathbf{S}_i := g_1^{\binom{\boldsymbol{s}_i'}{\hat{s}_i'}} \in \mathbb{G}_1^{d+1} ;$$

Return

$$\mathrm{CT}^* := \left(\mathrm{cInd}^*, C_0, \overrightarrow{\mathbf{C}}, \mathbf{C}''\right) := \mathrm{EncAlt}(\kappa, \mathrm{msk}, \mathrm{cInd}^*, m_b, \mathbb{H}, \boldsymbol{U}, \boldsymbol{V}, \mathrm{H}, \mathbf{S}_0, \ldots, \mathbf{S}_{w_2});$$

Phase II

Realize the oracles using known msk as defined in the experiment G_0; Thereby store for every decryption query the chosen reconstruction matrix $\boldsymbol{E}$;

$\mathcal{B}$ is given among other elements group description $\mathbb{GD}$ which is correctly distributed by

definition of Assumption 6.1. In the setup phase $\mathcal{B}$ implicitly sets

$$\boldsymbol{B} := \tilde{\boldsymbol{B}} \cdot \begin{pmatrix} \boldsymbol{I}_d & \boldsymbol{M}^{-\top} \cdot \boldsymbol{c} \\ \boldsymbol{0} & -1 \end{pmatrix} ,$$

$$\tilde{\boldsymbol{D}} := \boldsymbol{M} \cdot \boldsymbol{J} . \tag{7.2}$$

These matrices are uniformly distributed over $\mathbb{GL}_{p,d+1}$ and $\mathbb{GL}_{p,d}$ respectively due to the choice of $\tilde{\boldsymbol{B}}$ and $\boldsymbol{J}$ respectively, since $\left(\begin{smallmatrix} \boldsymbol{I}_d & \boldsymbol{M}^{-\top}\cdot\boldsymbol{c} \\ \boldsymbol{0} & -1 \end{smallmatrix}\right) \in \mathbb{GL}_{p,d+1}$ and $\boldsymbol{M} \in \mathbb{GL}_{p,d}$ by definition of Assumption 6.1.

Let us first consider matrix $\boldsymbol{Z}$ as defined in algorithm Setup on page 146. For matrix $\boldsymbol{B}$ as above it holds

$$\boldsymbol{B}^{-1} = \begin{pmatrix} \boldsymbol{I}_d & \boldsymbol{M}^{-\top} \cdot \boldsymbol{c} \\ \boldsymbol{0} & -1 \end{pmatrix} \cdot \tilde{\boldsymbol{B}}^{-1} ,$$

since $\left(\begin{smallmatrix} \boldsymbol{I}_d & \boldsymbol{M}^{-\top}\cdot\boldsymbol{c} \\ \boldsymbol{0} & -1 \end{smallmatrix}\right)^2 = \boldsymbol{I}_{d+1}$. Hence, it holds

$$\begin{aligned} \boldsymbol{B}^{-\top} &= \tilde{\boldsymbol{B}}^{-\top} \cdot \begin{pmatrix} \boldsymbol{I}_d & \boldsymbol{M}^{-\top} \cdot \boldsymbol{c} \\ \boldsymbol{0} & -1 \end{pmatrix}^{\top} \\ &= \tilde{\boldsymbol{B}}^{-\top} \cdot \begin{pmatrix} \boldsymbol{I}_d & \boldsymbol{0} \\ \boldsymbol{c}^{\top}\boldsymbol{M}^{-1} & -1 \end{pmatrix} , \end{aligned} \tag{7.3}$$

and $\boldsymbol{Z}$ is as follows (recall $\boldsymbol{D} \overset{\text{by def.}}{=} \left(\begin{smallmatrix} \tilde{\boldsymbol{D}} & \boldsymbol{0} \\ \boldsymbol{0} & 1 \end{smallmatrix}\right)$)

$$\begin{aligned} \boldsymbol{Z} &\overset{\text{by def.}}{=} \boldsymbol{B}^{-\top} \cdot \boldsymbol{D} \\ &\overset{(7.3),(7.2)}{=} \tilde{\boldsymbol{B}}^{-\top} \cdot \begin{pmatrix} \boldsymbol{I}_d & \boldsymbol{0} \\ \boldsymbol{c}^{\top}\boldsymbol{M}^{-1} & -1 \end{pmatrix} \cdot \begin{pmatrix} \boldsymbol{M}\cdot\boldsymbol{J} & \boldsymbol{0} \\ \boldsymbol{0} & 1 \end{pmatrix} \\ &= \tilde{\boldsymbol{B}}^{-\top} \cdot \begin{pmatrix} \boldsymbol{I}_d & \boldsymbol{0} \\ \boldsymbol{c}^{\top}\boldsymbol{M}^{-1} & -1 \end{pmatrix} \cdot \begin{pmatrix} \boldsymbol{M} & \boldsymbol{0} \\ \boldsymbol{0} & -1 \end{pmatrix} \cdot \begin{pmatrix} \boldsymbol{J} & \boldsymbol{0} \\ \boldsymbol{0} & -1 \end{pmatrix} \\ &= \tilde{\boldsymbol{B}}^{-\top} \cdot \begin{pmatrix} \boldsymbol{M} & \boldsymbol{0} \\ \boldsymbol{c}^{\top} & 1 \end{pmatrix} \cdot \begin{pmatrix} \boldsymbol{J} & \boldsymbol{0} \\ \boldsymbol{0} & -1 \end{pmatrix} \\ &= \tilde{\boldsymbol{B}}^{-\top} \cdot \boldsymbol{T} \cdot \begin{pmatrix} \boldsymbol{J} & \boldsymbol{0} \\ \boldsymbol{0} & -1 \end{pmatrix} . \end{aligned}$$

Next, we prove that $\mathbf{G}_1$, $\mathbf{G}_2$ and $\widehat{\mathbf{G}}_2$ are generated according to the choices of $\boldsymbol{B}$ and $\tilde{\boldsymbol{D}}$. Recall the definition of $\mathbf{G}_1$, $\mathbf{G}_2$, $\widehat{\mathbf{G}}_2$ from algorithm Setup on page 146 and from algorithm SFSetup on page 149. We deduce that it holds:

$$\begin{aligned} \begin{pmatrix} \mathbf{G}_2 & \widehat{\mathbf{G}}_2 \end{pmatrix} &\overset{\text{by def.}}{=} g_2^{\boldsymbol{Z}} \\ &= \mathbf{T}^{\tilde{\boldsymbol{B}}^{-\top}\left(\begin{smallmatrix} \boldsymbol{J} & \boldsymbol{0} \\ \boldsymbol{0} & -1 \end{smallmatrix}\right)} , \end{aligned}$$

$$\begin{aligned} \mathbf{G}_1 &\overset{\text{by def.}}{=} g_1^{\boldsymbol{B}\cdot\left(\begin{smallmatrix} \boldsymbol{I}_d \\ \boldsymbol{0} \end{smallmatrix}\right)} \\ &= g_1^{\tilde{\boldsymbol{B}}\cdot\left(\begin{smallmatrix} \boldsymbol{I}_d & \boldsymbol{M}^{-\top}\cdot\boldsymbol{c} \\ \boldsymbol{0} & -1 \end{smallmatrix}\right)\cdot\left(\begin{smallmatrix} \boldsymbol{I}_d \\ \boldsymbol{0} \end{smallmatrix}\right)} \\ &= g_1^{\tilde{\boldsymbol{B}}\cdot\left(\begin{smallmatrix} \boldsymbol{I}_d \\ \boldsymbol{0} \end{smallmatrix}\right)} . \end{aligned}$$

All other elements are computed as defined in the setup algorithm using $\mathbf{G}_1$ and $\mathbf{G}_2$. Furthermore, in both query phases $\mathcal{B}$ generates the normal keys as defined in the scheme using msk and perfectly answers the queries of $\mathcal{A}$.

Remark 7.8. Notice that $\widehat{\mathbf{G}}_2$ is not used by $\mathcal{B}$. We include the simulation of this element since this will be important for the proof of the following lemma which use similar simulation algorithm (cf. Lemma 7.13).

Next, let us consider the challenge ciphertext. It holds $\boldsymbol{B} \in \mathbb{GL}_{p,d+1}$ by definition, and hence for all $i \in [w_2]_0$ there exist unique $\boldsymbol{s}_i \in \mathbb{Z}_p^d$, $\hat{s}_i \in \mathbb{Z}_p$ such that $\boldsymbol{S}_i$ computed by $\mathcal{B}$ satisfies

$$\forall_{i\in[w_2]_0} : \mathbf{S}_i \overset{\text{by const.}}{=} g_1^{\binom{\boldsymbol{s}'_i}{\hat{s}'_i}} = g_1^{\boldsymbol{B}\cdot\binom{\boldsymbol{s}_i}{\hat{s}_i}} \overset{\text{by def.}}{=} \left(\mathbf{G}_1 \quad \widehat{\mathbf{G}}_1\right)^{\binom{\boldsymbol{s}_i}{\hat{s}_i}} = \mathbf{G}_1^{\boldsymbol{s}_i} \cdot \widehat{\mathbf{G}}_1^{\hat{s}_i} \in \mathbb{G}_1^{d+1} \ .$$

Elements $\boldsymbol{s}_i \in \mathbb{Z}_p^d$ and $\hat{s}_i \in \mathbb{Z}_p$ are uniformly and independently distributed by the choice of $\boldsymbol{s}'_i$ and $\hat{s}'_i$. Hence, by Lemma 6.22 in Line 12 $\mathcal{B}$ computes challenge ciphertext

$$\left(\text{cInd}, C_0, \overrightarrow{\mathbf{C}}, \mathbf{C}''\right) = \text{SFEnc}\left(1^\lambda, \text{pp}_\kappa, \widehat{\text{pp}}_\kappa, \text{cInd}, m; \boldsymbol{s}_0, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_{w_2}\hat{s}_0, \hat{s}_1, \ldots, \hat{s}_{w_2}\right) \ .$$

We deduce that CT* is a correctly distributed semi-functional ciphertext of m_b. Hence, $\mathcal{B}$ perfectly simulates G_0 for $\mathcal{A}$.

Now we are ready to show how $\mathcal{B}$ can break the challenge if the BadQuery event occurs.

Algorithm 20: Guess Phase of Algorithm 19

Guess
- $\mathcal{A}$ outputs a bit b'. $\mathcal{B}$ ignores this bit and performs the following computation:
- **foreach** *decryption query on* $\text{CT} = \left(\text{cInd}, C_0, \overrightarrow{\mathbf{C}}, \mathbf{C}''\right)$ *and* $i \in \mathbb{N}$ *such that* CT *pass the consistency checks* **do**
  - **if** $\mathbf{C}'' \neq {}^{t\cdot\boldsymbol{U}+\boldsymbol{V}}\mathbf{C}_1$ **then** Compute $\mathbf{X} := \mathbf{C}'' \cdot \left({}^{t\cdot\boldsymbol{U}+\boldsymbol{V}}\mathbf{C}_1\right)^{-1}$; **go to** Line 37 ;
  - **foreach** $\tau \in \Gamma$ **do**
    - Compute $\mathbf{X}' := \prod_{i\in I} {}^{b_{\tau,i}\cdot\boldsymbol{I}_{d+1}+\sum_{j\in[n]} b_{\tau,i,j}\cdot\boldsymbol{H}_j}\mathbf{C}_{\tau_i}$
    - **if** $\mathbf{C}_\tau \neq \mathbf{X}'$ **then** Compute $\mathbf{X} := \mathbf{C}_\tau \cdot (\mathbf{X}')^{-1}$; **go to** Line 37; ;
  - **end**
  - **foreach** $\tau_k \in \overline{\Gamma}$ **do** Compute $\mathbf{Z}_k := \mathbf{C}_{\tau_k} \cdot \left(\prod_{i\in I} {}^{-b_{\tau_k,i}\cdot\boldsymbol{I}_{d+1}-\sum_{j\in[n]} b_{\tau_k,i,j}\cdot\boldsymbol{H}_j}\mathbf{C}_{\tau_i}\right)$;
  - **foreach** $\tau \in [m_1]$ **do** Compute $\overline{\mathbf{Z}}_\tau := \prod_{\tau_k\in\overline{\Gamma}} \mathbf{Z}_k^{e_{\tau,\tau_k}}$, where $\boldsymbol{E} = \left(e_{i,j}\right)_{m_1,w_1}$ is the corresponding reconstruction matrix used for the decryption of CT ;
  - Compute $\mathbf{X} := \prod_{\tau\in[m_1]} \overline{\mathbf{Z}}_\tau^{a_\tau}$;
  - **if** $\mathbf{X} \neq \mathbf{1}$ **then go to** Line 37;
  - **foreach** $i \in [m_2]$ **do**
    - Compute $\mathbf{X} := \prod_{\tau\in[m_1]} \overline{\mathbf{Z}}_\tau^{a_{\tau,i}}$;
    - **if** $\mathbf{X} \neq \mathbf{1}$ **then go to** Line 37;
    - **foreach** $j \in [n]$ **do**
      - Compute $\mathbf{X} := \prod_{\tau\in[m_1]} \overline{\mathbf{Z}}_\tau^{a_{\tau,i,j}}$;
      - **if** $\mathbf{X} \neq \mathbf{1}$ **then go to** Line 37;
    - **end**
  - **end**
- **end**
- Output 0;

Break
- Compute $\mathbf{Q} := {}^{\tilde{\boldsymbol{B}}^{-\top}}\mathbf{Z}$;
- Output 1 if $\text{e}(\mathbf{X}, \mathbf{Q}) = 1_{\mathbb{G}_\text{T}}$ and output 0 otherwise;

In order to analyze the success probability of $\mathcal{B}$ let us first consider the value $\mathbf{Q} = {}^{\tilde{\boldsymbol{B}}^{-\top}}\mathbf{Z}$ computed by $\mathcal{B}$ in Line 38 and prove the following claim.

Claim. Let $\mathbf{Z} = g_2^{\boldsymbol{T}\cdot\binom{\boldsymbol{y}}{\hat{y}}}$ be as defined in Assumption 6.1, $\tilde{\boldsymbol{B}}$, $\mathbf{G}_2$, and $\widehat{\mathbf{G}}_2$ be as defined in the computation of $\mathcal{B}$. Then

$$\tilde{\boldsymbol{B}}^{-\top}\mathbf{Z} \;=\; \mathbf{G}_2^{\boldsymbol{r}} \cdot \widehat{\mathbf{G}}_2^{r} \;,$$

where $r = -\hat{y}$. In particular, $r \neq 0 \pmod p$ if and only if $\hat{y} \neq 0 \pmod p$.

Proof. Notice that $\boldsymbol{r} \in \mathbb{Z}_p^d$, $r \in \mathbb{Z}_p$ which satisfy the equation in the claim exist and are unique by Lemma 6.1, since $(\,\mathbf{G}_2\ \widehat{\mathbf{G}}_2\,) \overset{\text{by def.}}{=} g_2^{\boldsymbol{Z}}$ for $\boldsymbol{Z} \in \mathbb{GL}_{p,d+1}$ and $\tilde{\boldsymbol{B}}^{-\top}\mathbf{Z} \in \mathbb{G}_2^{d+1}$. As shown above, $\boldsymbol{Z} \in \mathbb{Z}_p^{(d+1)\times(d+1)}$ from the public parameters is equal $\boldsymbol{Z} = \tilde{\boldsymbol{B}}^{-\top} \cdot \boldsymbol{T} \cdot \begin{pmatrix} \boldsymbol{J} & \mathbf{0} \\ \mathbf{0} & -1 \end{pmatrix}$. Hence, $\boldsymbol{T}$ can be written as follows

$$\boldsymbol{T} = \tilde{\boldsymbol{B}}^{\top} \cdot \boldsymbol{Z} \cdot \begin{pmatrix} \boldsymbol{J}^{-1} & \mathbf{0} \\ \mathbf{0} & -1 \end{pmatrix} \;. \tag{7.4}$$

We deduce

$$\begin{aligned}
\tilde{\boldsymbol{B}}^{-\top}\mathbf{Z} \;&=\; \tilde{\boldsymbol{B}}^{-\top}\begin{pmatrix} g_2^{\boldsymbol{T}\cdot\binom{\boldsymbol{y}}{\hat{y}}} \end{pmatrix} \\
&\overset{(7.4)}{=}\; g_2^{\tilde{\boldsymbol{B}}^{-\top}\cdot\tilde{\boldsymbol{B}}^{\top}\cdot\boldsymbol{Z}\cdot\left(\begin{smallmatrix}\boldsymbol{J}^{-1} & \mathbf{0}\\ \mathbf{0} & -1\end{smallmatrix}\right)\cdot\binom{\boldsymbol{y}}{\hat{y}}} \\
&=\; \begin{pmatrix} \mathbf{G}_2 & \widehat{\mathbf{G}}_2 \end{pmatrix}^{\binom{\boldsymbol{J}^{-1}\cdot\boldsymbol{y}}{-\hat{y}}} \\
&=\; \mathbf{G}_2^{\boldsymbol{J}^{-1}\cdot\boldsymbol{y}} \cdot \widehat{\mathbf{G}}_2^{-\hat{y}} \;.
\end{aligned}$$

This proves the claim. □

By this claim we deduce that given $\widehat{\mathbf{G}}_1^{\hat{u}}$ for some $\hat{u} \neq 0 \pmod p$, algorithm $\mathcal{B}$ is able to break the own challenge by the following simple test:

$$\begin{aligned}
\mathrm{e}\left(\widehat{\mathbf{G}}_1^{\hat{u}}, \tilde{\boldsymbol{B}}^{-\top}\mathbf{Z}\right) \;&=\; \mathrm{e}\left(\widehat{\mathbf{G}}_1^{\hat{u}}, \mathbf{G}_2^{\boldsymbol{r}} \cdot \widehat{\mathbf{G}}_2^{r}\right) \\
&=\; \mathrm{e}\left(\widehat{\mathbf{G}}_1^{\hat{u}}, \mathbf{G}_2^{\boldsymbol{r}}\right) \cdot \mathrm{e}\left(\widehat{\mathbf{G}}_1^{\hat{u}}, \widehat{\mathbf{G}}_2^{-\hat{y}}\right) \\
&\overset{\text{Lemma 6.12}}{=}\; 1_{\mathbb{G}_\mathrm{T}} \cdot \mathrm{e}\left(g_1, g_2\right)^{-\hat{y}\cdot\hat{u}} \\
&\overset{?}{=}\; 1_{\mathbb{G}_\mathrm{T}} \;.
\end{aligned}$$

Namely, since $\hat{u} \neq 0 \pmod p$ it holds $\mathrm{e}\left(\widehat{\mathbf{G}}_1^{\hat{u}}, \tilde{\boldsymbol{B}}^{-\top}\mathbf{Z}\right) = 1_{\mathbb{G}_\mathrm{T}}$ if and only if $\hat{y} = 0 \pmod p$. Algorithm $\mathcal{B}$ uses this fact. Namely, we claim that $\mathcal{B}$ goes into Step 37 if and only if event BadQuery occurs and then, it holds $\mathbf{X} = \widehat{\mathbf{G}}_1^{\hat{u}}$ for some $\hat{u} \neq 0 \pmod p$. This allow us to deduce the lemma. We prove this claim next step by step.

Let $\mathrm{CT} = \left(\mathrm{cInd}, C_0, \overrightarrow{\mathbf{C}}, \mathbf{C}''\right)$ be a ciphertext, which pass the consistency checks and $t = \mathrm{H}\left(\mathrm{cInd}, C_0, \overrightarrow{\mathbf{C}}\right)$ be the corresponding hash value.

1. If the verification check in (6.27) for $\mathbf{C}''$ is satisfied, than by Lemma 6.24 it holds

$$\exists_{\xi\in\mathbb{Z}_p} : \mathbf{C}'' = {}^{t\cdot\boldsymbol{U}+\boldsymbol{V}}\mathbf{C}_1 \cdot \widehat{\mathbf{G}}_1^{\xi} \;.$$

If and only if the first condition $\mathbf{C}'' \neq {}^{t\cdot\boldsymbol{U}+\boldsymbol{V}}\mathbf{C}_1$ defined for the BadQuery event is satisfied (checked by $\mathcal{B}$ in Line 18), it holds $\xi \neq 0 \pmod p$, $\mathcal{B}$ computes

$$\begin{aligned}\mathbf{X} &= \mathbf{C}'' \cdot \left({}^{t\cdot\boldsymbol{U}+\boldsymbol{V}}\mathbf{C}_1\right)^{-1} \\ &= {}^{t\cdot\boldsymbol{U}+\boldsymbol{V}}\mathbf{C}_1 \cdot \widehat{\mathbf{G}}_1^{\xi} \cdot {}^{-t\cdot\boldsymbol{U}-\boldsymbol{V}}\mathbf{C}_1 \\ &= \widehat{\mathbf{G}}_1^{\xi} \ ,\end{aligned}$$

and solves the own challenge as explained above.

2. If all verification checks in (6.28) are satisfied, then by Lemma 6.25 it holds:

$$\begin{aligned}\forall_{\tau\in\Gamma}\exists_{\xi_\tau\in\mathbb{Z}_p} : \mathbf{C}_\tau &= \prod_{i\in I}(\mathbf{C}_{\tau_i})^{b_{\tau,i}} \cdot \prod_{j\in[n]}\prod_{i\in I}{}^{\boldsymbol{H}_j}(\mathbf{C}_{\tau_i})^{b_{\tau,i,j}} \cdot \widehat{\mathbf{G}}_1^{\xi_\tau} \\ &= \prod_{i\in I}{}^{b_{\tau,i}\cdot\boldsymbol{I}_{d+1}+\sum_{j\in[n]} b_{\tau,i,j}\cdot\boldsymbol{H}_j}\mathbf{C}_{\tau_i} \cdot \widehat{\mathbf{G}}_1^{\xi_\tau} \ .\end{aligned}$$

If and only if the second condition $\exists_{\tau\in\Gamma} : \mathbf{C}_\tau \neq \prod_{i\in I}{}^{b_{\tau,i}\cdot\boldsymbol{I}_{d+1}+\sum_{j\in[n]} b_{\tau,i,j}\cdot\boldsymbol{H}_j}\mathbf{C}_{\tau_i}$ defined for BadQuery is satisfied (checked by $\mathcal{B}$ in Line 21), it holds $\xi_\tau \neq 0 \pmod p$, $\mathcal{B}$ computes

$$\begin{aligned}\mathbf{X} &= \mathbf{C}_\tau \cdot \left(\prod_{i\in I}(\mathbf{C}_{\tau_i})^{b_{\tau,i}} \cdot \prod_{j\in[n]}\prod_{i\in I}{}^{\boldsymbol{H}_j}(\mathbf{C}_{\tau_i})^{b_{\tau,i,j}}\right)^{-1} \\ &= \widehat{\mathbf{G}}_1^{\xi_\tau}\end{aligned}$$

and solves the own challenge as explained above.

