Models and Constructions for Secure Reputation Systems

M.Sc. Jakob Juhnke

January 21, 2019

PADERBORN UNIVERSITY

The University for the Information Society

A dissertation 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
December 18, 2018

Acknowledgements

I would like to thank Johannes Blömer for giving me the opportunity to work in his research group, for expanding my knowledge on modern cryptography, and for interesting discussions about my research.

Furthermore, I thank my former colleagues Dr. Peter Günther and Dr. Gennadij Liske for unofficially mentoring me, for providing great hints regarding scientific practice, and for helpful comments on the security definition I developed for this dissertation.

Besides Peter and Gennadij, I also thank my colleagues Claudia Jahn, Fabian Eidens, and Sascha Brauer for being such great friends supporting me and my family whenever needed, and for having much fun both at the office and in our spare time. In particular, I thank Claudia for organizing wonderful events, for solving problems unrelated to research, and for handling the daily business.

Moreover, I want to thank Fabian Eidens, Jan Bobolz, Nils Löken, Sascha Brauer, Dr. Kathrin Bujna, and Denis Diemert for their help in solving some mathematical issues, for fruitful discussions about definitions and techniques I apply in this thesis, and for proof-reading.

Of course, I also thank my family, especially my wife Silvana and my children, for motivating me in times I wanted to quit, for reminding me to take breaks whenever it was stressful, and for their patience in general.

Finally, I want to thank Prof. Dr. Hanno Lefmann from the Chemnitz University of Technology for teaching me the basics of scientific practice and for motivating me to keep on researching after finishing my studies of computer science.

Abstract

In this thesis we consider reputation systems and their security from a cryptographic perspective. The security properties a reputation system has to provide, as well as attacks against them and appropriate countermeasures, are extensively discussed in the literature and well understood. However, no generally accepted security model has emerged. This is due to the fact that some properties seem to mutually exclude each other, which complicates the design of models for secure reputation systems. Interestingly, each of the security properties can be realized with cryptographic primitives, albeit not necessarily simultaneously. Hence, it is reasonable to analyze the security of reputation systems in a cryptographic context.

Our main contribution is the design of two models for cryptographically secure reputation systems. The first model we propose follows an experiment-based approach to define security and extends the model of (dynamic) group signatures. Experiment-based security definitions have the advantage that they allow one to precisely formalize the desired security properties. But this approach is also susceptible to miss subtle details in the security definition. Therefore, we propose a second model, defined as an ideal functionality in the Universal Composability Framework. This ideal functionality defines security implicitly and is therefore able to cover subtle security properties, and is also able to overcome certain other disadvantages of the first model. Furthermore, the second model eliminates some disadvantages of the first model. Moreover, the Universal Composability Framework guarantees security for concurrently composed applications, which is an important property for reputation systems. For both security models we additionally provide efficient constructions of reputation systems that are provably secure in their respective model.

Zusammenfassung

In dieser Dissertationsschrift betrachten wir Reputationssysteme und deren Sicherheit vor einem kryptographischen Hintergrund. Die Sicherheitseigenschaften, die ein Reputationssystem erfüllen sollte, werden in der Literatur ausgiebig diskutiert und verstanden. Gleiches gilt für Angriffe gegen Reputationssysteme und deren Gegenmaßnahmen. Allerdings hat sich bisher kein allgemein akzeptiertes Sicherheitsmodell etablieren können. Dies liegt vor allem daran, dass sich einige Sicherheitseigenschaften zu widersprechen scheinen und sich vermutlich gegenseitig ausschließen, was die Definition eines Sicherheitsmodells erschwert. Interessanterweise ist jede dieser Eigenschaften, einzeln betrachtet, mit Hilfe von kryptographischen Verfahren realisierbar. Daher erscheint es vernünftig, die Sicherheitseigenschaften von Reputationssystemen im kryptographischen Kontext zu analysieren.

Das Hauptresultat dieser Arbeit ist der Entwurf von zwei Modellen für sichere Reputationssysteme. Das erste Modell nutzt experiment-basierte Sicherheitsdefinitionen auf der Grundlage des Sicherheitsmodells von Gruppensignaturen. Experiment-basierte Sicherheitsdefinitionen haben den Vorteil, dass sich mit ihnen die gewünschten Eigenschaften sehr präzise formalisieren lassen. Ein wesentlicher Nachteil ist jedoch, dass feine Details von Sicherheitsanforderungen leicht übersehen werden und somit nicht formalisiert werden. Aus diesem Grund stellen wir ein zweites Modell vor, welches als eine ideale Funktionalität im sogenannten Universal Composability Framework alle Sicherheitseigenschaften nur implizit definiert. Dadurch ist es jedoch möglich, auch Details von Sicherheitseigenschaften abzudecken. Darüber hinaus hat das erste Sicherheitsmodell einige Nachteile, welche im zweiten Modell behoben werden. Außerdem bewahren Applikationen, die im Universal Composability Framework definiert sind, ihre Sicherheitseigenschaften, wenn sie mit anderen Applikationen kombiniert werden, was insbesondere für Reputationssysteme wichtig ist. Für jedes der beiden Sicherheitsmodelle präsentieren wir ebenfalls effiziente Konstruktionen von Reputationssystemen, die im jeweiligen Modell als sicher bewiesen werden.

Contents

1 Introduction

In a nutshell, the goal of a reputation system is to provide reliable information about previous transactions in an application with interacting users. To explain the meaning of that, it is necessary to consider the application, the reputation system, and the different roles of the interacting users successively. Typically, in an application users interact with each other in two different roles, as a consumer or as a provider of some object. The kinds of available objects depend on the application, for example in online marketplaces the objects are products offered for purchase, in file-sharing communities they can be the files to share or the bandwidth supplied by other users, and in bulletin board systems every comment is an object. In such an application the consumers have to decide which object from which provider they want to consume. This can be problematic, especially when similar objects are available or the consumers have not interacted with a specific provider before. To resolve this issue, a reputation system can be incorporated into the application. Such a system enables users to rate the quality of consumed objects and the behavior of their corresponding providers. In the context of reputation systems, the consumption of objects is called a transaction and a user rating a transaction acts in the role of a rater. Thus, by collecting, evaluating, and aggregating ratings from all raters, the reputation system can provide condensed information about previous transactions. Therefore, when future behavior of providers and the quality of provided objects can be estimated based on the information supplied by the reputation system, they can be helpful in a user's decision process. To allow such estimations, the reputation information must be reliable, which can only be guaranteed when this information is protected against manipulation.

Basically, protection against manipulation is possible by preventing attacks, or mitigating an attack's effects when it cannot be prevented. Throughout the literature many different attacks are discussed and appropriate countermeasures are proposed to secure various aspects of reputation systems. Nevertheless, no generally accepted security model for reputation systems has emerged. This is because conflicting security properties complicate the design of holistic security models. This is exemplified by considering the security properties *linkability* and *anonymity*. Linkability requires that every user can check whether or not two ratings for the same transaction were created by the same rater. Thus, linkability is a mechanism to prevent specific attacks in reputation systems. In contrast to that, without anonymity a rater must fear discrimination due to negative ratings, which leads to dishonest ratings to avoid reprisals. But with dishonest ratings a reputation system cannot provide reliable information. Therefore, raters must be anonymous when supplying ratings, which seems to be in conflict when ratings are linkable. This further demonstrates that

more properties than those motivated by countermeasures against specific attacks have to be considered in models for secure reputation systems.

By further analyzing the concrete security requirements of reputation systems it turns out that besides anonymity and linkability also traceability and non-frameability need to be considered. Traceability ensures that the identity of any rater can be determined by a designated manager of the reputation system. Thereby, accountability can be realized, which is another mechanism to prevent attacks in reputation systems conflicting with anonymity. Related to traceability, non-frameability guarantees that honest raters are not blamed for a rating they did not create. The combination of traceability and non-frameability enables penalizing dishonest raters.

Having identified a set of natural security properties for reputation systems, in this thesis we want to answer the following research question:

How can one design a holistic security model for reputation systems that supports all of the identified security properties simultaneously, and that can be combined with arbitrary applications?

We address this question by considering reputation systems from a cryptographic perspective. The described security properties are similar to those provided by group signatures, which have been studied extensively in cryptography. As a consequence, formal security models for group signatures exist and it is well known how to realize the described properties in that context, although not necessarily simultaneously. Therefore, it is reasonable to use group signatures as a basis to define security models for reputation systems.

Contribution and Outline of this Thesis

As pointed out above, no generally accepted security model for reputation systems has emerged yet. However, since reputation systems are used more and more frequently, their security concerns gain in importance. So there is a need for a holistic security model that supports arbitrary applications, as formulated above as our research question. To answer this question, in this thesis we propose two comprehensive models for secure reputation systems and provide constructions that fulfill all defined security properties.

For the first proposed model we use experiment-based security definitions. These allow to precisely formalize the desired security properties. However, with this approach it is possible that subtle details are missing. A more intuitive approach is used for the second model, where we formulate an ideal functionality for reputation systems in the Universal Composability Framework. Here, the security is defined implicitly and hence specific security properties are harder to identify. For both security models we also provide constructions that are provably secure. These constructions show some similarities, but the security proofs are remarkably different. In the experiment-based security proofs it is possible to focus on an individual security property, whereas all security properties have to be considered simultaneously in the Universal Composability Framework. Nevertheless, we provide the technical preliminaries for both approaches in this thesis, which is organized as follows.

Chapter 2: Due to their previously noted similarities to group signatures, it is sensible to consider reputation systems from a cryptographic perspective. Indubitably, reputation systems are more complex than group signatures, so further cryptographic primitives are needed to construct secure reputation systems. The cryptographic primitives used in this work are hash functions, commitment schemes, public-key encryption, digital signatures, and proof systems, which are introduced in Chapter 2. Additionally, this chapter contains notational conventions, foundational mathematical concepts related to bilinear groups, and the cryptographic hardness assumptions used in this work.

Chapter 3: Before presenting to the definition of a security model for reputation systems, we introduce their functionality and typical attacks in Chapter 3. Based on known countermeasures against those attacks we further describe the similarities of reputation systems and group signatures in detail. Finally, we sketch how group signatures can be used to formulate a security model for reputation systems.

Chapter 4: After introducing the essentials in Chapter 2 and Chapter 3, we propose the first comprehensive model for secure reputation systems in Chapter 4. Initially, we describe the architecture of secure reputation systems. Subsequently, we formally define and discuss the four security properties anonymity, public linkability, traceability, and strong exculpability that a secure reputation system must fulfill. Besides that, we also provide a construction based on group signatures that is provably secure in the proposed model.

Chapter 5: To further improve the model proposed in Chapter 4, we consider reputation systems in the Universal Composability Framework. This framework is designed to guarantee security under concurrent composition of arbitrary protocols, which is a very important property for practical applications. In Chapter 5 we review the Universal Composability Framework by introducing its basic concept and discussing various technical details.

Chapter 6: With the foundational concepts from Chapter 5, we propose a second security model for reputation systems in the Universal Composability Framework in Chapter 6. Since the security properties in this model are harder to identify than those presented in Chapter 4, we also provide an exhaustive discussion about security. As for the first model, we present a construction of a secure reputation system in the second part of this chapter.

Chapter 7: Albeit proposing comprehensive models for secure reputation systems, there are some technical issues and additional interesting properties that need to be considered in future research. These aspects are discussed in Chapter 7.

Chapter 8: To increase readability of this thesis, Chapters 4 and 6 do not contain proofs showing that the provided constructions of reputation systems are secure in their respective model. Instead, we provide these proofs in Chapter 8, which finalizes this thesis.

Publications related to this Thesis

This thesis, especially the chapters 4, 6, 7, and 8, is based on the following publications with substantial contribution of the author.

[BEJ18] Johannes Blömer, Fabian Eidens, and Jakob Juhnke. „Practical, Anonymous, and Publicly Linkable Universally-Composable Reputation Systems“. In: *Topics in Cryptology – CT-RSA 2018*. Ed. by Nigel P. Smart. Vol. 10808. Lecture Notes in Computer Science. San Francisco, CA, USA: Springer, Heidelberg, Germany, Apr. 2018, pp. 470–490. DOI: 10.1007/978-3-319-76953-0_25. Full Version: Cryptology ePrint Archive, Report 2018/029. https://eprint.iacr.org/2018/029. 2018.

[BJK15] Johannes Blömer, Jakob Juhnke, and Christina Kolb. „Anonymous and Publicly Linkable Reputation Systems“. In: *FC 2015: 19th International Conference on Financial Cryptography and Data Security*. Ed. by Rainer Böhme and Tatsuaki Okamoto. Vol. 8975. Lecture Notes in Computer Science. San Juan, Puerto Rico: Springer, Heidelberg, Germany, Jan. 2015, pp. 478–488. DOI: 10.1007/978-3-662-47854-7_29. Full Version: Cryptology ePrint Archive, Report 2014/546. http://eprint.iacr.org/2014/546. 2014.

Furthermore, the author of this thesis also contributed to the following publications that are related to advanced signature schemes.

[BEJ18a] Johannes Blömer, Fabian Eidens, and Jakob Juhnke. „Enhanced Security of Attribute-Based Signatures“. In: *Cryptology and Network Security - 17th International Conference, CANS 2018, Naples, Italy, September 30 - October 3, 2018, Proceedings*. Ed. by Jan Camenisch and Panos Papadimitratos. Vol. 11124. Lecture Notes in Computer Science. Springer, 2018, pp. 235–255. DOI: 10.1007/978-3-030-00434-7_12.

[BJL15] Johannes Blömer, Jakob Juhnke, and Nils Löken. „Short Group Signatures with Distributed Traceability“. In: *Mathematical Aspects of Computer and Information Sciences - 6th International Conference, MACIS 2015, Berlin, Germany, November 11-13, 2015, Revised Selected Papers*. Ed. by Ilias S. Kotsireas, Siegfried M. Rump, and Chee K. Yap. Vol. 9582. Lecture Notes in Computer Science. Springer, 2015, pp. 166–180. DOI: 10.1007/978-3-319-32859-1_14.

2 Preliminaries

In this chapter we introduce notational conventions used throughout this thesis and provide the required cryptographic background. The notational conventions are defined in Section 2.1. In Section 2.2 we introduce the required cryptographic primitives and their formal definitions. Our constructions of reputation systems rely on cyclic groups of prime order and bilinear mappings, which are defined in Section 2.3. Finally, the cryptographic hardness assumptions guaranteeing security of our constructions are defined in Section 2.4.

2.1 Notation

By $x := y$ we denote the assignment of the value y to the variable x. For a finite set $\mathcal{S}$ we denote the sampling of an element of $\mathcal{S}$ according to the uniform distribution and its assignment to variable x by $x \xleftarrow{u} \mathcal{S}$. For a probabilistic algorithm $\mathcal{A}$ running on input y we denote the operation of assigning the output of $\mathcal{A}$ to the variable x by $x \leftarrow \mathcal{A}(y)$. When an algorithm $\mathcal{A}$ expects an input y from some specific set $\mathcal{S}$, we implicitly assume that $\mathcal{A}$ can efficiently verify the membership of $y \in \mathcal{S}$ and that $\mathcal{A}$ outputs the special error symbol $\perp$ in case $y \notin \mathcal{S}$.

The set of all possible outputs of a (randomized) algorithm $\mathcal{A}$ on input y we denote by $[\mathcal{A}(y)]$. We denote by $\mathcal{A}(y\colon \mathcal{O}_1, \mathcal{O}_2, \ldots)$ or $\mathcal{A}^{\mathcal{O}_1,\mathcal{O}_2,\cdots}(y)$ an algorithm $\mathcal{A}$ on input y with access to the oracles $\mathcal{O}_1, \mathcal{O}_2, \ldots$. By $\mathcal{A}(y; r)$ we denote running a de-randomized algorithm $\mathcal{A}$ on input y and randomness r.

For interactive algorithms $\mathcal{A}$ and $\mathcal{B}$ we denote by $(a, b) \leftarrow \langle \mathcal{A}(y) \leftrightarrow \mathcal{B}(z) \rangle$ the output of their interaction, where a is the output of $\mathcal{A}(y)$ and b is the output of $\mathcal{B}(z)$. If only one of both parties generates output, according to the definition of the interaction, we write $x \leftarrow \langle \mathcal{A}(y) \leftrightarrow \mathcal{B}(z) \rangle$.

Unless stated otherwise, a *probabilistic polynomial-time algorithm* $\mathcal{A}$ is an algorithm running in *strict probabilistic polynomial-time*. That is, there exists a bound, polynomial in the length of the inputs, on the number of steps in each possible run of $\mathcal{A}$, regardless of the outcome of its internal random choices. We also say an algorithm $\mathcal{A}$ is *efficient* when it runs in probabilistic polynomial-time. In contrast to that, a problem is *infeasible* when no efficient algorithm can solve it with more than negligible probability, where a negligible function is defined as follows:

Definition 2.1: Negligible Function - [KL07]
A function $f\colon \mathbb{N} \to \mathbb{R}$ is *negligible* if for every positive polynomial p there exists a value $N \in \mathbb{N}$ such that for all $n > N$, where $n \in \mathbb{N}$, it holds that $f(n) < \frac{1}{p(n)}$. △

2.2 Cryptographic Primitives

In this section we briefly introduce and formally define the most important cryptographic primitives that are used in this thesis.

2.2.1 Hash Functions and Random Oracles

A function $\mathcal{H}\colon X \to Y$ is called a *hash function*, if Y is a finite nonempty set and $|X| > |Y|$. This definition implies that for every hash function $\mathcal{H}$ there exist different inputs x, x' such that $\mathcal{H}(x) = \mathcal{H}(x')$, so called *collisions*. In cryptography, finding collisions of a hash function must be infeasible. This property is called *collision-resistance* and was first formally defined by Damgård [Dam88]. For a complexity theoretic treatment the definition of collision-resistance considers *families of hash functions* instead of a just one hash function.

Definition 2.2: Hash Function Family - [Dam88; Can+07]
A collection of functions $\mathcal{H} = \{\mathcal{H}_\lambda\colon K_\lambda \times X_\lambda \to Y_\lambda\}_{\lambda\in\mathbb{N}}$ is a *family of hash functions*, if K_λ and Y_λ are nonempty finite sets and $|X_\lambda| > |Y_\lambda|$ holds for all $\lambda \in \mathbb{N}$. △

Definition 2.3: Collision-Resistant Hash Function Family - [Dam88; Can+07]
A family of hash functions is a *family of collision-resistant hash functions*, if the following conditions hold:

- for all $\lambda \in \mathbb{N}$, elements from K_λ can be sampled efficiently according to the uniform distribution,
- there exists an algorithm Eval that on input $(1^\lambda, k, x)$ outputs $\mathcal{H}_\lambda(k, x)$ in polynomial-time, where $k \in K_\lambda$ and $x \in X_\lambda$,
- for all probabilistic polynomial-time adversaries $\mathcal{A}$ there exists a negligible function negl such that

$$\Pr\left[k \xleftarrow{u} K_\lambda, (x_1, x_2) \leftarrow \mathcal{A}(1^\lambda, k)\colon x_1 \neq x_2 \wedge \mathcal{H}_\lambda(k, x_1) = \mathcal{H}_\lambda(k, x_2)\right] \leq \mathsf{negl}(\lambda),$$

  where the probability is taken over the random choice of k and the random bits used by $\mathcal{A}$.

A member of a collision-resistant hash function family is also called collision-resistant. △

In this work we will always use hash functions in combination with other cryptographic primitives. To simplify the notation of hash functions in such contexts we omit the security

parameter λ and the key k. That means, when no confusion is possible, we denote the hash function $\mathcal{H}_\lambda(k, x)$ by $\mathcal{H}(x)$ and assume that k is publicly available.

An idealization for hash functions is the so called *random oracle model*, introduced by Bellare and Rogaway [BR93]. In this model a hash function $\mathcal{H}\colon X \to Y$ is chosen uniformly at random from the set of all functions from X to Y and the only way to evaluate $\mathcal{H}$ is to query a public oracle $\mathcal{O}$. In general, cryptographic schemes proven secure in the random oracle model are more efficient than schemes in the standard model, but the security guarantees are weaker. This is because random oracles cannot be realized in practice, as proven by Canetti, Goldreich, and Halevi [CGH04]. Nevertheless, replacing random oracles with collision-resistant hash functions in practical applications is still a reasonable heuristic for the purposes of *practical* security.

2.2.2 Commitment Schemes

A commitment scheme enables a party, the *committer*, to commit to some secret value m by sending a *commitment* c to a *receiver*. After this *commit phase*, the committer can open the commitment in a *decommit phase* to *reveal* m to the receiver. Before the committer initiated the decommit phase the receiver should gain no information about m from c, which is called *hiding*. During the decommit phase the committer should not be able to reveal some $m' \neq m$ to the receiver, which is called *binding*. The formal definition of a (non-interactive) commitment scheme is given in Definition 2.4. The security properties *hiding* and *binding* are defined in Definitions 2.5 and 2.6, respectively.

Definition 2.4: Commitment Scheme - [Dam00; DG03]
A *(non-interactive) commitment scheme* Π for message space $\mathcal{M}$ consists of three probabilistic polynomial-time algorithms:

KeyGen(1^λ) takes as input the security parameter λ. It outputs a public key pk.

Commit(pk, m) takes as input the public key pk and a message $m \in \mathcal{M}$. It outputs a commitment c and a decommitment d.

Reveal(pk, c, d) takes as input the public key pk, a commitment c, and a decommitment d. It outputs a message $m \in \mathcal{M}$ or the error symbol $\perp \notin \mathcal{M}$.

A non-interactive commitment scheme is *correct*, if for every $\lambda \in \mathbb{N}$ and every $m \in \mathcal{M}$ it holds

$$\Pr\left[pk \leftarrow \mathsf{KeyGen}(1^\lambda), (c, d) \leftarrow \mathsf{Commit}(pk, m) : \mathsf{Reveal}(pk, c, d) = m\right] = 1. \qquad \triangle$$

Remark: In a more general definition of commitment schemes, as given by Halevi and Micali [HM96] or Fischlin [Fis01], Commit and Reveal are interactive protocols between the committer and a receiver. In the commit phase both parties interactively form a commitment by executing the protocol Commit. Analogously, in the decommit phase the committer and the receiver run the protocol Reveal to reveal the value m. Non-interactive commitment schemes are the special case of such protocols, where only a single message is sent in both phases. That means, in the commit phase the committer computes $(c, d) \leftarrow \mathsf{Commit}(pk, m)$

and sends c to the receiver, whereas in the decommit phase the committer sends d to the receiver computing $m \leftarrow \mathsf{Reveal}(pk, c, d)$. In this work we will only consider non-interactive commitments, hence we omit the definition of interactive commitment schemes. ¤

Definition 2.5: Hiding Commitment Schemes - [Dam00; Fis01]
The hiding property of a non-interactive commitment scheme Π is defined via the following experiment:

$$
\begin{aligned}
&\text{Experiment } \mathrm{Exp}^{hide}_{\mathcal{A},\Pi}(\lambda)\\
&\quad pk \leftarrow \mathsf{KeyGen}(1^\lambda)\\
&\quad (m_0, m_1, \mathrm{St}) \leftarrow \mathcal{A}(pk)\\
&\quad b \leftarrow\!\!\!\!\text{\tiny$\$$}\ \{0,1\}\\
&\quad c \leftarrow \mathsf{Commit}(pk, m_b)\\
&\quad b' \leftarrow \mathcal{A}(\mathrm{St}, c)\\
&\quad \textbf{If } b = b' \wedge m_0, m_1 \in \mathcal{M} \textbf{ Then}\\
&\qquad \text{output } 1\\
&\quad \textbf{Else } \text{output } 0
\end{aligned}
$$

We say a (non-interactive) commitment scheme $\Pi = (\mathsf{KeyGen}, \mathsf{Commit}, \mathsf{Reveal})$ is *computationally hiding*, if for all probabilistic polynomial-time adversaries $\mathcal{A}$ there exists a negligible function negl such that

$$\Pr\left[\mathrm{Exp}^{hide}_{\mathcal{A},\Pi}(\lambda) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$ △

Definition 2.6: Binding Commitment Schemes - [Dam00; Fis01]
The binding property of a non-interactive commitment scheme Π is defined via the following experiment:

$$
\begin{aligned}
&\text{Experiment } \mathrm{Exp}^{bind}_{\mathcal{A},\Pi}(\lambda)\\
&\quad pk \leftarrow \mathsf{KeyGen}(1^\lambda)\\
&\quad (c, d_1, d_2) \leftarrow \mathcal{A}(pk)\\
&\quad \textbf{If } \mathsf{Reveal}(pk, c, d_1) \neq \mathsf{Reveal}(pk, c, d_2) \textbf{ Then}\\
&\qquad \text{output } 1\\
&\quad \textbf{Else } \text{output } 0
\end{aligned}
$$

We say a (non-interactive) commitment scheme $\Pi = (\mathsf{KeyGen}, \mathsf{Commit}, \mathsf{Reveal})$ is *computationally binding*, if for all probabilistic polynomial-time adversaries $\mathcal{A}$ there exists a negligible function negl such that

$$\Pr\left[\mathrm{Exp}^{bind}_{\mathcal{A},\Pi}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda).$$ △

A special variant of commitment schemes are *trapdoor commitments*. In such schemes there exists a trapdoor that allows to open a commitment to arbitrary values. This trapdoor

is neither handed to the committer nor to the receiver. Instead, it is used within security proofs to generate the adversary's view.