3. If the check in (6.29) is satisfied, than by Lemma 6.26 it holds:

$$\forall_{i\in\overline{I}}\exists_{\boldsymbol{s}_i\in\mathbb{Z}_p^d}\forall_{\tau\in\overline{\Gamma}}\exists_{\xi_\tau\in\mathbb{Z}_p} : \mathbf{C}_\tau = \widehat{\mathbf{G}}_1^{\xi_\tau}\cdot\prod_{i\in\overline{I}}(\mathbf{G}_1^{\boldsymbol{s}_i})^{b_{\tau,i}}\cdot\prod_{i\in I}(\mathbf{C}_{\tau_i})^{b_{\tau,i}}\cdot\prod_{j\in[n]}\prod_{i\in I}{}^{\boldsymbol{H}_j}(\mathbf{C}_{\tau_i})^{b_{\tau,i,j}} \ . \quad (7.5)$$

This last case will be analyzed next in detail.

Consider the computation of $\mathcal{B}$ beginning with Line 23. For every $\tau_k \in \overline{\Gamma}$ the algorithm $\mathcal{B}$ computes first of all

$$\begin{aligned}\mathbf{Z}_k &\overset{\text{by const.}}{=} \mathbf{C}_{\tau_k}\cdot\left(\prod_{i\in I}{}^{-b_{\tau_k,i}\cdot\boldsymbol{I}_{d+1}-\sum_{j\in[n]} b_{\tau_k,i,j}\cdot\boldsymbol{H}_j}\mathbf{C}_{\tau_i}\right) \\ &= \mathbf{C}_{\tau_k}\cdot\left(\prod_{i\in I}(\mathbf{C}_{\tau_i})^{b_{\tau_k,i}}\cdot\prod_{j\in[n]}\prod_{i\in I}{}^{\boldsymbol{H}_j}(\mathbf{C}_{\tau_i})^{b_{\tau_k,i,j}}\right)^{-1} \\ &\overset{(7.5)}{=} \widehat{\mathbf{G}}_1^{\xi_{\tau_k}}\cdot\prod_{i\in\overline{I}}(\mathbf{G}_1^{\boldsymbol{s}_i})^{b_{\tau_k,i}} \\ &= \mathbf{G}_1^{\sum_{i\in\overline{I}} b_{\tau_k,i}\cdot\boldsymbol{s}_i}\cdot\widehat{\mathbf{G}}_1^{\xi_{\tau_k}} \ . \quad (7.6)\end{aligned}$$

Next, $\mathcal{B}$ computes for all $\tau \in [m_1]$

$$\begin{aligned}\overline{\mathbf{Z}}_\tau &\overset{\text{by const.}}{=} \prod_{\tau_k\in\overline{\Gamma}}\mathbf{Z}_k^{e_{\tau,\tau_k}} \\ &\overset{(7.6)}{=} \prod_{\tau_k\in\overline{\Gamma}}\left(\mathbf{G}_1^{\sum_{i\in\overline{I}} b_{\tau_k,i}\cdot\boldsymbol{s}_i}\cdot\widehat{\mathbf{G}}_1^{\xi_{\tau_k}}\right)^{e_{\tau,\tau_k}} \\ &= \mathbf{G}_1^{\sum_{i\in\overline{I}}\boldsymbol{s}_i\cdot\sum_{\tau_k\in\overline{\Gamma}} b_{\tau_k,i}\cdot e_{\tau,\tau_k}}\cdot\widehat{\mathbf{G}}_1^{\sum_{\tau_k\in\overline{\Gamma}} e_{\tau,\tau_k}\cdot\xi_{\tau_k}} \ . \quad (7.7)\end{aligned}$$

Notice that $\mathbf{Z}_k$ and $\overline{\mathbf{Z}}_\tau$ are exactly as defined in the last condition of BadQuery event. Next, we show that the three following checks of $\mathcal{B}$ correspond to the three properties in the last condition of BadQuery event.

1. Recall that if $\mathrm{R}_\kappa(\mathrm{kInd}, \mathrm{cInd}) = 1$ then, by (1.7) from Lemma 1.15 it holds:

$$\forall_{i' \in [w_2]} : \sum_{\tau' \in [w_1]} \sum_{\tau \in [m_1]} e_{\tau,\tau'} \cdot a_\tau \cdot b_{\tau',i'} = 0 \ .$$

Let us consider the value $\prod_{\tau \in [m_1]} \overline{\mathbf{Z}}_\tau^{a_\tau}$ used by $\mathcal{B}$ in Line 26:

$$\begin{aligned} \prod_{\tau \in [m_1]} \overline{\mathbf{Z}}_\tau^{a_\tau} &\overset{(7.7)}{=} \prod_{\tau \in [m_1]} \mathbf{G}_1^{a_\tau \cdot \sum_{i \in \overline{I}} \boldsymbol{s}_i \cdot \sum_{\tau_k \in \overline{\Gamma}} b_{\tau_k,i} \cdot e_{\tau,\tau_k}} \cdot \widehat{\mathbf{G}}_1^{a_\tau \cdot \sum_{\tau_k \in \overline{\Gamma}} e_{\tau,\tau_k} \cdot \xi_{\tau_k}} \\ &= \mathbf{G}_1^{\sum_{i \in \overline{I}} \boldsymbol{s}_i \cdot \sum_{\tau \in [m_1]} \sum_{\tau_k \in \overline{\Gamma}} e_{\tau,\tau_k} \cdot a_\tau \cdot b_{\tau_k,i}} \cdot \widehat{\mathbf{G}}_1^{\sum_{\tau \in [m_1]} a_\tau \cdot \sum_{\tau_k \in \overline{\Gamma}} e_{\tau,\tau_k} \cdot \xi_{\tau_k}} \\ &\overset{(7.9)}{=} \widehat{\mathbf{G}}_1^{\sum_{\tau \in [m_1]} a_\tau \cdot \sum_{\tau_k \in \overline{\Gamma}} e_{\tau,\tau_k} \cdot \xi_{\tau_k}} \ . \end{aligned} \tag{7.8}$$

where in the last step we used

$$\begin{aligned} \sum_{i \in \overline{I}} \boldsymbol{s}_i \cdot \sum_{\tau \in [m_1]} \sum_{\tau_k \in \overline{\Gamma}} e_{\tau,\tau_k} \cdot a_\tau \cdot b_{\tau_k,i} &\overset{\text{Lemma 1.14}}{=} \sum_{i \in \overline{I}} \boldsymbol{s}_i \cdot \sum_{\tau \in [m_1]} \sum_{\tau' \in [w_1]} e_{\tau,\tau'} \cdot a_\tau \cdot b_{\tau',i} \\ &\overset{\text{Lemma 1.15,(1.7)}}{=} \sum_{i \in \overline{I}} \boldsymbol{s}_i \cdot 0 \\ &= \mathbf{0} \ . \end{aligned} \tag{7.9}$$

Hence, if the first property $\prod_{\tau \in [m_1]} \overline{\mathbf{Z}}_\tau^{a_\tau} \neq \mathbf{1}_{\mathbb{G}_1^{d+1}}$ of the third condition of BadQuery is satisfied (checked by $\mathcal{B}$ in Line 26), then $\hat{u} = \sum_{\tau \in [m_1]} a_\tau \cdot \sum_{\tau_k \in \overline{\Gamma}} e_{\tau,\tau_k} \cdot \xi_{\tau_k} \neq 0 \pmod p$, $\mathcal{B}$ computes

$$\mathbf{X} = \widehat{\mathbf{G}}_1^{\sum_{\tau \in [m_1]} a_\tau \cdot \sum_{\tau_k \in \overline{\Gamma}} e_{\tau,\tau_k} \cdot \xi_{\tau_k}} = \widehat{\mathbf{G}}_1^{\hat{u}} \ ,$$

and solves the own challenge as explained above.

2. Recall that if $\mathrm{R}_\kappa(\mathrm{kInd}, \mathrm{cInd}) = 1$ then, by (1.9) from Lemma 1.15 it holds:

$$\forall_{i \in [m_2]} \forall_{i' \in [w_2]_0} : \sum_{\tau \in [m_1]} \sum_{\tau' \in [w_1]} e_{\tau,\tau'} \cdot a_{\tau,i} \cdot b_{\tau',i'} = 0 \ .$$

Let $\hat{i} \in [m_2]$ be arbitrary. Let us consider the value $\prod_{\tau \in [m_1]} \overline{\mathbf{Z}}_\tau^{a_{\tau,\hat{i}}}$ used by $\mathcal{B}$ in Line 28:

$$\begin{aligned} \prod_{\tau \in [m_1]} \overline{\mathbf{Z}}_\tau^{a_{\tau,\hat{i}}} &\overset{(7.7)}{=} \prod_{\tau \in [m_1]} \left(\mathbf{G}_1^{\sum_{i \in \overline{I}} \boldsymbol{s}_i \cdot \sum_{\tau_k \in \overline{\Gamma}} b_{\tau_k,i} \cdot e_{\tau,\tau_k}} \cdot \widehat{\mathbf{G}}_1^{\sum_{\tau_k \in \overline{\Gamma}} e_{\tau,\tau_k} \cdot \xi_{\tau_k}} \right)^{a_{\tau,\hat{i}}} \\ &= \mathbf{G}_1^{\sum_{i \in \overline{I}} \boldsymbol{s}_i \cdot \sum_{\tau \in [m_1]} \sum_{\tau_k \in \overline{\Gamma}} e_{\tau,\tau_k} \cdot a_{\tau,\hat{i}} \cdot b_{\tau_k,i}} \cdot \widehat{\mathbf{G}}_1^{\sum_{\tau \in [m_1]} a_{\tau,\hat{i}} \cdot \sum_{\tau_k \in \overline{\Gamma}} e_{\tau,\tau_k} \cdot \xi_{\tau_k}} \\ &\overset{(7.11)}{=} \widehat{\mathbf{G}}_1^{\sum_{\tau \in [m_1]} a_{\tau,\hat{i}} \cdot \sum_{\tau_k \in \overline{\Gamma}} e_{\tau,\tau_k} \cdot \xi_{\tau_k}} \end{aligned} \tag{7.10}$$

where in the last step we used

$$\begin{aligned} \sum_{i \in \overline{I}} \boldsymbol{s}_i \cdot \sum_{\tau \in [m_1]} \sum_{\tau_k \in \overline{\Gamma}} e_{\tau,\tau_k} \cdot a_{\tau,\hat{i}} \cdot b_{\tau_k,i} &\overset{\text{Lemma 1.14}}{=} \sum_{i \in \overline{I}} \boldsymbol{s}_i \cdot \sum_{\tau \in [m_1]} \sum_{\tau' \in [w_1]} e_{\tau,\tau'} \cdot a_{\tau,\hat{i}} \cdot b_{\tau',i} \\ &\overset{\text{Lemma 1.15,(1.9)}}{=} \sum_{i \in \overline{I}} \boldsymbol{s}_i \cdot 0 \\ &= \mathbf{0} \end{aligned} \tag{7.11}$$

Hence, if the second property $\exists_{i\in[m_2]} : \prod_{\tau\in[m_1]} \overline{\mathbf{Z}}_\tau^{a_{\tau,i}} \neq \mathbf{1}_{\mathbb{G}_1^{d+1}}$ of the third condition of BadQuery is satisfied (checked by $\mathcal{B}$ in Line 28), then $\hat{u} = \sum_{\tau\in[m_1]} a_{\tau,\hat{i}} \cdot \sum_{\tau_k\in\overline{\Gamma}} e_{\tau,\tau_k} \cdot \xi_{\tau_k} \neq 0 \pmod p$, $\mathcal{B}$ computes

$$\mathbf{X} = \widehat{\mathbf{G}}_1^{\sum_{\tau\in[m_1]} a_{\tau,\hat{i}} \cdot \sum_{\tau_k\in\overline{\Gamma}} e_{\tau,\tau_k}\cdot\xi_{\tau_k}} = \widehat{\mathbf{G}}_1^{\hat{u}} \ ,$$

where $\hat{u} \neq 0 \pmod p$ and solves the own challenge as explained above.

3. Recall that if $\mathrm{R}_\kappa(\mathrm{kInd}, \mathrm{cInd}) = 1$ then, by (1.10) from Lemma 1.15 it holds:

$$\forall_{i\in[m_2]}\forall_{i'\in[w_2]_0}\forall_{j\in[n]} : \sum_{\tau\in[m_1]}\sum_{\tau'\in[w_1]} e_{\tau,\tau'} \cdot \left(a_{\tau,i,j}\cdot b_{\tau',i'} + a_{\tau,i}\cdot b_{\tau',i',j}\right) = 0 \ .$$

Assume $\hat{i} \in [m_2]$, and $\hat{j} \in [n]$ be arbitrary. Let us consider the value $\prod_{\tau\in[m_1]} \overline{\mathbf{Z}}_\tau^{a_{\tau,\hat{i},\hat{j}}}$ used by $\mathcal{B}$ in Line 31:

$$\begin{aligned}
\prod_{\tau\in[m_1]} \overline{\mathbf{Z}}_\tau^{a_{\tau,\hat{i},\hat{j}}} &\overset{(7.7)}{=} \prod_{\tau\in[m_1]} \left(\mathbf{G}_1^{\sum_{i\in\overline{I}} \boldsymbol{s}_i \cdot \sum_{\tau_k\in\overline{\Gamma}} b_{\tau_k,i}\cdot e_{\tau,\tau_k}} \cdot \widehat{\mathbf{G}}_1^{\sum_{\tau_k\in\overline{\Gamma}} e_{\tau,\tau_k}\cdot\xi_{\tau_k}}\right)^{a_{\tau,\hat{i},\hat{j}}} \\
&= \mathbf{G}_1^{\sum_{i\in\overline{I}} \boldsymbol{s}_i \cdot \sum_{\tau\in[m_1]}\sum_{\tau_k\in\overline{\Gamma}} e_{\tau,\tau_k}\cdot a_{\tau,\hat{i},\hat{j}}\cdot b_{\tau_k,i}} \cdot \widehat{\mathbf{G}}_1^{\sum_{\tau\in[m_1]} a_{\tau,\hat{i},\hat{j}}\cdot\sum_{\tau_k\in\overline{\Gamma}} e_{\tau,\tau_k}\cdot\xi_{\tau_k}} \\
&\overset{(7.13)}{=} \widehat{\mathbf{G}}_1^{\sum_{\tau\in[m_1]} a_{\tau,\hat{i},\hat{j}}\cdot\sum_{\tau_k\in\overline{\Gamma}} e_{\tau,\tau_k}\cdot\xi_{\tau_k}} \ , \qquad (7.12)
\end{aligned}$$

where in the last step we used

$$\begin{aligned}
&\sum_{i\in\overline{I}} \boldsymbol{s}_i \cdot \sum_{\tau\in[m_1]}\sum_{\tau_k\in\overline{\Gamma}} e_{\tau,\tau_k}\cdot a_{\tau,\hat{i},\hat{j}}\cdot b_{\tau_k,i} \\
\overset{\text{Lemma 1.14}}{=}\ &\sum_{i\in\overline{I}} \boldsymbol{s}_i \cdot \sum_{\tau\in[m_1]}\sum_{\tau'\in[w_1]} e_{\tau,\tau'}\cdot a_{\tau,\hat{i},\hat{j}}\cdot b_{\tau',i} \\
\overset{\text{Lemma 1.14}}{=}\ &\sum_{i\in\overline{I}} \boldsymbol{s}_i \cdot \left(\sum_{\tau\in[m_1]}\sum_{\tau'\in[w_1]} e_{\tau,\tau'}\cdot\left(a_{\tau,\hat{i},\hat{j}}\cdot b_{\tau',i} + a_{\tau,\hat{i}}\cdot b_{\tau',i,\hat{j}}\right)\right) \\
\overset{\text{Lemma 1.15,(1.10)}}{=}\ &\sum_{i\in\overline{I}} \boldsymbol{s}_i \cdot 0 \\
=\ &\mathbf{0} \qquad (7.13)
\end{aligned}$$

Hence, if and only if the last property $\exists_{i\in[m_2]}\exists_{j\in[n]} : \prod_{\tau\in[m_1]} \overline{\mathbf{Z}}_\tau^{a_{\tau,i,j}} \neq \mathbf{1}_{\mathbb{G}_1^{d+1}}$ of the third condition of BadQuery is satisfied (checked by $\mathcal{B}$ in Line 31), then $\hat{u} = \sum_{\tau\in[m_1]} a_{\tau,\hat{i},\hat{j}} \cdot \sum_{\tau_k\in\overline{\Gamma}} e_{\tau,\tau_k}\cdot\xi_{\tau_k} \neq 0 \pmod p$, $\mathcal{B}$ computes

$$\mathbf{X} = \widehat{\mathbf{G}}_1^{\sum_{\tau\in[m_1]} a_{\tau,\hat{i},\hat{j}}\cdot\sum_{\tau_k\in\overline{\Gamma}} e_{\tau,\tau_k}\cdot\xi_{\tau_k}} = \widehat{\mathbf{G}}_1^{\hat{u}} \ ,$$

where $\hat{u} \neq 0 \pmod p$ and solves the own challenge as explained above.

All together we deduce that if the event BadQuery occurs, $\mathcal{B}$ outputs 1 if and only if $\hat{y} = 0 \pmod p$. If BadQuery does not occur, $\mathcal{B}$ outputs 0 by construction. The advantage of $\mathcal{B}$ is as follows:

$$\begin{aligned}
\mathrm{Adv}_{\mathcal{B}}^{\mathcal{D}_d-\mathrm{mDH}}(\lambda) &\overset{\text{by def.}}{=} \left|\Pr\left[\mathcal{B}\left(\mathbb{GD}, g_1^{\boldsymbol{A}}, g_1^{\boldsymbol{A}\cdot\binom{\boldsymbol{y}}{0}}\right) = 1\right] - \Pr\left[\mathcal{B}\left(\mathbb{GD}, g_1^{\boldsymbol{A}}, g_1^{\boldsymbol{A}\cdot\binom{\boldsymbol{y}}{\hat{y}}}\right) = 1\right]\right| \\
&= \left|\Pr\left[\mathcal{B}\text{ outputs }1 \mid \hat{y} = 0\right] - \Pr\left[\mathcal{B}\text{ outputs }1 \;\middle|\; \hat{y} \in \mathbb{Z}_p^*\right]\right| \\
&\overset{\text{by const.}}{=} \Pr\left[\mathrm{BadQuery} \mid \hat{y} = 0\right] - 0 \\
&= \Pr\left[\mathrm{BadQuery}\right] \ .
\end{aligned}$$

Thereby the last equation holds since the view of $\mathcal{A}$ is as in the experiment G_0 independently of $\mathcal{B}$'s challenge $\mathbf{Z}$, since $\mathcal{B}$ uses $\mathbf{Z}$ only in the guess phase. This finally proves the lemma. □

Third Reduction $\mathrm{G}_0 \to \mathrm{G}_0'$

G_0' is exactly the same as G_0 except for decryption queries which are answered using separately generated normal secret keys. Also here we can view the experiment G_0 and G_0' as defined over the same probability space. G_0' additionally requires mutually independent random variables for the separately generated normal secret keys. The corresponding random variables are ignored in G_0.

Lemma 7.9. *For every ppt* $\mathcal{A} \in \mathbf{A}_\Pi$ *and every* $\mathrm{des} \in \Omega$ *there exists a ppt* $\mathcal{B}$ *such that it holds*

$$\left|\mathrm{Adv}^{\mathrm{G}_0}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) - \mathrm{Adv}^{\mathrm{G}_0'}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})\right| \leq \Pr[\mathrm{BadQuery}] = \mathrm{Adv}^{\mathcal{D}_d\text{-mDH}}_{\mathcal{B}}(\lambda) \quad .$$

Furthermore, if the event BadQuery *does not occur, the output of the decryption algorithm on every* CT *which passes the consistency checks is* $m = C_0 \cdot \mathrm{e}(\mathbf{C}_1, \mathrm{msk})^{-1}$ *in both experiments.*

Proof. By Lemma 7.7 the probability for BadQuery in G_0 is negligible. We prove that both experiments identically proceed as long as the BadQuery event does not occur. Using Lemma 7.7 we then finalize the proof.

Let $\mathrm{CT} = \left(\mathrm{cInd}, C_0, \overrightarrow{\mathbf{C}}, \mathbf{C}''\right)$ be a ciphertext submitted by $\mathcal{A}$, which pass the consistency checks, $t = \mathrm{H}\left(\mathrm{cInd}, C_0, \overrightarrow{\mathbf{C}}\right)$ be the corresponding hash value and $\boldsymbol{E}$ be the reconstruction matrix computed for the corresponding decryption query. Furthermore, let sk and sk' be normal secret keys computed for this query in G_0 and in G_0', respectively. We prove that $\mathrm{Dec}(\mathrm{sk}, \mathrm{CT}; \boldsymbol{E}) \neq \mathrm{Dec}\left(\mathrm{sk}', \mathrm{CT}; \boldsymbol{E}\right)$ implies that the BadQuery event is caused by CT. We prove it by contradiction.

Conditioned on the assumption that CT does not cause the event BadQuery the elements in CT are as follows:

1. By the first condition defined for BadQuery it holds

$$\mathbf{C}'' = {}^{t\cdot\boldsymbol{U}+\boldsymbol{V}}\mathbf{C}_1 \quad . \tag{7.14}$$

   We conclude that $\mathbf{C}''$ is formed as by a correctly generated semi-functional ciphertext by Lemma 6.21.

2. By the second condition defined for BadQuery it holds

$$\forall_{\tau\in\Gamma} : \mathbf{C}_\tau = \prod_{i\in I}(\mathbf{C}_{\tau_i})^{b_{\tau,i}} \cdot \prod_{j\in[n]}\prod_{i\in I}{}^{\boldsymbol{H}_j}(\mathbf{C}_{\tau_i})^{b_{\tau,i,j}} \quad . \tag{7.15}$$

   Hence, for every $\tau \in \Gamma$ the element $\mathbf{C}_\tau$ is formed as by a correctly generated semi-functional ciphertext by Lemma 6.21.

For the remaining elements $\left\{\mathbf{C}_\tau \in \overrightarrow{\mathbf{C}}\right\}_{\tau\in\overline{\Gamma}}$ we cannot deduce that they are formed as by a correctly generated semi-functional ciphertext. However, due to the verification checks and due to the last condition defined for BadQuery, we are able to prove that the answer of decryption algorithm will be the same independently of the concrete user secret key for corresponding key index kInd.

If the verification checks in (6.29) are satisfied, then by Lemma 6.26 for all $i \in \overline{I}$ there exists $\boldsymbol{s}_i \in \mathbb{Z}_p^d$ such that it holds:

$$\forall_{\tau\in\overline{\Gamma}}\exists_{\xi_\tau\in\mathbb{Z}_p} : \mathbf{C}_\tau = \widehat{\mathbf{G}}_1^{\xi_\tau} \cdot \prod_{i\in\overline{I}}(\mathbf{G}_1^{\boldsymbol{s}_i})^{b_{\tau,i}} \cdot \prod_{i\in I}(\mathbf{C}_{\tau_i})^{b_{\tau,i}} \cdot \prod_{j\in[n]}\prod_{i\in I}{}^{\boldsymbol{H}_j}(\mathbf{C}_{\tau_i})^{b_{\tau,i,j}} \quad .$$

Consider the case that for every $i \in \overline{I}$ there exists $\hat{s}_i \in \mathbb{Z}_p$ such that for every $\tau \in \overline{\Gamma}$ it holds $\xi_\tau = \sum_{i\in\overline{I}} b_{\tau,i} \cdot \hat{s}_i$. That is,

$$\forall_{\tau\in\overline{\Gamma}} : \mathbf{C}_\tau = \prod_{i\in\overline{I}} \left(\widehat{\mathbf{G}}_1^{\hat{s}_i}\right)^{b_{\tau,i}} \cdot \prod_{i\in\overline{I}} (\mathbf{G}_1^{\boldsymbol{s}_i})^{b_{\tau,i}} \cdot \prod_{i\in I} (\mathbf{C}_{\tau_i})^{b_{\tau,i}} \cdot \prod_{j\in[n]} \prod_{i\in I} {}^{\boldsymbol{H}_j}(\mathbf{C}_{\tau_i})^{b_{\tau,i,j}} \quad . \tag{7.16}$$

Then, for all $\tau \in \overline{\Gamma}$ the element $\mathbf{C}_\tau$ is formed as by a correctly generated semi-functional ciphertext due to Lemma 6.21. Then, from (7.14), (7.15), and (7.16) and Lemma 6.21 we deduce that the ciphertext CT is a correctly formed semi-functional ciphertext and by Lemma 6.29 the output of the decryption algorithm will be the same for every normal secret keys for the corresponding key index kInd. Hence, the answer to the decryption query on CT will be the same in both experiment.

Next, consider the remaining case – that is,

$$\begin{aligned}
\mathbf{C}'' &= {}^{t\cdot\boldsymbol{U}+\boldsymbol{V}}\mathbf{C}_1 \ , \\
\forall_{\tau\in\Gamma} : \mathbf{C}_\tau &= \prod_{i\in I} (\mathbf{C}_{\tau_i})^{b_{\tau,i}} \cdot \prod_{j\in[n]} \prod_{i\in I} {}^{\boldsymbol{H}_j}(\mathbf{C}_{\tau_i})^{b_{\tau,i,j}} \ , \\
\forall_{\tau\in\overline{\Gamma}} \exists_{\xi_\tau\in\mathbb{Z}_p} : \mathbf{C}_\tau &= \widehat{\mathbf{G}}_1^{\xi_\tau} \cdot \prod_{i\in\overline{I}} (\mathbf{G}_1^{\boldsymbol{s}_i})^{b_{\tau,i}} \cdot \prod_{i\in I} (\mathbf{C}_{\tau_i})^{b_{\tau,i}} \cdot \prod_{j\in[n]} \prod_{i\in I} {}^{\boldsymbol{H}_j}(\mathbf{C}_{\tau_i})^{b_{\tau,i,j}} \ ,
\end{aligned}$$

but there do not exist $\{\hat{s}_i \in \mathbb{Z}_p\}_{i\in\overline{I}}$ such that for every $\tau \in \overline{\Gamma}$ it holds $\xi_\tau = \sum_{i\in\overline{I}} b_{\tau,i} \cdot \hat{s}_i$. Once again, we intend to prove that in this case $\mathrm{Dec}\left(\mathrm{sk}, \mathrm{CT}; \boldsymbol{E}\right) \neq \mathrm{Dec}\left(\mathrm{sk}', \mathrm{CT}; \boldsymbol{E}\right)$ implies that the BadQuery event is caused by CT, namely one of the cases in the last condition defined for this event.

Let us consider the computation of Dec on a ciphertext CT as above given a reconstruction matrix $\boldsymbol{E}$. The decryption algorithm computes the value $X \in \mathbb{G}_\mathrm{T}$ in an intermediate step. Let $X' \in \mathbb{G}_\mathrm{T}$ be the value, which would be computed by the decryption algorithm when $\forall_{\tau\in\overline{\Gamma}} : \xi_\tau = 0$. In this case CT is a correctly formed (semi-functional) ciphertext with $\forall_{i\in\overline{I}} : \hat{s}_i = 0$, and hence X' will be the same for all normal keys as argued above. Let $\mathrm{sk} = \left(\mathrm{kInd}, \overrightarrow{\mathbf{K}}\right)$ with $\overrightarrow{\mathbf{K}} = (\mathbf{K}_1, \ldots, \mathbf{K}_{m_1})$ be an arbitrary, but fixed normal secret key for kInd. We deduce that decryption algorithm on input CT and sk computes:

$$\begin{aligned}
X &\overset{\text{by def.}}{=} \prod_{\tau\in[m_1]} \prod_{\tau'\in[w_1]} \mathrm{e}\left(\mathbf{C}_{\tau'}, \mathbf{K}_\tau\right)^{e_{\tau,\tau'}} \\
&= X' \cdot \prod_{\tau\in[m_1]} \prod_{\tau_k\in\overline{\Gamma}} \mathrm{e}\left(\widehat{\mathbf{G}}_1^{\xi_{\tau_k}}, \mathbf{K}_\tau\right)^{e_{\tau,\tau_k}} \quad .
\end{aligned}$$

Hence, if the output of the decryption algorithm is not the same for different keys, then at least for one of the keys in question it holds

$$\prod_{\tau\in[m_1]} \prod_{\tau_k\in\overline{\Gamma}} \mathrm{e}\left(\widehat{\mathbf{G}}_1^{\xi_{\tau_k}}, \mathbf{K}_\tau\right)^{e_{\tau,\tau_k}} \neq 1_{\mathbb{G}_\mathrm{T}} \ .$$

Let us consider this factor. Recall that for a normal user secret key, for every $\tau \in [m_1]$ it holds

$$\begin{aligned}
k_\tau\left(\boldsymbol{\alpha}, \boldsymbol{R}, \overline{\mathbb{H}}\right) &= a_\tau \cdot \boldsymbol{\alpha} + \sum_{i\in[m_2]} \left(a_{\tau,i} \cdot \boldsymbol{Z} \cdot \begin{pmatrix} \boldsymbol{r}_i \\ 0 \end{pmatrix} + \sum_{j\in[n]} a_{\tau,i,j} \cdot \boldsymbol{H}_j^\top \cdot \boldsymbol{Z} \cdot \begin{pmatrix} \boldsymbol{r}_i \\ 0 \end{pmatrix} \right) \\
&= a_\tau \cdot \boldsymbol{\alpha} + \sum_{i\in[m_2]} \left(a_{\tau,i} \cdot \boldsymbol{R}_i + \sum_{j\in[n]} a_{\tau,i,j} \cdot \boldsymbol{H}_j^\top \cdot \boldsymbol{R}_i \right) \ ,
\end{aligned}$$

where $\boldsymbol{R}_i = \left(\begin{smallmatrix} \boldsymbol{r}_i \\ 0 \end{smallmatrix}\right)$. Hence, it holds

$$\begin{aligned} X \cdot \left(X'\right)^{-1} \quad \overset{\text{by const.}}{=} \quad & \prod_{\tau \in [m_1]} \prod_{\tau_k \in \overline{\Gamma}} \mathrm{e}\left(\widehat{\mathbf{G}}_1^{\xi_{\tau_k}}, \mathbf{K}_\tau\right)^{e_{\tau,\tau_k}} \\ = \quad & \prod_{\tau \in [m_1]} \mathrm{e}\left(\widehat{\mathbf{G}}_1, \mathbf{K}_\tau\right)^{\sum_{\tau_k \in \overline{\Gamma}} e_{\tau,\tau_k} \cdot \xi_{\tau_k}} \\ = \quad & \mathrm{e}\left(\widehat{\mathbf{G}}_1, \prod_{\tau \in [m_1]} \mathbf{K}_\tau^{\sum_{\tau_k \in \overline{\Gamma}} e_{\tau,\tau_k} \cdot \xi_{\tau_k}}\right) \\ = \quad & \mathrm{e}\left(\widehat{\mathbf{G}}_1, \prod_{\tau \in [m_1]} \left(g_2^{\boldsymbol{k}_\tau\left(\boldsymbol{\alpha}, \boldsymbol{R}, \overline{\mathbb{H}}\right)}\right)^{\sum_{\tau_k \in \overline{\Gamma}} e_{\tau,\tau_k} \cdot \xi_{\tau_k}}\right) \\ = \quad & \mathrm{e}\left(\widehat{\mathbf{G}}_1, g_2^{\sum_{\tau \in [m_1]} \sum_{\tau_k \in \overline{\Gamma}} \boldsymbol{k}_\tau\left(\boldsymbol{\alpha}, \boldsymbol{R}, \overline{\mathbb{H}}\right) \cdot e_{\tau,\tau_k} \cdot \xi_{\tau_k}}\right) \quad . \end{aligned}$$

We conclude

$$\begin{aligned} X \cdot \left(X'\right)^{-1} \quad &\neq \quad 1_{\mathbb{G}_\mathrm{T}} \Rightarrow \\ \sum_{\tau \in [m_1]} \sum_{\tau_k \in \overline{\Gamma}} \boldsymbol{k}_\tau\left(\boldsymbol{\alpha}, \boldsymbol{R}, \overline{\mathbb{H}}\right) \cdot e_{\tau,\tau_k} \cdot \xi_{\tau_k} \quad &\neq \quad \mathbf{0} \ . \end{aligned}$$

But

$$\begin{aligned} \mathbb{Z}_p^{d+1} \ni & \sum_{\tau \in [m_1]} \sum_{\tau_k \in \overline{\Gamma}} \boldsymbol{k}_\tau\left(\boldsymbol{\alpha}, \boldsymbol{R}, \overline{\mathbb{H}}\right) \cdot e_{\tau,\tau_k} \cdot \xi_{\tau_k} \\ = & \sum_{\tau \in [m_1]} \sum_{\tau_k \in \overline{\Gamma}} \left(a_\tau \cdot \boldsymbol{\alpha} + \sum_{i \in [m_2]} \left(a_{\tau,i} \cdot \boldsymbol{R}_i + \sum_{j \in [n]} a_{\tau,i,j} \cdot \boldsymbol{H}_j^\top \cdot \boldsymbol{R}_i\right)\right) \cdot e_{\tau,\tau_k} \cdot \xi_{\tau_k} \\ = & \ \boldsymbol{\alpha} \cdot \sum_{\tau \in [m_1]} a_\tau \cdot \sum_{\tau_k \in \overline{\Gamma}} e_{\tau,\tau_k} \cdot \xi_{\tau_k} \\ & + \sum_{i \in [m_2]} \boldsymbol{R}_i \cdot \sum_{\tau \in [m_1]} a_{\tau,i} \cdot \sum_{\tau_k \in \overline{\Gamma}} e_{\tau,\tau_k} \cdot \xi_{\tau_k} \\ & + \sum_{i \in [m_2]} \sum_{j \in [n]} \boldsymbol{H}_j^\top \cdot \boldsymbol{R}_i \cdot \sum_{\tau \in [m_1]} a_{\tau,i,j} \cdot \sum_{\tau_k \in \overline{\Gamma}} e_{\tau,\tau_k} \cdot \xi_{\tau_k} \quad . \end{aligned}$$

We deduce that $X \cdot \left(X'\right)^{-1} \neq 1_{\mathbb{G}_\mathrm{T}}$ implies that at least one of the following three cases appears:

$$\begin{aligned} \boldsymbol{\alpha} \cdot \sum_{\tau \in [m_1]} a_\tau \cdot \sum_{\tau_k \in \overline{\Gamma}} e_{\tau,\tau_k} \cdot \xi_{\tau_k} \neq \mathbf{0} \Rightarrow \\ \sum_{\tau \in [m_1]} a_\tau \cdot \sum_{\tau_k \in \overline{\Gamma}} e_{\tau,\tau_k} \cdot \xi_{\tau_k} \neq 0 \ , \end{aligned}$$

or

$$\begin{aligned} \sum_{i \in [m_2]} \boldsymbol{R}_i \cdot \sum_{\tau \in [m_1]} a_{\tau,i} \cdot \sum_{\tau_k \in \overline{\Gamma}} e_{\tau,\tau_k} \cdot \xi_{\tau_k} \quad &\neq \quad \mathbf{0} \Rightarrow \\ \exists_{i \in [m_2]} : \sum_{\tau \in [m_1]} a_{\tau,i} \cdot \sum_{\tau_k \in \overline{\Gamma}} e_{\tau,\tau_k} \cdot \xi_{\tau_k} \quad &\neq \quad 0 \ , \end{aligned}$$

or

$$\sum_{i\in[m_2]}\sum_{j\in[n]} \boldsymbol{H}_j^\top \cdot \boldsymbol{R}_i \cdot \sum_{\tau\in[m_1]} a_{\tau,i,j} \cdot \sum_{\tau_k\in\overline{\Gamma}} e_{\tau,\tau_k}\cdot\xi_{\tau_k} \neq \mathbf{0} \Rightarrow$$
$$\exists_{i\in[m_2]}\exists_{j\in[n]} : \sum_{\tau\in[m_1]} a_{\tau,i,j} \cdot \sum_{\tau_k\in\overline{\Gamma}} e_{\tau,\tau_k}\cdot\xi_{\tau_k} \neq 0 \ .$$

Consider these three cases. We claim that all of them cause event BadQuery, which finalize the proof of the lemma:

1. Assume $\hat{u} := \sum_{\tau\in[m_1]} a_\tau \cdot \sum_{\tau_k\in\overline{\Gamma}} e_{\tau,\tau_k}\cdot\xi_{\tau_k} \neq 0$, then

$$\prod_{\tau\in[m_1]} \overline{\mathbf{Z}}_\tau^{a_\tau} \overset{(7.8)}{=} \widehat{\mathbf{G}}_1^{\sum_{\tau\in[m_1]} a_\tau\cdot\sum_{\tau_k\in\overline{\Gamma}} e_{\tau,\tau_k}\cdot\xi_{\tau_k}}$$
$$= \widehat{\mathbf{G}}_1^{\hat{u}} \neq \mathbf{1}_{\mathbb{G}_1^{d+1}} \ .$$

   This is exactly the first property of the last condition of BadQuery.

2. Assume $\exists_{\hat{i}\in[m_2]}$ with $\hat{u} := \sum_{\tau\in[m_1]} a_{\tau,\hat{i}} \cdot \sum_{\tau_k\in\overline{\Gamma}} e_{\tau,\tau_k}\cdot\xi_{\tau_k} \neq 0$, then

$$\prod_{\tau\in[m_1]} \overline{\mathbf{Z}}_\tau^{a_{\tau,\hat{i}}} \overset{(7.10)}{=} \widehat{\mathbf{G}}_1^{\sum_{\tau\in[m_1]} a_{\tau,\hat{i}}\cdot\sum_{\tau_k\in\overline{\Gamma}} e_{\tau,\tau_k}\cdot\xi_{\tau_k}}$$
$$= \widehat{\mathbf{G}}_1^{\hat{u}} \neq \mathbf{1}_{\mathbb{G}_1^{d+1}} \ .$$

   This is exactly the second property of the last condition of BadQuery.

3. Assume $\exists_{\hat{i}\in[m_2]}\exists_{\hat{j}\in[n]}$ with $\hat{u} := \sum_{\tau\in[m_1]} a_{\tau,\hat{i},\hat{j}} \cdot \sum_{\tau_k\in\overline{\Gamma}} e_{\tau,\tau_k}\cdot\xi_{\tau_k} \neq 0$, then

$$\prod_{\tau\in[m_1]} \overline{\mathbf{Z}}_\tau^{a_{\tau,\hat{i},\hat{j}}} \overset{(7.12)}{=} \widehat{\mathbf{G}}_1^{\sum_{\tau\in[m_1]} a_{\tau,\hat{i},\hat{j}}\cdot\sum_{\tau_k\in\overline{\Gamma}} e_{\tau,\tau_k}\cdot\xi_{\tau_k}}$$
$$= \widehat{\mathbf{G}}_1^{\hat{u}} \neq \mathbf{1}_{\mathbb{G}_1^{d+1}} \ .$$

   This is exactly the third property of the last condition of BadQuery.

We conclude that if BadQuery does not occur, then $X\cdot(X')^{-1} = 1_{\mathbb{G}_T}$. That is, the output of the decryption oracle is $C_0\cdot(X')^{-1} = C_0\cdot(X)^{-1}$ independently of the concrete normal secret key for kInd used in the decryption algorithm.

We deduce that both experiments proceed identically until the event BadQuery occurs. Hence,

$$\mathrm{G}_{0,\Pi,\mathcal{A}}(\lambda,\mathrm{des}) = 1 \wedge \overline{\mathrm{BadQuery}} \Leftrightarrow \mathrm{G}'_{0,\Pi,\mathcal{A}}(\lambda,\mathrm{des}) = 1 \wedge \overline{\mathrm{BadQuery}}$$

and by Difference Lemma 5.8 it holds

$$\left|\Pr\left[\mathrm{G}_{0,\Pi,\mathcal{A}}(\lambda,\mathrm{des})=1\right] - \Pr\left[\mathrm{G}'_{0,\Pi,\mathcal{A}}(\lambda,\mathrm{des})=1\right]\right| \leq \Pr\left[\mathrm{BadQuery}\right] \ .$$

By Lemma 7.7, there exists ppt $\mathcal{B}$ such that

$$\begin{aligned}
&\left|\mathrm{Adv}^{\mathrm{G}_0}_{\Pi,\mathcal{A}}(\lambda,\mathrm{des}) - \mathrm{Adv}^{\mathrm{G}'_0}_{\Pi,\mathcal{A}}(\lambda,\mathrm{des})\right| \\
\overset{\text{by def.}}{=}\ &\left|\Pr\left[\mathrm{G}_{0,\Pi,\mathcal{A}}(\lambda,\mathrm{des})=1\right] - \Pr\left[\mathrm{G}'_{0,\Pi,\mathcal{A}}(\lambda,\mathrm{des})=1\right]\right| \\
\leq\ &\Pr\left[\overline{\mathrm{BadQuery}}\right] \\
=\ &\mathrm{Adv}^{\mathcal{D}_d\text{-mDH}}_{\mathcal{B}}(\lambda) \ .
\end{aligned}$$

We additionally deduce that if the event BadQuery does not occur, the decryption algorithm outputs

$$m := C_0 \cdot X^{-1} \ ,$$

where by definition above X is the value computed for correctly generated semi-functional ciphertext, as explained above. That is $X = \mathbf{Y}^{\boldsymbol{s}_0} \cdot \widehat{\mathbf{Y}}^{\hat{s}_0}$, where $\boldsymbol{s}_0$ and $\hat{s}_0$ are defined through $\mathbf{C}_1 = \mathbf{G}_1^{\boldsymbol{s}_0} \cdot \widehat{\mathbf{G}}_1^{\hat{s}_0}$. Hence,

$$\begin{aligned} m &= C_0 \cdot \mathbf{Y}^{-\boldsymbol{s}_0} \cdot \widehat{\mathbf{Y}}^{-\hat{s}_0} \\ &= C_0 \cdot \mathrm{e}\left(\mathbf{C}_1, \mathrm{msk}\right)^{-1} \ , \end{aligned}$$

since

$$\begin{aligned} \mathrm{e}\left(\mathbf{C}_1, \mathrm{msk}\right) &= \mathrm{e}\left(\mathbf{G}_1^{\boldsymbol{s}_0} \cdot \widehat{\mathbf{G}}_1^{\hat{s}_0}, g_2^{\boldsymbol{\alpha}}\right) \\ &\overset{\text{by def.}}{=} \mathbf{Y}^{\boldsymbol{s}_0} \cdot \widehat{\mathbf{Y}}^{\hat{s}_0} \ . \end{aligned}$$

This finalizes the proof. □

We explicitly notice that $\hat{s}_0 \in I$ is not necessarily 0. Furthermore, we only proved that $\prod_{\tau \in [m_1]} \prod_{\tau_k \in \overline{\Gamma}} \mathrm{e}\left(\widehat{\mathbf{G}}_1^{\xi_{\tau_k}}, \mathbf{K}_\tau\right)^{e_{\tau,\tau_k}} \neq 1_{\mathbb{G}_\mathrm{T}}$ can happen only if the BadQuery event occurs. We did not prove that there exist $\{\hat{s}_i \in \mathbb{Z}_p\}_{i \in \overline{I}}$ such that for every $\tau \in \overline{\Gamma}$ it holds $\xi_\tau = \sum_{i \in \overline{I}} b_{\tau,i} \cdot \hat{s}_i$. That is, we did not prove that the ciphertext CT has the form of an semi-functional ciphertext if it pass the verification checks and does not cause BadQuery event.

From G_0' to G_0''

Experiment G_0'' is the same as G_0' except for decryption queries on $\mathrm{CT} = \left(\mathrm{cInd}, C_0, \overrightarrow{\mathbf{C}}, \mathbf{C}''\right)$ which are answered by $C_0 \cdot \mathrm{e}\left(\mathbf{C}_1, \mathrm{msk} \cdot \widehat{\mathbf{G}}_2^{\sigma}\right)^{-1}$ in the case that the consistency checks are satisfied. Thereby $\sigma \leftarrow \mathbb{Z}_p$ is chosen once in the setup phase. This step plays similar role as the reduction step $\mathrm{G}_{q_1+3} \rightarrow \mathrm{G}'_{q_1+3}$ for our framework in composite-order groups (cf. Lemma 5.23 on page 112). Recall that in both experiments the challenge ciphertext is semi-functional and all keys given to $\mathcal{A}$ are normal.

Before we state and prove the actual lemma let us consider two additional statements. First of all we argue that probability for event BadQuery are negligible in G_0'. Recall the definition of event BadQuery from Subsection 7.2.1 (on page 181). Notice that we could also deduce this from previous results, since the experiments G_0 and G_0' are indistinguishable. However, using direct proofs, we achieve improved bounds.

Lemma 7.10. *For every ppt $\mathcal{A} \in \mathbf{A}_\Pi$, and every $\mathrm{des} \in \Omega$ there exists a ppt algorithm $\mathcal{B}$ such that*

$$\mathrm{Adv}_{\mathcal{B}}^{\mathcal{D}_d\text{-mDH}}(\lambda) = \Pr\left[\mathrm{BadQuery}\right] \ ,$$

where the probability space on the right side is defined by experiment $\mathrm{G}'_{0,\mathcal{A},\Pi}(\lambda, \mathrm{des})$.

Proof. We deduce the statement from the proof of Lemma 7.7. We just have to modify $\mathcal{B}$ from the proof of this lemma such that the decryption queries are answered as defined in $\mathrm{G}'_{0,\mathcal{A},\Pi}(\lambda, \mathrm{des})$ – that is, using separately generated normal user secret keys. All arguments in the proof are exactly the same. □

Intuitively, from Lemma 7.10 we deduce that event HashAbort $\vee$ BadQuery can be neglected. Combined with the following lemma we can argue that the hash values for ciphertexts submitted by adversary are all unequal the hash value of the challenge ciphertext except for negligible probability. The next lemma plays similar role as Lemma 5.12 for composite-order framework.

Lemma 7.11. *Let* $\mathcal{A} \in \mathbf{A}_\Pi$ *and* $\mathrm{des} \in \Omega$ *be arbitrary. Consider conditional probability distribution derived from* $\mathrm{G}'_{0,\mathcal{A},\Pi}(\lambda, \mathrm{des})$ *given* $\overline{\mathrm{HashAbort}} \wedge \overline{\mathrm{BadQuery}}$*. Let* CT^* *be the challenge ciphertext. Then, for every* $\mathrm{CT} \neq \mathrm{CT}^*$ *submitted by* $\mathcal{A}$ *to the decryption oracle such that* CT *pass the consistency checks it holds*

$$t \neq t^* \pmod{p} \ ,$$

where $t = \mathrm{H}(\mathrm{HInput}(\mathrm{CT}))$ *and* $t^* = \mathrm{H}(\mathrm{HInput}(\mathrm{CT}^*))$.