Definition 2.7: Trapdoor Commitment Scheme - [Abe+10; DG03; Gro09]
A *(non-interactive) trapdoor commitment scheme* Π for message space $\mathcal{M}$ consists of five probabilistic polynomial-time algorithms:

KeyGen(1^λ) takes as input the security parameter. It outputs a public key pk and a trapdoor td.

Commit(pk, m) takes as input the public key pk and a message $m \in \mathcal{M}$. It outputs a commitment c and a decommitment d.

Reveal(pk, c, d) takes as input the public key pk, a commitment c, and a decommitment d. It outputs a message $m \in \mathcal{M}$ or the error symbol $\perp \notin \mathcal{M}$.

TCommit(pk, td) takes as input the public key pk and the trapdoor td. It outputs an equivocal commitment $\hat{c}$ and an equivocation key ek.

TReveal($pk, \hat{c}, ek, m$) takes as input the public key pk, an equivocal commitment $\hat{c}$, the corresponding equivocation key ek, and a message m. It outputs a decommitment $\hat{d}$.

A non-interactive trapdoor commitment scheme is *correct*, if for every $\lambda \in \mathbb{N}$ and every $m \in \mathcal{M}$ it holds

$$\Pr\left[(pk, td) \leftarrow \mathsf{KeyGen}(1^\lambda), (c, d) \leftarrow \mathsf{Commit}(pk, m) : \mathsf{Reveal}(pk, c, d) = m\right] = 1$$

and

$$\Pr\left[\begin{array}{l}(pk, td) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ (\hat{c}, ek) \leftarrow \mathsf{TCommit}(pk, td) \\ \hat{d} \leftarrow \mathsf{TReveal}(pk, \hat{c}, ek, m)\end{array} : \mathsf{Reveal}(pk, \hat{c}, \hat{d}) = m\right] = 1. \qquad \triangle$$

Besides hiding and binding, secure trapdoor commitments have to fulfill the *trapdoor property*, which states that commitments and decommitments generated by executing Commit are indistinguishable from those generated by executing TCommit and TReveal.

Definition 2.8: Trapdoor Property - [Abe+10; DG03; Gro09]
The trapdoor property of a non-interactive trapdoor commitment scheme Π is defined via the following experiment:

Experiment $\mathrm{Exp}^{td}_{\mathcal{A},\Pi}(\lambda)$
  $(pk, td) \leftarrow \mathsf{KeyGen}(1^\lambda)$
  $(m, \mathrm{St}) \leftarrow \mathcal{A}(pk)$
  $b \xleftarrow{\$} \{0,1\}$
  If $b = 0$ **Then**
    $(c, d) \leftarrow \mathsf{Commit}(pk, m)$
  Else
    $(c, ek) \leftarrow \mathsf{TCommit}(pk, td)$
    $d \leftarrow \mathsf{TReveal}(pk, c, ek, m)$
  $b' \leftarrow \mathcal{A}(\mathrm{St}, c, d)$
  If $b = b' \wedge m \in \mathcal{M}$ **Then**
    output 1
  Else output 0

We say a (non-interactive) trapdoor commitment scheme $\Pi = (\mathsf{KeyGen}, \mathsf{Commit}, \mathsf{Reveal})$ fulfills the *computational trapdoor property*, if for all probabilistic polynomial-time adversaries $\mathcal{A}$ there exists a negligible function negl such that

$$\Pr\left[\mathrm{Exp}^{td}_{\mathcal{A},\Pi}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda). \qquad \triangle$$

2.2.3 Public-Key Encryption

Public-key encryption schemes enable parties to communicate privately with each other without having to share secret information. For this to work, every party that wants to receive private messages generates a personal pair (pk, sk) of public and secret keys. A sender can use the public key pk to encrypt a message and send it to the receiver. Using the secret key sk the receiver can then decrypt the message. It is important not to reveal sk to anyone, as the security of public-key encryption relies on the secrecy of sk. The formal definitions of public-key encryption and its security are given in Definitions 2.9 and 2.10.

Definition 2.9: Public-Key Encryption Scheme - [Bel+98; NY90]
A *public-key encryption scheme* Π for message space $\mathcal{M}$ consists of three probabilistic polynomial-time algorithms:

$\mathsf{KeyGen}(1^\lambda)$ takes as input the security parameter. It outputs a public key pk and a corresponding secret key sk.

$\mathsf{Enc}(pk, m)$ takes as input the public key pk and a message $m \in \mathcal{M}$. It outputs a ciphertext ct.

$\mathsf{Dec}(sk, ct)$ takes as input the secret key sk and a ciphertext ct. It outputs a message $m \in \mathcal{M}$ or the error symbol $\perp \notin \mathcal{M}$.

A public-key encryption scheme is *correct*, if for every $\lambda \in \mathbb{N}$ and every $m \in \mathcal{M}$ it holds

$$\Pr\left[(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda) : \mathsf{Dec}(sk, \mathsf{Enc}(pk, m)) = m\right] = 1. \qquad \triangle$$

Definition 2.10: Security against Chosen Ciphertext Attacks - [Bel+98; NY90]
The security of a public-key encryption scheme Π against an adversary $\mathcal{A}$ that runs an *adaptive chosen ciphertext attack* is defined via the following experiment:

Experiment $\mathrm{Exp}^{cca}_{\mathcal{A},\Pi}(\lambda)$
$(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda)$
$(m_0, m_1, \mathrm{St}) \leftarrow \mathcal{A}(pk : \mathsf{Dec}_{sk})$
$b \xleftarrow{u} \{0, 1\}$
$ct^* \leftarrow \mathsf{Enc}(pk, m_b)$
$b' \leftarrow \mathcal{A}(\mathrm{St}, ct^* : \mathsf{Dec}_{sk})$
If $(b = b') \wedge (\mathcal{A}$ did not query $\mathsf{Dec}_{sk}(ct))$
$\wedge (|m_0| = |m_1|) \wedge (m_0, m_1 \in \mathcal{M})$ **Then**
output 1
Else output 0

Oracle $\mathsf{Dec}_{sk}(ch)$
Output $m \leftarrow \mathsf{Dec}(sk, ch)$

In this experiment the adversary has access to a *decryption oracle* Dec_{sk} which can be used to obtain decryptions of adversarially chosen ciphertexts.

We say the public-key encryption scheme Π is *secure against adaptive chosen ciphertext attacks* (or CCA-secure), if for all probabilistic polynomial-time adversaries $\mathcal{A}$ there exists a negligible function negl such that

$$\Pr\left[\mathrm{Exp}^{cca}_{\mathcal{A},\Pi}(\lambda) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda). \qquad \triangle$$

Remark: Another important security definition for public-key encryption schemes is that of security against *chosen plaintext attacks* (CPA-security). This weaker notion is defined analogously to CCA-security, but the adversary has no access to the decryption oracle. ¤

2.2.4 Digital Signatures

Digital signature schemes enable parties to exchange messages such that every receiver can determine the origin of a message and verify its integrity. For this purpose a sender generates a personal *secret signing key sk* and a corresponding *public verification key pk*. Using the signing key a sender can compute a signature for some message. The message-signature pair is then sent to a receiver who is able to verify it using the public verification key of the sender. This basic idea is formalized in Definition 2.11. Certainly, signatures can guarantee integrity as long as the signer is the only party that can generate them. This requirement is formally defined in Definition 2.12.

Definition 2.11: Digital Signature Scheme - [GMR88]
A *digital signature scheme* Π for message space $\mathcal{M}$ consists of three probabilistic polynomial-time algorithms:

KeyGen(1^λ) takes as input the security parameter. It outputs a public key pk and a corresponding secret key sk.

Sign(sk, m) takes as input the secret key sk and a message $m \in \mathcal{M}$. It outputs a signature σ.

Verify(pk, m, σ) takes as input the public key pk, a message $m \in \mathcal{M}$, and a signature σ. It outputs a bit $v \in \{0, 1\}$.

We say a signature σ is *valid* for message $m \in \mathcal{M}$ with respect to public key pk if $\mathsf{Verify}(pk, m, \sigma) = 1$, and *invalid* otherwise.

A digital signature scheme is *correct*, if for all $\lambda \in \mathbb{N}$ and all $m \in \mathcal{M}$ it holds

$$\Pr\left[(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda) : \mathsf{Verify}(pk, m, \mathsf{Sign}(sk, m)) = 1\right] = 1. \qquad \triangle$$

Definition 2.12: Existential Unforgeability under Chosen Message Attacks - [GMR88]
The existential unforgeability of a digital signature scheme Π against an adversary $\mathcal{A}$ that runs an *adaptive chosen message attack* is defined via the following experiment:

Experiment $\mathrm{Exp}^{euf-cma}_{\mathcal{A},\Pi}(\lambda)$
$\mathcal{Q} := \emptyset$
$(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda)$
$(m^*, \sigma^*) \leftarrow \mathcal{A}(pk : \mathsf{Sign}_{sk})$
If $\mathsf{Verify}(pk, m^*, \sigma^*) = 1$
$\wedge\, m^* \notin \mathcal{Q} \wedge m^* \in \mathcal{M}$ **Then**
output 1
Else output 0

Oracle $\mathsf{Sign}_{sk}(m)$
$\sigma \leftarrow \mathsf{Sign}(sk, m)$
$\mathcal{Q} := \mathcal{Q} \cup \{m\}$

In this experiment the adversary has access to a *sign oracle* Sign_{sk} which can be used to obtain signatures on adversarial chosen messages.

We say the digital signature scheme Π is *existentially unforgeable under a chosen message attack* (or EUF-CMA-secure), if for all probabilistic polynomial-time adversaries $\mathcal{A}$ there exists a negligible function negl such that

$$\Pr\left[\mathrm{Exp}^{euf-cma}_{\mathcal{A},\Pi}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda). \qquad \triangle$$

In Definition 2.12 we require an adversary to output a forgery σ^* for some message m^* that was not queried to the signing oracle Sign_{sk} previously. An alternative formulation would be to add the tuple (m, σ) to the set $\mathcal{Q}$ in the oracle definition and to test whether $(m^*, \sigma^*) \in \mathcal{Q}$ in the winning condition. This is an important difference as in the first case

randomized signatures are not considered as a forgery while they are in the second case. Since we will use randomizable signatures in this work we rely on the slightly weaker notion of EUF-CMA security.

2.2.5 Group Signatures

Group signatures, introduced by Chaum and van Heyst [Cv91], are a generalization of digital signatures and allow a group of users to anonymously produce signatures on behalf of the entire group. In this section we informally restate the concepts of static and dynamic group signature schemes and their security properties. We do not provide formal definitions, because the security definitions for the reputation systems proposed in Chapter 4 are similar to those of (dynamic) group signatures.

Static Group Signature Schemes

A formal model for static group signatures schemes and their security is defined by Bellare, Micciancio, and Warinschi [BMW03]. Such a scheme consists of a single *group manager* and a fixed group of *users*. To initialize the group signature scheme a trusted setup phase is necessary which generates the *group public key* and the *group manager's public key*, hands the *group manager's secret key* to the group manager, and outputs personal *secret signing keys* to every user. Using these secret signing keys every user can *sign* messages on behalf of the group. With respect to the group public key everybody, even outsiders, can *verify* the validity of message-signature-pairs. In case of misbehaving users, the group manager is able to extract the signer's identity from *valid* signatures, which is called *opening a signature*.

As discussed in [BMW03], *full-anonymity* and *full-traceability* are the two security properties to consider for static group signatures:

Full-anonymity means that an adversary in possession of all secret signing keys of the users and with access to an opening oracle is not able to tell which user generated a signature. This strong property also implies unlinkability, which in turn means that it is hard to tell whether or not two signatures were produced by the same user.

To ensure that the opening mechanism is useful, full-traceability requires that no group of colluding users can produce signatures which cannot be opened to a user of the coalition. This property has to hold even when all users collude and the group manager's secret key is known to the coalition, which also implies that it is hard to produce signatures which cannot be opened at all.

By combining digital signatures, public-key encryption, and non-interactive zero-knowledge proofs Bellare, Micciancio, and Warinschi [BMW03] present a generic construction of static group signatures and prove its security.

Dynamic Group Signature Schemes

In contrast to static groups, dynamic group signature schemes are more flexible as they allow users to join the group over time. This seems to be a minor modification of the model for static group signatures, but it requires many adjustments - especially regarding the security properties.

Comprehensive formal definitions for dynamic group signatures are given by Bellare, Shi, and Zhang [BSZ05]. Following their notation, dynamic group signatures consist of a single *opener*, a group of *users* with unique identities, and a dedicated *issuer*. During a trusted setup phase the *group public key*, the *opener's public key*, and the *issuer's public key* are generated and published, the *opener's secret key* and the *issuer's secret key* are handed to the opener and the issuer, respectively. To become a group member a user has to generate a personal *public and secret key pair*, which in turn is used to execute an *interactive group joining protocol* with the issuer. During this protocol the user's *secret signing key* is generated such that the issuer does not know the entire key. By combining the personal public key, the corresponding secret key, and the secret signing key a user is able to sign messages on behalf of the entire group. Everyone in possession of the group public key can verify the *validity* of message-signature-pairs. As in static group signatures, the opener can *open* signatures to extract the identity of a signer by using the opener's secret key. But instead of simply outputting the obtained identity, the opener has to generate a proof that the claimed user is really the author of a given signature. For security reasons this proof can be verified, which is called *judging*, to ensure that the opener does not blame an honest group member.

The security properties of dynamic group signatures according to [BSZ05] are *anonymity*, *traceability*, and *non-frameability*:

To break anonymity an adversary is asked to determine which user generated a specific signature. In this process the adversary acts as the issuer, can obtain all secret keys from group members by corrupting them, and has access to an opening oracle. As for static group signatures this implies the unlinkability of signatures.

A dynamic group signature scheme is traceable, if it is infeasible to produce signatures that cannot be opened, or for which no opening proof can be generated. In this scenario the issuer must be honest to prevent the creation of dummy-users that are unknown to the opener. But an adversary can create honest group members, corrupt them to obtain their secret keys and interact with the issuer on their behalf.

Non-frameability states that it is hard to produce an opening proof blaming an honest group member having generated a signature, when he did not. An adversary against non-frameability can corrupt the opener, the issuer, and all but one group members. As a consequence, this property relies on the user's personal secret key used during the interactive group joining protocol, but which is unknown to the issuer.

To achieve these properties simultaneously it is important that every party can obtain authenticated copies of the different public keys. Otherwise, users could modify their keys over time. Therefore, Bellare, Shi, and Zhang [BSZ05] assume the existence of a public-key infrastructure, but also other models are suitable to achieve this.

As for static group signatures Bellare, Shi, and Zhang [BSZ05] present a generic and provably secure construction for dynamic group signatures based on digital signatures, public-key encryption, and non-interactive zero-knowledge proofs systems.

2.2.6 Interactive Proofs

Interactive proofs enable a party $\mathcal{P}$ to *convince* another party $\mathcal{V}$ *interactively* that some statement is true. At the same time they ensure that $\mathcal{P}$ can not convince $\mathcal{V}$ when a statement is false.

Definition 2.13: Interactive Proof System - [GMR89; MY08]
A pair of interactive algorithms $(\mathcal{P}, \mathcal{V})$, the computationally unbounded prover $\mathcal{P}$ and the probabilistic polynomial-time verifier $\mathcal{V}$, is an *interactive proof system* for language $\mathcal{L}$ if there exists a negligible function negl such that

- *Completeness:* $\forall x \in \mathcal{L}\colon \Pr\left[1 \leftarrow \langle \mathcal{P}(x) \leftrightarrow \mathcal{V}(x) \rangle\right] \geq 1 - \mathsf{negl}(|x|)$
- *Soundness:* $\forall x \notin \mathcal{L}\colon$ for every interactive algorithm $\mathcal{P}^*$ it holds

$$\Pr\left[1 \leftarrow \langle \mathcal{P}^*(x) \leftrightarrow \mathcal{V}(x) \rangle\right] \leq \mathsf{negl}(|x|).$$

We say $\mathcal{V}$ *accepts* x after interacting with $\mathcal{P}$ if $\mathcal{V}$ outputs 1, and $\mathcal{V}$ *rejects* x otherwise. △

For practical (cryptographic) applications a computationally unbounded prover is not useful. Therefore, it is reasonable to reduce the prover's computational power to probabilistic polynomial-time. But then the prover is as powerful as the verifier and hence is only able to prove trivial statements (statements the verifier can verify on its own). For this reason the prover gets an *auxiliary input* w, called a *witness*, to make him more powerful than the verifier. Such a system is called *interactive argument*.

Definition 2.14: Interactive Argument System - [MY08]
A pair of interactive probabilistic polynomial-time algorithms $(\mathcal{P}, \mathcal{V})$, the prover $\mathcal{P}$ and the verifier $\mathcal{V}$, is an *interactive argument system* (or *computationally sound proof*) for language $\mathcal{L}$ if there exists a negligible function negl such that

- *Completeness:* $\forall x \in \mathcal{L}\ \exists w : \Pr\left[1 \leftarrow \langle \mathcal{P}(x, w) \leftrightarrow \mathcal{V}(x) \rangle\right] \geq 1 - \mathsf{negl}(|x|)$
- *Soundness:* $\forall x \notin \mathcal{L}\colon$ for every interactive probabilistic polynomial-time algorithm $\mathcal{P}^*$ it holds

$$\Pr\left[1 \leftarrow \langle \mathcal{P}^*(x) \leftrightarrow \mathcal{V}(x) \rangle\right] \leq \mathsf{negl}(|x|).$$

We say $\mathcal{V}$ *accepts* x after interacting with $\mathcal{P}$ if $\mathcal{V}$ outputs 1, and $\mathcal{V}$ *rejects* x otherwise. △

Remark: Restricting the prover $\mathcal{P}$ to probabilistic polynomial-time means that $\mathcal{P}$'s running time is bounded by $p(|x|)$ for some polynomial p and all w. Hence, increasing the length of w does not increase $\mathcal{P}$'s running time bound. ¤

The definitions for interactive proofs and arguments capture the concept of *interactively convincing* another party of the validity of some statement. It requires that for all valid statements accepting interactions can be implemented (completeness) and that all accepting interactions are for valid statements (soundness). But there is no requirement on the strategy the prover executes. For cryptographic applications it is desirable to *hide* as much information as possible from a verifier. This property is captured by the definition of *zero-knowledge*. Since this definition requires the concept of (computationally) indistinguishable probability distributions, this is defined first.

Definition 2.15: Indistinguishability of Probability Distributions - [MY08]
Two probability distribution ensembles $\{X_z\}_{z\in\mathcal{S}}$ and $\{Y_z\}_{z\in\mathcal{S}}$, indexed by strings in some finite set $\mathcal{S}$, are *computationally indistinguishable*, denoted by $\{X_z\}_{z\in\mathcal{S}} \stackrel{c}{\equiv} \{Y_z\}_{z\in\mathcal{S}}$, if for all probabilistic polynomial-time algorithms $\mathcal{D}$ there exists a negligible function negl such that for all $z \in \mathcal{S}$ it holds

$$|\Pr[\mathcal{D}(X_z, z) = 1] - \Pr[\mathcal{D}(Y_z, z) = 1]| \leq \mathsf{negl}(\lambda). \qquad \triangle$$

Definition 2.16: Black-Box Zero-Knowledge - [GMR89; MY08]
An interactive proof system $(\mathcal{P}, \mathcal{V})$ for language $\mathcal{L}$ is *black-box zero-knowledge* if there exists a probabilistic polynomial-time algorithm $\mathcal{S}$, called the *simulator*, such that for every verifier $\mathcal{V}^*$ it holds

$$\left\{\mathrm{VIEW}_{\mathcal{V}^*}(\langle\mathcal{P}(x) \leftrightarrow \mathcal{V}^*(x, z)\rangle)\right\}_{x\in\mathcal{L},\, z\in\{0,1\}^*} \stackrel{c}{\equiv} \left\{\mathcal{S}^{\mathcal{V}^*}(x, z)\right\}_{x\in\mathcal{L},\, z\in\{0,1\}^*}.$$

By $\mathrm{VIEW}_{\mathcal{V}^*}(\langle\mathcal{P}(x) \leftrightarrow \mathcal{V}^*(x, z)\rangle)$ we denote the *computation history* of $\mathcal{V}^*$ consisting of $\mathcal{V}^*$'s inputs, the contents of its random tape and all messages $\mathcal{V}^*$ received from $\mathcal{P}$ during the protocol execution. $\triangle$

Remark: Interactive black-box zero-knowledge argument systems are defined analogously: the only difference to Definition 2.16 is that $\mathcal{P}$ is given the witness w as auxiliary input. The auxiliary input z given to $\mathcal{V}^*$ in Definition 2.16 models additional knowledge about x that $\mathcal{V}^*$ already possesses, for example from previous interactions with $\mathcal{P}$. $\square$

The basic idea of zero-knowledge is that the knowledge a verifier obtains by interacting with a prover can be computed by the verifier itself. This is expressed by the definition of a simulator that is able to produce the verifier's view while being as powerful as the verifier: $\mathcal{S}$ is a probabilistic polynomial-time algorithm *without* access to a witness w. Instead, the computational power of $\mathcal{S}$ comes from the oracle (black-box) access to the verifier. A weaker variant of black-box zero-knowledge is to only consider the honest verifier $\mathcal{V}$ in Definition 2.16, instead of arbitrary verifiers $\mathcal{V}^*$. Interactive proofs and arguments satisfying the weaker notion are called *honest-verifier zero-knowledge*.

With the notions of interactive proofs and arguments the concept of *convincing a verifier of the validity of some statement* is captured. However, these notions do not state explicitly what it means for a prover to *know something* about the statement to prove. This is captured by the notion of *proofs of knowledge*.