Proof. Let $\mathrm{CT}^* = \left(\mathrm{cInd}^*, C_0^*, \overrightarrow{\mathbf{C}}^*, \mathbf{C}''^*\right)$ be the challenge ciphertext. Assume that it holds $t = t^*$ for some submitted ciphertext $\mathrm{CT} \neq \mathrm{CT}^*$ which satisfies the consistency checks. If the HashAbort event does not occur, then it holds $\mathrm{HInput}(\mathrm{CT}) = \mathrm{HInput}(\mathrm{CT}^*)$ by definition of this event. Then, from the definition of function HInput (on page 147) we deduce that CT has the form $\mathrm{CT} = \left(\mathrm{cInd}^*, C_0^*, \overrightarrow{\mathbf{C}}^*, \mathbf{C}''\right)$. We deduce that it must hold $\mathbf{C}'' \neq \mathbf{C}''^*$.

Next, by Lemma 6.21 it holds for CT^*, which is a correctly generated semi-functional ciphertext for cInd^*:

$$\mathbf{C}''^* = {}^{t^* \cdot \boldsymbol{U} + \boldsymbol{V}}\mathbf{C}_1^* \ ,$$

where $\boldsymbol{U}, \boldsymbol{V}$ are parameters chosen in the setup algorithm and $\mathbf{C}_1^*$ is the first element of $\overrightarrow{\mathbf{C}}^*$. Furthermore, if CT does not cause the BadQuery event, it holds (by the first property defined for BadQuery)

$$\begin{aligned} \mathbf{C}'' &\stackrel{!}{=} {}^{t \cdot \boldsymbol{U} + \boldsymbol{V}}\mathbf{C}_1 \\ &= {}^{t^* \cdot \boldsymbol{U} + \boldsymbol{V}}\mathbf{C}_1^* \\ &= \mathbf{C}''^* \ . \end{aligned}$$

Where in the middle equation we used $t = t^*$ and $\overrightarrow{\mathbf{C}}^* \ni \mathbf{C}_1^* = \mathbf{C}_1 \in \overrightarrow{\mathbf{C}}$. We conclude $\mathrm{CT} = \mathrm{CT}^*$ which contradicts previous statement. This proves the lemma. □

Notice that the additional condition $\mathrm{CT} \neq \mathrm{CT}^*$ is not a real restriction, since in the first query phase the probability that any submitted ciphertext CT is equal CT^* is at most $\frac{1}{p^d} \leq \frac{1}{2^\lambda}$ over the random choice of exponent $\boldsymbol{s}_0^* \in \mathbb{Z}_p^d$ of $\mathbf{C}_1^* \in \overrightarrow{\mathbf{C}}^*$. In turn, in the second query phase adversary $\mathcal{A} \in \mathbf{A}_\Pi$ never asks for decryption of CT^*. Now we are ready to prove the next reduction step.

Lemma 7.12. *For every ppt* $\mathcal{A} \in \mathbf{A}_\Pi$ *and every* $\mathrm{des} \in \Omega$ *there exist ppt algorithms* $\mathcal{B}$ *such that*

$$\left|\mathrm{Adv}^{\mathrm{G}'_0}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) - \mathrm{Adv}^{\mathrm{G}''_0}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})\right| \leq \mathrm{Adv}^{\mathcal{D}_d\text{-mDH}}_{\mathcal{B}}(\lambda) + \frac{2 \cdot q_{\mathrm{dec}}}{2^\lambda} \ ,$$

where q_{dec} *is the upper bound for the number of decryption queries of* $\mathcal{A}$.

Proof. Let $\mathcal{A} \in \mathbf{A}_\Pi$ and $\mathrm{des} \in \Omega$ be arbitrary but fixed. By definitions of the experiments, they can differ only if event HashAbort does not occur and $\mathcal{A}$ submits a ciphertext $\mathrm{CT} = \left(\mathrm{cInd}, C_0, \overrightarrow{\mathbf{C}}, \mathbf{C}''\right)$ to the decryption oracle such that CT pass the consistency checks but the answers of the decryption oracles differ. We denote this last failure event by F. As long as $\mathrm{F} \wedge \overline{\mathrm{HashAbort}}$ does not occur, the experiments proceed identically and in particular, the outputs of the experiments are identical. We deduce that it holds

$$\mathrm{G}'_{0,\Pi,\mathcal{A}}(\lambda, \mathrm{des}) = 1 \wedge \overline{\mathrm{F} \wedge \overline{\mathrm{HashAbort}}} \Leftrightarrow \mathrm{G}''_{0,\Pi,\mathcal{A}}(\lambda, \mathrm{des}) = 1 \wedge \overline{\mathrm{F} \wedge \overline{\mathrm{HashAbort}}}$$

and by Difference Lemma 5.8 it holds

$$\begin{aligned} & \left|\mathrm{Adv}^{\mathrm{G}'_0}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) - \mathrm{Adv}^{\mathrm{G}''_0}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})\right| \\ \overset{\text{by def.}}{=} & \left|\Pr\left[\mathrm{G}'_{0,\Pi,\mathcal{A}}(\lambda, \mathrm{des}) = 1\right] - \Pr\left[\mathrm{G}''_{0,\Pi,\mathcal{A}}(\lambda, \mathrm{des}) = 1\right]\right| \\ \leq & \Pr\left[\mathrm{F} \wedge \overline{\mathrm{HashAbort}}\right] \ . \end{aligned}$$

Our aim is to analyze the probability for $\mathrm{F} \wedge \overline{\mathrm{HashAbort}}$, which is the same in both experiments. Hence, we estimate it based on probability experiment $\mathrm{G}'_{0,\Pi,\mathcal{A}}(\lambda, \mathrm{des})$.

Let BDdec be the events that $\mathcal{A}$ submits $\mathrm{CT} = \mathrm{CT}^*$ in the first query phase. By union bound and due to the random choice of exponent $\boldsymbol{s}^*_0 \in \mathbb{Z}_p^d$ of $\mathbf{C}^*_1 \in \overrightarrow{\mathbf{C}}^* \in \mathrm{CT}^*$ the probability for BDdec is at most $\frac{q_{\mathrm{dec}_1}}{p^d}$, where q_{dec_1} is the upper bound for the number of decryption queries of $\mathcal{A}$ in the first query phase. Next, let E be the event defined by

$$\mathrm{E} := \mathrm{BadQuery} \vee \mathrm{BDdec}.$$

Then, by the law of total probability and using the upper bound from Lemma 7.10 and the upper bound for BDdec we deduce

$$\begin{aligned} \Pr\left[\mathrm{F} \wedge \overline{\mathrm{HashAbort}}\right] &\leq \Pr\left[\mathrm{F} \wedge \overline{\mathrm{HashAbort}} \wedge \overline{\mathrm{E}}\right] + \Pr\left[\mathrm{E}\right] \\ &\leq \Pr\left[\mathrm{F} \wedge \overline{\mathrm{HashAbort}} \wedge \overline{\mathrm{E}}\right] + \Pr\left[\mathrm{BadQuery}\right] + \Pr\left[\mathrm{BDdec}\right] \\ &\leq \Pr\left[\mathrm{F} \wedge \overline{\mathrm{HashAbort}} \wedge \overline{\mathrm{E}}\right] + \mathrm{Adv}^{\mathcal{D}_d\text{-mDH}}_{\mathcal{B}}(\lambda) + \frac{q_{\mathrm{dec}_1}}{p^d}. \end{aligned}$$

It remains to analyze the probability for $\mathrm{F} \wedge \overline{\mathrm{HashAbort}} \wedge \overline{\mathrm{E}}$. Let BD be the event that there is a ciphertext $\mathrm{CT} = \left(\mathrm{cInd}, C_0, \overrightarrow{\mathbf{C}}, \mathbf{C}''\right)$ submitted by $\mathcal{A}$ to the decryption oracle which pass the consistency checks and satisfies two following conditions:

- $\mathbf{C}'' = {}^{t\cdot\boldsymbol{U}+\boldsymbol{V}}\mathbf{C}_1$.
- $\hat{s}_0 \neq 0 \pmod p$, where $\hat{s}_0 \in \mathbb{Z}_p$ is uniquely defined by $\mathbf{C}_1 = \mathbf{G}_1^{\boldsymbol{s}_0} \cdot \widehat{\mathbf{G}}_1^{\hat{s}_0}$ (according to Lemma 6.1).

Next we prove that $\Pr\left[\mathrm{F} \wedge \overline{\mathrm{HashAbort}} \wedge \overline{\mathrm{E}} \wedge \overline{\mathrm{BD}}\right] = 0$. Let $\mathrm{CT} \neq \mathrm{CT}^*$ be a ciphertext submitted by $\mathcal{A}$ to the decryption oracle which causes event F. In particular CT passes the consistency checks. In the case of $\overline{\mathrm{BadQuery}}$ the decryption query on CT in experiment G'_0 results in $m = C_0 \cdot \mathrm{e}(\mathbf{C}_1, \mathrm{msk})^{-1}$ by additional statement of Lemma 7.9. By definition of G''_0, the answer to this query is

$$\begin{aligned} m' &= C_0 \cdot \mathrm{e}\left(\mathbf{C}_1, \mathrm{msk} \cdot \widehat{\mathbf{G}}_2^{\sigma}\right)^{-1} \\ &= C_0 \cdot \mathrm{e}(\mathbf{C}_1, \mathrm{msk})^{-1} \cdot \mathrm{e}\left(\mathbf{G}_1^{\boldsymbol{s}_0} \cdot \widehat{\mathbf{G}}_1^{\hat{s}_0}, \widehat{\mathbf{G}}_2^{\sigma}\right)^{-1} \\ &= m \cdot \mathrm{e}\left(\widehat{\mathbf{G}}_1, \widehat{\mathbf{G}}_2\right)^{-\hat{s}_0 \cdot \sigma} \\ &= m \cdot \mathrm{e}(g_1, g_2)^{-\hat{s}_0 \cdot \sigma} \ . \end{aligned}$$

Event $\overline{\text{BadQuery}}$ implies that it holds $\mathbf{C}'' = {}^{t\cdot\boldsymbol{U}+\boldsymbol{V}}\mathbf{C}_1$ (by the first property defined for event BadQuery). Together with $\overline{\text{BD}}$ this implies $\hat{s}_0 = 0 \pmod p$ and thus $m' = m$. This contradicts the assumption that CT causes event F. We conclude that $\Pr\left[\text{F} \wedge \overline{\text{E}} \wedge \overline{\text{BD}}\right] = 0$.

Finally we argue that the probability for $\overline{\text{E}} \wedge \overline{\text{HashAbort}} \wedge \text{BD} = \text{BD} \wedge \overline{\text{HashAbort}} \wedge \overline{\text{BadQuery}} \wedge \overline{\text{BDdec}}$ is negligible. Let $\text{CT} = \left(\text{cInd}, C_0, \overrightarrow{\mathbf{C}}, \mathbf{C}''\right)$ be such that the conditions of BD are satisfied for this ciphertext. By definition of this event it holds $\hat{s}_0 \neq 0$ for CT. Furthermore, conditioned on $\overline{\text{BadQuery}}$ it holds

$$\begin{aligned}\mathbf{C}'' &\overset{!}{=} {}^{t\cdot\boldsymbol{U}+\boldsymbol{V}}\mathbf{C}_1 \\ &= {}^{t\cdot\boldsymbol{U}+\boldsymbol{V}}\left(\mathbf{G}_1^{\boldsymbol{s}_0} \cdot \widehat{\mathbf{G}}_1^{\hat{s}_0}\right) \\ &= {}^{t\cdot\boldsymbol{U}+\boldsymbol{V}}\mathbf{G}_1^{\boldsymbol{s}_0} \cdot {}^{t\cdot\boldsymbol{U}+\boldsymbol{V}}\widehat{\mathbf{G}}_1^{\hat{s}_0} \ , \end{aligned} \tag{7.17}$$

where $\hat{s}_0 \neq 0$. However, by Lemma 6.8 there are p possible values of $t \cdot \boldsymbol{U} + \boldsymbol{V}$ in the view of adversary $\mathcal{A}$ as long as $t \neq t^*$, which is guarantied by $\overline{\text{BadQuery}} \wedge \overline{\text{HashAbort}} \wedge \overline{\text{BDdec}}$ and by Lemma 7.11. Notice that all elements from Lemma 6.8 are information-theoretically known to $\mathcal{A}$. Namely, values $\boldsymbol{B}$ and $\boldsymbol{Z}$ determine $\mathbf{G}_1, \mathbf{G}_2 \in \text{pp}_\kappa$ and $\widehat{\mathbf{G}}_1$, $\widehat{\mathbf{G}}_2$ used to generate semi-functional ciphertext and semi-functional keys. Similarly $\boldsymbol{U} \cdot \boldsymbol{B} \cdot \left(\begin{smallmatrix}\boldsymbol{I}_d \\ \mathbf{0}\end{smallmatrix}\right)$ $\boldsymbol{V} \cdot \boldsymbol{B} \cdot \left(\begin{smallmatrix}\boldsymbol{I}_d \\ \mathbf{0}\end{smallmatrix}\right)$ determine $\mathbf{U}, \mathbf{V} \in \text{pp}_\kappa$ whereas $\boldsymbol{U}^\top \cdot \boldsymbol{Z} \cdot \left(\begin{smallmatrix}\boldsymbol{I}_d \\ \mathbf{0}\end{smallmatrix}\right)$, $\boldsymbol{V}^\top \cdot \boldsymbol{Z} \cdot \left(\begin{smallmatrix}\boldsymbol{I}_d \\ \mathbf{0}\end{smallmatrix}\right)$, determine $\overline{\mathbf{U}}, \overline{\mathbf{V}} \in \text{pp}_\kappa$. Finally, $t^* \cdot \boldsymbol{U} + \boldsymbol{V}$ is contained in $\mathbf{C}''^*$ (cf. Lemma 6.21) and all other elements in the view of $\mathcal{A}$ are independent of $\boldsymbol{U}$ and $\boldsymbol{V}$. Hence, the probability that $\mathbf{C}''$ has the form of (7.17) is $\frac{1}{p}$ over the random choice of $\boldsymbol{U}$ and $\boldsymbol{V}$. By union bound over q_{dec} many decryption queries it holds

$$\Pr\left[\text{F} \wedge \overline{\text{HashAbort}} \wedge \overline{\text{E}} \wedge \text{BD}\right] \leq \Pr\left[\text{BD} \wedge \overline{\text{HashAbort}} \wedge \overline{\text{BadQuery}} \wedge \overline{\text{BDdec}}\right] \leq \frac{q_{\text{dec}}}{p} \ .$$

Summarizing our results we deduce that there exist ppt algorithms $\mathcal{B}$ such that

$$\begin{aligned}\left|\text{Adv}^{\text{G}'_0}_{\Pi,\mathcal{A}}(\lambda, \text{des}) - \text{Adv}^{\text{G}''_0}_{\Pi,\mathcal{A}}(\lambda, \text{des})\right| &\leq \Pr\left[\text{F} \wedge \overline{\text{HashAbort}}\right] \\ &\leq \Pr\left[\text{F} \wedge \overline{\text{HashAbort}} \wedge \overline{\text{E}}\right] + \text{Adv}^{\mathcal{D}_d\text{-mDH}}_{\mathcal{B}}(\lambda) + \frac{q_{\text{dec}_1}}{p^d} \\ &= \Pr\left[\text{F} \wedge \overline{\text{HashAbort}} \wedge \overline{\text{E}} \wedge \text{BD}\right] \\ &\quad + \text{Adv}^{\mathcal{D}_d\text{-mDH}}_{\mathcal{B}}(\lambda) + \frac{q_{\text{dec}_1}}{p^d} \\ &\leq \frac{q_{\text{dec}}}{p} + \text{Adv}^{\mathcal{D}_d\text{-mDH}}_{\mathcal{B}}(\lambda) + \frac{q_{\text{dec}_1}}{p^d} \\ &\leq \text{Adv}^{\mathcal{D}_d\text{-mDH}}_{\mathcal{B}}(\lambda) + \frac{2 \cdot q_{\text{dec}}}{2^\lambda} \ , \end{aligned}$$

where the last inequality holds since $|p| = \lambda$ by our convention and since $d \geq 2$, $q_{\text{dec}_1} \leq q_{\text{dec}}$. This finally proves the lemma. □

7.2.2. Reduction Steps with Simple Extensions

In this subsection we present reduction steps which are similar to the corresponding reduction steps in the proof of the original CPA-secure framework. We formally prove all this reduction based on the lemmata proved in the previous chapter, which simplifies the proofs.

From $\text{G}_{k-1,3}$ to $\text{G}_{k,1}$

Let $k \in [q_1]$. In experiment $\text{G}_{k-1,3}$ the first $k-1$ keys in Phase 1 are semi-functional of type 3 (random element β is chosen separately for every key) and all other keys are normal. All decryption queries are answered as defined in G''_0. We notice that $\text{G}_{0,3}$ is equivalent to G''_0. Experiment $\text{G}_{i,1}$ is the same except for the i'th key, which is semi-functional of type 1.

Lemma 7.13 (cf. Lemma 5 in [Att15]). *For every $k \in [q_1]$, every* $\mathrm{des} \in \Omega$*, every $\mathcal{A} \in \mathbf{A}_\Pi$ there exists a ppt algorithm $\mathcal{B}$ such that*

$$\mathrm{Adv}_{\mathcal{B}}^{\mathcal{D}_d-\mathrm{mDH}}(\lambda) = \left|\mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{k-1,3}}(\lambda,\mathrm{des}) - \mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{k,1}}(\lambda,\mathrm{des})\right| \ .$$

Proof. Let $k \in [q_1]$, $\mathrm{des} \in \Omega$, and $\mathcal{A} \in \mathbf{A}_\Pi$ be arbitrary, but fixed. We present an algorithm $\mathcal{B}$, which receives des as an additional input. According to Assumption 6.1 algorithm $\mathcal{B}$ is also given the group description $\mathbb{GD} = (p, (g_1, \mathbb{G}_1), (g_2, \mathbb{G}_2), \mathbb{G}_\mathrm{T}, \mathrm{e})$ and challenge $\mathbf{T} = g_2^{\boldsymbol{T}} \in \mathbb{G}^{(d+1)\times(d+1)}$ as well as $\mathbf{Z} = g_2^{\boldsymbol{T}\cdot\binom{\boldsymbol{y}}{\hat{y}}} \in \mathbb{G}_1^{d+1}$. The elements are chosen by $\mathbb{GD} \leftarrow \mathcal{G}(1^\lambda)$, $\boldsymbol{T} = \begin{pmatrix}\boldsymbol{M} & \boldsymbol{0}\\ \boldsymbol{c}^\top & 1\end{pmatrix} \leftarrow \mathcal{D}_d$, $\boldsymbol{y} \leftarrow \mathbb{Z}_p^d$, where $\boldsymbol{M} \in \mathbb{GL}_{p,d}$ and $\boldsymbol{c} \in \mathbb{Z}_p^d$. Furthermore, $\hat{y}$ is either zero or uniformly distributed in $\mathbb{Z}_p^*$ and $\mathcal{B}$ has to distinguish between these both cases.

The setup phase of $\mathcal{B}$ is almost the same as in Algorithm 19. $\mathcal{B}$ additionally initialize $j := 0$, chooses $\sigma \leftarrow \mathbb{Z}_p$ and computes $\mathbf{Q} := \mathrm{msk} \cdot \widehat{\mathbf{G}}_2^\sigma$ in order to answer decryption queries. In Algorithm 19 we already showed how to compute $\widehat{\mathbf{G}}_2$. Furthermore, notice that σ is chosen as required and $\mathbf{Q}$ can be directly used to answer decryption queries on $\mathrm{CT} = \left(\mathrm{cInd}, C_0, \overrightarrow{\mathbf{C}}, \mathbf{C}''\right)$ by $C_0 \cdot \mathrm{e}\left(\mathbf{C}_1, \mathrm{msk} \cdot \widehat{\mathbf{G}}_2^\sigma\right)^{-1} = C_0 \cdot \mathrm{e}(\mathbf{C}_1, \mathbf{Q})^{-1}$ as required. The challenge and all normal keys are also generated as in Algorithm 19. Hence, we show only the first query phase and in particular how to generate the semi-functional keys of type 3 and the key k which will be normal or semi-functional of type 1 depending on the challenge of $\mathcal{B}$.

Recall that by construction of Algorithm 19 it holds

$$\begin{pmatrix}\mathbf{G}_2 & \widehat{\mathbf{G}}_2\end{pmatrix} = \mathbf{T}^{\tilde{\boldsymbol{B}}^{-\top}\begin{pmatrix}\boldsymbol{J} & \boldsymbol{0}\\ \boldsymbol{0} & -1\end{pmatrix}} = g_2^{\tilde{\boldsymbol{B}}^{-\top}\cdot\boldsymbol{T}\cdot\begin{pmatrix}\boldsymbol{J} & \boldsymbol{0}\\ \boldsymbol{0} & -1\end{pmatrix}} \ .$$

The first $k-1$ keys are semi-functional of type 3 by construction. Furthermore, we state that key k is semi-functional of type 1 (if $\hat{y} \in \mathbb{Z}_p^*$) or normal (if $\hat{y} = 0$). Namely, for this key $\mathcal{B}$ first of all randomizes the own challenge and computes

$$\mathbf{R} = g_1^{\tilde{\boldsymbol{B}}^{-\top}\cdot\boldsymbol{T}\cdot\begin{pmatrix}\boldsymbol{r}'_1 & \cdots & \boldsymbol{r}'_{m_2}\\ \hat{r}'_1 & \cdots & \hat{r}'_{m_2}\end{pmatrix}} \leftarrow \mathrm{ReRand}\left(\mathbb{G}_1, g_1^{\tilde{\boldsymbol{B}}^{-\top}\cdot\boldsymbol{T}}, g_1^{\tilde{\boldsymbol{B}}^{-\top}\cdot\boldsymbol{T}\cdot\binom{\boldsymbol{y}}{\hat{y}}}, m_2\right) \ .$$

By Lemma 6.9 $\boldsymbol{r}'_1, \ldots, \boldsymbol{r}'_{m_2} \in \mathbb{Z}_p^d$ are uniformly and independently distributed. Furthermore, if $\hat{y} \neq 0$ the elements $\hat{r}'_1, \ldots, \hat{r}'_{m_2} \in \mathbb{Z}_p$ are also uniformly and independently distributed. In turn, if $\hat{y} = 0$ it holds $\hat{r}'_1 = \cdots = \hat{r}'_{m_2} = 0$. Now we use the fact that $\left(\begin{smallmatrix}\boldsymbol{J} & \boldsymbol{0}\\ \boldsymbol{0} & -1\end{smallmatrix}\right) \in \mathbb{GL}_{p,d+1}$, and hence there exists unique $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_{m_2} \in \mathbb{Z}_p^d$ and $\hat{r}_1, \ldots, \hat{r}_{m_2} \in \mathbb{Z}_p$ such that

$$\begin{pmatrix}\boldsymbol{J} & \boldsymbol{0}\\ \boldsymbol{0} & -1\end{pmatrix} \cdot \begin{pmatrix}\boldsymbol{r}_1 & \cdots & \boldsymbol{r}_{m_2}\\ \hat{r}_1 & \cdots & \hat{r}_{m_2}\end{pmatrix} = \begin{pmatrix}\boldsymbol{r}'_1 & \cdots & \boldsymbol{r}'_{m_2}\\ \hat{r}'_1 & \cdots & \hat{r}'_{m_2}\end{pmatrix} \ .$$

Furthermore, the distribution on matrix $\left(\begin{smallmatrix}\boldsymbol{r}_1 & \cdots & \boldsymbol{r}_{m_2}\\ \hat{r}_1 & \cdots & \hat{r}_{m_2}\end{smallmatrix}\right) \in \mathbb{Z}_p^{(d+1)\times m_2}$ is exactly the same as the distribution on $\left(\begin{smallmatrix}\boldsymbol{r}'_1 & \cdots & \boldsymbol{r}'_{m_2}\\ \hat{r}'_1 & \cdots & \hat{r}'_{m_2}\end{smallmatrix}\right)$. In particular if $\hat{y} = 0$ it holds $\hat{r}_1 = \cdots = \hat{r}_{m_2} = 0$. But now we can

Algorithm 21: $\mathcal{B}$ against Assumption 6.1

Input : $(\mathrm{des}, \mathbb{GD}, \mathbf{T}, \mathbf{Z})$.

Require: $\mathrm{des} \in \Omega$, $\mathbb{GD} = \left(p, \left(g_1, \mathbb{G}_1\right), \left(g_2, \mathbb{G}_2\right), \mathbb{G}_\mathrm{T}, \mathrm{e}\right) \in \left[\mathcal{G}\left(1^\lambda\right)\right]$, $\mathbf{T} \in \mathbb{G}_2^{(d+1)\times(d+1)}$, $\mathbf{Z} \in \mathbb{G}_2^{d+1}$.

Setup

  ...

  $\sigma \leftarrow \mathbb{Z}_p$, $\mathbf{Q} := \mathrm{msk} \cdot \widehat{\mathbf{G}}_2^{\sigma}$.

Phase I

  CKGen (kInd_i) **with** $\mathrm{kInd}_i \in \mathbb{X}_\kappa$**:**

    Store (i, kInd_i).

  Open (i)**:**

    $j := j + 1$

    case $j < k$ **do**

      Choose $\beta_j \leftarrow \mathbb{Z}_p$ and return

      $\mathrm{sk} \leftarrow \mathrm{SFKeyGen}\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{msk}, \left(\widehat{\mathbf{G}}_2, \_\right), \mathrm{kInd}, 3; \beta_j\right)$

    case $j = k$ **do**

      Compute $(\boldsymbol{k}, m_2) := \mathrm{Enc1}\left(\kappa, \mathrm{kInd}\right)$.

      Compute $\mathbf{R} = (\mathbf{R}_1, \ldots, \mathbf{R}_{m_2}) \in \mathbb{G}_2^{(d+1)\times(m_2)}$ by

$$\mathbf{R} \leftarrow \mathrm{ReRand}\left(\mathbb{G}_2, \tilde{\boldsymbol{B}}^{-\top}\mathbf{T}, \tilde{\boldsymbol{B}}^{-\top}\mathbf{Z}, m_2\right) \ ;$$

      Compute $\overrightarrow{\mathbf{K}} \leftarrow \mathrm{KeyGenAlt}\left(\boldsymbol{k}, \mathrm{msk}, \mathbf{R}, \overline{\mathbb{H}}\right)$ and return $\mathrm{sk} = \left(\mathrm{kInd}, \overrightarrow{\mathbf{K}}\right)$.

    case $j > k$ **do**

      Return $\mathrm{sk} \leftarrow \mathrm{KeyGen}\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{kInd}\right)$

  Dec (CT, i)**:**

    Realize the decryption oracles as defined in the experiment using $\mathbf{Q}$;

Challenge (given cInd^***,** $m_0, m_1 \in \mathcal{M} = \mathbb{G}_\mathrm{T}$ **from** $\mathcal{A}$**)**

  ...

Phase II

  ...

Guess

  ...

rewrite $\mathbf{R}$ as follows:

$$\begin{aligned}\mathbf{R} &= g_1^{\tilde{\boldsymbol{B}}^{-\top}\cdot\boldsymbol{T}\cdot\left(\begin{smallmatrix}\boldsymbol{r}'_1 & \cdots & \boldsymbol{r}'_{m_2}\\ \hat{r}'_1 & \cdots & \hat{r}'_{m_2}\end{smallmatrix}\right)}\\ &= g_1^{\tilde{\boldsymbol{B}}^{-\top}\cdot\boldsymbol{T}\cdot\left(\begin{smallmatrix}\boldsymbol{J} & \mathbf{0}\\ \mathbf{0} & -1\end{smallmatrix}\right)\cdot\left(\begin{smallmatrix}\boldsymbol{r}_1 & \cdots & \boldsymbol{r}_{m_2}\\ \hat{r}_1 & \cdots & \hat{r}_{m_2}\end{smallmatrix}\right)}\\ &= \left(\begin{array}{cc}\mathbf{G}_2 & \widehat{\mathbf{G}}_2\end{array}\right)^{\left(\begin{smallmatrix}\boldsymbol{r}_1 & \cdots & \boldsymbol{r}_{m_2}\\ \hat{r}_1 & \cdots & \hat{r}_{m_2}\end{smallmatrix}\right)} \ .\end{aligned}$$

That is, for every $i \in [m_2]$ it holds $\mathbf{R}_i = \mathbf{G}_2^{\boldsymbol{r}_i} \cdot \widehat{\mathbf{G}}_2^{\hat{r}_i}$. Hence, in the next step by Lemma 6.17 $\mathcal{B}$ computes the user secret key

$$\left(\mathrm{kInd}, \overrightarrow{\mathbf{K}}\right) = \mathrm{SFKeyGen}\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{msk}, \widehat{\mathrm{msk}}, \mathrm{kInd}, 1; \boldsymbol{r}_1, \ldots, \boldsymbol{r}_{m_2}, \hat{r}_1, \ldots, \hat{r}_{m_2}\right) \ .$$

We deduce that in the case $\mathbf{Z} = g_1^{\boldsymbol{T}\cdot\left(\begin{smallmatrix}\boldsymbol{y}\\ 0\end{smallmatrix}\right)}$ (that is $\hat{y} = 0$) the generated key is a correctly distributed

normal key and in the case of $\mathbf{Z} = g_1^{\boldsymbol{T}\cdot\binom{\boldsymbol{y}}{\hat{y}}}$ (that is $\hat{y} \in \mathbb{Z}_p^*$) the generated key is a correctly distributed semi-functional key of type 1.

All together the view of $\mathcal{A}$ is as defined in game $\mathrm{G}_{k-1,3}$ if $\hat{y} = 0$ and it is as defined in game $\mathrm{G}_{k,1}$ if $\hat{y} \neq 0$. In the guess phase $\mathcal{B}$ outputs 1 if and only if the event HashAbort did not occurred and $\mathcal{A}$ outputs the correct bit – that is, if and only if the output of the corresponding experiment is equal one. Hence, for every $\mathcal{A} \in \mathbf{A}_\Pi$ and every $\mathrm{des} \in \Omega$ there is $\mathcal{B} = \mathcal{B}_{\mathcal{A}}(\mathrm{des}, \cdot)$ as defined in Algorithm 21 such that it holds

$$\begin{aligned}
\mathrm{Adv}_{\mathcal{B}}^{\mathcal{D}_d\text{-mDH}}(\lambda) &\overset{\text{by def.}}{=} \left|\Pr\left[\mathcal{B}\left(\mathbb{GD}, g_1^{\boldsymbol{A}}, g_1^{\boldsymbol{A}\cdot\binom{\boldsymbol{y}}{0}}\right) = 1\right] - \Pr\left[\mathcal{B}\left(\mathbb{GD}, g_1^{\boldsymbol{A}}, g_1^{\boldsymbol{A}\cdot\binom{\boldsymbol{y}}{\hat{y}}}\right) = 1\right]\right| \\
&= \left|\Pr\left[\mathrm{G}_{k-1,3,\Pi,\mathcal{A}}(\lambda, \mathrm{des}) = 1\right] - \Pr\left[\mathrm{G}_{k,1,\Pi,\mathcal{A}}(\lambda, \mathrm{des}) = 1\right]\right| \\
&\overset{\text{by def.}}{=} \left|\mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{k-1,3}}(\lambda, \mathrm{des}) - \mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{k,1}}(\lambda, \mathrm{des})\right| .
\end{aligned}$$

This finally proves the lemma. □

From $\mathrm{G}_{k,1}$ to $\mathrm{G}_{k,2}$

Let $k \in [q_1]$. In experiment $\mathrm{G}_{k,1}$ the first $k-1$ keys in Phase 1 are semi-functional of type 3, key k is semi-functional of type 1 and all other keys are normal. The experiment $\mathrm{G}_{k,2}$ is the same except for key k which is semi-functional of type 2.

The following lemma extensively use the parameter hiding lemma. We only need to extend the challenge ciphertext, which is possible since $\boldsymbol{U}, \boldsymbol{V}$ are known. For the decryption queries it is important that $\widehat{\mathbf{G}}_2$ and msk are known.

Lemma 7.14 (cf. Lemma 7 in [Att15])**.** *For every $k \in [q_1]$, every $\mathrm{des} \in \Omega$ and every $\mathcal{A} \in \mathbf{A}_\Pi$ there exists a ppt algorithm $\mathcal{B}$ such that*

$$\mathrm{Adv}_{\mathcal{B}}^{\mathrm{CMH}}(\lambda) = \left|\mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{k,1}}(\lambda, \mathrm{des}) - \mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{k,2}}(\lambda, \mathrm{des})\right| .$$

Let $\mathrm{kInd} \in \mathbb{X}_\kappa$, $n = \mathrm{Param}(\kappa)$, $(\boldsymbol{k}, m_2) = \mathrm{Enc1}(\kappa, \mathrm{kInd})$, $m_1 = |\boldsymbol{k}|$. Before we present the proof let us define set $J \subseteq [m_2]$ similar to the set I for ciphertext encodings:

$$i \in J \Leftrightarrow \exists_{\tau \in [m_1]} : k_\tau = \mathrm{X}_{r_i} .$$

Then, since P is a regular pair encoding we deduce that for every $\tau \in [m_1]$ it holds

$$k_\tau = a_\tau \cdot \mathrm{X}_\alpha + \sum_{i \in [m_2]} a_{\tau,i} \cdot \mathrm{X}_{r_i} + \sum_{i \in J} \sum_{j \in [n]} a_{\tau,i,j} \cdot \mathrm{X}_{h_j} \cdot \mathrm{X}_{r_i} .$$

Proof. We present an algorithm $\mathcal{B}$, which receives des as an additional input. According to the CMH security property $\mathcal{B}$ is also given $\mathbb{GD} = (p, (g_1, \mathbb{G}_1), (g_2, \mathbb{G}_2), \mathbb{G}_\mathrm{T}, \mathrm{e})$ (see the definitions in Subsection 1.3.5).

$\mathcal{B}$ computes correctly generated semi-functional public parameter and master secret key by construction (see Algorithm 22). The matrices $\boldsymbol{H}_1, \ldots, \boldsymbol{H}_n$ are not fixed is the view of $\mathcal{A}$ and are fixed later according to Lemma 6.7.

The semi-functional keys of type 3 and normal keys are computed using corresponding algorithms, and hence are correctly distributed. All these keys are independent of $\widehat{\mathbf{H}}_1, \ldots, \widehat{\mathbf{H}}_n \in \mathbb{G}_1^{d+1}$ and $\widehat{\overline{\mathbf{H}}}_1, \ldots, \widehat{\overline{\mathbf{H}}}_n \in \mathbb{G}_2^{d+1}$, which are not defined yet.

Algorithm 22: $\mathcal{B}$ against CMH security property of P

Input : $\mathbb{GD}$, des
Require: $\mathbb{GD} = (p, (g_1, \mathbb{G}_1), (g_2, \mathbb{G}_2), \mathbb{G}_T, e)$, des $\in \Omega$

Setup
- Set $j := 0$, $\kappa := (\text{des}, p)$;
- Choose $\boldsymbol{B} \leftarrow \mathbb{GL}_{p,d+1}$, $\tilde{\boldsymbol{D}} \leftarrow \mathbb{GL}_{p,d}$, $\boldsymbol{H}'_1, \ldots, \boldsymbol{H}'_n, \boldsymbol{U}, \boldsymbol{V} \leftarrow \mathbb{Z}_p^{(d+1)\times(d+1)}$, $\boldsymbol{\alpha} \leftarrow \mathbb{Z}_p^{d+1}$;
- Compute $\left(\text{pp}_\kappa, \text{msk}, \widehat{\text{pp}_\kappa}, \widehat{\text{msk}}\right) \leftarrow \text{SFSetup}\left(1^\lambda, \text{des}; \boldsymbol{B}, \tilde{\boldsymbol{D}}, \boldsymbol{H}'_1, \ldots, \boldsymbol{H}'_n, \boldsymbol{U}, \boldsymbol{V}\right)$;
- Parse $\widehat{\text{pp}_\kappa} = \left(\widehat{\mathbf{Y}}, \widehat{\mathbf{G}}_1, \_, \widehat{\mathbf{U}}, \widehat{\mathbf{V}}\right)$, $\widehat{\text{msk}} = \left(\widehat{\mathbf{G}}_2, \_\right)$. Let $\boldsymbol{Z} := \boldsymbol{B}^{-\top} \cdot \boldsymbol{D}$;
- Choose $\sigma \leftarrow \mathbb{Z}_p$ and compute $\mathbf{Q} := \text{msk} \cdot \widehat{\mathbf{G}}_2^{\sigma}$.
- Simulate $\mathcal{A}\left(1^\lambda, \text{pp}_\kappa\right)$;

Phase I
- **CKGen** (kInd_i) **with** $\text{kInd}_i \in \mathbb{X}_\kappa$**:**
  - Store (i, kInd_i);
- **Open** (i)**:**
  - Set $j := j + 1$;
  - **case** $j < k$ **do**
    - Choose $\beta_j \leftarrow \mathbb{Z}_p$, return $\text{sk} \leftarrow \text{SFKeyGen}\left(1^\lambda, \text{pp}_\kappa, \text{msk}, \left(\widehat{\mathbf{G}}_2, \_\right), \text{kInd}_i, 3; \beta_j\right)$.
  - **case** $j = k$ **do**
    - Compute a normal secret key $\left(\text{kInd}, \overrightarrow{\mathbf{K}}\right) = \text{sk} \leftarrow \text{KeyGen}\left(\lambda, \text{pp}_\kappa, \text{msk}, \text{kInd}_i\right)$;
    - Query $\mathbb{G}_2^{m_1} \ni \widehat{\mathbf{K}}' = \left(\widehat{K}'_1, \ldots, \widehat{K}'_{m_1}\right) \leftarrow \mathcal{O}^1_{\text{CMH},\nu,\hat{\alpha},\hat{\boldsymbol{h}}}\left(\text{kInd}_i\right)$;
    - For all $\tau \in [m_1]$ compute $\mathbf{K}_\tau := \mathbf{K}_\tau \cdot \widehat{\mathbf{K}}_\tau$, where

$$\widehat{\mathbf{K}}_\tau := \left(\widehat{K}'_\tau\right)^{\boldsymbol{Z}\cdot\binom{\mathbf{0}}{1}} \cdot \prod_{j\in[n]} \prod_{i\in J} \left(\widehat{K}'_{\tau_i}\right)^{a_{\tau,i,j}\cdot \boldsymbol{H}'^{\top}_j \cdot \boldsymbol{Z}\cdot\binom{\mathbf{0}}{1}} ,$$

      where $\boldsymbol{Z} = \boldsymbol{B}^{-\top} \cdot \left(\begin{smallmatrix} \tilde{\boldsymbol{D}} & \mathbf{0} \\ \mathbf{0} & 1 \end{smallmatrix}\right)$;
    - Return the modified key $\text{sk} = \left(\text{kInd}_i, \overrightarrow{\mathbf{K}}\right)$;
  - **case** $j > k$ **do** Return $\text{sk} \leftarrow \text{KeyGen}\left(1^\lambda, \text{pp}_\kappa, \text{msk}, \text{kInd}_i\right)$;
- **Dec** (CT, i)**:**
  - Realize the decryption oracles as defined in the experiment using $\mathbf{Q}$;

User secret key number k for kInd_i is computed using the first oracle of CMH security game. Let us denote the corresponding index kInd_i by kInd. The oracle query to $\mathcal{O}^1_{\text{CMH},\nu,\hat{\alpha},\hat{\boldsymbol{h}}}(\text{kInd})$ results in

$$\widehat{\mathbf{K}}' = \begin{cases} g_2^{\boldsymbol{k}(0,\hat{\boldsymbol{r}},\hat{\boldsymbol{h}})} & \text{if } \nu = 0 \\ g_2^{\boldsymbol{k}(\hat{\alpha},\hat{\boldsymbol{r}},\hat{\boldsymbol{h}})} & \text{if } \nu = 1 \end{cases},$$

where ν is the challenge bit for $\mathcal{B}$. That is,for every $\tau \in [m_1]$ it holds

$$\begin{aligned} \mathbb{G}_2 \ni \widehat{K}'_\tau &= g_2^{k_\tau(\hat{\alpha},\hat{\boldsymbol{r}},\hat{\boldsymbol{h}})} \\ &= g_2^{a_\tau\cdot\hat{\alpha} + \sum_{i\in[m_2]}\left(a_{\tau,i}\cdot\hat{r}_i + \sum_{j\in[n]} a_{\tau,i,j}\cdot\hat{h}_j\cdot\hat{r}_i\right)} . \end{aligned}$$

We implicitly define $\forall_{j\in[n]} : \boldsymbol{H}_j := \boldsymbol{H}'_j + \boldsymbol{B} \cdot \left(\begin{smallmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \hat{h}_j \end{smallmatrix}\right) \cdot \boldsymbol{B}^{-1}$, where $\hat{h}_j \in \mathbb{Z}_p$ is the j'th component of $\hat{\boldsymbol{h}} \in \mathbb{Z}_p^n$ chosen uniformly at random by definition of CMH experiment. By Lemma 6.6

the public parameters given to $\mathcal{A}$ are consistent with these choices. Hence, $\forall_{j\in[n]} : \boldsymbol{H}_j^\top = \boldsymbol{H}_j'^\top + \boldsymbol{B}^{-\top} \cdot \left(\begin{smallmatrix}\mathbf{0} & \mathbf{0}\\ \mathbf{0} & \hat{h}_j\end{smallmatrix}\right) \cdot \boldsymbol{B}^\top$. Hence, it holds

$$\begin{aligned}
\widehat{\mathbf{H}}_j &\overset{\text{by def.}}{=} \boldsymbol{H}_j^\top \widehat{\mathbf{G}}_2 \\
&= g_2^{\left(\boldsymbol{H}_j'^\top + \boldsymbol{B}^{-\top}\cdot\left(\begin{smallmatrix}\mathbf{0} & \mathbf{0}\\ \mathbf{0} & \hat{h}_j\end{smallmatrix}\right)\cdot\boldsymbol{B}^\top\right)\cdot\boldsymbol{Z}\cdot\left(\begin{smallmatrix}\mathbf{0}\\ 1\end{smallmatrix}\right)} \\
&= g_2^{\boldsymbol{H}_j'^\top\cdot\boldsymbol{Z}\cdot\left(\begin{smallmatrix}\mathbf{0}\\ 1\end{smallmatrix}\right) + \boldsymbol{B}^{-\top}\cdot\left(\begin{smallmatrix}\mathbf{0} & \mathbf{0}\\ \mathbf{0} & \hat{h}_j\end{smallmatrix}\right)\cdot\boldsymbol{B}^\top\cdot\boldsymbol{B}^{-\top}\cdot\left(\begin{smallmatrix}\tilde{\boldsymbol{D}} & \mathbf{0}\\ \mathbf{0} & 1\end{smallmatrix}\right)\cdot\left(\begin{smallmatrix}\mathbf{0}\\ 1\end{smallmatrix}\right)} \\
&= g_2^{\boldsymbol{H}_j'^\top\cdot\boldsymbol{Z}\cdot\left(\begin{smallmatrix}\mathbf{0}\\ 1\end{smallmatrix}\right) + \boldsymbol{B}^{-\top}\cdot\left(\begin{smallmatrix}\mathbf{0}\\ \hat{h}_j\end{smallmatrix}\right)} \\
&= g_2^{\boldsymbol{H}_j'^\top\cdot\boldsymbol{Z}\cdot\left(\begin{smallmatrix}\mathbf{0}\\ 1\end{smallmatrix}\right) + \boldsymbol{Z}\cdot\left(\begin{smallmatrix}\mathbf{0}\\ \hat{h}_j\end{smallmatrix}\right)} .
\end{aligned} \tag{7.18}$$

Furthermore, it holds (will be required for the ciphertext)

$$\begin{aligned}
\widehat{\mathbf{H}}_j &\overset{\text{by def.}}{=} \boldsymbol{H}_j\widehat{\mathbf{G}}_1 \\
&= g_1^{\left(\boldsymbol{H}_j' + \boldsymbol{B}\cdot\left(\begin{smallmatrix}\mathbf{0} & \mathbf{0}\\ \mathbf{0} & \hat{h}_j\end{smallmatrix}\right)\cdot\boldsymbol{B}^{-1}\right)\cdot\boldsymbol{B}\cdot\left(\begin{smallmatrix}\mathbf{0}\\ 1\end{smallmatrix}\right)} \\
&= g_1^{\boldsymbol{H}_j'\cdot\boldsymbol{B}\cdot\left(\begin{smallmatrix}\mathbf{0}\\ 1\end{smallmatrix}\right) + \boldsymbol{B}\cdot\left(\begin{smallmatrix}\mathbf{0}\\ \hat{h}_j\end{smallmatrix}\right)} .
\end{aligned} \tag{7.19}$$

Now, let us consider how the key k is extended using $\widehat{\mathbf{K}}'$. We claim that $\mathcal{B}$ implicitly sets the semi-functional randomness $\beta, \hat{r}_1, \ldots, \hat{r}_{m_2}$ of the secret key to $\hat{\alpha}, \hat{\boldsymbol{r}}$, which are correctly distributed, since chosen by $\hat{\alpha} := 0$ (if $\nu = 0$) $\hat{\alpha} \leftarrow \mathbb{Z}_p$ (if $\nu = 1$) and $\hat{\boldsymbol{r}} \leftarrow \mathbb{Z}_p^{m_2}$ respectively. We have to show how to compute the semi-functional part of type 1 respectively type 2 key. Let $\tau \in [m_1]$ be arbitrary, then we are first of all given

$$\widehat{K}'_\tau = g_2^{a_\tau\cdot\hat{\alpha} + \sum_{i\in[m_2]} a_{\tau,i}\cdot\hat{r}_i + \sum_{i\in J}\sum_{j\in[n]} a_{\tau,i,j}\cdot\hat{h}_j\cdot\hat{r}_i} .$$