Definition 2.17: Proofs of Knowledge - [BG93]
Let $\kappa\colon \{0,1\}^* \to [0,1]$ be a function and $\mathcal{R} \subseteq \{0,1\}^* \times \{0,1\}^*$ a binary relation. Further, define $\mathcal{L}_\mathcal{R} = \{x \mid \exists w : (x,w) \in \mathcal{R}\}$ and $R(x) = \{w \mid (x,w) \in \mathcal{R}\}$.

An interactive probabilistic polynomial-time algorithm $\mathcal{V}$ is a *knowledge verifier for relation* $\mathcal{R}$ *with knowledge error* κ if the following two conditions hold:

- *Non-triviality:* There exists an interactive algorithm $\mathcal{P}$ such that for all $x \in \mathcal{L}_\mathcal{R}$ it holds $\Pr\left[1 \leftarrow \langle \mathcal{P}(x) \leftrightarrow \mathcal{V}(x) \rangle\right] = 1$.
- *Validity:* There exists a probabilistic expected polynomial-time algorithm $\mathcal{E}$ such that for every interactive algorithm $\mathcal{P}^*$ and for all $x \in \mathcal{L}_\mathcal{R}$ it is the case that $\mathcal{E}^{\mathcal{P}^*}(x) \in R(x) \cup \{\perp\}$ and it holds

$$\Pr[\mathcal{E}^{\mathcal{P}^*}(x) \in R(x)] \geq \Pr\left[1 \leftarrow \langle \mathcal{P}^*(x) \leftrightarrow \mathcal{V}(x) \rangle\right] - \kappa(x).$$

The algorithm $\mathcal{E}$ is called a *(universal) extractor* that either outputs a *witness* w for x such that $(x,w) \in \mathcal{R}$, or the special symbol $\perp$. △

As shown by Bellare and Goldreich [BG93], the knowledge error κ can be reduced via m sequential repetitions to essentially κ^m and via parallel repetitions to 0, when κ is small enough. Regardless of this, proofs of knowledge for a relation $\mathcal{R}$ are not necessarily interactive proof systems for $\mathcal{L}_\mathcal{R}$ because soundness is not required. However, an interactive proof system $(\mathcal{P}, \mathcal{V})$ is also a proof of knowledge when $\mathcal{V}$ is a knowledge verifier. By restricting the class of provers in Definition 2.17 to interactive probabilistic polynomial-time provers with auxiliary input one obtains *arguments of knowledge.*

A special class of efficient protocols, combining interactive argument systems, honest-verifier zero-knowledge, and arguments of knowledge, is called Σ-protocols.

Definition 2.18: Σ-Protocol - [Dam02]
Let $\mathcal{R} \subseteq \{0,1\}^* \times \{0,1\}^*$ be a binary relation and define $\mathcal{L}_\mathcal{R} = \{x \mid \exists w : (x,w) \in \mathcal{R}\}$ and $R(x) = \{w \mid (x,w) \in \mathcal{R}\}$.

An interactive argument system $(\mathcal{P}, \mathcal{V})$ for $\mathcal{L}_\mathcal{R}$ is a *Σ-protocol for relation* $\mathcal{R}$ if it is of the form

1. $\mathcal{P}$ sends a message a, the *announcement*, to $\mathcal{V}$.
2. $\mathcal{V}$ sends a random t-bit string ch, the *challenge*, to $\mathcal{P}$.
3. $\mathcal{P}$ sends a message r, the *response*, to $\mathcal{V}$. Then $\mathcal{V}$ accepts or rejects depending on (x, a, ch, r).

and the following three conditions hold:

- *Completeness:* $\forall (x,w) \in \mathcal{R}\colon \Pr\left[1 \leftarrow \langle \mathcal{P}(x,w) \leftrightarrow \mathcal{V}(x) \rangle\right] = 1$
- *Special Soundness:* For all $x \in \{0,1\}^*$ and all pairs of accepting interactions (a, ch, r), (a, ch', r') on input x, where $ch \neq ch'$, it is possible to compute a witness w in probabilistic polynomial-time such that $(x,w) \in \mathcal{R}$.

- *Special Honest-Verifier Zero-Knowledge:* There exists a probabilistic polynomial-time algorithm $\mathcal{S}$ that on input $x \in L_{\mathcal{R}}$ and a uniformly random ch outputs an accepting interaction of the form (a, ch, r) with the same probability distribution as interactions between an honest $\mathcal{P}$ and an honest $\mathcal{V}$ on input x. △

As shown by Damgård [Dam02], completeness and special soundness imply that every Σ-protocol is an interactive argument of knowledge with knowledge error 2^{-t}, where t is the length of the verifier's challenge. The special honest-verifier zero-knowledge property is a stronger notion than honest-verifier black-box zero-knowledge, because it requires the simulator to output accepting interactions that are distributed *identically* to real interactions and that this is possible for *every* challenge ch.

Concurrent Composition of Interactive Protocols

The definitions for interactive proofs (arguments), zero-knowledge and proofs of knowledge only consider the execution of some protocol between two parties. A more realistic scenario is to consider an execution of several protocol instances where messages from different parties are arbitrarily interleaved. The message scheduling is controlled by an adversary under the restriction that all messages are scheduled in the right order. This concurrent interaction is defined in Definition 2.19.

Definition 2.19: Concurrent Interaction - [DNS98; DNS04; Ros06]
The concurrent execution of an interactive two-party protocol $(\mathcal{P}, \mathcal{V})$, denoted by $(\hat{\mathcal{P}}, \hat{\mathcal{V}})$, is controlled by an adversary $\mathcal{A}$ that schedules the messages exchanged between $\hat{\mathcal{P}}$ and $\hat{\mathcal{V}}$. The parties $\hat{\mathcal{P}}$ and $\hat{\mathcal{V}}$ act as follows:

- When $\hat{\mathcal{P}}$ ($\hat{\mathcal{V}}$) receives $(\texttt{init}, m, \mathrm{ID})$ from $\mathcal{A}$, create a new copy of $\mathcal{P}$ ($\mathcal{V}$) and run $\mathcal{P}(m)$ ($\mathcal{V}(m)$). For the i'th `init`-message we denote this copy of $\mathcal{P}$ ($\mathcal{V}$) by $\mathcal{P}_i$ ($\mathcal{V}_i$) and its input by m_i. The identifier ID is used as a label to identify the protocol instance of $(\mathcal{P}, \mathcal{V})$. When $\mathcal{P}$ ($\mathcal{V}$) generates a message m' to send to its corresponding $\mathcal{V}$ ($\mathcal{P}$), $\hat{\mathcal{P}}$ ($\hat{\mathcal{V}}$) sends (m', ID) to $\mathcal{A}$.

- When $\hat{\mathcal{P}}$ ($\hat{\mathcal{V}}$) receives $(\texttt{message}, m, \mathrm{ID})$ from $\mathcal{A}$, hand m to $\mathcal{P}$ ($\mathcal{V}$) as a message sent within $(\mathcal{P}, \mathcal{V})$ associated with ID. If no such associated $\mathcal{P}$ ($\mathcal{V}$) exists, ignore the message. When $\mathcal{P}$ ($\mathcal{V}$) generates a message m' to send to its corresponding $\mathcal{V}$ ($\mathcal{P}$), $\hat{\mathcal{P}}$ ($\hat{\mathcal{V}}$) sends (m', ID) to $\mathcal{A}$.

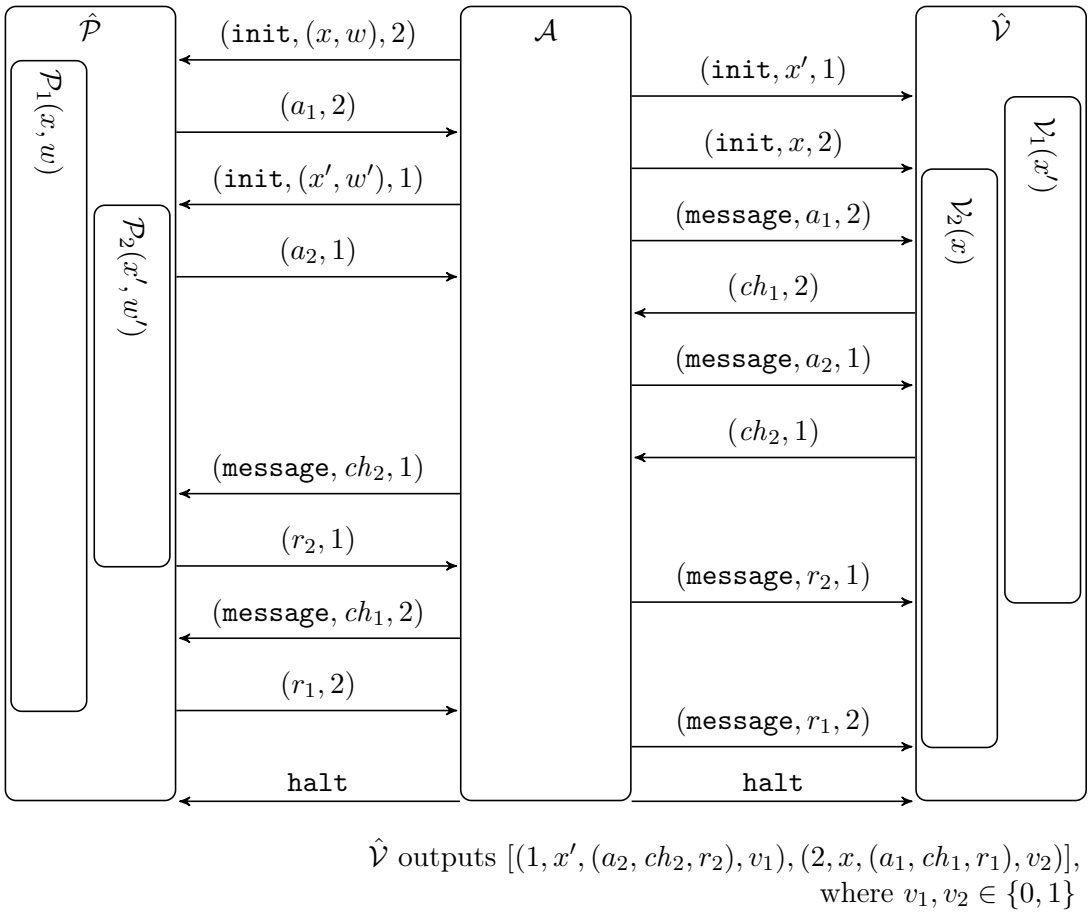
- When $\hat{\mathcal{P}}$ ($\hat{\mathcal{V}}$) receives `halt` from $\mathcal{A}$, stop the execution of all protocol instances and generate the output.

The output of $\hat{\mathcal{P}}$ is the vector of all outputs of the instantiated $\mathcal{P}_i$'s with their associated identifiers, namely $[(\mathrm{ID}_1, p_1), (\mathrm{ID}_2, p_2), \ldots]$, where p_i denotes $\mathcal{P}_i$'s output. The output of $\hat{\mathcal{V}}$ is a vector of all inputs, transcripts and outputs of the instantiated $\mathcal{V}_i$'s with their associated identifiers, namely $[(\mathrm{ID}_1, x_1, t_1, v_1), (\mathrm{ID}_2, x_2, t_2, v_2), \ldots]$, where x_i denotes an input, t_i denotes a complete transcript and v_i denotes $\mathcal{V}_i$'s output. △

Remark: To emphasize that $\hat{\mathcal{P}}$ and $\hat{\mathcal{V}}$ get multiple inputs during a concurrently composed interaction, we denote their inputs as vectors. Concretely, we denote the execution of a concurrently composed protocol on input $\vec{x}$ by $\langle \hat{\mathcal{P}}(\vec{x}) \leftrightarrow \hat{\mathcal{V}}(\vec{x}) \rangle$. ¤

In Definition 2.19 $\mathcal{A}$ models a higher-level process that initiates protocol executions, provides the parties' inputs, and controls an asynchronous network. The actual protocol instances are executed independently of each other and are managed by $\hat{\mathcal{P}}$ and $\hat{\mathcal{V}}$, respectively.

To give an example, consider the possible concurrent execution of two instances of a Σ-protocol in Figure 2.1.



$\hat{\mathcal{V}}$ outputs $[(1, x', (a_2, ch_2, r_2), v_1), (2, x, (a_1, ch_1, r_1), v_2)]$,
where $v_1, v_2 \in \{0, 1\}$

Figure 2.1: Example of two concurrently executed Σ-protocol instances

By considering concurrent interaction according to Definition 2.19 for interactive proof systems and arguments, it is not hard to see that completeness is preserved, because all protocol instances are executed independently. Moreover, if there would exist a composed prover $\hat{\mathcal{P}}^*$ that is able to convince a composed verifier $\hat{\mathcal{V}}$ about some false statement with more than negligible probability, then $\hat{\mathcal{P}}^*$ could be transformed into an ordinary prover $\mathcal{P}^*$ executing an interactive proof (or argument) by emulating the concurrent conditions. Hence, also soundness is preserved under concurrent composition of interactive proofs and

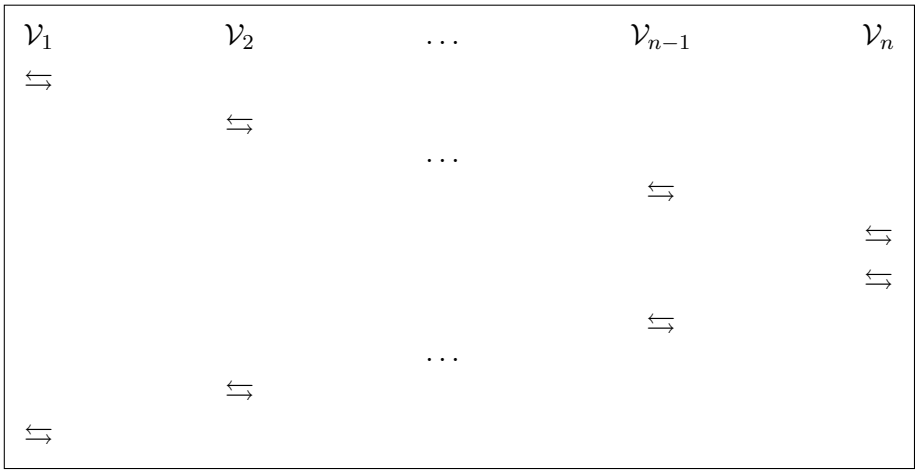
arguments. Unfortunately, a similar statement regarding zero-knowledge protocols does not hold.

Definition 2.20: Concurrent Black-Box Zero-Knowledge - [DNS98; DNS04]
An interactive proof system $(\mathcal{P}, \mathcal{V})$ for language $\mathcal{L}$ is *concurrent black-box zero-knowledge* if there exists a probabilistic polynomial-time algorithm $\mathcal{S}$, called the *simulator*, such that for every composed verifier $\hat{\mathcal{V}}^*$ it holds

$$\left\{\mathrm{VIEW}_{\hat{\mathcal{V}}^*}(\langle \hat{\mathcal{P}}(\vec{x}) \leftrightarrow \hat{\mathcal{V}}^*(\vec{x}, z)\rangle)\right\}_{\vec{x}\in\mathcal{L}^*,\, z\in\{0,1\}^*} \stackrel{c}{\equiv} \left\{\mathcal{S}^{\hat{\mathcal{V}}^*}(\vec{x}, z)\right\}_{\vec{x}\in\mathcal{L}^*,\, z\in\{0,1\}^*}.$$

By $\mathrm{VIEW}_{\hat{\mathcal{V}}^*}(\langle \hat{\mathcal{P}}(\vec{x}) \leftrightarrow \hat{\mathcal{V}}^*(\vec{x}, z)\rangle)$ we denote the *computation history* of $\hat{\mathcal{V}}^*$ consisting of $\hat{\mathcal{V}}^*$'s inputs, the contents of its random tape and all messages $\hat{\mathcal{V}}^*$ received from $\hat{\mathcal{P}}$ during the protocol execution. △

The definition of concurrent black-box zero-knowledge considers composed verifiers $\hat{\mathcal{V}}^*$ that are able to influence the message scheduling. This is because $\hat{\mathcal{V}}^*$ is not forced to respond immediately to messages from $\hat{\mathcal{P}}$. Furthermore, $\hat{\mathcal{V}}^*$ could send messages to $\hat{\mathcal{P}}$ that depend on messages received earlier in other protocol instances. An example of a problematic message scheduling, originally suggested by [DNS98] [DNS98], is given in Figure 2.2.



An arrow from the right to the left indicates a message sent from the verifier $\mathcal{V}_i$ to the prover $\hat{\mathcal{P}}$, whereas an arrow from the left to the right indicates a message sent from the prover $\hat{\mathcal{P}}$ to the verifier $\mathcal{V}_i$. Since $\hat{\mathcal{V}}^*$ controls all $\mathcal{V}_i$'s, the messages sent by $\mathcal{V}_i$ can depend on all previous messages.

Figure 2.2: Example of a message schedule for n instances of a 4-move interactive protocol

To simulate the interaction shown in Figure 2.2 the straight-forward approach of simulating each of the n instances as in the case without concurrency does not work. The problem that arises is that simulating an instance could influence the simulation of another instance, due to the possible dependency of messages. This results in an exponential running time for the straight-forward simulator to simulate instance 1, as shown in more detail by Dwork, Naor, and Sahai [DNS98]. Hence, zero-knowledge is not preserved under concurrent composition.

Fortunately, for Σ-protocols Damgård [Dam00] shows how obtain concurrent black-box zero-knowledge in the auxiliary string model by combining them with trapdoor commitments. In the auxiliary string model, also known as the *common reference string model* [CF01], an auxiliary string with a predefined distribution is given as an additional input to all parties involved in a protocol. It is often assumed that a trusted third party chooses and publishes that string. Damgård's technique to construct concurrent black-box zero-knowledge arguments from Σ-protocols and trapdoor commitment schemes works as follows.

Lemma 2.1: Damgård's Technique - [Dam00]
Let pk be the public key of a secure trapdoor commitment scheme and let $(\mathcal{P}, \mathcal{V})$ be a Σ-protocol for some relation $\mathcal{R}$. Then the following protocol $(\mathcal{P}', \mathcal{V}')$ is a concurrent black-box zero-knowledge argument of knowledge.

1: On input (x, w, pk), where $(x, w) \in \mathcal{R}$, the prover $\mathcal{P}'$ computes a by executing $\mathcal{P}(x, w)$, computes $(c, d) \leftarrow \mathsf{Commit}(pk, a)$, and sends c to the verifier $\mathcal{V}'$.
2: On input (x, pk) the verifier $\mathcal{V}'$ sends a random challenge ch to $\mathcal{P}'$.
3: The prover $\mathcal{P}'$ hands the challenge ch to $\mathcal{P}$ to obtain r as the provers response and sends (r, a, d) to $\mathcal{V}'$.
4: The verifier $\mathcal{V}'$ accepts x, if and only if $\mathcal{V}$ accepts (x, a, ch, r) and $a = \mathsf{Reveal}(pk, c, d)$.

We call this composed protocol a $\hat{\Sigma}$-protocol. ◇

Sketch of Proof. We have to show that the protocol is complete, sound, concurrent black-box zero-knowledge, and an argument of knowledge.

Completeness follows from the completeness of the underlying Σ-protocol and the correctness of the trapdoor commitment scheme. Analogously, soundness follows from the soundness of the Σ-protocol. To prove that the $\hat{\Sigma}$-protocol is concurrent black-box zero-knowledge we define the following simulator $\mathcal{S}$ that interacts with a concurrently composed verifier $\hat{\mathcal{V}}^*$:

1: On input $\vec{x}$ the simulator $\mathcal{S}$ runs the KeyGen algorithm of the trapdoor commitment scheme to obtain (pk, td) and hands $(\vec{x}, pk)$ to $\hat{\mathcal{V}}^*$.
2: For all $i \in \{1, \ldots, |\vec{x}|\}$, $\mathcal{S}$ runs $\mathsf{TCommit}(pk, td)$ to obtain $(\hat{c}_i, ek_i)$ and sends $\hat{c}_i$ to $\hat{\mathcal{V}}^*$.
3: When $\mathcal{S}$ receives a challenge ch_i from $\hat{\mathcal{V}}^*$, he runs the special honest-verifier zero-knowledge simulator of the Σ-protocol on input (x_i, ch_i) to obtain an accepting transcript (a_i, ch_i, r_i). Then $\mathcal{S}$ runs $\mathsf{TReveal}(pk, \hat{c}_i, ek_i, a_i)$ to obtain a decommitment $\hat{d}_i$ and sends $(r_i, a_i, \hat{d}_i)$ to $\hat{\mathcal{V}}^*$.

Since the trapdoor commitment scheme is secure, especially the trapdoor property is fulfilled, transcripts generated by $\mathcal{S}$ are indistinguishable from real protocol transcripts. Furthermore, $\mathcal{S}$ can instantly react on challenges from $\hat{\mathcal{V}}^*$ such that no rewinding is needed. Hence, the $\hat{\Sigma}$-protocol is concurrent black-box zero-knowledge. That the $\hat{\Sigma}$-protocol is also an argument of knowledge follows from binding property of the trapdoor commitment scheme and the special soundness of the underlying Σ-protocol. □

2.2.7 Non-Interactive Proofs

The purpose of non-interactive proofs is basically the same as for their interactive counterpart. They enable a party $\mathcal{P}$ to convince another party $\mathcal{V}$ that some statement is true, but *without interacting with* $\mathcal{V}$. Naturally, the question of whether non-interactive proofs can be zero-knowledge or proofs of knowledge arises. Indeed this is true, albeit the missing interaction needs to be compensated. For this reason Blum, Feldman, and Micali [BFM88] introduce a common reference string in their definition for non-interactive proof systems that replaces interaction to support zero-knowledge. Unfortunately, this definition only allows to prove a single statement for a fixed common references string. A definition supporting polynomially many proofs is given by Feige, Lapidot, and Shamir [FLS99].

Definition 2.21: Non-interactive Argument System - [FLS99; GOS06]
Let $\mathcal{R} \subseteq \{0,1\}^* \times \{0,1\}^*$ be a binary $\mathcal{NP}$-relation and define $\mathcal{L}_\mathcal{R} = \{x \mid \exists w : (x,w) \in \mathcal{R}\}$. A non-interactive argument system Π consists of three probabilistic polynomial-time algorithms:

$\mathsf{KeyGen}(1^\lambda)$ takes as input the security parameter. It outputs a common reference string crs.

$\mathcal{P}(crs, x, w)$ takes as input a common reference string crs and a tuple $(x,w) \in \{0,1\}^* \times \{0,1\}^*$. It outputs an argument π or the error symbol $\perp$.

$\mathcal{V}(crs, x, \pi)$ takes as input a common reference string crs, a bitstring $x \in \{0,1\}^*$, and an argument π. It outputs a bit $b \in \{0,1\}$.

We call Π a *non-interactive argument system for relation* $\mathcal{R}$, if there exists a negligible function negl such that

- *Completeness:* for every (unbounded) algorithm $\mathcal{A}$ it holds

$$\Pr\left[crs \leftarrow \mathsf{KeyGen}(1^\lambda), (x,w) \leftarrow \mathcal{A}(crs), \pi \leftarrow \mathcal{P}(crs,x,w) : \mathcal{V}(crs,x,\pi) = 1\right] = 1,$$

  if $(x,w) \in \mathcal{R}$.

- *Soundness:* for every probabilistic polynomial-time algorithm $\mathcal{P}^*$ it holds

$$\Pr\left[crs \leftarrow \mathsf{KeyGen}(1^\lambda), (x,\pi) \leftarrow \mathcal{P}^*(crs) : \mathcal{V}(crs,x,\pi) = 1\right] \leq \mathsf{negl}(\lambda),$$

  if $x \notin \mathcal{L}_\mathcal{R}$.