Furthermore, for $i \in J \subseteq [m_2]$ we are also given the corresponding random elements

$$\widehat{K}'_{\tau_i} = g_2^{\hat{r}_i} .$$

Hence, $\mathcal{B}$ computes

$$\begin{aligned}
\widehat{\mathbf{K}}_\tau &\overset{\text{by const.}}{=} \left(\widehat{K}'_\tau\right)^{\boldsymbol{Z}\cdot\left(\begin{smallmatrix}\mathbf{0}\\ 1\end{smallmatrix}\right)} \cdot \prod_{j\in[n]}\prod_{i\in J}\left(\widehat{K}'_{\tau_i}\right)^{a_{\tau,i,j}\cdot\boldsymbol{H}_j'^\top\cdot\boldsymbol{Z}\cdot\left(\begin{smallmatrix}\mathbf{0}\\ 1\end{smallmatrix}\right)} \\
&= \left(g_2^{a_\tau\cdot\beta + \sum_{i\in[m_2]} a_{\tau,i}\cdot\hat{r}_i + \sum_{j\in[n]}\sum_{i\in J} a_{\tau,i,j}\cdot\hat{r}_i\cdot\hat{h}_j}\right)^{\boldsymbol{Z}\cdot\left(\begin{smallmatrix}\mathbf{0}\\ 1\end{smallmatrix}\right)} \cdot \prod_{j\in[n]}\prod_{i\in J}\left(g_2^{\hat{r}_i}\right)^{a_{\tau,i,j}\cdot\boldsymbol{H}_j'^\top\cdot\boldsymbol{Z}\cdot\left(\begin{smallmatrix}\mathbf{0}\\ 1\end{smallmatrix}\right)} \\
&= \left(g_2^{a_\tau\cdot\beta + \sum_{i\in[m_2]} a_{\tau,i}\cdot\hat{r}_i}\right)^{\boldsymbol{Z}\cdot\left(\begin{smallmatrix}\mathbf{0}\\ 1\end{smallmatrix}\right)} \cdot \left(\prod_{j\in[n]} g_2^{\hat{h}_j\cdot\sum_{i\in J} a_{\tau,i,j}\cdot\hat{r}_i}\right)^{\boldsymbol{Z}\cdot\left(\begin{smallmatrix}\mathbf{0}\\ 1\end{smallmatrix}\right)} \\
&\quad \cdot \prod_{j\in[n]}\left(g_2^{\boldsymbol{H}_j'^\top\cdot\boldsymbol{Z}\cdot\left(\begin{smallmatrix}\mathbf{0}\\ 1\end{smallmatrix}\right)\cdot\sum_{i\in J}\hat{r}_i\cdot a_{\tau,i,j}}\right) \\
&= g_2^{\boldsymbol{Z}\cdot\left(\begin{smallmatrix}\mathbf{0}\\ 1\end{smallmatrix}\right)\cdot\left(a_\tau\cdot\beta + \sum_{i\in[m_2]} a_{\tau,i}\cdot\hat{r}_i\right)} \cdot \prod_{j\in[n]}\left(g_2^{\boldsymbol{H}_j'^\top\cdot\boldsymbol{Z}\cdot\left(\begin{smallmatrix}\mathbf{0}\\ 1\end{smallmatrix}\right) + \boldsymbol{Z}\cdot\left(\begin{smallmatrix}\mathbf{0}\\ \hat{h}_j\end{smallmatrix}\right)}\right)^{\sum_{i\in J} a_{\tau,i,j}\cdot\hat{r}_i}
\end{aligned}$$

Algorithm 23: Continuation of Algorithm 22

Challenge (given cInd^***,** $m_0, m_1 \in \mathcal{M} = \mathbb{G}_\mathrm{T}$ **from** $\mathcal{A}$**)**

Choose $b \leftarrow \{0,1\}$, $\boldsymbol{s}_0 \leftarrow \mathbb{Z}_p^d$ compute

$$\mathrm{CT} = \left(\mathrm{cInd}^*, C_0, \overrightarrow{\mathbf{C}}, \_\right) \leftarrow \mathrm{Enc}\left(\mathrm{pp}_\kappa, \mathrm{cInd}^*, m_b; \boldsymbol{s}_0\right);$$

Query $\mathbb{G}_1^{w_1} \ni \widehat{\boldsymbol{C}}' = \left(\widehat{C}'_1, \ldots, \widehat{C}'_{w_1}\right) \leftarrow \mathcal{O}^2_{\mathrm{CMH},\nu,\hat{\alpha},\hat{\boldsymbol{h}}}\left(\mathrm{cInd}^*\right)$;

Compute $C_0 := C_0 \cdot \widehat{C}_0$, where

$$\widehat{C}_0 := \mathrm{e}\left(\left(\widehat{C}'_1\right)^{\boldsymbol{B}\cdot\binom{\mathbf{0}}{1}}, g_2^{\boldsymbol{\alpha}}\right) \ ;$$

For all $\tau \in [w_1]$ compute $\mathbf{C}_\tau := \mathbf{C}_\tau \cdot \widehat{\mathbf{C}}_\tau$, where

$$\widehat{\mathbf{C}}_\tau := \left(\widehat{C}'_\tau\right)^{\boldsymbol{B}\cdot\binom{\mathbf{0}}{1}} \cdot \prod_{i\in I}\left(\widehat{C}'_{\tau_i}\right)^{\sum_{j\in[n]} b_{\tau,i,j}\cdot \boldsymbol{H}'_j\cdot\boldsymbol{B}\cdot\binom{\mathbf{0}}{1}} \ ;$$

Compute $t := \mathrm{H}\left(\mathrm{cInd}, C_0, \overrightarrow{\mathbf{C}}\right)$ and then $\mathbf{C}'' = \left(\mathbf{U}^t \cdot \mathbf{V}\right)^{\boldsymbol{s}_0} \cdot \mathbf{X}$, where

$$\mathbf{X} := \left(\widehat{C}'_1\right)^{(t\cdot\boldsymbol{U}+\boldsymbol{V})\cdot\boldsymbol{B}\cdot\binom{\mathbf{0}}{1}}$$

Return $\mathrm{CT}^* = \left(\mathrm{cInd}^*, C_0, \overrightarrow{\mathbf{C}}, \mathbf{C}''\right)$.

Phase II

Realize the oracles using msk and $\mathbf{Q}$ as defined in the experiment;

Guess

Perform as defined in the experiments;

$$\overset{(7.18)}{=} \quad \widehat{\mathbf{G}}_2^{a_\tau\cdot\beta+\sum_{i\in[m_2]} a_{\tau,i}\cdot\hat{r}_i} \cdot \prod_{j\in[n]} \widehat{\mathbf{H}}_j^{\sum_{i\in J} a_{\tau,i,j}\cdot\hat{r}_i}$$

$$\overset{J}{=} \quad \widehat{\mathbf{G}}_2^{a_\tau\cdot\beta+\sum_{i\in[m_2]} a_{\tau,i}\cdot\hat{r}_i} \cdot \prod_{j\in[n]} \widehat{\mathbf{H}}_j^{\sum_{i\in[m_2]} a_{\tau,i,j}\cdot\hat{r}_i} \ .$$

This is exactly the form of the semi-functional components as defined in (6.30) on page 150. Namely, if $\beta = \hat{\alpha} = 0$ (case $\nu = 0$) key number k is a semi-functional key of type 1 and otherwise (case $\nu = 1$) this key is semi-functional of type 2.

Next in Algorithm 23 we show how to generate the semi-functional ciphertext using the second oracle given in CMH security experiment.

Given $m_0, m_1 \in \mathcal{M}$ the challenger computes a normal ciphertext for m_b (except for $\mathbf{C}''$) and than asks the oracle for cInd^*. According to the CMH experiment $\mathcal{B}$ receives

$$\widehat{\boldsymbol{C}}' = g_1^{\boldsymbol{c}\left(\hat{s},\hat{\boldsymbol{s}},\hat{\boldsymbol{h}}\right)} \ ,$$

where $(\boldsymbol{c}, w_2) := \mathrm{Enc2}\left(\kappa, \mathrm{cInd}^*\right)$, and $\hat{s} \in \mathbb{Z}_p$ and $\hat{\boldsymbol{s}} \in \mathbb{Z}_p^{w_2}$ are chosen uniformly at random. Hence, for all $\tau \in [w_1]$ it holds

$$\widehat{C}'_\tau = g_1^{\sum_{i\in[w_2]_0} b_{\tau,i}\cdot\hat{s}_i + \sum_{i\in I}\sum_{j\in[n]} b_{\tau,i,j}\cdot\hat{h}_j\cdot\hat{s}_i} \ ,$$

In particular, $\widehat{C}'_1 = g_1^{\hat{s}_0}$ due to the regularity of the pair encoding and for all $i \in I$ it holds $\widehat{C}'_{\tau_i} = g_1^{\hat{s}_i}$. Let us consider the elements $\widehat{C}_0$, $\left\{\widehat{\mathbf{C}}_\tau\right\}_{\tau\in[w_1]}$, and $\mathbf{C}''$ computed by $\mathcal{B}$:

- Element $\widehat{C}_0 = \widehat{\mathbf{Y}}^{\hat{s}_0} \in \mathbb{G}_\mathrm{T}$ is computed from $C'_1 = g_1^{\hat{s}_0}$ and is as follows

$$\begin{aligned}
\widehat{C}_0 &\overset{\text{by const.}}{=} \mathrm{e}\left(\left(\widehat{C}'_1\right)^{\boldsymbol{B}\cdot\binom{\mathbf{0}}{1}}, g_2^{\boldsymbol{\alpha}}\right) \\
&= \mathrm{e}\left(\left(g_1^{\hat{s}_0}\right)^{\boldsymbol{B}\cdot\binom{\mathbf{0}}{1}}, g_2^{\boldsymbol{\alpha}}\right) \\
&= \mathrm{e}\left(\widehat{\mathbf{G}}_1, g_2^{\boldsymbol{\alpha}}\right)^{\hat{s}_0} \\
&\overset{\text{by def.}}{=} \widehat{\mathbf{Y}}^{\hat{s}_0}
\end{aligned}$$

- Elements $\widehat{\mathbf{C}}_\tau \in \mathbb{G}_1^{d+1}$ for all $\tau \in [w_1]$ are as follows

$$\begin{aligned}
\widehat{\mathbf{C}}_\tau &\overset{\text{by const.}}{=} \left(\widehat{C}'_\tau\right)^{\boldsymbol{B}\cdot\binom{\mathbf{0}}{1}} \cdot \prod_{i\in I}\left(\widehat{C}'_{\tau_i}\right)^{\sum_{j\in[n]} b_{\tau,i,j}\cdot\boldsymbol{H}'_j\cdot\boldsymbol{B}\cdot\binom{\mathbf{0}}{1}} \\
&= \left(g_1^{\sum_{i\in[w_2]_0} b_{\tau,i}\cdot\hat{s}_i + \sum_{j\in[n]}\sum_{i\in I} b_{\tau,i,j}\cdot\hat{s}_i\cdot\hat{h}_j}\right)^{\boldsymbol{B}\cdot\binom{\mathbf{0}}{1}} \cdot \prod_{i\in I}\left(g_1^{\hat{s}_i}\right)^{\sum_{j\in[n]} b_{\tau,i,j}\cdot\boldsymbol{H}'_j\cdot\boldsymbol{B}\cdot\binom{\mathbf{0}}{1}} \\
&= \left(g_1^{\sum_{i\in[w_2]_0} b_{\tau,i}\cdot\hat{s}_i}\right)^{\boldsymbol{B}\cdot\binom{\mathbf{0}}{1}} \cdot \left(\prod_{j\in[n]} g_1^{\hat{h}_j\cdot\sum_{i\in I} b_{\tau,i,j}\cdot\hat{s}_i}\right)^{\boldsymbol{B}\cdot\binom{\mathbf{0}}{1}} \\
&\quad \cdot \prod_{j\in[n]}\left(g_1^{\boldsymbol{H}'_j\cdot\boldsymbol{B}\cdot\binom{\mathbf{0}}{1}}\right)^{\sum_{i\in I}\hat{s}_i\cdot b_{\tau,i,j}} \\
&= \left(g_1^{\boldsymbol{B}\cdot\binom{\mathbf{0}}{1}}\right)^{\sum_{i\in[w_2]_0} b_{\tau,i}\cdot\hat{s}_i} \cdot \prod_{j\in[n]}\left(g_1^{\boldsymbol{H}'_j\cdot\boldsymbol{B}\cdot\binom{\mathbf{0}}{1}+\boldsymbol{B}\cdot\binom{\mathbf{0}}{\hat{h}_j}}\right)^{\sum_{i\in I} b_{\tau,i,j}\cdot\hat{s}_i} \\
&\overset{(7.19)}{=} \widehat{\mathbf{G}}_1^{\sum_{i\in[w_2]_0} b_{\tau,i}\cdot\hat{s}_i} \cdot \prod_{j\in[n]} \widehat{\mathbf{H}}_j^{\sum_{i\in I} b_{\tau,i,j}\cdot\hat{s}_i} \quad .
\end{aligned}$$

- C'' is computed using $t := \mathrm{H}\left(\mathrm{cInd}, C_0, \overrightarrow{\mathbf{C}}\right) \in \mathbb{Z}_p$ (which itself is computed using updated values C_0 and $\overrightarrow{\mathbf{C}}$) and $\widehat{C}'_1$ and is as follows. We consider only the semi-functional component, since the normal component is computed as defined in the algorithm Enc.

$$\begin{aligned}
\mathbf{X} = \left(\widehat{C}'_1\right)^{(t\cdot\boldsymbol{U}+\boldsymbol{V})\cdot\boldsymbol{B}\cdot\binom{\mathbf{0}}{1}} &= g_1^{\hat{s}_0\cdot(t\cdot\boldsymbol{U}+\boldsymbol{V})\cdot\boldsymbol{B}\cdot\binom{\mathbf{0}}{1}} \\
&= \left({}^{\boldsymbol{U}}\widehat{\mathbf{G}}_1^{t} \cdot {}^{\boldsymbol{V}}\widehat{\mathbf{G}}_1\right)^{\hat{s}_0} \\
&= \left(\widehat{\mathbf{U}}^t \cdot \widehat{\mathbf{V}}\right)^{\hat{s}_0} \quad .
\end{aligned}$$

We deduce that CT^* is a correctly generated semi-functional ciphertext of m_b according to definition of SFEnc on page 151.

All together the view of $\mathcal{A}$ is as defined in game $\mathrm{G}_{k,1}$ if $\nu = 0$ and it is as defined in game $\mathrm{G}_{k,2}$ if $\nu = 1$. In the guess phase $\mathcal{B}$ outputs 1 if and only if the event HashAbort did not occurred and $\mathcal{A}$ outputs the correct bit – that is, if and only if the output of the corresponding experiment is equal one. Hence, for every $\mathcal{A} \in \mathbf{A}_\Pi$ and every $\mathrm{des} \in \Omega$ there is $\mathcal{B}$ as defined in Algorithm 22 such that it holds

$$\begin{aligned}\mathrm{Adv}^{\mathrm{CMH}}_{\mathcal{B}}(\lambda) &\overset{\text{by def.}}{=} \left|\Pr\left[\mathrm{Exp}^{\mathrm{CMH}}_{\mathrm{P},\mathcal{G},0,\mathcal{A}}(\lambda,\mathrm{des})=1\right]-\Pr\left[\mathrm{Exp}^{\mathrm{CMH}}_{\mathrm{P},\mathcal{G},1,\mathcal{A}}(\lambda,\mathrm{des})=1\right]\right| \\ &= \left|\mathrm{Adv}^{\mathrm{G}_{k,1}}_{\Pi,\mathcal{A}}(\lambda,\mathrm{des})-\mathrm{Adv}^{\mathrm{G}_{k,2}}_{\Pi,\mathcal{A}}(\lambda,\mathrm{des})\right| .\end{aligned}$$

This finally proves the lemma. □

From $\mathrm{G}_{k,2}$ to $\mathrm{G}_{k,3}$

Let $k \in [q_1]$. In $\mathrm{G}_{k,2}$ the first $k-1$ keys in Phase 1 are semi-functional of type 3, key k is of type 2 and all other keys are normal. The experiment $\mathrm{G}_{i,3}$ is the same except for key k which is of semi-functional of type 3.

Lemma 7.15 (cf. Lemma 8 in [Att15])**.** *For every $k \in [q_1]$, every* $\mathrm{des} \in \Omega$*, every $\mathcal{A} \in \mathbf{A}_\Pi$ there exists a ppt algorithm $\mathcal{B}$ such that*

$$\mathrm{Adv}^{\mathcal{D}_d-\mathrm{mDH}}_{\mathcal{B}}(\lambda) = \left|\mathrm{Adv}^{\mathrm{G}_{k,2}}_{\Pi,\mathcal{A}}(\lambda,\mathrm{des})-\mathrm{Adv}^{\mathrm{G}_{k,3}}_{\Pi,\mathcal{A}}(\lambda,\mathrm{des})\right| .$$

Proof. Algorithm $\mathcal{B}$, presented in Algorithm 24 is almost the same as Algorithm 21 except for the generation of key number k.

Algorithm 24: $\mathcal{B}$ against Assumption 6.1

Input : $(\mathrm{des}, \mathbb{GD}, \mathbf{T}, \mathbf{Z})$.

Require: $\mathrm{des} \in \Omega$, $\mathbb{GD} = (p, (g_1, \mathbb{G}_1), (g_2, \mathbb{G}_2), \mathbb{G}_\mathrm{T}, \mathrm{e}) \in \left[\mathcal{G}\left(1^\lambda\right)\right]$, $\mathbf{T} \in \mathbb{G}_2^{(d+1)\times(d+1)}$, $\mathbf{Z} \in \mathbb{G}_2^{d+1}$.

Setup

...

Phase I

...

Open (j)**:**

...

case $j = k$ **do**

Choose $\beta_k \leftarrow \mathbb{Z}_p$, compute $(\boldsymbol{k}, m_2) := \mathrm{Enc1}(\kappa, \mathrm{kInd})$.

Compute $\mathbf{R} = (\mathbf{R}_1, \ldots, \mathbf{R}_{m_2}) \in \mathbb{G}_2^{(d+1)\times(m_2)}$ by

$$\mathbf{R} \leftarrow \mathrm{ReRand}\left(\mathbb{G}_2, {}^{\tilde{\boldsymbol{B}}^{-\top}}\mathbf{T}, {}^{\tilde{\boldsymbol{B}}^{-\top}}\mathbf{Z}, m_2\right) ;$$

Compute $\overrightarrow{\mathbf{K}}' \leftarrow \mathrm{KeyGenAlt}\left(\boldsymbol{k}, \mathrm{msk}, \mathbf{R}, \overline{\mathbb{H}}\right)$.

Return $\mathrm{sk} = \left(\mathrm{kInd}, \overrightarrow{\mathbf{K}}' \cdot \widehat{\mathbf{G}}_2^{a_\tau \cdot \beta_k}\right)$, where a_τ is defined by $\boldsymbol{k}$.

...

...

Challenge (given cInd^***,** $m_0, m_1 \in \mathcal{M} = \mathbb{G}_\mathrm{T}$ **from** $\mathcal{A}$**)**

...

Phase II

...

Recall (6.30) from page 150. By definition, semi-functional key of type 3 is a normal key with additional factor $\widehat{\mathbf{G}}_2^{a_\tau \cdot \beta_k}$, whereas semi-functional key of type 2 is a semi-functional key of type 1 with the same additional factor $\widehat{\mathbf{G}}_2^{a_\tau \cdot \beta_k}$. Hence, by construction (cf. Line 11) and using the analysis of Lemma 7.13 the statement of the lemma holds. □

From $\mathrm{G}_{q_1,3}$ to G_{q_1+1}

In $\mathrm{G}_{q_1,3}$ the keys in Phase 1 are semi-functional of type 3 and the keys in Phase 2 are normal. The experiment G_{q_1+1} is the same except for the keys in Phase 2 which are semi-functional of type 1.

Lemma 7.16 (cf. Lemma 9 in [Att15])**.** *For every every* $\mathrm{des} \in \Omega$ *and every* $\mathcal{A} \in \mathbf{A}_\Pi$ *there exists a ppt algorithm* $\mathcal{B}$ *such that*

$$\mathrm{Adv}_{\mathcal{B}}^{\mathcal{D}_d-\mathrm{mDH}}(\lambda) = \left|\mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{q_1,3}}(\lambda, \mathrm{des}) - \mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{q_1+1}}(\lambda, \mathrm{des})\right| .$$

Proof. Algorithm $\mathcal{B}$ simulates the setup phase in the same way as Algorithm 21 which in turn extends the simulation of Algorithm 19. The semi-functional keys of type 3 for the first query phase are computed in the same way as in Algorithm 21. The semi-functional challenge is computed in the same way as in Algorithm 19. All the keys in the second query phase are computed as key k in Algorithm 21 and are all correctly distributed semi-functional keys of type 1 or normal keys depending on the challenge of $\mathcal{B}$. This already proves the lemma. □

From G_{q_1+1} to G_{q_1+2}

In G_{q_1+1} all keys in Phase 2 are semi-functional of type 1, whereas in G_{q_1+2} these keys are semi-functional of type 2.

The following lemma extensively use the parameter hiding lemma.

Lemma 7.17 (cf. Lemma 10 in [Att15])**.** *For every* $\mathcal{A} \in \mathbf{A}_\Pi$ *and every* $\mathrm{des} \in \Omega$ *there exists a ppt algorithm* $\mathcal{B}$ *such that*

$$\mathrm{Adv}_{\mathcal{B}}^{\mathrm{SMH}}(\lambda) = \left|\mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{q_1+1}}(\lambda, \mathrm{des}) - \mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{q_1+2}}(\lambda, \mathrm{des})\right| .$$

Proof. $\mathcal{B}$ against SMH security property of P is similar to Algorithm 22 and is presented in Algorithm 25. The setup phase is exactly the same. In the first query phase all keys are semi-functional of type 3 and are generated in the same way as semi-functional keys of this type in Algorithm 22. Hence, we directly start with the challenge phase.

The challenge is exactly the same as in Algorithm 22 since the first oracle in the SMH experiment and the second oracle in the CMH experiment are equal.