We say $\mathcal{V}$ *accepts* π *for* x if $\mathcal{V}$ outputs 1, and $\mathcal{V}$ *rejects* π *for* x otherwise. △

Remark: Analogously to interactive proofs and arguments, we call Π a *non-interactive proof system*, when the soundness conditions holds for unbounded provers $\mathcal{P}^*$. In this case the output π of the prover $\mathcal{P}$ is called a *proof*, rather than an argument. ¤

To define zero-knowledge for non-interactive proofs (arguments) it is important to note that the same common reference string should be used for polynomially many proofs. Furthermore, depending on the application a non-interactive proof system is used in, it may be possible that an adversary is able to query proofs for adaptively chosen statements. A definition capturing this adversarial behavior is called *adaptive multi-theorem zero-knowledge*.

Definition 2.22: Adaptive Multi-Theorem Zero-Knowledge - [FLS99; GOS06]
A non-interactive proof system Π for an $\mathcal{NP}$-relation $\mathcal{R}$ is *adaptively multi-theorem zero-knowledge*, if there exists a probabilistic polynomial-time algorithm $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$, called the *simulator*, such that for every probabilistic polynomial-time algorithm $\mathcal{V}^*$ there exists a negligible function negl such that

$$\Pr[\mathrm{Exp}^{zk}_{\mathcal{V}^*,\Pi}(1^\lambda) = 1] \leq \frac{1}{2} + \mathsf{negl}(\lambda),$$

where the experiment $\mathrm{Exp}^{zk}_{\mathcal{V}^*,\Pi}$ is defined as follows:

$\mathrm{Exp}^{zk}_{\mathcal{V}^*,\Pi}(1^\lambda)$
1: $b \xleftarrow{u} \{0,1\}$
2: **If** $b = 0$ **Then** $crs \leftarrow \mathsf{KeyGen}(1^\lambda)$
3: **Else** $(crs, td) \leftarrow \mathcal{S}_1(1^\lambda)$
4: $b' \leftarrow \mathcal{V}^{*\mathcal{O}_b}(crs)$
5: **If** $b = b'$ **Then**
6: output 1
7: **Else** output 0

$\mathcal{O}_0(x, w)$
1: $\pi \leftarrow \mathcal{P}(crs, x, w)$
2: return π

$\mathcal{O}_1(x, w)$
1: **If** $(x, w) \notin \mathcal{R}$ **Then**
2: return $\perp$
3: **Else** $\pi \leftarrow \mathcal{S}_2(crs, td, x)$
4: return π

In this experiment the malicious verifier $\mathcal{V}^*$ can query an oracle that either outputs proofs generated by an honest prover $\mathcal{P}$ or by the simulator $\mathcal{S}$ using a *simulator trapdoor* td. △

This definition basically states that simulated and real proofs are computational indistinguishable, but it also implies that common reference strings generated by KeyGen are indistinguishable from simulated ones. A similar approach is used to define non-interactive zero-knowledge (NIZK) proofs of knowledge.

Definition 2.23: NIZK Proof of Knowledge - [De +01]
A non-interactive zero-knowledge proof system Π for an $\mathcal{NP}$-relation $\mathcal{R}$ is a *proof of knowledge*, if there exists a probabilistic polynomial-time algorithm $\mathcal{E} = (\mathcal{E}_1, \mathcal{E}_2)$, called the *extractor*, such that the following conditions hold:

- *Uniformity:* for every algorithm $\mathcal{A}$ it holds

$$\Pr[crs \leftarrow \mathsf{KeyGen}(1^\lambda) \colon \mathcal{P}^*(crs) = 1] = \Pr[(crs, td) \leftarrow \mathcal{E}_1(1^\lambda) \colon \mathcal{P}^*(crs) = 1]$$

- *Extractability:* for all algorithms $\mathcal{P}^*$ there exists a negligible function negl such that

$$\begin{aligned}&\Pr[(crs, td) \leftarrow \mathcal{E}_1(1^\lambda), (x, \pi) \leftarrow \mathcal{P}^*(crs), w \leftarrow \mathcal{E}_2(crs, td, x, \pi) \colon (x, w) \in \mathcal{R}] \\ &\quad \geq \Pr[crs \leftarrow \mathsf{KeyGen}(1^\lambda), (x, \pi) \leftarrow \mathcal{P}^*(crs) \colon \mathcal{V}(crs, x, \pi) = 1] - \mathsf{negl}(\lambda).\end{aligned}$$

△

When considering proofs of knowledge that are also zero-knowledge, simulated proofs should not enable an adversary to prove false statements. This property is not covered by Definition 2.23, since the malicious prover $\mathcal{P}^*$ has no access to simulated proofs. The notion of *simulation sound extractability* extends proofs of knowledge in that effect.

Definition 2.24: Simulation Sound Extractability - [Gro06]
A non-interactive adaptive multi-theorem zero-knowledge proof of knowledge Π for an $\mathcal{NP}$-relation $\mathcal{R}$ is *simulation sound extractable*, if there exists a probabilistic polynomial-time algorithm $\mathcal{SE}$ such that for every probabilistic polynomial-time adversary $\mathcal{A}$ there exists a negligible function negl such that

$$\Pr[\mathrm{Exp}^{sse}_{\mathcal{A},\Pi}(1^\lambda) = 1] \leq \mathsf{negl}(\lambda),$$

where the experiment $\mathrm{Exp}^{sse}_{\mathcal{A},\Pi}$ is defined as follows:

$\mathrm{Exp}^{sse}_{\mathcal{A},\Pi}(1^\lambda)$
1: $(crs, td_s, td_e) \leftarrow \mathcal{SE}(1^\lambda)$
2: $\mathcal{Q} := \emptyset$
3: $(x, \pi) \leftarrow \mathcal{A}^{\mathcal{S}(\cdot)}(crs)$
4: $w \leftarrow \mathcal{E}_2(crs, td_e, x, \pi)$
5: **If** $(x, \pi) \notin \mathcal{Q}$
$\wedge\, (x, w) \notin \mathcal{R}$
$\wedge\, \mathcal{V}(crs, x, \pi) = 1$ **Then**
6: return 1
7: **Else** return 0

$\mathcal{S}(x)$
1: $\pi \leftarrow \mathcal{S}_2(crs, td_s, x)$
2: $\mathcal{Q} := \mathcal{Q} \cup \{(x, \pi)\}$
3: return π

In this experiment $\mathcal{SE}$ is a probabilistic polynomial-time algorithm that outputs a common reference string crs and corresponding trapdoors td_s and td_e for simulation and extraction. The probabilistic polynomial-time algorithms $\mathcal{S}_2$ and $\mathcal{E}_2$ are the second-stage simulator from Definition 2.22 and the second-stage extractor from Definition 2.23, respectively. △

Remark: In this definition the adversary has direct access to the simulator $\mathcal{S}_2$ and can query proofs for any $x \in \{0,1\}^*$. ¤

Both definitions of witness extraction consider extractors that generate common reference strings together with appropriate extraction trapdoors. This is because rewinding a prover, as for interactive proofs of knowledge, does not seem to be helpful, since there is no interaction an extractor could use to ask „tricky“ questions. Interestingly, this problem does not arise for proofs of knowledge in the random oracle model. The most popular technique to construct zero-knowledge proofs of knowledge in the random oracle model is the Fiat-Shamir heuristic.

The Fiat-Shamir Heuristic

The Fiat-Shamir heuristic, introduced by Fiat and Shamir [FS87] and refined by Bernhard, Pereira, and Warinschi [BPW12], is used to transform Σ-protocols into non-interactive

simulation sound extractable proof systems that are secure in the random oracle model. It is important to note that in the random oracle model no common reference string exists. Hence, the definitions for non-interactive proof systems, zero-knowledge, and proofs of knowledge differ from those in the common reference string model.

Informally, non-interactive proof systems and zero-knowledge proofs in the random oracle model are defined as in Definitions 2.21 and 2.22 by removing the KeyGen algorithm, the common reference string crs, and the simulation trapdoor td. The simulator is responsible to answer queries to the random oracle and to generate valid proofs for any statement, even for false statements. Since the simulator controls the random oracle, he is able to *patch* it as needed to generate valid proofs. For proofs of knowledge in the random oracle model also no common reference string exists and hence no extraction trapdoor. But the extractor can rewind provers and controls the random oracle. Thereby, its behavior is more comparable to the extractor from Definition 2.17 of interactive proofs of knowledge, when the interaction is emulated via the random oracle. Since the prover must query the random oracle to generate a valid proof, the extractor can answer differently to queries in the first run and the rewound runs of a prover. Formal definitions for non-interactive proofs, zero-knowledge, and simulation sound extractability are given in [BPW12].

Definition 2.25: Fiat-Shamir Transformation - [FS87; BPW12]
Let $(\mathcal{P}_\Sigma, \mathcal{V}_\Sigma)$ be a Σ-protocol and $\mathcal{H}$ a collision-resistant hash function. The Fiat-Shamir transformation of $(\mathcal{P}_\Sigma, \mathcal{V}_\Sigma)$ is the non-interactive proof system $\mathrm{FS}_\mathcal{H}(\mathcal{P}_\Sigma, \mathcal{V}_\Sigma) = (\mathcal{P}, \mathcal{V})$, defined as follows:

$\mathcal{P}(x, w)$
1: run $\mathcal{P}_\Sigma(x, w)$ to obtain a
2: compute $ch \coloneqq \mathcal{H}(x, a)$
3: finish the run of $\mathcal{P}_\Sigma$ on ch to obtain r
4: output $\pi \coloneqq (a, ch, r)$

$\mathcal{V}(x, \pi)$
1: **If** $ch = \mathcal{H}(x, a)$ **Then**
2: output $(\mathcal{V}_\Sigma(x, a, ch, r) = 1)$
3: **Else** output 0

△

The zero-knowledge simulator $\mathcal{S}$ for this transformed Σ-protocol works as follows. On input x, $\mathcal{S}$ chooses a random challenge ch as an honest verifier would do and runs the special honest-verifier zero-knowledge simulator $\mathcal{S}_\Sigma(x, ch)$ to obtain an accepting transcript (a, ch, r). Then $\mathcal{S}$ patches the random oracle $\mathcal{H}$ such that $\mathcal{H}(x, a) \coloneqq ch$. It is possible that during the patching a collision occurs, but this happens only with negligible probability.

The knowledge extractor $\mathcal{E}$ for $(\mathcal{P}, \mathcal{V})$ runs a prover $\mathcal{P}^*$ to obtain an accepting transcript (a, ch, r) for some statement x, rewinds $\mathcal{P}^*$ up to the point where the random oracle was queried for (x, a) by $\mathcal{P}^*$, and lets the random oracle output a different value ch' for the same query (x, a) to obtain another accepting transcript (a, ch', r'). By running the special soundness extractor $\mathcal{E}_\Sigma$ from the Σ-protocol, $\mathcal{E}$ obtains a witness w such that $(x, w) \in \mathcal{R}$.

The Fiat-Shamir heuristic is often used to construct *signatures of knowledge* [CL06; GM17]. These are digital signatures proving knowledge of the secret signing key that corresponds to the public key of the scheme. Before presenting the construction of signatures of knowledge from Σ-protocols, we need to introduce hard relations.

Definition 2.26: Hard Relation - [Dam02]
Let $\mathcal{R} \subseteq \{0,1\}^* \times \{0,1\}^*$ be a binary $\mathcal{NP}$-relation. We say the *relation* $\mathcal{R}$ *is hard*, if the following conditions hold:

- There exists a probabilistic polynomial-time algorithm $\mathcal{G}$, called the *generator*, that on input 1^λ outputs a pair $(x, w) \in \mathcal{R}$ such that $|x| = \lambda$.
- For all probabilistic polynomial-time algorithms $\mathcal{A}$ there exists a negligible function negl, such that

$$\Pr[(x, w) \leftarrow \mathcal{G}(1^\lambda), w' \leftarrow \mathcal{A}(x) : (x, w') \in \mathcal{R}] \leq \mathsf{negl}(\lambda).$$ △

When $\mathcal{R}$ is a hard relation with generator $\mathcal{G}$, $(\mathcal{P}, \mathcal{V})$ is a non-interactive proof system obtained from the Fiat-Shamir transformation of a Σ-protocol for $\mathcal{R}$, and $\mathcal{H}$ is a collision-resistant hash function, the following construction is a signature of knowledge:

KeyGen(1^λ): run $\mathcal{G}(1^\lambda)$ to obtain $(x, w) \in \mathcal{R}$ and output $(pk \coloneqq x, sk \coloneqq w)$

Sign(sk, m): run $\mathcal{P}(pk, sk)$ to obtain the proof π, but compute the challenge by setting $ch \coloneqq \mathcal{H}(m, pk, a)$, and output the signature $\sigma \coloneqq \pi$

Verify(pk, m, σ): run $\mathcal{V}(pk, \sigma)$, but compute the challenge by setting $ch \coloneqq \mathcal{H}(m, pk, a)$ and output 1, if and only if $\mathcal{V}$ accepts.

The correctness of this signature scheme follows from the correctness of the underlying Σ-protocol. Due to the simulation sound extractability of $(\mathcal{P}, \mathcal{V})$ the signature scheme is also existentially unforgeable under chosen message attacks. In the proof of this statement a potential forger can query signatures for messages of his choice and the Sign-oracle responds with simulated proofs. If this forger is then able to output a valid forgery σ_1 for some message m, the extractor is invoked to obtain a second forgery σ_2 for the same message. From these two forgeries it is possible to compute a witness w' such that $(x, w') \in \mathcal{R}$, due to the special soundness property of the Σ-protocol. Since the relation $\mathcal{R}$ is assumed to be hard, the forger can output the forgery σ_1 only with negligible probability. Hence, the signature scheme is existentially unforgeable under a chosen message attack.

A general proof for the security of signatures of knowledge in the random oracle model is given by Pointcheval and Stern [PS00], called the *Forking Lemma*. Unfortunately, generating the second forgery needed to compute a witness is only possible in *expected polynomial runtime*. An improved version of the Forking Lemma is given by Bellare and Neven [BN06], called the *General Forking Lemma*, that provides worst-case guarantees.

Lemma 2.2: General Forking Lemma - [BN06]
Fix $q \in \mathbb{N}$ and a set H of size $h \coloneqq |H| \geq 2$. Let $\mathbf{IG}$ be a probabilistic algorithm, called the *input generator*, and let $\mathbf{A}$ be a probabilistic algorithm that on input $(x, h_1, \ldots, h_q)$ returns a tuple (J, St), where x is the output of $\mathbf{IG}$, $h_1, \ldots, h_q \in H$, $J \in \{0, \ldots, q\}$ and St is some state information. Further let

$$\varepsilon_{\mathbf{A}} \coloneqq \Pr[x \leftarrow \mathbf{IG}, h_1, \ldots, h_q \overset{\pi}{\leftarrow} H, (J, \mathrm{St}) \leftarrow \mathbf{A}(x, h_1, \ldots, h_q) : J \geq 1]$$

and the *forking algorithm* $\mathbf{F}_{\mathbf{A}}$ associated to $\mathbf{A}$ be the probabilistic algorithm that takes x as input and proceeds as follows:

$\mathbf{F}_{\mathbf{A}}(x)$
1: Pick random bits ω for $\mathbf{A}$
2: $h_1, \ldots, h_q \xleftarrow{u} H$
3: $(I, \mathrm{St}) \leftarrow \mathbf{A}(x, h_1, \ldots, h_q; \omega)$
4: **If** $I = 0$ **Then** return $(0, \perp, \perp)$
5: $h'_I, \ldots, h'_q \xleftarrow{u} H$
6: $(I', \mathrm{St}') \leftarrow \mathbf{A}(x, h_1, \ldots, h_{I-1}, h'_I, \ldots, h'_q; \omega)$
7: **If** $I = I' \wedge h_I \neq h'_I$ **Then** return $(1, \mathrm{St}, \mathrm{St}')$
8: **Else** return $(0, \perp, \perp)$

Define the success probability of $\mathbf{F}_{\mathbf{A}}$ as

$$\varepsilon_{\mathbf{F}_{\mathbf{A}}} := \Pr[x \leftarrow \mathbf{IG}, (b, \mathrm{St}, \mathrm{St}') \leftarrow \mathbf{F}_{\mathbf{A}}(x) : b = 1].$$

Then it holds

$$\varepsilon_{\mathbf{F}_{\mathbf{A}}} \geq \varepsilon_{\mathbf{A}} \cdot \left(\frac{\varepsilon_{\mathbf{A}}}{q} - \frac{1}{h} \right). \qquad \diamond$$

Applying this lemma to signatures of knowledge leads to the following matching:

- q is the number of queries to a random oracle $\mathcal{H}$,
- H is the image space of the random oracle $\mathcal{H}$,
- **IG** is the generator $\mathcal{G}$ of the hard relation $\mathcal{R}$,
- x is the statement generated by $\mathcal{G}$,
- the h_i's are the queried hash values,
- $\mathbf{A}$ reduces the security of the signature scheme to the hardness of $\mathcal{R}$,
- St and St$'$ are a forged signatures,
- and J is the index of the hash query needed to compute the forgery.

When $\mathbf{F}_{\mathbf{A}}$ outputs $(1, \mathrm{St}, \mathrm{St}')$, the special soundness extractor can be used to compute a witness w such that $(x, w) \in \mathcal{R}$.

It is not hard to see that $\mathbf{F}_{\mathbf{A}}$ is a probabilistic polynomial-time algorithm, when $\mathbf{A}$ is probabilistic polynomial-time, since $\mathbf{F}_{\mathbf{A}}$ invokes $\mathbf{A}$ only twice. Hence, this lemma can be used when hard relations are considered with respect to *strict* probabilistic polynomial-time rather than *expected* probabilistic polynomial-time.

2.3 Group Generators and Bilinear Maps

In this work we define security properties with respect to prime order groups and their corresponding group generators.

Definition 2.27: Group Generator - [KL07]
A group generator, denoted by GrGen, is a probabilistic polynomial-time algorithm that, on input 1^λ, $\lambda \in \mathbb{N}$, outputs a description of a group $\mathbb{GD} = (p, \mathbb{G}, g)$, where p is the prime order of the cyclic group $\mathbb{G}$, $|p| = \lambda$, and $g \in \mathbb{G}$ is a generator of $\mathbb{G}$. △

Before defining bilinear group generators we review some concepts related to bilinear maps, following the notation of [BLS04].

Definition 2.28: Bilinear Groups - [BLS04]
Let $\mathbb{G}_1, \mathbb{G}_2$, and $\mathbb{G}_T$ be cyclic groups of prime order p. A *bilinear map* $\mathrm{e}\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a map with the following properties:

1. Bilinearity: for all $u \in \mathbb{G}_1$, $v \in \mathbb{G}_2$ and $a, b \in \mathbb{Z}_p$: $\mathrm{e}(u^a, v^b) = \mathrm{e}(u, v)^{ab}$.
2. Non-degeneracy: for $u \neq 1_{\mathbb{G}_1}$ and $v \neq 1_{\mathbb{G}_2}$: $\mathrm{e}(u, v) \neq 1_{\mathbb{G}_T}$.

Further, we call $(\mathbb{G}_1, \mathbb{G}_2)$ a *bilinear group pair*, if the group operations in $\mathbb{G}_1, \mathbb{G}_2$, and $\mathbb{G}_T$, and the map e are efficiently computable. The bilinear map e is also called a *pairing*. △

According to [GPS08], pairings over prime order groups are categorized into three types:

Type-1: The symmetric pairing setting where $\mathbb{G}_1 = \mathbb{G}_2$.

Type-2: An asymmetric pairing setting where $\mathbb{G}_1 \neq \mathbb{G}_2$, but there exists an efficiently computable isomorphism $\phi\colon \mathbb{G}_2 \to \mathbb{G}_1$.

Type-3: An asymmetric pairing setting where $\mathbb{G}_1 \neq \mathbb{G}_2$, but there does not exist an efficiently computable isomorphism.

Throughout this work only Type-2 and Type-3 pairings will be used, which are generated by a *bilinear group generator*. As a convention, for Type-1 pairings we assume that $g_1 = g_2$ holds, whereas $g_1 = \phi(g_2)$ holds in the Type-2 case.

Definition 2.29: Bilinear Group Generator
A bilinear group generator, denoted by BiGrGen, is a probabilistic polynomial-time algorithm that, on input 1^λ, $\lambda \in \mathbb{N}$, outputs a description of bilinear groups $\mathbb{GD} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathrm{e}, g_1, g_2)$, where p is the prime order of the groups $\mathbb{G}_1, \mathbb{G}_2$, and $\mathbb{G}_T$, $|p| = \lambda$, e is the pairing, $g_1 \in \mathbb{G}_1$ is a generator of $\mathbb{G}_1$, and $g_2 \in \mathbb{G}_2$ is a generator of $\mathbb{G}_2$. For Type-2 pairings BiGrGen also outputs the isomorphism ϕ. △

Note that bilinear group generators can be used as group generators according to Definition 2.27 by reducing the output to $(p, \mathbb{G}_1, g_1)$, $(p, \mathbb{G}_2, g_2)$, or $(p, \mathbb{G}_T, \mathrm{e}(g_1, g_2))$.

2.4 Cryptographic Hardness Assumptions

In this section we give the formal definitions of all cryptographic hardness assumptions our constructions rely on.

2.4.1 Decisional Assumptions

The decisional assumptions introduced in this section are mainly, but not exclusively, used to build public-key encryption schemes. Depending on the pairing type we have to consider different but related assumptions.

Definition 2.30: Decision Diffie-Hellman – DDH [Bon98]
Given a group description $\mathbb{GD} = (p, \mathbb{G}, g)$ and a tuple $(g^\alpha, g^\beta, g^\gamma) \in \mathbb{G}^3$, where $\alpha, \beta \xleftarrow{u} \mathbb{Z}_p$, the Decision Diffie-Hellman Problem is to decide whether $\gamma = \alpha \cdot \beta$ or $\gamma \xleftarrow{u} \mathbb{Z}_p$.

We say the Decision Diffie-Hellman Assumption holds for group generator GrGen if for all probabilistic polynomial-time adversaries $\mathcal{A}$ there exists a negligible function negl such that

$$\left|\Pr\left[\mathcal{A}\left(\mathbb{GD}, g^\alpha, g^\beta, g^{\alpha\cdot\beta}\right) = 1\right] - \Pr\left[\mathcal{A}\left(\mathbb{GD}, g^\alpha, g^\beta, g^\gamma\right) = 1\right]\right| \leq \mathsf{negl}(\lambda),$$

where $\mathbb{GD} \leftarrow \mathsf{GrGen}(1^\lambda)$, and $\alpha, \beta, \gamma \xleftarrow{u} \mathbb{Z}_p$. The probability is taken over the random bits used by GrGen, the uniform random choices of $\alpha, \beta, \gamma \xleftarrow{u} \mathbb{Z}_p$ and the random bits used by $\mathcal{A}$. △

For Type-3 pairings it is believed that the Decision Diffie-Hellman Problem is hard in both $\mathbb{G}_1$ and $\mathbb{G}_2$. This assumption is often referred to as the Symmetric External Diffie-Hellman Assumption (SXDH).

Definition 2.31: Symmetric External Diffie-Hellman – SXDH [GSW10]
We say the Symmetric External Diffie-Hellman Assumption holds for bilinear group generator $\mathsf{BiGrGen}$ if for all probabilistic polynomial-time adversaries $\mathcal{A}$ and $\mathcal{B}$ there exist negligible functions $\mathsf{negl}_\mathcal{A}$ and $\mathsf{negl}_\mathcal{B}$ such that

$$\left|\Pr\left[\mathcal{A}\left(\mathbb{GD}, g_1^\alpha, g_1^\beta, g_1^{\alpha\cdot\beta}\right) = 1\right] - \Pr\left[\mathcal{A}\left(\mathbb{GD}, g_1^\alpha, g_1^\beta, g_1^\gamma\right) = 1\right]\right| \leq \mathsf{negl}_\mathcal{A}(\lambda)$$

and

$$\left|\Pr\left[\mathcal{B}\left(\mathbb{GD}, g_2^\delta, g_2^\varepsilon, g_2^{\delta\cdot\varepsilon}\right) = 1\right] - \Pr\left[\mathcal{B}\left(\mathbb{GD}, g_2^\delta, g_2^\varepsilon, g_2^\omega\right) = 1\right]\right| \leq \mathsf{negl}_\mathcal{B}(\lambda)$$

hold, where $\mathbb{GD} \leftarrow \mathsf{BiGrGen}(1^\lambda)$ and $\alpha, \beta, \gamma, \delta, \varepsilon, \omega \xleftarrow{u} \mathbb{Z}_p$. The probability is taken over the random bits used by $\mathsf{BiGrGen}$, the uniform random choices of $\alpha, \beta, \gamma, \delta, \varepsilon, \omega \xleftarrow{u} \mathbb{Z}_p$, and the random bits used by $\mathcal{A}$ and $\mathcal{B}$, respectively. △

Note that the SXDH Assumption does not hold for Type-1 and Type-2 pairings:

Type-1: Since $\mathbb{G}_1 = \mathbb{G}_2$, on input $(\mathbb{GD}, A, B, C)$, adversary $\mathcal{A}$ can compute $\mathrm{e}(A, B)$ and compare it to $\mathrm{e}(g_1, C)$. If both are equal he knows that $C = g_1^{\alpha\cdot\beta}$, otherwise C is a random group element. An analogous argument can be given for $\mathcal{B}$.

Type-2: Using the isomorphism $\phi : \mathbb{G}_2 \to \mathbb{G}_1$, on input $(\mathbb{GD}, A, B, C)$, adversary $\mathcal{B}$ can compute $\mathrm{e}(\phi(A), B)$ and compare it to $\mathrm{e}(\phi(g_2), C)$. As in the Type-1 setting, if both are equal $\mathcal{B}$ knows that $C = g_2^{\delta \cdot \varepsilon}$, otherwise C is a random group element.

Hence, in these settings cryptographic primitives relying on the hardness of Decision Diffie-Hellman can not be used. For this reason, Boneh, Boyen and Shacham introduce the Decision Linear Problem [BBS04].

Definition 2.32: Decision Linear – DLIN [BBS04]
Given a group description $\mathbb{GD} = (p, \mathbb{G}, g)$ and a tuple $(u, v, w, u^\alpha, v^\beta, w^\gamma) \in \mathbb{G}^6$, where $\alpha, \beta \xleftarrow{u} \mathbb{Z}_p$, the Decision Linear Problem is to decide whether $\gamma = \alpha + \beta$ or $\gamma \xleftarrow{u} \mathbb{Z}_p$.

We say the Decision Linear Assumption holds for group generator GrGen if for all probabilistic polynomial-time adversaries $\mathcal{A}$ there exists a negligible function negl such that

$$\left| \begin{array}{l} \Pr\left[\mathcal{A}(\mathbb{GD}, u, v, w, u^\alpha, v^\beta, w^{\alpha+\beta}) = 1\right] \\ -\Pr\left[\mathcal{A}(\mathbb{GD}, u, v, w, u^\alpha, v^\beta, w^\gamma) = 1\right] \end{array} \right| \leq \mathsf{negl}(\lambda),$$

where $\mathbb{GD} \leftarrow \mathsf{GrGen}(1^\lambda)$, $u, v, w \xleftarrow{u} \mathbb{G}$ and $\alpha, \beta, \gamma \xleftarrow{u} \mathbb{Z}_p$. The probability is taken over the random bits used by GrGen, the uniform random choices of $u, v, w \xleftarrow{u} \mathbb{G}$, $\alpha, \beta, \gamma \xleftarrow{u} \mathbb{Z}_p$ and the random bits used by $\mathcal{A}$. △

Although defined for a single group $(p, \mathbb{G}, g)$, Boneh, Boyen and Shacham prove that the Decision Linear Assumption holds in generic bilinear groups [BBS04].

2.4.2 Computational Assumptions

The security of our constructions relies on the following computational assumptions: the Discrete Logarithm Assumption, the Strong Diffie-Hellman Assumption [BBS04] for Type-1 and Type-2 pairings and the Pointcheval-Sanders Assumption [PS16] for Type-3 pairings.

Definition 2.33: Discrete Logarithm – DLog [KL07]
Given a group description $\mathbb{GD} = (p, \mathbb{G}, g)$ and a tuple $(h, \mathcal{D})$, the Discrete Logarithm Problem is to output the the value $x \in \mathbb{Z}_p$ such that $h^x = \mathcal{D}$.

We say the Discrete Logarithm Assumption holds for group generator GrGen if for all probabilistic polynomial-time adversaries $\mathcal{A}$ there exists a negligible function negl such that

$$\Pr\left[\mathcal{A}\left(\mathbb{GD}, h, \mathcal{D}\right) = \log_h(\mathcal{D})\right] \leq \mathsf{negl}(\lambda),$$

where $\mathbb{GD} \leftarrow \mathsf{GrGen}(1^\lambda)$ and $h, \mathcal{D} \xleftarrow{u} \mathbb{G}$. The probability is over the random bits used by GrGen, the uniform random choice of $h, \mathcal{D} \xleftarrow{u} \mathbb{Z}_p$ and over the random bits used by $\mathcal{A}$. △

Definition 2.34: Strong Diffie-Hellman – SDH [BB04]
Given a group description $\mathbb{GD} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathrm{e}, g_1, g_2)$ of Type-1 or Type-2 pairings and a tuple $\left(g_2^\gamma, g_2^{(\gamma^2)}, \ldots, g_2^{(\gamma^{q(\lambda)})}\right)$, the Strong Diffie-Hellman Problem is to output a pair $\left(g_1^{\frac{1}{x+\gamma}}, x\right)$, where $x \in \mathbb{Z}_p$.

We say the Strong Diffie-Hellman Assumption holds for bilinear group generator $\mathsf{BiGrGen}$ if for all probabilistic polynomial-time adversaries $\mathcal{A}$ and for every polynomially bounded function $q : \mathbb{Z} \to \mathbb{Z}$ there exists a negligible function negl such that

$$\Pr\left[\mathcal{A}\left(\mathbb{GD}, g_2^{\gamma}, g_2^{(\gamma^2)}, \ldots, g_2^{(\gamma^{q(\lambda)})}\right) = \left(g_1^{\frac{1}{x+\gamma}}, x\right)\right] \leq \mathsf{negl}(\lambda),$$

where $\mathbb{GD} \leftarrow \mathsf{BiGrGen}(1^{\lambda})$ and $\gamma \xleftarrow{u} \mathbb{Z}_p$. The probability is over the random bits used by $\mathsf{BiGrGen}$, the uniform random choice of $\gamma \xleftarrow{u} \mathbb{Z}_p$ and over the random bits used by $\mathcal{A}$. △

Definition 2.35: Pointcheval-Sanders – PS1 [PS16]
Given a group description $\mathbb{GD} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathrm{e}, g_1, g_2)$ of a Type-3 pairing, choose $x, y \xleftarrow{u} \mathbb{Z}_p$, $g \xleftarrow{u} \mathbb{G}_1$, $\tilde{g} \xleftarrow{u} \mathbb{G}_2$, and set $X \coloneqq g^x$, $Y \coloneqq g^y$, $\tilde{X} \coloneqq \tilde{g}^x$, $\tilde{Y} \coloneqq \tilde{g}^y$ and define the oracle $\mathcal{O}(m)$ as follows: on input $m \in \mathbb{Z}_p$, choose $h \xleftarrow{u} \mathbb{G}_1$ and output $(h, h^{x+m\cdot y})$.

Given $(\mathbb{GD}, g, Y, \tilde{g}, \tilde{X}, \tilde{Y})$ and unlimited access to oracle $\mathcal{O}$, the Pointcheval-Sanders Problem is to output a tuple $(m^*, s, s^{x+m^*\cdot y})$, where $s \neq 1_{\mathbb{G}_1}$ and m^* was not queried to $\mathcal{O}$.

We say the Pointcheval-Sanders Assumption holds for bilinear group generator $\mathsf{BiGrGen}$ if for all probabilistic polynomial-time adversaries $\mathcal{A}$ there exists a negligible function negl such that

$$\Pr\left[\mathcal{A}^{\mathcal{O}(\cdot)}\left(\mathbb{GD}, g, Y, \tilde{g}, \tilde{X}, \tilde{Y}\right) = \left(m^*, s, s^{x+m^*\cdot y}\right)\right] \leq \mathsf{negl}(\lambda),$$

where $\mathbb{GD} \leftarrow \mathsf{BiGrGen}(1^{\lambda})$. The probability is taken over the random bits used by $\mathsf{BiGrGen}$ the uniform random choices of $x, y \xleftarrow{u} \mathbb{Z}_p$, $g \xleftarrow{u} \mathbb{G}_1$, $\tilde{g} \xleftarrow{u} \mathbb{G}_2$, and over the random bits used by $\mathcal{A}$. △

3 Reputation Systems and their Security

A reputation system is the digital equivalent of word-of-mouth advertising and recommendation, meaning that a reputation system collects and aggregates feedback about attributes of individual objects and publishes the aggregated results as reputation scores. Such systems are always used in conjunction with other applications in which many users need to interact. The purpose of a reputation system in those situations is to provide reliable information about the past behavior of the users, expressed by their reputation score. Hence, reputation systems are an important tool to manage risk when engaging in transactions with users who have not interacted before. In business applications they are also an important marketing tool, as high reputation is assumed to imply high reliability of the business partner.

3.1 Essential Functionality

While the architecture of a reputation system depends on the application it is used for, its functionality is always the same. To illustrate this functionality, we describe how users, acting as consumers and providers of some object, interact with each other, utilizing the reputation system. In the first step, the consumer uses the reputation scores, provided by a reputation system, to select a provider of the application for future interactions. Then the consumer initiates an interaction by sending an application-specific request to the selected provider, which in turn handles the request and sends a response back to the consumer. After consuming the response according to the application, the consumer, now acting as a rater, evaluates it to generate a rating, which contains feedback about the selected provider and its response. The rating is then handed to the reputation system.

Since the reputation system collects all ratings from all raters, it can aggregate them by evaluating a reputation function to generate reputation scores for the providers. These scores are published again to support other consumers in their decision and selection process. A depiction of this functional cycle of reputation systems, originally provided by Jøsang and Golbeck [JG09], is given in Figure 3.1.

Based on the assumption that future experiences will be similar to the past, reputation scores represent a measure of reliability that is deduced from prior experiences various consumers have made. Hence, prospective consumers can use reputation scores as a basis to decide whether or not to interact with providers. Unfortunately, reputation scores lose their expressiveness, and are therefore not reliable, when they can be manipulated.

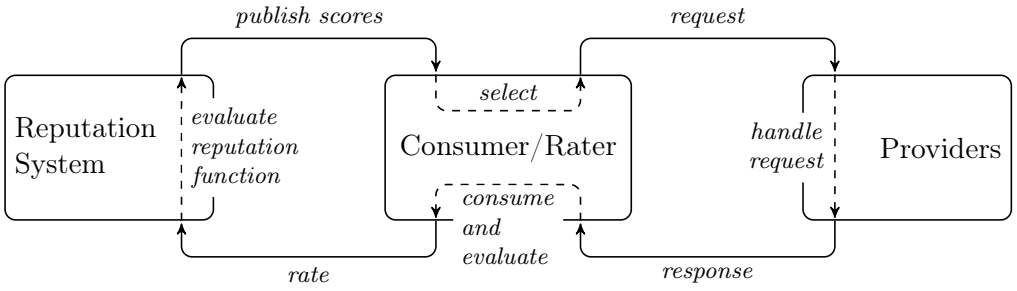


Figure 3.1: Functional Integration of Reputation Systems

When manipulations in reputation systems are considered, it is important to distinguish between preventing manipulations and mitigating their effect. This distinction is necessary, because there exist attacks against reputation systems, discussed in the next section, that cannot be prevented. Nevertheless, their effects on the reputation scores must be mitigated to provide reliable scores. Preventing an attack means that, no matter what strategy a malicious party executes, it is infeasible to successfully run the attack. In contrast to this, mitigating the effect of an attack means that it is possible to detect the attack enabling further mechanisms to reduce its influence on reputation scores.

One approach to mitigate the effect of manipulations is to consider reputation functions that are robust, where robustness, according to Zhang et al. [Zha+12], means that reputation scores for honestly behaving parties are higher than scores for malicious parties. For this purpose various metrics and reputation functions based on different mathematical models have been proposed. Such metrics include, but are not limited to, models based on the Dempster–Shafer theory (or evidence theory) [YS02], Hidden Markov Models [MAB09], β-probability density functions [IJ02; Tea+06], or clustering based approaches [Liu+11]. As shown by Marsh, Basu, and Dwyer [MBD12], a major drawback of such complex reputation functions is that in most cases users do not understand them. Consequently, the users do not trust the complete reputation system, which makes it useless. To address this problem, Sänger and Pernul [SP18] propose a new concept for the presentation of reputation scores that allows users to understand complex metrics and hence increases trust in the reputation system.

Another approach against manipulations is to prevent them by securing the process of rating generation. Since this process already starts when some user selects a provider for interaction, various attacks against reputation systems with different goals exist. The most important attacks and appropriate countermeasures are discussed in the next section.

3.2 Attacks against Reputation Systems

With robust reputation functions and a user-friendly presentation of reputation scores, it is possible to mitigate the effect of manipulations while preserving the users' trust in the reputation system, as briefly discussed in the previous section. To further strengthen the security of reputation systems, it is important to impede and prevent different attacks against them.

Typical attacks against reputation systems are discussed in various works [HZN09; JG09; SL12] and can be categorized into three different classes, according to Koutrouli and Tsalgatidou [KT12]. These categories are *Unfair Feedback*, *Inconsistent Behavior*, and *Identity-based Attacks*. In the following we describe the different classes and typical attacks of each class.

To simplify the following discussion, from now on we merge the roles of consumers and raters and call a party acting in one of both roles a *user* or a *rater*. Furthermore, the objects consumed and provided by different parties are called *products* (or services). These products are *purchased*, which means that the transaction is complete and the provider hands a rating token as a part of his response to the rater. We stress that this denotation does not imply a restricted applicability of the provided reputation systems to business applications. It is only introduced to simplify the description of the security models and constructions.

3.2.1 Unfair Feedback

The class of unfair feedback attacks comprises all attacks where raters generate unfairly high or low ratings to increase the reputation score of colluding providers or to decrease the reputation score of competing providers. Providing unfairly high ratings is called *ballot stuffing*, whereas providing unfairly low ratings is called *bad mouthing*. Ballot stuffing can be used as a strategic attack, but it can also be the result of pressure from the provider, when a rater fears negative consequences from negative ratings. To prevent such situations, the rater must be anonymous or the rating must be kept secret until both parties have sent their ratings to the reputation system.

A special case of unfairly high ratings is the *self-rating attack* (also known as self-promoting). For self-rating attacks a malicious party tries to increase its own reputation by rating himself. Such an attack is possible when users are not accountable for ratings and the cost for generating ratings is low. With a tracing mechanism that enables a dedicated party to find out the identity of a rater accountability can be realized. To increase the cost for generating ratings a reputation system can require a rater to bind its rating to a previously executed transaction. When multiple ratings bound to the same transaction are linkable, this further impedes self-ratings attacks.

3.2.2 Inconsistent Behavior

Parties executing inconsistent behavior attacks exploit the reputation system by behaving differently as expected. These attacks are especially performed in business applications, where users have to pay for interactions. Typical attacks of this type are *traitor attacks*. Here, a malicious provider behaves appropriately for a while to build a high reputation score, but suddenly changes its behavior. The malicious provider accepts negative ratings for its bad behavior, because they are compensated by many positive ratings deserved previously. Mechanisms against bad mouthing further prohibit the adequate reduction of the reputation score. To give an example for this attack, consider a malicious provider that gets positive ratings for some low-priced products, and hence has a high reputation

score. When selling small numbers of some high-priced products, the provider can provide products of low quality to increase its financial profit, without the fear of a decreased reputation. This concrete attack is called *value imbalance attack*. A variation of this attack can be executed by discriminating a minority of users, while to the majority of users a malicious provider behaves properly. Both attacks allow the provider to preserve a good reputation, as long as the positive ratings compensate the negative ones.

A completely different behavioral attack is to not provide ratings for others. The motivation behind this is to not support competitors and increase their reputation. By that, the attacker retains his advantage for being selected for interactions, while not apprehending accusations for providing unfair feedback.

Unfortunately, inconsistent behavior attacks cannot be prevented, because such attacks do not directly exploit properties of the rating generation process, but address the reputation function in use. Indeed, inconsistent behavior is still valid behavior within a reputation system. Nevertheless, Chen et al. [Che+10] propose a scheme resilient to traitor attacks, based on the Cobweb Theorem [Eze38], that can be incorporated into reputation functions and also other works consider inconsistent behavior attacks [IJ02; SXL05; Sun+06] to mitigate their effects.

3.2.3 Identity-based Attacks

Since reputation systems are intended to provide reliable information about previous transactions with concrete providers, reputation scores must be bound on the identities of those providers. Hence, it might be beneficial for malicious providers to attack the mechanism used to manage identities in the reputation system.

In a *Sybil attack* an adversary generates multiple identities and is hence able to behave as multiple users and providers simultaneously. Therefore, the adversary can influence the reputation system more than honest parties with a single identity. The susceptibility of a reputation system to Sybil attacks depends on the cost for identity generation and the initial influence a newly generated identity has in the system. Sybil attacks can be prevented with a trusted certification mechanism, where the identity of a user is bound to a cryptographic certificate. This solution impacts the privacy of users as multiple shows of a certificate are linkable, which enables the certificate verifier to generate user profiles. Hence, to preserve privacy other techniques than simple certificate shows are necessary.

With a *whitewashing attack* a malicious provider evades negative consequences of low reputation scores by generating a new identity. This is only useful in reputation systems that provide an initial reputation score for new identities that is higher than the attacker's current score. Since this attack is similar to the Sybil attack, whitewashing can be prevented with an appropriate identity management.

Other identity-based attacks do not exploit the identity management directly, but focus on the authenticity of messages and parties. Attacks of this category comprise *impersonation*, *man-in-the-middle attacks*, and *repudiation*. Impersonation means that an attacker operates on behalf of a noninvolved user to initiate interactions with providers or to generate unfair ratings. With a man-in-the-middle attack a malicious party can try to alter ratings

or reputation scores during the transmission between a user and the reputation system. Repudiation means that users and providers are not accountable for their actions, which allows the execution of arbitrary attacks without fearing consequences. All these attacks can be prevented by authenticating all actions of parties to guarantee accountability.

3.3 Desired Properties and Cryptographic Considerations

As discussed in the previous sections, reputation systems need to be protected against various attacks to provide reliable and honest ratings. Albeit many attacks and their countermeasures have been considered in the literature [Del00; Ste06; And+08; HZN09; Ker09; CSK13; Has+13; PLS14; Zha+16; Fig+17], no generally accepted security model for reputation systems has emerged. With robust metrics and reputation functions the effect of manipulations can be mitigated, but they cannot be prevented. Therefore, also the process of rating generation must be secured. In this section we discuss the security properties of the rating generation process and further propose techniques to achieve them.

Averting manipulations in reputation systems means that concrete attacks must be prevented or at least weakened. For the identity-based Sybil attack Douceur [Dou02] has shown that it can only be avoided with cryptographically secure certificates that bind the identity of a party to a certificate. This mechanism also prevents whitewashing. Furthermore, certificates and digital signatures provide accountability, authenticity, and integrity, which are needed to prohibit impersonation, man-in-the-middle attacks, and repudiation. So it is reasonable to also consider other attacks and security properties in a cryptographic context.

Besides authenticity, also privacy has been identified as a key property of reputation systems [Del00; Ste06; Ker09; CSK13; MK14]. Here, privacy means that providers do not learn the identity of a rater and the actual value of a rating at the same time. This can be achieved in different ways. To give an example, Dellarocas [Del00] proposes anonymous interactions to prevent unfair ratings. The anonymity ensures that the identity of a rater is hidden, so this approach provides privacy. In contrast to this, Kerschbaum [Ker09] encrypts ratings without guaranteeing anonymity, so the identity of the rater is known, but the rating value is hidden. Therefore, this approach also provides privacy, which shows that anonymity is only one possible way to achieve privacy. But since anonymity is an often desired property in many applications, it should also be ensured in a reputation system.

With anonymity raters cannot be punished for negative ratings, which allows them to rate honestly. But at the same time anonymity facilitates bad mouthing. This demonstrates the conflict between anonymity and authenticity in reputation systems. To address this problem, a tracing mechanism can be implemented that enables a dedicated party to rescind the anonymity of misbehaving raters. But then the question arises how misbehaving raters can be detected, especially when bad mouthing, ballot stuffing, or self-ratings are considered simultaneously. For this purpose, ratings need to be bound to a previous transaction [SL03; SXL05] and the reputation system has to ensure that every transaction can only be rated once. This allows to detect multiple ratings from the same rater for a specific transaction and the misbehaving rater can be identified using the tracing mechanism.

Summarizing the discussed aspects, a secure reputation system should provide:

- an identity management based on cryptographic certificates,
- a binding of ratings to transactions,
- linkability of multiple ratings from one author for a single transaction,
- authenticity and integrity of ratings,
- anonymity for raters, and
- a tracing mechanism to rescind anonymity.

Furthermore, a model for secure reputation systems should allow to build systems independently from specific applications and reputation functions. To achieve this it is important that ratings can be arbitrary objects, for example numerical values or textual reviews. But also encrypted ballots, as used by Kerschbaum [Ker09], and other complex constructions are conceivable. With some extensions, all of these properties are achievable by dynamic group signatures. However, the concept of group signatures does not meet all the requirements for reputation systems. In particular, reputation systems do not consist of a single group of users. Rather one can think of reputation systems as a family of group signature schemes - one for each product. Moreover, a single provider might have multiple products that have to be treated differently. Hence, when looking at security and anonymity, group signature schemes for different products cannot be considered in isolation.

Let us provide some intuition how dynamic group signatures can be used as the basic cryptographic primitive to formulate a security model for reputation systems. Dynamic group signature schemes already provide an identity management, authenticity and integrity of signed messages, anonymity for signers, and a tracing mechanism to rescind anonymity. By replacing the signed message of a group signature with the actual rating value, the same properties can be achieved in a reputation system. When for each product a new dynamic group signature scheme is set up, a secret signing key, generated during the interactive group joining protocol, can be used as a rating token. Only with this token it is possible to generate a rating, which basically is a group signature that binds a rating to a transaction. Due to the anonymity property of group signatures this binding does not reveal the identity of a rater. Furthermore, the security properties traceability and non-frameability of dynamic group signatures ensure that rating tokens cannot be forged, which prevents faking ratings. In addition to this, traceability and non-frameability also guarantee that ratings cannot be generated on behalf of other users, that every rating can be traced back to its author, and that the tracing party is not able to blame an honest user being the author of a rating when it is not. The last property needed for secure reputation systems is linkability. Since linkability weakens anonymity, it is an undesired property for group signatures. Moreover, extending group signatures to support linkability is not expedient, because ratings for different products from the same rater should not be linkable. Hence, the separation of different products must explicitly be considered in the definition of linkability. The basic idea to achieve this is to introduce *unique tags* that depend on the rater's identity and the product under consideration, but hide the identity of the rater. Including such tags in the ratings allows to define a linking mechanism that finds linkable ratings.