The keys in the second query phase are also generated as in Algorithm 22. However, whereas in this algorithm only key number k is generated using the oracle, here all keys in Phase 2 are generated using the corresponding oracle, which can be queried polynomially many times by definition of SMH. All these keys are (independent) semi-functional keys of type 1 or of type 2. In the second case all random elements β of these keys are the same as required by the definition of G_{q_1+2}. This finalizes the proof. □

Algorithm 25: $\mathcal{B}$ against SMH security property of P

Input : $\mathbb{GD}$, des,
Require: $\mathbb{GD} = (p, (g_1, \mathbb{G}_1), (g_2, \mathbb{G}_2), \mathbb{G}_T, e)$, des $\in \Omega$

Setup
  ...
Phase I
  ...
Challenge (given cInd***,** $m_0, m_1 \in \mathcal{M} = \mathbb{G}_T$ **from** $\mathcal{A}$**)**
  Query $\widehat{C}' \leftarrow \mathcal{O}^1_{\mathrm{SMH},\nu,\hat{\alpha},\hat{\boldsymbol{h}}}(\mathrm{cInd}^*)$;
  Choose $b \leftarrow \{0,1\}$, $\boldsymbol{s}_0 \leftarrow \mathbb{Z}_p^d$ compute
  $\mathrm{CT} = \left(\mathrm{cInd}, C_0, \overrightarrow{\mathbf{C}}, \_\right) \leftarrow \mathrm{Enc}(\mathrm{pp}_\kappa, \mathrm{cInd}^*, m_b; \boldsymbol{s}_0)$;
  Compute $C_0 := C_0 \cdot \widehat{C}_0$, where

$$\widehat{C}_0 := e\left(\left(C'_1\right)^{\boldsymbol{B}\cdot\binom{\mathbf{0}}{1}}, g_2^{\boldsymbol{\alpha}}\right) \ ;$$

  For all $\tau \in [w_1]$ compute $\mathbf{C}_\tau := \mathbf{C}_\tau \cdot \widehat{\mathbf{C}}_\tau$, where

$$\widehat{\mathbf{C}}_\tau := \left(C'_\tau\right)^{\boldsymbol{B}\cdot\binom{\mathbf{0}}{1}} \cdot \prod_{i\in I} \left(C'_{\tau_i}\right)^{\sum_{j\in[n]} b_{\tau,i,j}\cdot \boldsymbol{H}'_j\cdot\boldsymbol{B}\cdot\binom{\mathbf{0}}{1}} \ ;$$

  Compute $t := \mathrm{H}\left(\mathrm{cInd}, C_0, \overrightarrow{\mathbf{C}}\right)$ and then $\mathbf{C}'' = \left(\mathbf{U}^t \cdot \mathbf{V}\right)^{\boldsymbol{s}_0} \cdot \left(\widehat{\mathbf{U}}^t \cdot \widehat{\mathbf{V}}\right)^{\hat{s}_0}$, where

$$\left(\widehat{\mathbf{U}}^t \cdot \widehat{\mathbf{V}}\right)^{\hat{s}_0} := \left(C'_1\right)^{(t\cdot\boldsymbol{U}+\boldsymbol{V})\cdot\boldsymbol{B}\cdot\binom{\mathbf{0}}{1}}$$

From G_{q_1+2} **to** G_{q_1+3}

In G_{q_1+2} the keys Phase 2 are semi-functional of type 2. The experiment G_{q_1+3} is the same except for the keys in Phase 2 which are semi-functional of type 3.

Lemma 7.18 (cf. Lemma 11 in [Att15])**.** *For every every* des $\in \Omega$ *and every* $\mathcal{A} \in \mathbf{A}_\Pi$ *there exists a ppt algorithm* $\mathcal{B}$ *such that*

$$\mathrm{Adv}^{\mathcal{D}_d-\mathrm{mDH}}_{\mathcal{B}}(\lambda) = \left|\mathrm{Adv}^{\mathrm{G}_{q_1+2}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) - \mathrm{Adv}^{\mathrm{G}_{q_1+3}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})\right| \ .$$

Proof. $\mathcal{B}$ is the same as in the proof of Lemma 7.16 except for keys in Phase 2. All keys in the second query phase are generated as key number k in the proof of Lemma 7.15. Only the random element β_k is chosen ones and used for all the keys as required. All other random elements for these keys are independent due to the choice of algorithm ReRand from Lemma 6.9. This proves the lemma. □

7.2.3. Final Steps

In this subsection we present the last reduction step and the final analysis of the main theorem.

Algorithm 26: Continuation of Algorithm 25

Phase II

CKGen (kInd_i) **with** $\mathrm{kInd}_i \in \mathbb{X}_\kappa$**:**

Store (i, kInd_i);

Open (i)**:**

Compute a normal secret key $\left(\mathrm{kInd}, \overrightarrow{\mathbf{K}}\right) = \mathrm{sk} \leftarrow \mathrm{KeyGen}\left(\mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{kInd}_i\right)$;

Query $\mathbb{G}_2^{m_1} \ni \widehat{\mathbf{K}}' = \left(\widehat{K}'_1, \ldots, \widehat{K}'_{m_1}\right) \leftarrow \mathcal{O}^2_{\mathrm{CMH},\nu,\hat{\alpha},\hat{\boldsymbol{h}}}\left(\mathrm{kInd}_i\right)$;

For all $\tau \in [m_1]$ compute $\mathbf{K}_\tau := \mathbf{K}_\tau \cdot \widehat{\mathbf{K}}_\tau$, where

$$\widehat{\mathbf{K}}_\tau := \left(\widehat{K}'_\tau\right)^{\boldsymbol{Z}\cdot\left(\begin{smallmatrix}\mathbf{0}\\1\end{smallmatrix}\right)} \cdot \prod_{j\in[n]} \prod_{i\in J} \left(\widehat{K}'_{\tau_i}\right)^{a_{\tau,i,j}\cdot \boldsymbol{H}'^{\top}_j \cdot \boldsymbol{Z}\cdot\left(\begin{smallmatrix}\mathbf{0}\\1\end{smallmatrix}\right)} .$$

(recall $\boldsymbol{Z} = \boldsymbol{B}^{-\top} \cdot \left(\begin{smallmatrix}\tilde{\boldsymbol{D}} & \mathbf{0}\\ \mathbf{0} & 1\end{smallmatrix}\right)$);

Return the modified key $\mathrm{sk} = \left(\mathrm{kInd}, \overrightarrow{\mathbf{K}}\right)$;

Dec (CT, i)**:**

Realize the decryption oracles as defined in the experiment using $\mathbf{Q}$;

Guess

Perform as defined in the experiments;

Final Step $\mathrm{G}'_{q_1+3} \to \mathrm{G}_{\mathrm{Final}}$

In the experiment G'_{q_1+3} the challenge ciphertext is semi-functional and all keys are semi-functional of type 3. Experiment $\mathrm{G}_{\mathrm{Final}}$ is the same as G'_{q_1+3} except for the challenge ciphertext, which is a semi-functional ciphertext of message $m \in \mathbb{G}_\mathrm{T}$ chosen uniformly and independently at random. All decryption queries in both experiments are answered by $C_0 \cdot \mathrm{e}\left(\mathbf{C}_1, \mathrm{msk} \cdot \widehat{\mathbf{G}}_2^\sigma\right)^{-1}$, where $\sigma \leftarrow \mathbb{Z}_p$ is chosen once in the setup phase. We remark that our method to answer the decryption queries without the user secret keys is exploited in the proof of the following lemma.

Lemma 7.19 (cf. Lemma 12 in [Att15])**.** *Experiments* G'_{q_1+3} *and* $\mathrm{G}_{\mathrm{Final}}$ *are (almost) identical. In particular, for every* $\mathcal{A} \in \mathbf{A}_\Pi$ *it holds*

$$\left|\mathrm{Adv}^{\mathrm{G}'_{q_1+3}}_{\mathcal{A},\Pi}(\mathrm{des}, \lambda) - \mathrm{Adv}^{\mathrm{G}_{\mathrm{Final}}}_{\mathcal{A},\Pi}(\mathrm{des}, \lambda)\right| \leq \frac{1}{2^{\lambda-1}} .$$

Proof. In order to prove the lemma we define three additional probability distributions $\widetilde{\mathrm{G}}'_{q_1+3}$, $\widehat{\mathrm{G}}_{\mathrm{Final}}$, and $\widetilde{\mathrm{G}}_{\mathrm{Final}}$ and prove that it holds

$$\mathrm{G}'_{q_1+3} \equiv \widetilde{\mathrm{G}}'_{q_1+3} \equiv \widehat{\mathrm{G}}_{\mathrm{Final}} \approx \widetilde{\mathrm{G}}_{\mathrm{Final}} \equiv \mathrm{G}_{\mathrm{Final}} ,$$

where $\equiv$ denotes equal distributions and $\approx$ statistically close distributions.

First of all, $\mathrm{G}_{\mathrm{Final}}$ can be alternatively seen as G'_{q_1+3}, where $C_0 \in \mathrm{CT}^*$ is additionally multiplied by $\mathrm{e}(g_1, g_2)^u$, where $u \in \mathbb{Z}_p$ is chosen uniformly and independently at random. We denote this probability experiment by $\widetilde{\mathrm{G}}_{\mathrm{Final}}$. Obviously, it holds $\widetilde{\mathrm{G}}_{\mathrm{Final}} \equiv \mathrm{G}_{\mathrm{Final}}$ since CT^* in $\widetilde{\mathrm{G}}_{\mathrm{Final}}$ corresponds to the encryption of $m = m_b \cdot \mathrm{e}(g_1, g_2)^u \in \mathbb{G}_\mathrm{T}$, which is uniformly distributed message due to the choice of u.

Next, we define $\widehat{\mathrm{G}}_{\mathrm{Final}}$ to be the same as $\widetilde{\mathrm{G}}_{\mathrm{Final}}$ except for the choice of u, which is defined to be $u := \hat{s}_0 \cdot u'$, where $u' \leftarrow \mathbb{Z}_p$ and $\hat{s}_0$ is the semi-functional exponent defined by $\mathbf{C}_1 \in \mathrm{CT}^*$.

By definition of SFEnc $\hat{s}_0$ is chosen uniformly at random from $\mathbb{Z}_p$. Statistical distance between $\widetilde{\mathrm{G}}_{\mathrm{Final}}$ and $\widehat{\mathrm{G}}_{\mathrm{Final}}$ is equal to the statistical distance between probability distribution $\mathcal{U}_{\mathbb{Z}_p} \times \mathcal{U}_{\mathbb{Z}_p}$ and probability distribution on $\mathbb{Z}_p^2$, where the first element x is chosen uniformly at random and the second element y is set to $x \cdot z$ for uniform z. In turn, statistical distance between the latter two distributions is equal $\frac{p-1}{p^2} < \frac{1}{p}$. Furthermore, by our convention $|p| = \lambda$, and hence $\frac{1}{p} \leq \frac{1}{2^{\lambda-1}}$. Hence, it holds for every $\mathcal{A} \in \mathbf{A}_\Pi$

$$\left|\mathrm{Adv}^{\widehat{\mathrm{G}}_{\mathrm{Final}}}_{\mathcal{A},\Pi}(\mathrm{des},\lambda) - \mathrm{Adv}^{\widetilde{\mathrm{G}}_{\mathrm{Final}}}_{\mathcal{A},\Pi}(\mathrm{des},\lambda)\right| \leq \frac{1}{2^{\lambda-1}} \ .$$

Next, let us consider the setup phase of G'_{q_1+3} and especially the distribution regarding the master secret. By the definition of the setup algorithm it holds $\boldsymbol{Z} \in \mathbb{GL}_{p,d+1}$ – that is, $\boldsymbol{Z}$ is invertible. Let $\boldsymbol{\alpha} \in \mathbb{Z}_p^{d+1}$ be the exponent of the master secret key – that is, $\mathrm{msk} = g_2^{\boldsymbol{\alpha}}$. Define the following vector

$$\begin{pmatrix}\boldsymbol{\delta}\\ \hat{\delta}\end{pmatrix} := \boldsymbol{Z}^{-1} \cdot \boldsymbol{\alpha} \ ,$$

where $\boldsymbol{\delta} \in \mathbb{Z}_p^d$ and $\hat{\delta} \in \mathbb{Z}_p$. Then, $\boldsymbol{\alpha} = \boldsymbol{Z} \cdot \begin{pmatrix}\boldsymbol{\delta}\\ \hat{\delta}\end{pmatrix} = \boldsymbol{Z} \cdot \begin{pmatrix}\boldsymbol{I}_d\\ \boldsymbol{0}\end{pmatrix} \cdot \boldsymbol{\delta} + \boldsymbol{Z} \cdot \begin{pmatrix}\boldsymbol{0}\\ 1\end{pmatrix} \cdot \hat{\delta}$. That is

$$\begin{aligned} g_2^{\boldsymbol{\alpha}} &= g_2^{\boldsymbol{Z}\cdot\begin{pmatrix}\boldsymbol{I}_d\\ \boldsymbol{0}\end{pmatrix}\cdot\boldsymbol{\delta}} \cdot g_2^{\boldsymbol{Z}\cdot\begin{pmatrix}\boldsymbol{0}\\ 1\end{pmatrix}\cdot\hat{\delta}} \\ &= \mathbf{G}_2^{\boldsymbol{\delta}} \cdot \widehat{\mathbf{G}}_2^{\hat{\delta}} \ . \end{aligned}$$

Hence, the random choice of $\boldsymbol{\alpha}$ can be alternatively seen as the random choice of $\boldsymbol{\delta} \leftarrow \mathbb{Z}_p^d$, $\hat{\delta} \leftarrow \mathbb{Z}_p$. Consider the probability experiment which is obtained from G'_{q_1+3} when $\hat{\delta}$ is determined by $\hat{\delta}', u' \leftarrow \mathbb{Z}_p$, and $\hat{\delta} := \hat{\delta}' + u'$. We denote this probability experiment by $\widetilde{\mathrm{G}}'_{q_1+3}$. Obviously, $\mathrm{G}'_{q_1+3} \equiv \widetilde{\mathrm{G}}'_{q_1+3}$ since $\hat{\delta}$ is still uniformly distributed.

Next, we prove that $\widetilde{\mathrm{G}}'_{q_1+3}$ and $\widehat{\mathrm{G}}_{\mathrm{Final}}$ are equal. Let $\mathcal{A} \in \mathbf{A}_\Pi$ be arbitrary not necessarily ppt adversary. Let us consider the view of $\mathcal{A}$ in G'_{q_1+3} with $\hat{\delta} = \hat{\delta}'$ and $\hat{\delta} = \hat{\delta}' + u'$. The main observation is that the view of $\mathcal{A}$ is independent of $\hat{\delta}$ except for $C_0 \in \mathrm{CT}^*$ in both experiments.

Claim. The view of $\mathcal{A}$ in G'_{q_1+3} is independent of the value $\hat{\delta}$ except for $C_0 \in \mathrm{CT}^*$.

Proof. Let us consider the elements which depend on msk (that is on $\boldsymbol{\alpha}$), and hence potentially on $\hat{\delta}$.

1. Public parameters: The value $\mathbf{Y} \in \mathrm{pp}_\kappa$ is as follows

$$\begin{aligned} \mathbf{Y} &\overset{\text{by def.}}{=} \mathrm{e}\left(\mathbf{G}_1, g_2^{\boldsymbol{\alpha}}\right) \\ &= \mathrm{e}\left(\mathbf{G}_1, \mathbf{G}_2^{\boldsymbol{\delta}} \cdot \widehat{\mathbf{G}}_2^{\hat{\delta}}\right) \\ &\overset{\text{Lemma 6.12}}{=} \mathrm{e}\left(\mathbf{G}_1, \mathbf{G}_2^{\boldsymbol{\delta}}\right) \ . \end{aligned}$$

2. Semi-functional keys:

- **Phase I**: By Corollary 6.16 the group elements of the semi-functional key number j in the first query phase are of the form:

$$\begin{aligned}\overrightarrow{\mathbf{K}}'_j &= g_2^{\boldsymbol{k}\left(\boldsymbol{\alpha}+\boldsymbol{Z}\cdot\binom{\mathbf{0}}{\beta_j},\boldsymbol{R}_j,\overline{\mathbb{H}}\right)} \\ &= g_2^{\boldsymbol{k}\left(\boldsymbol{Z}\cdot\binom{\boldsymbol{\delta}}{\hat{\delta}}+\boldsymbol{Z}\cdot\binom{\mathbf{0}}{\beta_j},\boldsymbol{R}_j,\overline{\mathbb{H}}\right)} \\ &= g_2^{\boldsymbol{k}\left(\boldsymbol{Z}\cdot\binom{\boldsymbol{\delta}}{\hat{\delta}+\beta_j},\boldsymbol{R}_j,\overline{\mathbb{H}}\right)},\end{aligned}$$

  where β_j is chosen uniformly at random from $\mathbb{Z}_p$. Hence, $\hat{\delta}+\beta_j$ is uniformly distributed over $\mathbb{Z}_p$ and independent of $\hat{\delta}$.

- **Phase II**: Analogously to the first phase, all keys in this phase are of the form

$$\overrightarrow{\mathbf{K}}'_j = g_2^{\boldsymbol{k}\left(\boldsymbol{Z}\cdot\binom{\boldsymbol{\delta}}{\hat{\delta}+\beta},\boldsymbol{R}_j,\overline{\mathbb{H}}\right)},$$

  where β is chosen uniformly at random from $\mathbb{Z}_p$. Hence, $\hat{\delta}+\beta$ is uniformly distributed over $\mathbb{Z}_p$ and independent of $\hat{\delta}$.

3. Decryption queries: Consider the elements in $\mathrm{msk}\cdot\widehat{\mathbf{G}}_2^{\sigma}$ used for decryption

$$\begin{aligned}\mathrm{msk}\cdot\widehat{\mathbf{G}}_2^{\sigma} &= \mathbf{G}_2^{\boldsymbol{\delta}}\cdot\widehat{\mathbf{G}}_2^{\hat{\delta}}\cdot\widehat{\mathbf{G}}_2^{\sigma} \\ &= \mathbf{G}_2^{\boldsymbol{\delta}}\cdot\widehat{\mathbf{G}}_2^{\hat{\delta}+\sigma},\end{aligned}$$

   where σ is chosen uniformly at random from $\mathbb{Z}_p$. Hence, $\hat{\delta}+\sigma$ is uniformly distributed over $\mathbb{Z}_p$ by the choice of σ and in particular independent of $\hat{\delta}$.

This proves the claim, since all other elements in the experiment except for C_0 are independent of $\boldsymbol{\alpha}$ by construction. □

Next, let us consider the value $C_0 \in \mathrm{CT}^*$ with respect to $\hat{\delta}$. First of all, consider the semi-functional parameter $\widehat{\mathbf{Y}} = \mathrm{e}\left(\widehat{\mathbf{G}}_1, g_2^{\boldsymbol{\alpha}}\right)$ used in the semi-functional ciphertext (is not given to $\mathcal{A}$):

$$\begin{aligned}\mathrm{e}\left(\widehat{\mathbf{G}}_1, g_2^{\boldsymbol{\alpha}}\right) &= \mathrm{e}\left(\widehat{\mathbf{G}}_1, \mathbf{G}_2^{\boldsymbol{\delta}}\cdot\widehat{\mathbf{G}}_2^{\hat{\delta}}\right) \\ &\overset{\text{Lemma 6.12}}{=} \mathrm{e}\left(\widehat{\mathbf{G}}_1, \widehat{\mathbf{G}}_2^{\hat{\delta}}\right) \\ &= \mathrm{e}\left(g_1, g_2\right)^{\hat{\delta}}.\end{aligned}$$

Hence, $C_0 \in \mathrm{CT}^*$ is as follows by definition

$$\begin{aligned}C_0 &\overset{\text{by def.}}{=} m_b\cdot\mathbf{Y}^{\boldsymbol{s}_0}\cdot\widehat{\mathbf{Y}}^{\hat{s}_0} \\ &= m_b\cdot\mathbf{Y}^{\boldsymbol{s}_0}\cdot\mathrm{e}\left(g_1, g_2\right)^{\hat{\delta}\cdot\hat{s}_0}.\end{aligned}$$

Hence, if $\hat{\delta}=\hat{\delta}'$ then $C_0 = m_b\cdot\mathbf{Y}^{\boldsymbol{s}_0}\cdot\mathrm{e}\left(g_1,g_2\right)^{\hat{\delta}'\cdot\hat{s}_0}$, whereas for $\hat{\delta}=\hat{\delta}'+u'$ we get $C_0 = \left(m_b\cdot\mathbf{Y}^{\boldsymbol{s}_0}\cdot\mathrm{e}\left(g_1,g_2\right)^{\hat{\delta}'\cdot\hat{s}_0}\right)\cdot\left(\mathrm{e}\left(g_1,g_2\right)^{\hat{s}_0}\right)^{u'}$. In the second case the view of $\mathcal{A}$ is as in $\widehat{\mathrm{G}}_{\mathrm{Final}}$ with $u=\hat{s}_0\cdot u'$. This finally proves the lemma. □

Final Analysis

In the last game $\mathrm{G}_{\mathrm{Final}}$ the adversary gets no information about the challenge bit, and hence its advantage is equal to zero.

Lemma 7.20. *For every algorithm* $\mathcal{A} \in \mathbf{A}_{\Pi}$ *and every* $\mathrm{des} \in \Omega$ *it holds*

$$\mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{\mathrm{Final}}}(\lambda, \mathrm{des}) = 0 \ .$$

Summing up all factors from the lemmata in this chapter we get

$$\begin{aligned}
\mathrm{Adv\text{-}aP\text{-}KEM}_{\Pi,\mathcal{A}}^{\mathrm{CCA2}}(\lambda, \mathrm{des}) &= \mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{\mathrm{Real}}}(\lambda, \mathrm{des}) - \mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{\mathrm{resH}}}(\lambda, \mathrm{des}) \\
&\quad + \mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{\mathrm{resH}}}(\lambda, \mathrm{des}) - \mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_0}(\lambda, \mathrm{des}) \\
&\quad + \dots \\
&\quad + \mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}'_{q_1+3}}(\lambda, \mathrm{des}) - \mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{\mathrm{Final}}}(\lambda, \mathrm{des}) \\
&\quad + \mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{\mathrm{Final}}}(\lambda, \mathrm{des}) \\
&\leq \mathrm{Adv}_{\mathcal{H},\mathcal{B}_1}^{\mathrm{CR}}(\lambda) + (5 + 2 \cdot q_1) \cdot \mathrm{Adv}_{\mathcal{B}_2}^{\mathcal{D}_d\text{-}\mathrm{mDH}}(\lambda) \\
&\quad + q_1 \cdot \mathrm{Adv}_{\mathcal{B}_3}^{\mathrm{CMH}}(\lambda) + \mathrm{Adv}_{\mathcal{B}_4}^{\mathrm{SMH}}(\lambda) + \frac{q_{\mathrm{dec}} + 1}{2^{\lambda-1}} \ .
\end{aligned}$$

This finally proves Theorem 7.1.

Conclusion and Future Work

Based both on previous results and on our results from Part I, we conclude that for PE the simpler indistinguishability definition is appropriate for CPA and CCA attack scenarios, as long as the key-revealing selective-opening attacks are not concerned. Under chosen-ciphertext attack we suggest to use the CK-model to handle user secret keys; while under chosen-plaintext attack simpler OU-model is appropriate. This suggestion holds for PE as well as for P-KEM. Finally, under CCA the SE-model is the most advisable in order to handle the no-challenge-decryption condition for both PE and P-KEM. An interesting open question is if there are practical PE schemes for which the possibility to abort in the first query phase introduced in our security definitions significantly affects the achieved security guarantees.

The main contribution of Part II of this thesis consists in the introduced techniques to extend CPA-secure predicate-encryption schemes in order to achieve CCA-security. Our framework presented in Part III shows how these techniques can be adapted to other contexts. The same holds also for our proof techniques, which combine CCA-security and the dual system encryption methodology [Wat09a, LW10]. Due to the enormous relevance of the latter for constructions of CPA-secure PE schemes we believe that our techniques can be used to achieve CCA-security for further schemes that are not covered by the pair encoding frameworks [Att14a, Att16]. As the first instance, CCA-secure variants of encoding frameworks from [Wee14, CGW15, AC16, AC17b] could be interesting. We also note that our consistency checks developed for composite-order groups and for prime-order groups bring new insights into generic transformations from verifiability property presented in [YAHK11, YAS$^+$12]. Our consistency checks in groups of composite order can be used in these transformations without any modifications; whereas our checks in prime-order groups do not satisfy the required properties. An interesting open question is if one can adapt the known transformation from verifiability property and/or our consistency checks in prime-order groups in order to achieve CHK-like transformation from verifiability for schemes in prime-order groups applicable to more predicate encryption schemes. However, due to the more involved extensions required in the CHK-like transformations, in comparison to our constructions, this approach probably will not lead to more efficient schemes.

In Part III of this thesis we presented efficient fully CCA-secure PE schemes from computationally secure regular pair encoding schemes for the corresponding predicates. Although we do not present a transformation from any CPA-secure PE scheme into a CCA-secure scheme, our transformation lead to a variety of PE schemes for sophisticated predicates due to the pair encoding abstraction. In contrast, the known generic transformation do not lead to CCA-secure schemes for these predicates, due to the additional requirements of the transformations which significantly limit their applicability. Furthermore, a further essential advantage of our techniques is that the user secret keys do not have to be modified for every decryption as is the case in the generic transformations from delegatability. As already mentioned, this is a crucial property in practice if the user secret keys are stored in a cryptographic storage and the decryption has to be performed in a secured environment.

Finally, in the introduction to the second part of the thesis we mentioned that in the context of IBE there is a further technique to improve the efficiency of CCA-secure constructions. Namely, in [KV08] the authors used implicit rejection of malformed ciphertexts using hybrid construction and authenticated symmetric encryption. Considering our involved consistency checks it would be a great efficiency improvement if these checks could be dropped or at least simplified. However, in our opinion it is hardly feasible to apply this approach for general PE schemes, due to the more complex structure of the ciphertexts in PE schemes in comparison

to the IBE schemes. We believe that a combination of simpler consistency checks and implicit rejection techniques may lead to more efficient CCA-secure schemes.

Bibliography

[ABV+12] Shweta Agrawal, Xavier Boyen, Vinod Vaikuntanathan, Panagiotis Voulgaris, and Hoeteck Wee. Functional encryption for threshold functions (or fuzzy IBE) from lattices. In Fischlin et al. [FBM12], pages 280–297.

[AC16] Shashank Agrawal and Melissa Chase. A study of pair encodings: Predicate encryption in prime order groups. In Eyal Kushilevitz and Tal Malkin, editors, *Theory of Cryptography - 13th International Conference, TCC 2016-A, Tel Aviv, Israel, January 10-13, 2016, Proceedings, Part II*, volume 9563 of *Lecture Notes in Computer Science*, pages 259–288. Springer, 2016.

[AC17a] Shashank Agrawal and Melissa Chase. Simplifying design and analysis of complex predicate encryption schemes. Cryptology ePrint Archive, Report 2017/233, 2017.

[AC17b] Shashank Agrawal and Melissa Chase. Simplifying design and analysis of complex predicate encryption schemes. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part I*, volume 10210 of *Lecture Notes in Computer Science*, pages 627–656, 2017. Full version [AC17a].

[ACG+06] Nuttapong Attrapadung, Yang Cui, David Galindo, Goichiro Hanaoka, Ichiro Hasuo, Hideki Imai, Kanta Matsuura, Peng Yang, and Rui Zhang. Relations among notions of security for identity based encryption schemes. In José R. Correa, Alejandro Hevia, and Marcos A. Kiwi, editors, *LATIN 2006: Theoretical Informatics, 7th Latin American Symposium, Valdivia, Chile, March 20-24, 2006, Proceedings*, volume 3887 of *Lecture Notes in Computer Science*, pages 130–141. Springer, 2006.

[AGVW13] Shweta Agrawal, Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption: New perspectives and lower bounds. In Canetti and Garay [CG13], pages 500–518.

[AI09] Nuttapong Attrapadung and Hideki Imai. Dual-policy attribute based encryption. In Michel Abdalla, David Pointcheval, Pierre-Alain Fouque, and Damien Vergnaud, editors, *Applied Cryptography and Network Security, 7th International Conference, ACNS 2009, Paris-Rocquencourt, France, June 2-5, 2009. Proceedings*, volume 5536 of *Lecture Notes in Computer Science*, pages 168–185, 2009.

[AKW16] Shashank Agrawal, Venkata Koppula, and Brent Waters. Impossibility of simulation secure functional encryption even with random oracles. Cryptology ePrint Archive, Report 2016/959, 2016.

[AL10] Nuttapong Attrapadung and Benoît Libert. Functional encryption for inner product: Achieving constant-size ciphertexts with adaptive security or support for negation. In Phong Q. Nguyen and David Pointcheval, editors, *Public Key Cryptography - PKC 2010, 13th International Conference on Practice and Theory in Public Key Cryptography, Paris, France, May 26-28, 2010. Proceedings*, volume 6056 of *Lecture Notes in Computer Science*, pages 384–402. Springer, 2010.

[AL12] Nuttapong Attrapadung and Benoît Libert. Functional encryption for public-attribute inner products: Achieving constant-size ciphertexts with adaptive security or support for negation. *Journal of Mathematical Cryptology*, 5(2):115–158, 2012.

[ALdP11] Nuttapong Attrapadung, Benoît Libert, and Elie de Panafieu. Expressive key-policy attribute-based encryption with constant-size ciphertexts. In Catalano et al. [CFGN11], pages 90–108.

[Att14a] Nuttapong Attrapadung. Dual system encryption via doubly selective security: Framework, fully secure functional encryption for regular languages, and more. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, volume 8441 of *Lecture Notes in Computer Science*, pages 557–577. Springer, 2014. Full version [Att14b].

[Att14b] Nuttapong Attrapadung. Dual system encryption via doubly selective security: Framework, fully-secure functional encryption for regular languages, and more. Cryptology ePrint Archive, Report 2014/428, 2014.

[Att15] Nuttapong Attrapadung. Dual system encryption framework in prime-order groups. Cryptology ePrint Archive, Report 2015/390, 2015.

[Att16] Nuttapong Attrapadung. Dual system encryption framework in prime-order groups via computational pair encodings. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part II*, volume 10032 of *Lecture Notes in Computer Science*, pages 591–623, 2016. Full version [Att15].

[AY15] Nuttapong Attrapadung and Shota Yamada. Duality in ABE: converting attribute based encryption for dual predicate and dual policy via computational encodings. In Kaisa Nyberg, editor, *Topics in Cryptology - CT-RSA 2015, The Cryptographer's Track at the RSA Conference 2015, San Francisco, CA, USA, April 20-24, 2015. Proceedings*, volume 9048 of *Lecture Notes in Computer Science*, pages 87–105. Springer, 2015.

[BCHK07] Dan Boneh, Ran Canetti, Shai Halevi, and Jonathan Katz. Chosen-ciphertext security from identity-based encryption. *SIAM Journal on Computing*, 36(5):1301–1328, 2007.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. Springer, 2001.

[BF03] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. *SIAM Journal on Computing*, 32(3):586–615, 2003.

[BF13] Manuel Barbosa and Pooya Farshim. On the semantic security of functional encryption schemes. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *Public-Key Cryptography - PKC 2013 - 16th International Conference on Practice and Theory in Public-Key Cryptography, Nara, Japan, February 26 - March 1, 2013. Proceedings*, volume 7778 of *Lecture Notes in Computer Science*, pages 143–161. Springer, 2013.

[BH08] Dan Boneh and Michael Hamburg. Generalized identity based and broadcast encryption schemes. In Josef Pieprzyk, editor, *Advances in Cryptology - ASIACRYPT 2008, 14th International Conference on the Theory and Application of Cryptology and Information Security, Melbourne, Australia, December 7-11, 2008. Proceedings*, volume 5350 of *Lecture Notes in Computer Science*, pages 455–470. Springer, 2008.

[BHK15] Mihir Bellare, Dennis Hofheinz, and Eike Kiltz. Subtleties in the definition of IND-CCA: when and how should challenge decryption be disallowed? *Journal of Cryptology*, 28(1):29–48, 2015.

[BK05] Dan Boneh and Jonathan Katz. Improved efficiency for CCA-secure cryptosystems built using identity-based encryption. In Alfred Menezes, editor, *Topics in Cryptology - CT-RSA 2005, The Cryptographers' Track at the RSA Conference 2005, San Francisco, CA, USA, February 14-18, 2005, Proceedings*, volume 3376 of *Lecture Notes in Computer Science*, pages 87–103. Springer, 2005.

[BL13] Johannes Blömer and Gennadij Liske. Direct chosen-ciphertext secure attribute-based key encapsulations without random oracles. Cryptology ePrint Archive, Report 2013/646, 2013.

[BL14] Johannes Blömer and Gennadij Liske. Constructing CCA-secure predicate encapsulation schemes from CPA-secure schemes and universal one-way hash functions. Cryptology ePrint Archive, Report 2014/511, 2014.

[BL16a] Johannes Blömer and Gennadij Liske. Construction of fully CCA-secure predicate encryptions from pair encoding schemes. Cryptology ePrint Archive, Report 2016/206, 2016.

[BL16b] Johannes Blömer and Gennadij Liske. Construction of fully CCA-secure predicate encryptions from pair encoding schemes. In Kazue Sako, editor, *Topics in Cryptology - CT-RSA 2016 - The Cryptographers' Track at the RSA Conference 2016, San Francisco, CA, USA, February 29 - March 4, 2016, Proceedings*, volume 9610 of *Lecture Notes in Computer Science*, pages 431–447. Springer, 2016. Full version [BL16a].

[BL17] Johannes Blömer and Gennadij Liske. Subtleties in security definitions for predicate encryption with public index. Cryptology ePrint Archive, Report 2017/453, 2017.

[BMW05] Xavier Boyen, Qixiang Mei, and Brent Waters. Direct chosen ciphertext security from identity-based techniques. In Vijay Atluri, Catherine Meadows, and Ari Juels, editors, *Proceedings of the 12th ACM Conference on Computer and Communications Security, CCS 2005, Alexandria, VA, USA, November 7-11, 2005*, pages 320–329. ACM, 2005.

[BO13] Mihir Bellare and Adam O'Neill. Semantically-secure functional encryption: Possibility results, impossibility results and the quest for a general definition. In Michel Abdalla, Cristina Nita-Rotaru, and Ricardo Dahab, editors, *Cryptology and Network Security - 12th International Conference, CANS 2013, Paraty, Brazil, November 20-22. 2013. Proceedings*, volume 8257 of *Lecture Notes in Computer Science*, pages 218–234. Springer, 2013.

[BSW07] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *2007 IEEE Symposium on Security and Privacy (S&P 2007), 20-23 May 2007, Oakland, California, USA*, pages 321–334. IEEE Computer Society, 2007.

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In Yuval Ishai, editor, *Theory of Cryptography - 8th Theory of Cryptography Conference, TCC 2011, Providence, RI, USA, March 28-30, 2011. Proceedings*, volume 6597 of *Lecture Notes in Computer Science*, pages 253–273. Springer, 2011.

[CFGN11] Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors. *Public Key Cryptography - PKC 2011 - 14th International Conference on Practice and Theory in Public Key Cryptography, Taormina, Italy, March 6-9, 2011. Proceedings*, volume 6571 of *Lecture Notes in Computer Science*. Springer, 2011.

[CG13] Ran Canetti and Juan A. Garay, editors. *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, volume 8043 of *Lecture Notes in Computer Science*. Springer, 2013.

[CGW15] Jie Chen, Romain Gay, and Hoeteck Wee. Improved dual system ABE in prime-order groups via predicate encodings. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 595–624. Springer, 2015.

[CHK04] Ran Canetti, Shai Halevi, and Jonathan Katz. Chosen-ciphertext security from identity-based encryption. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, volume 3027 of *Lecture Notes in Computer Science*, pages 207–222. Springer, 2004.

[Coc01] Clifford Cocks. An identity based encryption scheme based on quadratic residues. In Bahram Honary, editor, *Cryptography and Coding, 8th IMA International Conference, Cirencester, UK, December 17-19, 2001, Proceedings*, volume 2260 of *Lecture Notes in Computer Science*, pages 360–363. Springer, 2001.

[Cra05] Ronald Cramer, editor. *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*. Springer, 2005.

[CS98] Ronald Cramer and Victor Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In Hugo Krawczyk, editor, *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*, volume 1462 of *Lecture Notes in Computer Science*, pages 13–25. Springer, 1998.

[CS02] Ronald Cramer and Victor Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In Lars R. Knudsen, editor, *Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28 - May 2, 2002, Proceedings*, volume 2332 of *Lecture Notes in Computer Science*, pages 45–64. Springer, 2002.

[CS03] Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, 33(1):167–226, 2003.

[CW13] Jie Chen and Hoeteck Wee. Fully, (almost) tightly secure IBE and dual system groups. In Canetti and Garay [CG13], pages 435–460.

[DDN03] Danny Dolev, Cynthia Dwork, and Moni Naor. Nonmalleable cryptography. *SIAM Review*, 45(4):727–784, 2003.

[DGKS10] Yvo Desmedt, Rosario Gennaro, Kaoru Kurosawa, and Victor Shoup. A new and improved paradigm for hybrid encryption secure against chosen-ciphertext attack. *Journal of Cryptology*, 23(1):91–120, 2010.

[EHK+15] Alex Escala, Gottfried Herold, Eike Kiltz, Carla Ràfols, and Jorge Villar. An algebraic framework for diffie–hellman assumptions. *Journal of Cryptology*, 30(1):242–288, oct 2015.

[FBM12] Marc Fischlin, Johannes A. Buchmann, and Mark Manulis, editors. *Public Key Cryptography - PKC 2012 - 15th International Conference on Practice and Theory in Public Key Cryptography, Darmstadt, Germany, May 21-23, 2012. Proceedings*, volume 7293 of *Lecture Notes in Computer Science*. Springer, 2012.

[FO13] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. *Journal of Cryptology*, 26(1):80–101, 2013.

[Fre10] David Mandell Freeman. Converting pairing-based cryptosystems from composite-order groups to prime-order groups. In Gilbert [Gil10], pages 44–61.

[Gen06] Craig Gentry. Practical identity-based encryption without random oracles. In Serge Vaudenay, editor, *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, volume 4004 of *Lecture Notes in Computer Science*, pages 445–464. Springer, 2006.

[Gil10] Henri Gilbert, editor. *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 - June 3, 2010. Proceedings*, volume 6110 of *Lecture Notes in Computer Science*. Springer, 2010.

[Gol04a] Oded Goldreich. *The Foundations of Cryptography - Volume I, Basic Tools*. Cambridge University Press, 2004.

[Gol04b] Oded Goldreich. *The Foundations of Cryptography - Volume II, Basic Applications*. Cambridge University Press, 2004.

[GPSW06] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, Ioctober 30 - November 3, 2006*, pages 89–98. ACM, 2006.

[Gui13] Aurore Guillevic. Comparing the pairing efficiency over composite-order and prime-order elliptic curves. In Michael J. Jacobson Jr., Michael E. Locasto, Payman Mohassel, and Reihaneh Safavi-Naini, editors, *Applied Cryptography and Network Security - 11th International Conference, ACNS 2013, Banff, AB, Canada, June 25-28, 2013. Proceedings*, volume 7954 of *Lecture Notes in Computer Science*, pages 357–372. Springer, 2013.

[Ham11] Mike Hamburg. Spatial encryption. Cryptology ePrint Archive, Report 2011/389, 2011.

[HHH+14] Gottfried Herold, Julia Hesse, Dennis Hofheinz, Carla Ràfols, and Andy Rupp. Polynomial spaces: A new framework for composite-to-prime-order transformations. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 261–279. Springer, 2014.

[KD04] Kaoru Kurosawa and Yvo Desmedt. A new paradigm of hybrid encryption scheme. In Matthew K. Franklin, editor, *Advances in Cryptology - CRYPTO 2004, 24th Annual International CryptologyConference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, volume 3152 of *Lecture Notes in Computer Science*, pages 426–442. Springer, 2004.

[KG06] Eike Kiltz and David Galindo. Direct chosen-ciphertext secure identity-based key encapsulation without random oracles. In Lynn Margaret Batten and Reihaneh Safavi-Naini, editors, *Information Security and Privacy, 11th Australasian Conference, ACISP 2006, Melbourne, Australia, July 3-5, 2006, Proceedings*, volume 4058 of *Lecture Notes in Computer Science*, pages 336–347. Springer, 2006.

[KG09] Eike Kiltz and David Galindo. Direct chosen-ciphertext secure identity-based key encapsulation without random oracles. *Theoretical Computer Science*, 410(47-49):5093–5111, 2009.

[KL15] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Cryptography and Network Security. Chapman and Hall/CRC Press, second edition edition, 2015.

[KM13] Neal Koblitz and Alfred Menezes. Another look at security definitions. *Adv. in Math. of Comm.*, 7(1):1–38, 2013.

[KV08] Eike Kiltz and Yevgeniy Vahlis. CCA2 secure IBE: standard model efficiency through authenticated symmetric encryption. In Tal Malkin, editor, *Topics in Cryptology - CT-RSA 2008, The Cryptographers' Track at the RSA Conference 2008, San Francisco, CA, USA, April 8-11, 2008. Proceedings*, volume 4964 of *Lecture Notes in Computer Science*, pages 221–238. Springer, 2008.

[Lew12] Allison B. Lewko. Tools for simulating features of composite order bilinear groups in the prime order setting. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, volume 7237 of *Lecture Notes in Computer Science*, pages 318–335. Springer, 2012.

[LOS+10] Allison B. Lewko, Tatsuaki Okamoto, Amit Sahai, Katsuyuki Takashima, and Brent Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In Gilbert [Gil10], pages 62–91.

[LW09] Allison Lewko and Brent Waters. New techniques for dual system encryption and fully secure HIBE with short ciphertexts. Cryptology ePrint Archive, Report 2009/482, 2009.

[LW10] Allison B. Lewko and Brent Waters. New techniques for dual system encryption and fully secure HIBE with short ciphertexts. In Daniele Micciancio, editor, *Theory of Cryptography, 7th Theory of Cryptography Conference, TCC 2010, Zurich, Switzerland, February 9-11, 2010. Proceedings*, volume 5978 of *Lecture Notes in Computer Science*, pages 455–479. Springer, 2010. Full version [LW09].

[LW11] Allison B. Lewko and Brent Waters. Unbounded HIBE and attribute-based encryption. In Kenneth G. Paterson, editor, *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, volume 6632 of *Lecture Notes in Computer Science*, pages 547–567. Springer, 2011.

[LW12] Allison B. Lewko and Brent Waters. New proof methods for attribute-based encryption: Achieving full security through selective techniques. In Safavi-Naini and Canetti [SC12], pages 180–198.

[NP15] Mridul Nandi and Tapas Pandit. Generic conversions from CPA to CCA secure functional encryption. Cryptology ePrint Archive, Report 2015/457, 2015.

[NY90] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In Harriet Ortiz, editor, *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 427–437. ACM, 1990.

[O'N10] Adam O'Neill. Definitional issues in functional encryption. Cryptology ePrint Archive, Report 2010/556, 2010.

[OSW07] Rafail Ostrovsky, Amit Sahai, and Brent Waters. Attribute-based encryption with non-monotonic access structures. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28-31, 2007*, pages 195–203. ACM, 2007.

[OT08] Tatsuaki Okamoto and Katsuyuki Takashima. Homomorphic encryption and signatures from vector decomposition. In Steven D. Galbraith and Kenneth G. Paterson, editors, *Pairing-Based Cryptography - Pairing 2008, Second International Conference, Egham, UK, September 1-3, 2008. Proceedings*, volume 5209 of *Lecture Notes in Computer Science*, pages 57–74. Springer, 2008.

[OT09] Tatsuaki Okamoto and Katsuyuki Takashima. Hierarchical predicate encryption for inner-products. In Mitsuru Matsui, editor, *Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings*, volume 5912 of *Lecture Notes in Computer Science*, pages 214–231. Springer, 2009.

[OT10a] Tatsuaki Okamoto and Katsuyuki Takashima. Fully secure functional encryption with general relations from the decisional linear assumption. In Tal Rabin, editor, *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, volume 6223 of *Lecture Notes in Computer Science*, pages 191–208. Springer, 2010. Full version [OT10b].

[OT10b] Tatsuaki Okamoto and Katsuyuki Takashima. Fully secure functional encryption with general relations from the decisional linear assumption. Cryptology ePrint Archive, Report 2010/563, 2010.

[OT11] Tatsuaki Okamoto and Katsuyuki Takashima. Achieving short ciphertexts or short secret-keys for adaptively secure general inner-product encryption. In Dongdai Lin, Gene Tsudik, and Xiaoyun Wang, editors, *Cryptology and Network Security - 10th International Conference, CANS 2011, Sanya, China, December 10-12, 2011. Proceedings*, volume 7092 of *Lecture Notes in Computer Science*, pages 138–159. Springer, 2011.

[PPSS13] Duong Hieu Phan, David Pointcheval, Siamak Fayyaz Shahandashti, and Mario Strefler. Adaptive CCA broadcast encryption with constant-size secret keys and ciphertexts. *International Journal of Information Security*, 12(4):251–265, 2013.

[RW13] Yannis Rouselakis and Brent Waters. Practical constructions and new proof methods for large universe attribute-based encryption. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 463–474. ACM, 2013.

[Sah99] Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 543–553. IEEE Computer Society, 1999.

[SC12] Reihaneh Safavi-Naini and Ran Canetti, editors. *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*. Springer, 2012.

[Sha84] Adi Shamir. Identity-based cryptosystems and signature schemes. In G. R. Blakley and David Chaum, editors, *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*, volume 196 of *Lecture Notes in Computer Science*, pages 47–53. Springer, 1984.

[Sho00] Victor Shoup. Using hash functions as a hedge against chosen ciphertext attack. In Bart Preneel, editor, *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, volume 1807 of *Lecture Notes in Computer Science*, pages 275–288. Springer, 2000.

[Sho04] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, 2004.

[SW05] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In Cramer [Cra05], pages 457–473.

[Wat05] Brent Waters. Efficient identity-based encryption without random oracles. In Cramer [Cra05], pages 114–127.

[Wat09a] Brent Waters. Dual system encryption: Realizing fully secure IBE and HIBE under simple assumptions. In Shai Halevi, editor, *Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*, volume 5677 of *Lecture Notes in Computer Science*, pages 619–636. Springer, 2009. Full version [Wat09b].

[Wat09b] Brent Waters. Dual system encryption: Realizing fully secure IBE and HIBE under simple assumptions. Cryptology ePrint Archive, Report 2009/385, 2009.

[Wat11] Brent Waters. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In Catalano et al. [CFGN11], pages 53–70.

[Wat12] Brent Waters. Functional encryption for regular languages. In Safavi-Naini and Canetti [SC12], pages 218–235.

[Wee14] Hoeteck Wee. Dual system encryption via predicate encodings. In Yehuda Lindell, editor, *Theory of Cryptography - 11th Theory of Cryptography Conference, TCC 2014, San Diego, CA, USA, February 24-26, 2014. Proceedings*, volume 8349 of *Lecture Notes in Computer Science*, pages 616–637. Springer, 2014.

[YAHK11] Shota Yamada, Nuttapong Attrapadung, Goichiro Hanaoka, and Noboru Kunihiro. Generic constructions for chosen-ciphertext secure attribute based encryption. In Catalano et al. [CFGN11], pages 71–89.

[YAS+12] Shota Yamada, Nuttapong Attrapadung, Bagus Santoso, Jacob C. N. Schuldt, Goichiro Hanaoka, and Noboru Kunihiro. Verifiable predicate encryption and applications to CCA security and anonymous predicate authentication. In Fischlin et al. [FBM12], pages 243–261.