Concrete models for secure reputation systems and constructions fulfilling the defined security properties are presented in Chapter 4 and Chapter 6. The security models focus on the process of rating generation, beginning with an interaction of a user and a provider.

4 Models and Constructions for Secure Reputation Systems

This chapter is based on our paper „Anonymous and Publicly Linkable Reputation Systems“, presented at the International Conference on Financial Cryptography and Data Security 2015 [BJK15].

We begin this chapter by defining our model for reputation systems based on group signatures and providing associated experiment-based security definitions. Subsequently, we present a construction that is secure in the proposed model.

4.1 A Model for Reputation Systems

In this section we present a model for reputation systems based on dynamic group signatures, as defined by Bellare, Shi, and Zhang [BSZ05]. We start by introducing the different parties and the algorithms they execute.

4.1.1 Architecture and Algorithms

A meaningful reputation system must provide reliable, and honest ratings. Furthermore, it should be flexible in the sense that it can be combined with many different applications. Therefore, we focus on the process of secure rating generation and exclude the aggregation of ratings and the evaluation of a specific reputation function from our model.

A reputation system consists of one authority called *System Manager*, a set of authorities called the *(product) providers*, and a set of *raters*. The system manager provides the system manager's public key *smpk* and is able to trace raters. Every provider maintains items (or *products*) with corresponding item-based public keys $ipk[item]$, which will be used by the raters to rate a specific item. Raters have unique identities $i \in \mathbb{N}$ and may become members of the system by registering at the system manager. Figure 4.1 illustrates the architecture of our reputation system.

The specification of a reputation system is a tuple $\mathcal{RS} = (\mathsf{KeyGen}_{SM}, \mathsf{KeyGen}_{P}, \mathsf{KeyGen}_{R}, \mathsf{Register}_{SM}, \mathsf{Register}_{R}, \mathsf{Join}, \mathsf{Issue}, \mathsf{Rate}, \mathsf{Verify}, \mathsf{Open}, \mathsf{Link}, \mathsf{Revoke})$ of polynomial-time algorithms. Their functionality is described as follows.

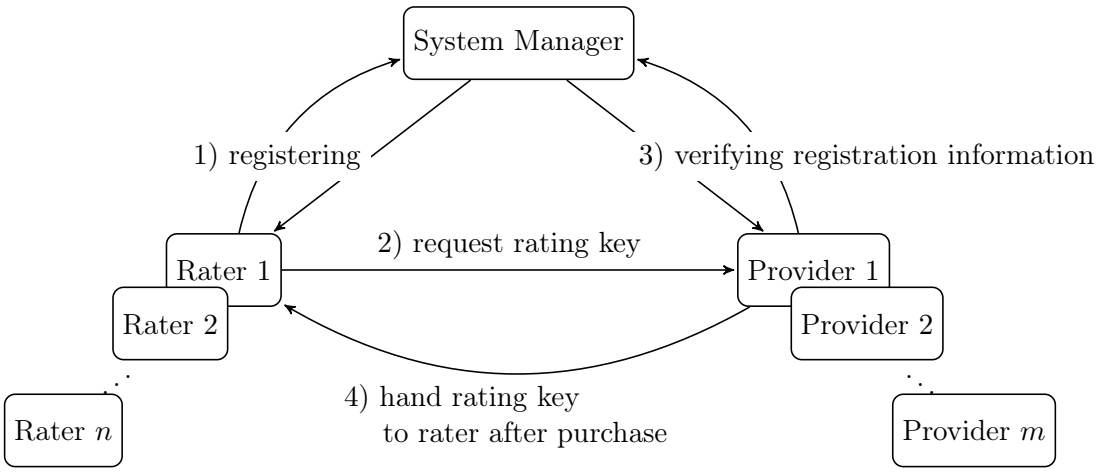


Figure 4.1: Informal architecture of our reputation system.

KeyGen$_{SM}(1^\lambda)$: This randomized algorithm is run in the setup phase by the system manager to create the system manager's public key *smpk* and the system manager's secret key *smsk*. The secret key *smsk* contains elements which allow tracing of raters and the creation of revocation tokens.

KeyGen$_P(1^\lambda, item)$: This randomized algorithm is run by a provider for every *item* he maintains. For the given *item* this algorithm creates an *item*-based public key $ipk[item]$ and a corresponding *item*-based secret key $isk[item]$. The tuple $(item, ipk[item])$ is added to the *ItemList*.

KeyGen$_R(1^\lambda, i)$: This randomized algorithm is run to create the rater's public and secret key pair $(rpk[i], rsk[i])$. The rater's public key $rpk[i]$ is used during the registration to the system, the corresponding secret key $rsk[i]$ is used to create ratings.

Register$_{SM}(\mathrm{St}_{SM}, M_{SM})$, Register$_R(\mathrm{St}_R, M_R)$: These randomized interactive algorithms are run by the system manager and a prospective rater $i \in \mathbb{N}$. During this protocol the rater's public and secret key pair $(rpk[i], rsk[i])$ is chosen by using the KeyGen$_R$ algorithm. If the system manager accepts, the tuple $(i, rpk[i])$ is added to the registration table *reg*. The input parameters of the algorithms are some state information and a message, which was received from the communicating partner. It is assumed that the rater starts the interaction.

Join(St_R, M_R), Issue(St_P, M_P): These randomized interactive algorithms are run by a rater $i \in \mathbb{N}$ and a provider. The input parameters of the algorithms are some state information and a message, which was received from the communicating partner. It is assumed that the rater starts the interaction. The first message of the rater must contain his public key $rpk[i]$, an *item*, and his identity i. If Issue accepts, the provider sends a personal rating key for the given *item* $rrsk[i, item]$ to the rater and saves the tuple $(rpk[i], rrsk[i, item])$ in the identification list IL_{item} for the specified *item*.

Rate$(item, smpk, ipk[item], rrsk[i, item], rsk[i], M)$: This randomized algorithm is run by a rater to create a rating for the specified *item*. Given an *item*, the system manager's public key *smpk*, an *item*-based public key $ipk[item]$, the rating key for

the given *item* of rater i $rrsk[i, item]$, the secret key of rater i $rsk[i]$, and a message M, Rate computes and outputs a rating σ on M under the given keys.

Verify($item$, $smpk$, $ipk[item]$, $\mathcal{RL}$, M, σ): This deterministic algorithm can be run by every party, even by an outsider, having access to the public *ItemList*, the system manager's public key *smpk*, the revocation list $\mathcal{RL}$, a message M and a candidate rating σ for M, to obtain a bit v. We say that σ is a *valid* rating for M with respect to the given keys, if and only if the bit v is 1.

Open($smpk$, $smsk$, M, σ): This deterministic algorithm is run by the system manager to *open* ratings. Given the system manager's public key *smpk*, the system manager's secret key *smsk*, a message M and a rating σ, output the identity of the rater or `failure`.

Link($item$, $smpk$, $ipk[item]$, (M', σ'), (M'', σ'')): This deterministic algorithm can be run by every party, even by an outsider, having access to the public *ItemList*, the system manager's public key *smpk* and two message-rating pairs $(M', \sigma'), (M'', \sigma'')$, to obtain a bit ℓ. If Link outputs $\ell = 1$, we call σ' and σ'' are *publicly linkable* ratings.

Revoke($smpk$, $smsk$, i): This deterministic algorithm is run by the system manager to revoke raters in case of misuse. Given the system manager's public key *smpk*, the system manager's secret key *smsk* and the identity of the rater to revoke, compute the revocation token $rrt[i]$ and add it to the public revocation list $\mathcal{RL}$.

Figure 4.2 illustrates the interaction of the described parties and the algorithms involved. It is not hard to see that the number of providers is not important in this model: a single provider has the same capabilities as a colluding set of providers. Therefore, in all formal definitions we will only consider a single provider. Additionally, we assume that the correctness of rating keys from the provider given to a rater can be checked using only public parameters.

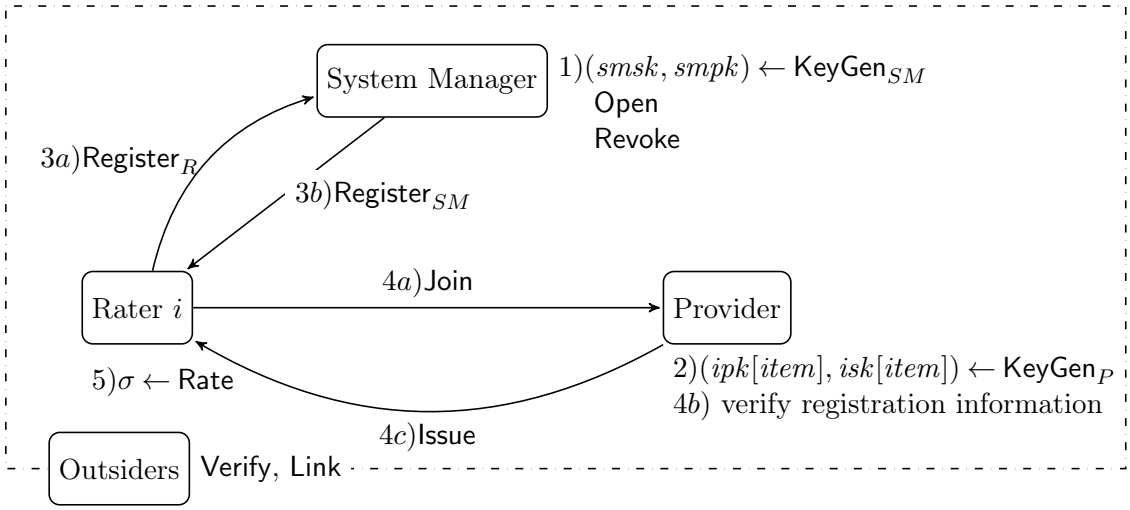


Figure 4.2: Interaction of the parties within a reputation system.

Correctness: A reputation system must satisfy the following correctness requirements: For all $i \in \mathbb{N}$, all $item \in \{0,1\}^*$, all $(smpk, smsk) \in [\mathsf{KeyGen}_{SM}]$, all $(ipk[item], isk[item]) \in [\mathsf{KeyGen}_P]$, all $(rpk[i], rsk[i]) \in [\mathsf{Register}_{SM}] \times [\mathsf{Register}_R]$, all $rrsk[i, item] \in [\mathsf{Join}] \times [\mathsf{Issue}]$, and all $rrt[i] \in [\mathsf{Revoke}]$:

$$\begin{aligned}&\mathsf{Verify}(item, smpk, ipk[item], \mathcal{RL}, M,\\ &\qquad \mathsf{Rate}(item, smpk, ipk[item], rrsk[i, item], rsk[i], M)) = 1\\ &\qquad \Longleftrightarrow rrt[i] \notin \mathcal{RL},\\ &\mathsf{Open}(smpk, smsk, M,\\ &\qquad \mathsf{Rate}(item, smpk, ipk[item], rrsk[i, item], rsk[i], M)) = i,\end{aligned}$$

and

$$\begin{aligned}&\mathsf{Link}(item, smpk, ipk[item],\\ &\qquad (M', \mathsf{Rate}(item, smpk, ipk[item], rrsk[i, item], rsk[i], M')),\\ &\qquad (M'', \mathsf{Rate}(item, smpk, ipk[item], rrsk[i, item], rsk[i], M''))) = 1.\end{aligned}$$

Informally, that means

- honestly generated ratings of non-revoked raters will be accepted by the Verify algorithm,
- honestly generated ratings can be traced back to the correct rater, and
- two different ratings for the same *item* generated by a single rater will be detected by the Link algorithm.

4.1.2 Security Notions

To model the different attack capabilities of an adversary, we introduce certain oracles, which will be used in the definitions of security. The oracle definitions given in Figure 4.3 and Figure 4.4 are based on those for dynamic group signatures [BSZ05]. Therefore, we assume that a security experiment has run KeyGen_{SM} to obtain $(smpk, smsk)$, and manages the global sets $\mathcal{HR}$, $\mathcal{CR}$, $\mathcal{RR}$, $\mathcal{JIR}$, $\mathcal{QR}$, reg, $ItemList$, and IL_{item}. Except $ItemList$, IL_{item}, and reg all sets are only used within the formal definitions of oracles and security experiments. With $\mathcal{HR}$ we denote the set of honest raters, with $\mathcal{CR}$ the set of corrupted raters. The set $\mathcal{RR}$ contains all identities of raters that currently engage in the registration protocol. The set $\mathcal{JIR}$ contains all identities of raters that currently engage in the join-issue protocol. With $\mathcal{QR}$ we denote the set of queried ratings. All sets are assumed to be initially empty.

AddR(i): To add honest raters to the system the adversary can call this *add rater* oracle with an identity $i \in \mathbb{N}$ as argument. The oracle adds i to the set of honest raters and executes the registration protocol by running $\mathsf{Register}_{SM}$ and $\mathsf{Register}_R$. When $\mathsf{Register}_{SM}$ accepts, the tuple $(i, rpk[i])$ is stored in the registration table reg. When $\mathsf{Register}_R$ accepts, the pair $(rsk[i], rpk[i])$ is the key pair of rater i. The oracle returns $rpk[i]$ to the adversary.

AddItem($item$): An adversary can add *item*s by using this *add item* oracle. The oracle then runs the KeyGen_P algorithm to obtain a secret and a public key for the specified *item*. Afterwards, the *item* is added to the *ItemList* and the public key of the *item* is returned to the adversary.

RSK(i): To get the secret key $rsk[i]$ of an honest rater $i \in \mathbb{N}$ an adversary can call the *rater secret key* oracle with an identity i as argument. Then the rater i is added to $\mathcal{CR}$ and $rsk[i]$ is returned to the adversary.

RRSK(i, $item$): To get the secret rating key $rrsk[i, item]$ of a corrupted rater $i \in \mathbb{N}$ for a specified *item*, an adversary can call the *rating key* oracle with an identity i and an already existing *item* as arguments. If no rating key is found, the oracle generates a new one and returns it to the adversary.

RevR(i): To get the *revocation token* of an honest rater $i \in \mathbb{N}$ an adversary can call this *revoke rater* oracle with an identity i as argument. Then the revocation token is added to $\mathcal{RL}$ and returned to the adversary.

GRate(i, $item$, M): An adversary can use the *generate rating* oracle to obtain a valid rating for the message M with respect to the rating key of rater $i \in \mathbb{N}$, and the *item*-based public key $ipk[item]$. The queried rating is added to $\mathcal{QR}$ and returned to the adversary.

SndToP(i, $item$, $rpk[i]$, M_P): After corruption of rater $i \in \mathbb{N}$, the adversary can use this *send to provider* oracle to engage in a join-issue protocol with the provider. The adversary provides the *item* and the public key of rater i for which he wants to get a secret rating key. Furthermore, the message M_P is sent to the provider. The oracle honestly executes the Issue protocol and computes a response to M_P. If Issue accepts the communication, the rater's secret rating key is saved in the identification list IL_{item} and M_R contains the tuple $(rpk[i], rrsk[i, item])$.

SndToSM(i, M_{SM}): Similarly to the SndToP oracle, the *send to system manager* oracle can be used by an adversary to engage in a registration protocol with the system manager. The adversary provides an identity $i \in \mathbb{N}$ and a message M_{SM} sent to the system manager. The oracle honestly executes the $\mathsf{Register}_{SM}$ protocol and saves $(i, rpk[i])$ in the registration table $reg[i]$, if and only if $\mathsf{Register}_{SM}$ accepts. The rater i is added to $\mathcal{CR}$.

WItemList($item$, ipk): An adversary can use the *write to item list* oracle to manipulate the *item*-based public key of the specified *item*. If $ipk = \varepsilon$ the *item* is deleted from the list. Otherwise, the specified public key is set.

WIdentList($item$, i, $rpk[i]$, $rrsk$): Using the *write to identification list* oracle an adversary can modify the secret rating key of rater $i \in \mathbb{N}$ for the specified *item*. If $rrsk = \varepsilon$ the key information about rater i is deleted from the list.

Open($item$, M, σ): The *opening* oracle can be used by the adversary with a message M, a rating σ and an *item* as arguments to get the output of the Open algorithm.

$\mathsf{AddR}(i)$: // everybody
  If $(i \in \mathcal{HR} \cup \mathcal{CR})$ **Then** return ε.
  $\mathcal{HR} \coloneqq \mathcal{HR} \cup \{i\}$
  $\mathrm{St}_R^i \coloneqq (smpk, i)$
  $M_R \coloneqq \varepsilon$
  $\mathrm{St}_{SM}^i \coloneqq (smpk, smsk)$
  $(\mathrm{St}_R^i, M_{SM}, \mathrm{mode}^i) \leftarrow \mathsf{Register}_R(\mathrm{St}_R^i, M_R)$
  While $(\mathrm{mode}^i = \texttt{continue})$ **do**
    $(\mathrm{St}_{SM}^i, M_R, \mathrm{mode}^i) \leftarrow \mathsf{Register}_{SM}(\mathrm{St}_{SM}^i, M_{SM})$
    If $(\mathrm{mode}^i = \texttt{accept})$ **Then** $reg[i] \coloneqq (i, rpk[i])$
    $(\mathrm{St}_R^i, M_{SM}, \mathrm{mode}^i) \leftarrow \mathsf{Register}_R(\mathrm{St}_R^i, M_R)$
  return $rpk[i]$

$\mathsf{RRSK}(i, item)$: // corrupted raters
  If $(i \notin \mathcal{CR})$ **Then** return ε.
  If $(item \notin ItemList)$ **Then** return ε.
  If $((rpk[i], \cdot) \notin IL_{item})$ **Then**
    $\mathrm{St}_R^i \coloneqq (smpk, ipk[item], rpk[i], i)$
    $M_R \coloneqq \varepsilon$
    $\mathrm{St}_P^i \coloneqq (smpk, rpk[i])$
    $(\mathrm{St}_R^i, M_P, \mathrm{mode}^i) \leftarrow \mathsf{Join}(\mathrm{St}_R^i, M_R)$
    While $(\mathrm{mode}^i = \texttt{continue})$ **do**
      $(\mathrm{St}_P^i, M_R, \mathrm{mode}^i) \leftarrow \mathsf{Issue}(\mathrm{St}_P^i, M_P)$
      If $(\mathrm{mode}^i = \texttt{accept})$ **Then**
        $IL_{item} \coloneqq IL_{item} \cup \{(rpk[i], rrsk[i, item])\}$
      $(\mathrm{St}_R^i, M_P, \mathrm{mode}^i) \leftarrow \mathsf{Join}(\mathrm{St}_R^i, M_R)$
  return $rrsk[i, item]$

$\mathsf{Open}(item, M, \sigma)$: // everybody
  If $(item \notin ItemList)$ **Then**
    return `failure`.
  return $\mathsf{Open}(smpk, smsk, M, \sigma)$

$\mathsf{RevR}(i)$: // corrupted raters
  If $(i \notin \mathcal{HR})$ **Then**
    return ε.
  $\mathcal{RL} \coloneqq \mathcal{RL} \cup \{rrt[i]\}$
  return $rrt[i]$

$\mathsf{RSK}(i)$: // corrupted raters
  If $(i \notin \mathcal{HR})$ **Then**
    return ε.
  $\mathcal{HR} \coloneqq \mathcal{HR} \setminus \{i\}$
  $\mathcal{CR} \coloneqq \mathcal{CR} \cup \{i\}$
  return $rsk[i]$

Figure 4.3: Oracles

$\mathsf{AddItem}(item)$: // everybody
If $(item \in ItemList)$ **Then** return ε.
$(ipk[item], isk[item]) \leftarrow \mathsf{KeyGen}_P(item)$
$ItemList := ItemList \cup \{(item, ipk[item])\}$
return $ipk[item]$

$\mathsf{WItemList}(item, ipk)$: // corrupted provider
$ItemList := ItemList \setminus \{(item, ipk[item])\}$
If $(ipk \neq \varepsilon)$ **Then**
$ItemList := ItemList \cup \{(item, ipk)\}$
return 1

$\mathsf{WIdentList}(item, i, rpk[i], rrsk)$:
If $(item \notin ItemList)$ **Then** return 0.
If $((i, rpk[i]) \neq reg[i])$ **Then** return 0.
$IL_{item} := IL_{item} \setminus \{(rpk[i], rrsk[i, item])\}$
If $(rrsk \neq \varepsilon)$ **Then**
$IL_{item} := IL_{item} \cup \{(rpk[i], rrsk)\}$
return 1

$\mathsf{SndToP}(i, item, rpk[i], M_P)$: // corrupted raters
If $(i \notin \mathcal{CR})$ **Then** return ε.
If $(item \notin ItemList)$ **Then** return ε.
If $(i \notin \mathcal{JIR})$ **Then** $\mathrm{St}_P^i := \varepsilon$.
$\mathcal{JIR} := \mathcal{JIR} \cup \{i\}$
$(\mathrm{St}_P^i, M_R, \mathrm{mode}^i) \leftarrow \mathsf{Issue}(\mathrm{St}_P^i, M_P)$
If $(\mathrm{mode}^i = \texttt{accept})$ **Then**
$(rpk[i], rrsk[i, item]) := M_R$
$IL_{item} := IL_{item} \cup \{(rpk[i], rrsk[i, item])\}$
$\mathcal{JIR} := \mathcal{JIR} \setminus \{i\}$
If $(\mathrm{mode}^i = \texttt{deny})$ **Then** $\mathcal{JIR} := \mathcal{JIR} \setminus \{i\}$
return (M_R, mode^i)

$\mathsf{SndToSM}(i, M_{SM})$: // corrupted raters
If $(i \in \mathcal{HR} \cup \mathcal{CR})$ **Then** return ε.
If $(i \notin \mathcal{RR})$ **Then** $\mathrm{St}_{SM}^i := (smpk, smsk)$
$\mathcal{RR} := \mathcal{RR} \cup \{i\}$
$(\mathrm{St}_{SM}^i, M_R, \mathrm{mode}^i) \leftarrow \mathsf{Register}_{SM}(\mathrm{St}_{SM}^i, M_{SM})$
If $(\mathrm{mode}^i = \texttt{accept})$ **Then**
$reg[i] := (i, rpk[i])$
$\mathcal{CR} := \mathcal{CR} \cup \{i\}$
$\mathcal{RR} := \mathcal{RR} \setminus \{i\}$
If $(\mathrm{mode}^i = \texttt{deny})$ **Then** $\mathcal{RR} := \mathcal{RR} \setminus \{i\}$
return (M_R, mode^i)

$\mathsf{GRate}(i, item, M)$: // everybody
If $(i \notin \mathcal{HR})$ **Then** return ε
If $(item \notin ItemList)$ **Then** return ε
If $((rpk[i], \cdot) \notin IL_{item})$ **Then**
$\mathrm{St}_R^i := (smpk, ipk[item], rpk[i], i)$
$M_R := \varepsilon$
$\mathrm{St}_P^i := (smpk, rpk[i])$
$(\mathrm{St}_R^i, M_P, \mathrm{mode}^i) \leftarrow \mathsf{Join}(\mathrm{St}_R^i, M_R)$
While $(\mathrm{mode}^i = \texttt{continue})$ **do**
$(\mathrm{St}_P^i, M_R, \mathrm{mode}^i) \leftarrow \mathsf{Issue}(\mathrm{St}_P^i, M_P)$
If $(\mathrm{mode}^i = \texttt{accept})$ **Then**
$IL_{item} := IL_{item} \cup \{(rpk[i], rrsk[i, item])\}$
$(\mathrm{St}_R^i, M_P, \mathrm{mode}^i) \leftarrow \mathsf{Join}(\mathrm{St}_R^i, M_R)$
$\sigma \leftarrow \mathsf{Rate}(item, smpk, ipk[item], rrsk[i, item], rsk[i], M)$
$\mathcal{QR} := \mathcal{QR} \cup \{(item, i, M, \sigma)\}$
return σ

Figure 4.4: Oracles

Experiment $\mathrm{Exp}^{anon}_{\mathcal{A},\mathcal{RS}}(\lambda)$

$\mathcal{HR} \coloneqq \emptyset, \mathcal{CR} \coloneqq \emptyset, \mathcal{RL} \coloneqq \emptyset, ItemList \coloneqq \emptyset, reg \coloneqq \emptyset, \mathcal{RR} \coloneqq \emptyset, \mathcal{JIR} \coloneqq \emptyset, \mathcal{QR} \coloneqq \emptyset$

$(smsk, smpk) \leftarrow \mathsf{KeyGen}_{SM}(1^\lambda)$

$(i_0, i_1, item, M, \mathrm{St}) \leftarrow \mathcal{A}(smpk\colon \mathsf{AddR}, \mathsf{RSK}, \mathsf{RevR}, \mathsf{GRate}, \mathsf{SndToSM}, \mathsf{WItemList}, \mathsf{WIdentList}, \mathsf{Open}, \mathtt{choose})$

$b \xleftarrow{u} \{0,1\}$

$\sigma \leftarrow \mathsf{Rate}(item, smpk, ipk[item], rrsk[i_b, item], rsk[i_b], M)$

$d \leftarrow \mathcal{A}(\sigma, \mathrm{St}\colon \mathsf{AddR}, \mathsf{RSK}, \mathsf{RevR}, \mathsf{GRate}, \mathsf{SndToSM}, \mathsf{WItemList}, \mathsf{WIdentList}, \mathsf{Open}, \mathtt{guess})$

If $((item, i_0, \cdot, \cdot) \in \mathcal{QR}) \vee ((item, i_1, \cdot, \cdot) \in \mathcal{QR})$
$\vee\, (i_0 \notin \mathcal{HR}) \vee (i_1 \notin \mathcal{HR}) \vee (rsk[i_0] \in \mathcal{RL}) \vee (rsk[i_1] \in \mathcal{RL})$
$\vee\, (\mathcal{A}$ queried $\mathsf{Open}(item, M, \sigma))$ **Then**
return 0

return $d = b$

Experiment $\mathrm{Exp}^{publink}_{\mathcal{A},\mathcal{RS}}(\lambda)$

$\mathcal{HR} \coloneqq \emptyset, \mathcal{CR} \coloneqq \emptyset, \mathcal{RL} \coloneqq \emptyset, ItemList \coloneqq \emptyset, reg \coloneqq \emptyset, \mathcal{RR} \coloneqq \emptyset, \mathcal{JIR} \coloneqq \emptyset, \mathcal{QR} \coloneqq \emptyset$

$(smsk, smpk) \leftarrow \mathsf{KeyGen}_{SM}(1^\lambda)$

$\{(item, m_i, \sigma_i)\}_{i=1}^{|\mathcal{CR}|+1} \leftarrow \mathcal{A}(smpk\colon \mathsf{AddItem}, \mathsf{SndToP}, \mathsf{SndToSM})$

If $\exists i \in \{1, \ldots, |\mathcal{CR}| + 1\}\colon$
$\mathsf{Verify}(item, smpk, ipk[item], \mathcal{RL}, m_i, \sigma_i) = 0$ **Then**
return 0

If $\exists i, j \in \{1, \ldots, |\mathcal{CR}| + 1\}\colon$
$i \neq j \wedge \mathsf{Link}(item,\ smpk,\ ipk[item],\ (m_i, \sigma_i),\ (m_j, \sigma_j)) = 1$ **Then**
return 0.

return 1

Figure 4.5: Experiment Definitions for Anonymity and public Linkability

Experiment $\mathrm{Exp}^{trace}_{\mathcal{A},\mathcal{RS}}(\lambda)$
$\mathcal{HR} \coloneqq \emptyset, \mathcal{CR} \coloneqq \emptyset, \mathcal{RL} \coloneqq \emptyset, \mathit{ItemList} \coloneqq \emptyset, \mathit{reg} \coloneqq \emptyset, \mathcal{RR} \coloneqq \emptyset, \mathcal{JIR} \coloneqq \emptyset, \mathcal{QR} \coloneqq \emptyset$
$(\mathit{smsk}, \mathit{smpk}) \leftarrow \mathsf{KeyGen}_{SM}(1^\lambda)$
$(\mathit{item}, m, \sigma) \leftarrow \mathcal{A}(\mathit{smpk}: \mathsf{AddR}, \mathsf{AddItem}, \mathsf{RSK}, \mathsf{RRSK}, \mathsf{RevR}, \mathsf{GRate}, \mathsf{SndToP}, \mathsf{SndToSM}, \mathsf{Open})$
If $(\mathsf{Verify}(\mathit{item}, \mathit{smpk}, \mathit{ipk}[\mathit{item}], \mathcal{RL}, m, \sigma) = 0)$ **Then**
return 0
If $(\mathsf{Open}(\mathit{smpk}, \mathit{smsk}, m, \sigma) = \texttt{failure})$ **Then**
return 1
$i \leftarrow \mathsf{Open}(\mathit{smpk}, \mathit{smsk}, m, \sigma)$
If $(i \in \mathcal{CR}) \vee ((\mathit{item}, i, m, \sigma) \in \mathcal{QR})$ **Then**
return 0
return 1

Experiment $\mathrm{Exp}^{str-ex}_{\mathcal{A},\mathcal{RS}}(\lambda)$
$\mathcal{HR} \coloneqq \emptyset, \mathcal{CR} \coloneqq \emptyset, \mathcal{RL} \coloneqq \emptyset, \mathit{ItemList} \coloneqq \emptyset, \mathit{reg} \coloneqq \emptyset, \mathcal{RR} \coloneqq \emptyset, \mathcal{JIR} \coloneqq \emptyset, \mathcal{QR} \coloneqq \emptyset$
$(\mathit{smsk}, \mathit{smpk}) \leftarrow \mathsf{KeyGen}_{SM}(1^\lambda)$
$(\mathit{item}, m, \sigma) \leftarrow \mathcal{A}(\mathit{smpk}: \mathsf{AddR}, \mathsf{RSK}, \mathsf{RevR}, \mathsf{GRate}, \mathsf{SndToSM}, \mathsf{WItemList}, \mathsf{WIdentList}, \mathsf{Open})$
If $(\mathsf{Verify}(\mathit{item}, \mathit{smpk}, \mathit{ipk}[\mathit{item}], \mathcal{RL}, m, \sigma) = 0)$ **Then**
return 0
If $(\mathsf{Open}(\mathit{smpk}, \mathit{smsk}, m, \sigma) = \texttt{failure})$ **Then**
return 0
$i \leftarrow \mathsf{Open}(\mathit{smpk}, \mathit{smsk}, m, \sigma)$
If $(i \notin \mathcal{HR}) \vee ((\mathit{item}, i, m, \sigma) \in \mathcal{QR})$ **Then**
return 0
return 1

Figure 4.6: Experiment Definitions for Traceability and Strong Exculpability

Using the oracle definitions from Figure 4.3 and Figure 4.4 we can define the security experiments. In our reputation system we need anonymity, traceability, public linkability and strong exculpability. The anonymity and traceability experiments are based on [BSZ05], the public linkability experiment is based on [FS07] and the strong exculpability experiment is based on [KY04; Ate+00; BSZ05]. The experiments are defined in Figures 4.5 and 4.6.

Definition 4.1:
A reputation system $\mathcal{RS}$ is *anonymous* if for all probabilistic polynomial-time adversaries $\mathcal{A}$ there exists a negligible function negl such that

$$\Pr[\mathrm{Exp}^{anon}_{\mathcal{A},\mathcal{RS}}(\lambda) = 1] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

The probability is taken over all random bits used in the experiment and by $\mathcal{A}$. △

The anonymity experiment $\mathrm{Exp}^{anon}_{\mathcal{A},\mathcal{RS}}(\lambda)$ asks an adversary to distinguish which of two honest raters generated a rating for some *item*, where the raters, the message, and the *item* are chosen by the adversary. In this setting it is possible to corrupt the provider and all raters, except those the adversary wants to be challenged on. This restriction is necessary because otherwise the adversary could possibly link different ratings to determine the rater's identities. Analogously, revocation of those raters is prohibited.

The anonymity requirements can slightly be relaxed to an experiment where an adversary is not allowed to query the Open oracle. We denote this modification by CPA-anonymity and the anonymity experiment as defined in Figure 4.5 by CCA2-anonymity, analogously to [BBS04].

Definition 4.2:
A reputation system $\mathcal{RS}$ is *publicly-linkable* if for all probabilistic polynomial-time adversaries $\mathcal{A}$ there exists a negligible function negl such that

$$\Pr[\mathrm{Exp}^{publink}_{\mathcal{A},\mathcal{RS}}(\lambda) = 1] \leq \mathsf{negl}(\lambda).$$

The probability is taken over all random bits used in the experiment and by $\mathcal{A}$. △

The public linkability experiment $\mathrm{Exp}^{publink}_{\mathcal{A},\mathcal{RS}}(\lambda)$ asks an adversary to output message-rating pairs, for the same *item* chosen by the adversary, such that all pairs are valid and there are no two pairs that can be linked. The number of pairs must be one more than the number of raters in the system. We allow the adversary to corrupt all raters, but the provider has to be honest. If the provider would be corrupt, he could create rating keys for non-existing raters.

Definition 4.3:
A reputation system $\mathcal{RS}$ is *traceable* if for all probabilistic polynomial-time adversaries $\mathcal{A}$ there exists a negligible function negl such that

$$\Pr[\mathrm{Exp}^{trace}_{\mathcal{A},\mathcal{RS}}(\lambda) = 1] \leq \mathsf{negl}(\lambda).$$

The probability is taken over all random bits used in the experiment and by $\mathcal{A}$. △

The traceability experiment $\mathrm{Exp}^{trace}_{\mathcal{A},\mathcal{RS}}(\lambda)$ asks an adversary to output a message-rating pair, for some *item* chosen by the adversary, which is valid but cannot be traced back to a corrupted rater. In this experiment the provider must be honest because he could generate rating keys for non-existing raters.

Definition 4.4:
A reputation system $\mathcal{RS}$ is *strong exculpable* if for all probabilistic polynomial-time adversaries $\mathcal{A}$ there exists a negligible function negl such that

$$\Pr[\mathrm{Exp}^{str-ex}_{\mathcal{A},\mathcal{RS}}(\lambda) = 1] \leq \mathsf{negl}(\lambda).$$

The probability is taken over all random bits used in the experiment and by $\mathcal{A}$. △

The strong exculpability experiment $\mathrm{Exp}^{str-ex}_{\mathcal{A},\mathcal{RS}}(\lambda)$ asks an adversary to output a message-rating pair, for some *item* chosen by the adversary, which is valid and can be traced back to an honest rater. We give an adversary the possibility to corrupt raters and the provider. Because the provider can always generate rating keys for non-existing raters, we force the adversary to output a rating on behalf of an honest rater.

4.1.3 Discussion

The defined experiments imply two different attack scenarios:

> In the first scenario, for anonymity and strong exculpability, we allow an adversary to corrupt the provider and a set of raters. We omit an oracle enabling the adversary to send corrupted data to honest raters in the $\mathsf{Join}, \mathsf{Issue}$-protocol, because this functionality is covered by the $\mathsf{WItemList}$ and $\mathsf{WIdentList}$ oracles. Due to the assumption that all rating keys are publicly verifiable (as stated in Section 4.1.1), an honest rater would only accept valid keys in the join-issue protocol. The same is implicitly done by our oracles and in the experiments.
>
> In the second scenario, for public linkability and traceability, providers are assumed to be honest, whereas a set of raters can be corrupted. In particular, this implies that raters and providers are disjoint sets. The restriction to honest providers is necessary because a corrupted provider could generate rating keys for non-existing raters. With an appropriate identity management this can be prevented and we could also allow corrupted providers in the experiments for public linkability and traceability.

Albeit based on dynamic group signatures, our model for reputation systems and the corresponding security properties are defined with respect to an honest system manager: during the Open algorithm no opening-proof is generated and hence non-frameability is not considered. This is just a simplification that allows us to concentrate on the combination of anonymity and linkability in a system of simultaneously existent group signature schemes. Nevertheless, we include strong exculpability as a related but weaker notion than non-frameability to protect honest raters. To add non-frameability to reputation systems the same techniques as in dynamic group signatures can be used. They are applied in the construction provided in Chapter 6.

Another important issue is that of timing the operations. The provider may correlate transactions and ratings by their timing, thereby threatening the anonymity of raters. Hence, our reputation systems needs a mechanism to prevent such attacks. In [CSK13; Ker09; GK11] different solutions to this problem are proposed, which can be incorporated into our construction.

4.2 Construction of a Reputation System

In this section we define our reputation system based on the Decision Linear Problem (Definition 2.32) and the Strong Diffie-Hellman Problem (Definition 2.34). To give some intuition for this system we provide an honest-verifier zero-knowledge proof of knowledge in Section 4.2.1. The complete construction of our reputation system is given in Section 4.2.2.

4.2.1 Building Blocks and Intuition

As discussed in Section 4.1.2 strong exculpability prevents honest raters from being blamed for having rated some product when the rater did not submit said rating. To achieve this property, every rater chooses his personal secret key *rsk* and uses it to generate ratings. Unfortunately, the SDH Problem is not well suited to support such a feature directly. Hence, we extend the SDH Problem and use the extension in our construction.

Definition 4.5: extended Strong Diffie-Hellman – eSDH
Given a group description $\mathbb{GD} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathrm{e}, g_1, g_2)$ of Type-1 or Type-2 pairings and a tuple $\left(h, g_2, g_2^{\gamma}, g_2^{(\gamma^2)}, \ldots, g_2^{(\gamma^{q(\lambda)})}\right)$, where $h \xleftarrow{u} \mathbb{G}_1$, the extended Strong Diffie-Hellman Problem is to output a triple $\left((g_1 \cdot h^y)^{\frac{1}{x+\gamma}}, x, y\right)$, where $x, y \in \mathbb{Z}_p$.

We say the extended Strong Diffie-Hellman Assumption holds for bilinear group generator $\mathsf{BiGrGen}$ if for all probabilistic polynomial time adversaries $\mathcal{A}$ and for every polynomially bounded function $q : \mathbb{Z} \to \mathbb{Z}$ there exists a negligible function negl such that

$$\Pr\left[\mathcal{A}\left(\mathbb{GD}, h, g_2^{\gamma}, g_2^{(\gamma^2)}, \ldots, g_2^{(\gamma^{q(\lambda)})}\right) = \left((g_1 \cdot h^y)^{\frac{1}{x+\gamma}}, x, y\right)\right] \leq \mathsf{negl}(\lambda),$$

where $\mathbb{GD} \leftarrow \mathsf{BiGrGen}(1^{\lambda})$, $h \xleftarrow{u} \mathbb{G}_1$, and $\gamma \xleftarrow{u} \mathbb{Z}_p$. The probability is over the random bits used by $\mathsf{BiGrGen}$, the uniform random choice of $\gamma \xleftarrow{u} \mathbb{Z}_p, h \xleftarrow{u} \mathbb{G}_1$, and over the random bits used by $\mathcal{A}$. △

Lemma 4.1:
If the Strong Diffie-Hellman Assumption holds for bilinear group generator $\mathsf{BiGrGen}$, then the extended Strong Diffie-Hellman Assumption also holds for $\mathsf{BiGrGen}$. ◇

Proof. Let $\mathbb{GD}$ be a group description $\mathbb{GD} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathrm{e}, g_1, g_2)$ output by $\mathsf{BiGrGen}(1^{\lambda})$ and let $\mathcal{A}$ be a probabilistic polynomial-time algorithm that solves the eSDH Problem for some polynomially bounded function q with probability $\varepsilon(\lambda)$. We construct a probabilistic

polynomial-time adversary $\mathcal{B}$ that solves the SDH Problem for function q with probability $\varepsilon(\lambda) - \frac{1}{p}$, as follows:

Given an instance of the SDH Problem $\left(\mathbb{GD}, g_2^{\gamma}, g_2^{(\gamma^2)}, \ldots, g_2^{(\gamma^{q(\lambda)})}\right)$, $\mathcal{B}$ chooses $\alpha \xleftarrow{u} \mathbb{Z}_p$ and computes $h \coloneqq g_1^{\alpha}$. Then $\mathcal{B}$ hands $\left(\mathbb{GD}, h, g_2^{\gamma}, g_2^{(\gamma^2)}, \ldots, g_2^{(\gamma^{q(\lambda)})}\right)$ as the eSDH Problem instance to adversary $\mathcal{A}$. When $\mathcal{A}$ outputs a solution $\left((g_1 \cdot h^y)^{\frac{1}{x+\gamma}}, x, y\right)$, $\mathcal{B}$ computes $c \coloneqq (1 + \alpha \cdot y)^{-1}$ and

$$\left((g_1 \cdot h^y)^{\frac{1}{x+\gamma}}\right)^c = (g_1 \cdot h^y)^{\frac{c}{x+\gamma}} = \left(g_1 \cdot g_1^{\alpha \cdot y}\right)^{\frac{c}{x+\gamma}} = g_1^{\frac{(1+\alpha \cdot y)}{x+\gamma} \cdot \frac{1}{(1+\alpha \cdot y)}} = g_1^{\frac{1}{x+\gamma}},$$

and outputs $\left(g_1^{\frac{1}{x+\gamma}}, x\right)$ as a solution for his SDH Problem instance.

The running time of $\mathcal{B}$ is composed of the random choice of α, the running time of $\mathcal{A}$, and the transformation of $\mathcal{A}$'s solution. Since all of these are probabilistic polynomial-time, $\mathcal{B}$ is a probabilistic polynomial-time algorithm.

In the case that $(1 + \alpha \cdot y)$ is not invertible $\mathcal{B}$ aborts, but this happens only with probability $\frac{1}{p}$. Hence, $\mathcal{B}$ outputs a solution to the SDH Problem for function q with probability $\varepsilon(\lambda) - \frac{1}{p}$. Since the SDH Assumption holds for bilinear group generator BiGrGen, the probability $\varepsilon(\lambda) - \frac{1}{p}$ is negligible, which is only possible when $\varepsilon(\lambda)$ is negligible. That in turn means that the eSDH Assumption holds for bilinear group generator BiGrGen. □

Analogously to dynamic group signatures, our construction of a reputation system is based on non-interactive zero-knowledge (NIZK) proofs of knowledge. In the following, we present an honest-verifier zero-knowledge protocol to prove possession of a solution to the extended Strong Diffie-Hellman Problem. This protocol will be transformed into a NIZK proof by applying the Fiat-Shamir heuristic 2.25.

In our protocol the prover has to convince an honest verifier that he knows a triple (A, x, y) such that $A^{x+\gamma} = g_1 \cdot h^y$, where $A, g_1, h \in \mathbb{G}_1$ and $x, y, \gamma \in \mathbb{Z}_p$. The value γ is neither known by the prover nor the verifier. The public values of the protocol are a group description $\mathbb{GD}$ of Type-2 bilinear groups as the output of a bilinear group generator BiGrGen, six random elements $d, f, h, u, v, w \in \mathbb{G}_1$, the value $C = h^y$, and the value $W = g_2^{\gamma}$. To obtain a provably secure protocol we assume that the Decision Linear Assumption and the extended Strong Diffie-Hellman Assumption hold for BiGrGen.

Remark: The triple (A, x, y) is a solution of the extended SDH Problem, if and only if the equation $\mathrm{e}(A, g_2)^x \cdot \mathrm{e}(A, W) \cdot \mathrm{e}(h, g_2)^{-y} = \mathrm{e}(g_1, g_2)$ holds. This will be used in the protocol. ¤

Protocol 4.1:
The following protocol is executed by a prover $\mathcal{P}$, proving knowledge of a solution (A, x, y) to an eSDH Problem, and an honest verifier $\mathcal{V}$. Both have access to the public values $(\mathbb{GD}, d, f, h, u, v, w, C, W)$, as described above.

$\mathcal{P}$: Choose $\alpha, \beta, \mu \leftarrow_u \mathbb{Z}_p$, compute

$$T_1 \coloneqq u^{\alpha} \quad T_2 \coloneqq v^{\beta} \quad T_3 \coloneqq A \cdot w^{\alpha+\beta} \quad T_4 \coloneqq d^{\mu} \quad T_5 \coloneqq f^{\mu+y}$$

and the helper values

$$\delta_1 \coloneqq x \cdot \alpha \qquad \delta_2 \coloneqq x \cdot \beta.$$

Now $\mathcal{P}$ and $\mathcal{V}$ run a Σ-protocol to prove knowledge of $(\alpha, \beta, x, y, \mu, \delta_1, \delta_2)$:

$\mathcal{P}$: Choose $r_\alpha, r_\beta, r_x, r_y, r_\mu, r_{\delta_1}, r_{\delta_2} \leftarrow_u \mathbb{Z}_p$, compute

$$\begin{aligned}
R_1 &\coloneqq u^{r_\alpha} & R_2 &\coloneqq v^{r_\beta} \\
R_3 &\coloneqq \mathrm{e}(T_3, g_2)^{r_x} \cdot \mathrm{e}(w, W)^{-r_\alpha - r_\beta} \cdot \mathrm{e}(w, g_2)^{-r_{\delta_1} - r_{\delta_2}} \cdot \mathrm{e}(h, g_2)^{-r_y} \\
R_4 &\coloneqq T_1^{r_x} \cdot u^{-r_{\delta_1}} & R_5 &\coloneqq T_2^{r_x} \cdot v^{-r_{\delta_2}} \\
R_6 &\coloneqq d^{r_\mu} & R_7 &\coloneqq f^{r_\mu + r_y}
\end{aligned}$$

and send $(T_1, T_2, T_3, T_4, T_5, R_1, R_2, R_3, R_4, R_5, R_6, R_7)$ to $\mathcal{V}$.

$\mathcal{V}$: Choose $c \leftarrow_u \mathbb{Z}_p$ as a challenge and send c to the prover $\mathcal{P}$.

$\mathcal{P}$: Compute

$$\begin{aligned}
s_\alpha &\coloneqq r_\alpha + c \cdot \alpha & s_\beta &\coloneqq r_\beta + c \cdot \beta \\
s_x &\coloneqq r_x + c \cdot x & s_y &\coloneqq r_y + c \cdot y \\
s_\mu &\coloneqq r_\mu + c \cdot \mu & & \\
s_{\delta_1} &\coloneqq r_{\delta_1} + c \cdot \delta_1 & s_{\delta_2} &\coloneqq r_{\delta_2} + c \cdot \delta_2
\end{aligned}$$

and send them to the verifier $\mathcal{V}$.

$\mathcal{V}$: Accept if all of the following equations hold:

$$\begin{aligned}
R_1 &= u^{s_\alpha} \cdot T_1^{-c} & R_2 &= v^{s_\beta} \cdot T_2^{-c} \\
R_3 &= \frac{\mathrm{e}(T_3, g_2)^{s_x} \cdot \mathrm{e}(T_3, W)^{c}}{\mathrm{e}(w, W)^{s_\alpha + s_\beta} \cdot \mathrm{e}(w, g_2)^{s_{\delta_1} + s_{\delta_2}} \cdot \mathrm{e}(h, g_2)^{s_y} \cdot \mathrm{e}(g_1, g_2)^{c}} & & \\
R_4 &= T_1^{s_x} \cdot u^{-s_{\delta_1}} & R_5 &= T_2^{s_x} \cdot v^{-s_{\delta_2}} \\
R_6 &= d^{s_\mu} \cdot T_4^{-c} & R_7 &= f^{s_\mu + s_y} \cdot T_5^{-c}
\end{aligned}$$

▲

Theorem 4.1:
Protocol 4.1 is an honest-verifier zero-knowledge argument of knowledge of an eSDH triple (A, x, y) under the Decision Linear Assumption. ◇

Proof. The proof follows from the Lemmas 4.2, 4.3, and 4.4. □

Lemma 4.2:
Protocol 4.1 is complete. ◇

Proof. If the prover $\mathcal{P}$ is honest and in possession of a triple (A, x, y) such that $A^{x+\gamma} \cdot h^{-y} = g_1$, he follows the computations specified for him in the protocol. In this case the verification equations hold:

$$u^{s_\alpha} \cdot T_1^{-c} = u^{r_\alpha + c \cdot \alpha} \cdot (u^{\alpha})^{-c} = u^{r_\alpha} = R_1 \tag{4.1}$$

$$v^{s_\beta} \cdot T_2^{-c} = v^{r_\beta + c \cdot \beta} \cdot (v^{\beta})^{-c} = v^{r_\beta} = R_2 \tag{4.2}$$

$$
\begin{aligned}
&\frac{\mathrm{e}(T_3, g_2)^{s_x} \cdot \mathrm{e}(T_3, W)^c}{\mathrm{e}(w, W)^{s_\alpha + s_\beta} \cdot \mathrm{e}(w, g_2)^{s_{\delta_1} + s_{\delta_2}} \cdot \mathrm{e}(h, g_2)^{s_y} \cdot \mathrm{e}(g_1, g_2)^c} \\
&= \frac{\mathrm{e}(T_3, g_2)^{r_x + c \cdot x} \cdot \mathrm{e}(T_3, W)^c}{\mathrm{e}(w, W)^{r_\alpha + c \cdot \alpha + r_\beta + c \cdot \beta} \cdot \mathrm{e}(w, g_2)^{r_{\delta_1} + c \cdot \delta_1 + r_{\delta_2} + c \cdot \delta_2} \cdot \mathrm{e}(h, g_2)^{r_y + c \cdot y} \cdot \mathrm{e}(g_1, g_2)^c} \\
&= R_3 \cdot \left(\frac{\mathrm{e}(T_3, g_2)^x \cdot \mathrm{e}(T_3, W)}{\mathrm{e}(w, W)^{\alpha+\beta} \cdot \mathrm{e}(w, g_2)^{\delta_1 + \delta_2} \cdot \mathrm{e}(h, g_2)^y \cdot \mathrm{e}(g_1, g_2)} \right)^c \\
&= R_3 \cdot \left(\frac{\mathrm{e}(A \cdot w^{\alpha+\beta}, g_2)^x \cdot \mathrm{e}(A \cdot w^{\alpha+\beta}, W)}{\mathrm{e}(w^{\alpha+\beta}, W) \cdot \mathrm{e}(w, g_2)^{x \cdot \alpha + x \cdot \beta} \cdot \mathrm{e}(h, g_2)^y \cdot \mathrm{e}(g_1, g_2)} \right)^c \\
&= R_3 \cdot \left(\frac{\mathrm{e}(A, g_2)^x \cdot \mathrm{e}(w^{\alpha+\beta}, g_2)^x \cdot \mathrm{e}(A, W) \cdot \mathrm{e}(w^{\alpha+\beta}, W)}{\mathrm{e}(w^{\alpha+\beta}, W) \cdot \mathrm{e}(w^{\alpha+\beta}, g_2)^x \cdot \mathrm{e}(h, g_2)^y \cdot \mathrm{e}(g_1, g_2)} \right)^c \\
&= R_3 \cdot \left(\mathrm{e}(A, g_2)^x \cdot \mathrm{e}(A, W) \cdot \mathrm{e}(h, g_2)^{-y} \cdot \mathrm{e}(g_1, g_2)^{-1} \right)^c \\
&= R_3 \cdot \left(\mathrm{e}(A^x, g_2) \cdot \mathrm{e}(A^\gamma, g_2) \cdot \mathrm{e}(h^{-y}, g_2) \cdot \mathrm{e}(g_1, g_2)^{-1} \right)^c \\
&= R_3 \cdot \left(\mathrm{e}(A^x \cdot A^\gamma \cdot h^{-y}, g_2) \cdot \mathrm{e}(g_1, g_2)^{-1} \right)^c \\
&= R_3 \cdot \left(\mathrm{e}(g_1, g_2) \cdot \mathrm{e}(g_1, g_2)^{-1} \right)^c \\
&= R_3 \qquad (4.3)
\end{aligned}
$$

$$
\begin{aligned}
&T_1^{s_x} \cdot u^{-s_{\delta_1}} = (u^\alpha)^{r_x + c \cdot x} \cdot u^{-r_{\delta_1} - c \cdot \delta_1} = (u^\alpha)^{r_x + c \cdot x} \cdot u^{-r_{\delta_1} - c \cdot x \cdot \alpha} \\
&= (u^\alpha)^{r_x} \cdot u^{-r_{\delta_1}} = R_4 \qquad (4.4) \\
&T_2^{s_x} \cdot v^{-s_{\delta_2}} = (v^\beta)^{r_x + c \cdot x} \cdot v^{-r_{\delta_2} - c \cdot \delta_2} = (v^\beta)^{r_x + c \cdot x} \cdot v^{-r_{\delta_2} - c \cdot x \cdot \beta} \\
&= (v^\beta)^{r_x} \cdot v^{-r_{\delta_2}} = R_5 \qquad (4.5) \\
&d^{s_\mu} \cdot T_4^{-c} = d^{r_\mu + c \cdot \mu} \cdot (d^\mu)^{-c} = d^{r_\mu} = R_6 \qquad (4.6) \\
&f^{s_\mu + s_y} \cdot T_5^{-c} = f^{r_\mu + c \cdot \mu + r_y + c \cdot y} \cdot (f^{\mu + y})^{-c} = f^{r_\mu + r_y} = R_7 \qquad (4.7)
\end{aligned}
$$

So the verifier will always accept when the prover is honest. □

Lemma 4.3:
For an honest verifier, transcripts of Protocol 4.1 can be simulated under the Decision Linear Assumption. ◇

Proof. We describe a simulator $\mathcal{S}$ that outputs transcripts for Protocol 4.1 that are indistinguishable from real protocol transcripts. At first, $\mathcal{S}$ chooses $\hat{A} \xleftarrow{u} \mathbb{G}_1$ and $\hat{y}, \alpha, \beta, \mu \xleftarrow{u} \mathbb{Z}_p$, and computes

$$T_1 := u^\alpha \qquad T_2 := v^\beta \qquad T_3 := \hat{A} \cdot w^{\alpha+\beta} \qquad T_4 := d^\mu \qquad T_5 := f^{\mu+\hat{y}}.$$

The remainder of the simulation is independent of the values $(\hat{A}, \hat{y}, \alpha, \beta, \mu)$ and defines the simulator for the Σ-protocol that is integrated in Protocol 4.1. Hence, it can be used for pre-specified tuples $(T_1, T_2, T_3, T_4, T_5)$.

$\mathcal{S}$ chooses $c, s_\alpha, s_\beta, s_x, s_{\hat{y}}, s_\mu, s_{\delta_1}, s_{\delta_2} \xleftarrow{u} \mathbb{Z}_p$ and computes

$$
\begin{aligned}
&R_1 := u^{s_\alpha} \cdot T_1^{-c} \qquad R_2 := v^{s_\beta} \cdot T_2^{-c} \\
&R_3 := \frac{\mathrm{e}(T_3, g_2)^{s_x} \cdot \mathrm{e}(T_3, W)^c}{\mathrm{e}(w, W)^{s_\alpha + s_\beta} \cdot \mathrm{e}(w, g_2)^{s_{\delta_1} + s_{\delta_2}} \cdot \mathrm{e}(h, g_2)^{s_{\hat{y}}} \cdot \mathrm{e}(g_1, g_2)^c} \\
&R_4 := T_1^{s_x} \cdot u^{-s_{\delta_1}} \qquad R_5 := T_2^{s_x} \cdot v^{-s_{\delta_2}}
\end{aligned}
$$

$R_6 \coloneqq d^{s_\mu} \cdot T_4^{-c}$ $\qquad$ $R_7 \coloneqq f^{s_\mu + s_{\hat{y}}} \cdot T_5^{-c}$

using the verification equations from Protocol 4.1.

Finally, $\mathcal{S}$ outputs the simulated transcript

$(T_1, T_2, T_3, T_4, T_5, R_1, R_2, R_3, R_4, R_5, R_6, R_7, c, s_\alpha, s_\beta, s_x, s_{\hat{y}}, s_\mu, s_{\delta_1}, s_{\delta_2})$.

Now we show that the transcripts created by $\mathcal{S}$ are indistinguishable from transcripts of the real protocol under the Decision Linear Assumption.

The triple (T_1, T_2, T_3) of real transcripts is a Linear Encryption of A under the public key (u, v, w). Hence, it is indistinguishable from a random triple output by $\mathcal{S}$ under the Decision Linear Assumption. Analogously, the 6-tuple (h, d, f, C, T_4, T_5) in real protocol transcripts is an instance of the Decision Linear Problem that is indistinguishable from a completely random 6-tuple of simulated transcripts.

By choosing the values $s_\alpha, s_\beta, s_x, s_{\hat{y}}, s_\mu, s_{\delta_1}, s_{\delta_2}$, and c uniformly at random during the simulation, the values $R_1, R_2, R_3, R_4, R_5, R_6$, and R_7 are fixed such that the verification equations hold. Therefore, the tuple $(R_1, R_2, R_3, R_4, R_5, R_6, R_7, c, s_\alpha, s_\beta, s_x, s_{\hat{y}}, s_\mu, s_{\delta_1}, s_{\delta_1})$ is simulated perfectly, as required for the special honest-verifier zero-knowledge property of Σ-protocols.

Using a standard hybrid argument, it follows that transcripts generated by the simulator $\mathcal{S}$ are indistinguishable from transcripts of the real protocol under the Decision Linear Assumption. □

Lemma 4.4:
Protocol 4.1 is an argument of knowledge. ◇

Proof. At first, we describe an Extractor $\mathcal{E}'$ for the Σ-protocol that is integrated in Protocol 4.1. Then we show how $\mathcal{E}'$ can be used to construct an extractor for Protocol 4.1.

According to the special soundness property of Σ-protocols the extractor $\mathcal{E}'$ gets two accepting transcripts (a, c, r) and (a, c', r') as input, where the announcement $a = (T_1, T_2, T_3, T_4, T_5, R_1, R_2, R_3, R_4, R_5, R_6, R_7)$ is identical for both transcripts, $c \neq c'$, and the responses $r = (s_\alpha, s_\beta, s_x, s_y, s_\mu, s_{\delta_1}, s_{\delta_2})$ and $r' = (s'_\alpha, s'_\beta, s'_x, s'_y, s'_\mu, s'_{\delta_1}, s'_{\delta_2})$ are different. From these inputs $\mathcal{E}'$ can compute the values $(\hat{\alpha}, \hat{\beta}, \hat{x}, \hat{y}, \hat{\mu}, \hat{\delta_1}, \hat{\delta_2})$ by setting

$$\Delta c \coloneqq c - c' \qquad \Delta s_\alpha \coloneqq s_\alpha - s'_\alpha \qquad \Delta s_\beta \coloneqq s_\beta - s'_\beta$$
$$\Delta s_x \coloneqq s_x - s'_x \qquad \Delta s_y \coloneqq s_y - s'_y \qquad \Delta s_\mu \coloneqq s_\mu - s'_\mu$$
$$\Delta s_{\delta_1} \coloneqq s_{\delta_1} - s'_{\delta_1} \qquad \Delta s_{\delta_2} \coloneqq s_{\delta_2} - s'_{\delta_2}$$

and computing

$$\hat{\alpha} \coloneqq \frac{\Delta s_\alpha}{\Delta c} \qquad \hat{\beta} \coloneqq \frac{\Delta s_\alpha}{\Delta c}$$
$$\hat{x} \coloneqq \frac{\Delta s_x}{\Delta c} \qquad \hat{y} \coloneqq \frac{\Delta s_y}{\Delta c} \qquad \hat{\mu} \coloneqq \frac{\Delta s_\mu}{\Delta c}$$
$$\hat{\delta_1} \coloneqq \frac{\Delta s_{\delta_1}}{\Delta c} \qquad \hat{\delta_2} \coloneqq \frac{\Delta s_{\delta_2}}{\Delta c}.$$

Using these values $\mathcal{E}$ can compute $\hat{A} = T_3 \cdot w^{-\hat{\alpha}-\hat{\beta}}$ and output $(\hat{A}, \hat{x}, \hat{y})$.

By inspecting the verification equations for both transcripts it follows that the extracted values are exactly the ones the prover knows:

$$\left.\begin{aligned} R_1 &= u^{s_\alpha} \cdot T_1^{-c} \\ R_1 &= u^{s'_\alpha} \cdot T_1^{-c'} \end{aligned}\right\} \Rightarrow 1_{\mathbb{G}_1} = \frac{u^{s_\alpha} \cdot T_1^{-c}}{u^{s'_\alpha} \cdot T_1^{-c'}} = u^{\Delta s_\alpha} \cdot T_1^{-\Delta c}$$
$$\Rightarrow T_1^{\Delta c} = u^{\Delta s_\alpha} \Rightarrow T_1 = u^{\hat{\alpha}}$$
$$\left.\begin{aligned} R_2 &= v^{s_\beta} \cdot T_2^{-c} \\ R_2 &= v^{s'_\beta} \cdot T_2^{-c'} \end{aligned}\right\} \Rightarrow 1_{\mathbb{G}_1} = \frac{v^{s_\beta} \cdot T_2^{-c}}{v^{s'_\beta} \cdot T_2^{-c'}} = v^{\Delta s_\beta} \cdot T_2^{-\Delta c}$$
$$\Rightarrow T_2^{\Delta c} = v^{\Delta s_\beta} \Rightarrow T_2 = v^{\hat{\beta}}$$
$$\left.\begin{aligned} R_6 &= d^{s_\mu} \cdot T_4^{-c} \\ R_6 &= d^{s'_\mu} \cdot T_4^{-c'} \end{aligned}\right\} \Rightarrow 1_{\mathbb{G}_1} = \frac{d^{s_\mu} \cdot T_4^{-c}}{d^{s'_\mu} \cdot T_4^{-c'}} = d^{\Delta s_\mu} \cdot T_4^{-\Delta c}$$
$$\Rightarrow T_4^{\Delta c} = d^{\Delta s_\mu} \Rightarrow T_4 = d^{\hat{\mu}}$$
$$\left.\begin{aligned} R_7 &= f^{s_\mu + s_y} \cdot T_5^{-c} \\ R_7 &= f^{s_\mu + s_y} \cdot T_5^{-c} \end{aligned}\right\} \Rightarrow 1_{\mathbb{G}_1} = \frac{f^{s_\mu + s_y} \cdot T_5^{-c}}{f^{s'_\mu + s'_y} \cdot T_5^{-c'}} = f^{\Delta s_\mu + \Delta s_y} \cdot T_5^{-\Delta c}$$
$$\Rightarrow T_5^{\Delta c} = f^{\Delta s_\mu + \Delta s_y} \Rightarrow T_5 = f^{\hat{\mu} + \hat{y}}$$
$$\left.\begin{aligned} R_4 &= T_1^{s_x} \cdot u^{-s_{\delta_1}} \\ R_4 &= T_1^{s'_x} \cdot u^{-s'_{\delta_1}} \end{aligned}\right\} \Rightarrow 1_{\mathbb{G}_1} = \frac{T_1^{s_x} \cdot u^{-s_{\delta_1}}}{T_1^{s'_x} \cdot u^{-s'_{\delta_1}}} = T_1^{\Delta s_x} \cdot u^{-\Delta s_{\delta_1}}$$
$$\Rightarrow T_1^{\Delta s_x} = u^{\Delta s_{\delta_1}} \Rightarrow \hat{\alpha} \cdot \Delta s_x = \Delta s_{\delta_1}$$
$$\left.\begin{aligned} R_5 &= T_2^{s_x} \cdot v^{-s_{\delta_2}} \\ R_5 &= T_2^{s'_x} \cdot v^{-s'_{\delta_2}} \end{aligned}\right\} \Rightarrow 1_{\mathbb{G}_1} = \frac{T_2^{s_x} \cdot v^{-s_{\delta_2}}}{T_2^{s'_x} \cdot v^{-s'_{\delta_2}}} = T_2^{\Delta s_x} \cdot v^{-\Delta s_{\delta_2}}$$
$$\Rightarrow T_2^{\Delta s_x} = v^{\Delta s_{\delta_2}} \Rightarrow \hat{\beta} \cdot \Delta s_x = \Delta s_{\delta_2}$$
$$\left.\begin{aligned} R_3 &= \frac{\mathrm{e}(T_3, g_2)^{s_x} \cdot \mathrm{e}(T_3, W)^{c} \cdot \mathrm{e}(g_1, g_2)^{-c}}{\mathrm{e}(w, W)^{s_\alpha + s_\beta} \cdot \mathrm{e}(w, g_2)^{s_{\delta_1} + s_{\delta_2}} \cdot \mathrm{e}(h, g_2)^{s_y}} \\ R_3 &= \frac{\mathrm{e}(T_3, g_2)^{s'_x} \cdot \mathrm{e}(T_3, W)^{c'} \cdot \mathrm{e}(g_1, g_2)^{-c'}}{\mathrm{e}(w, W)^{s'_\alpha + s'_\beta} \cdot \mathrm{e}(w, g_2)^{s'_{\delta_1} + s'_{\delta_2}} \cdot \mathrm{e}(h, g_2)^{s'_y}} \end{aligned}\right\}$$
$$\Rightarrow 1_{\mathbb{G}_T} = \frac{\mathrm{e}(T_3, g_2)^{\Delta s_x} \cdot \mathrm{e}(T_3, W)^{\Delta c} \cdot \mathrm{e}(g_1, g_2)^{-\Delta c}}{\mathrm{e}(w, W)^{\Delta s_\alpha + \Delta s_\beta} \cdot \mathrm{e}(w, g_2)^{\Delta s_{\delta_1} + \Delta s_{\delta_2}} \cdot \mathrm{e}(h, g_2)^{\Delta s_y}}$$
$$\Rightarrow \left(\frac{\mathrm{e}(g_1, g_2)}{\mathrm{e}(T_3, W)}\right)^{\Delta c} = \frac{\mathrm{e}(T_3, g_2)^{\Delta s_x}}{\mathrm{e}(w, W)^{\Delta s_\alpha + \Delta s_\beta} \cdot \mathrm{e}(w, g_2)^{\Delta s_{\delta_1} + \Delta s_{\delta_2}} \cdot \mathrm{e}(h, g_2)^{\Delta s_y}}$$

By taking Δc-th root we obtain

$$\frac{\mathrm{e}(g_1, g_2)}{\mathrm{e}(T_3, W)} = \mathrm{e}(T_3, g_2)^{\hat{x}} \cdot \mathrm{e}(w, W)^{-\hat{\alpha} - \hat{\beta}} \cdot \mathrm{e}(w, g_2)^{-\hat{\alpha} \cdot \hat{x} - \hat{\beta} \cdot \hat{x}} \cdot \mathrm{e}(h, g_2)^{-\hat{y}}$$

which can be rearranged as

$$\begin{aligned} \mathrm{e}(g_1, g_2) &= \mathrm{e}(T_3, g_2)^{\hat{x}} \cdot \mathrm{e}(w^{-\hat{\alpha} - \hat{\beta}}, W) \cdot \mathrm{e}(w^{-\hat{\alpha} - \hat{\beta}}, g_2)^{\hat{x}} \cdot \mathrm{e}(h, g_2)^{-\hat{y}} \cdot \mathrm{e}(T_3, W) \\ &= \mathrm{e}(T_3 \cdot w^{-\hat{\alpha} - \hat{\beta}}, g_2)^{\hat{x}} \cdot \mathrm{e}(h, g_2)^{-\hat{y}} \cdot \mathrm{e}(T_3 \cdot w^{-\hat{\alpha} - \hat{\beta}}, W) \end{aligned}$$

By setting $\hat{A} = T_3 \cdot w^{-\hat{\alpha} - \hat{\beta}}$ it holds

$$\mathrm{e}(g_1, g_2) = \mathrm{e}(\hat{A}, g_2)^{\hat{x}} \cdot \mathrm{e}(h, g_2)^{-\hat{y}} \cdot \mathrm{e}(\hat{A}, W). \qquad \square$$

4.2.2 The Reputation System

We apply the Fiat-Shamir transformation (Definition 2.25) on Protocol 4.1 to obtain a signature of knowledge which is secure in the random oracle model. By extending this signature scheme we construct a reputation system. We use the challenge c as a part of the signature rather than the values $R_1, \ldots, R_7$, modeling the value c as the output of a random oracle. This technique is widely used in the context of group signatures [BBS04; BS04; Ate+00].

In a reputation system, the provider publishes *item*s for which the ratings are created. Every rater can create a single rating for every *item* without losing anonymity. Due to the public linkability two ratings for one *item* by the same rater can be detected. In such a case, the anonymity of the cheating rater is revoked by the system manager. By publishing a *revocation token* the system manager can declare ratings from the cheating rater as *invalid*. This invalidity can be checked by every verifier using verifier-local revocation [BS04; NF06].

We assume the communication between raters and the system manager and between raters and the provider to take place via secure channels. Furthermore, the rater's public key $rpk[i]$ is certified by the system manager, such that the provider can verify the integrity of the public keys during the Join-Issue protocol.

In the following we assume that every party has access to a group description $\mathbb{GD}$ of Type-2 bilinear groups generated by a bilinear group generator BiGrGen on input 1^λ, and two hash functions $\mathcal{H}\colon \{0,1\}^* \to \mathbb{Z}_p$ and $\mathcal{H}_1\colon \{0,1\}^* \to \mathbb{G}_2$. Furthermore, as in [BBS04], we use Linear Encryption - a CPA-secure Elgamal-like encryption scheme based on the Decision Linear Problem (Definition 2.32).

$\mathsf{KeyGen}_{SM}(1^\lambda)$

1: Select $w \xleftarrow{u} \mathbb{G}_1$, $\xi_1, \xi_2 \xleftarrow{u} \mathbb{Z}_p$ and compute $u := w^{\frac{1}{\xi_1}}, v := w^{\frac{1}{\xi_2}}$. The values (u, v, w) are the public key of the Linear Encryption, the values (ξ_1, ξ_2) are the corresponding secret key.
2: Select $\hat{d} \xleftarrow{u} \mathbb{G}_2$, $\zeta \xleftarrow{u} \mathbb{Z}_p$ and compute $d := \phi(\hat{d}), h := d^\zeta$ as the basis for public linkability and revocation.
3: Set $smpk := (u, v, w, h, d, \hat{d})$ and $smsk := (\xi_1, \xi_2, \zeta)$ as the system manager's public and secret keys.

$\mathsf{KeyGen}_P(1^\lambda, item)$

1: Select $g_{2_{item}} \xleftarrow{u} \mathbb{G}_2$ and set $g_{1_{item}} := \phi(g_{2_{item}})$.
2: Select $\gamma_{item} \xleftarrow{u} \mathbb{Z}_p$ and set $W_{item} := g_{2_{item}}^{\gamma_{item}}$.
3: Add the *item*-based public key $ipk[item] := (g_{1_{item}}, g_{2_{item}}, W_{item})$ to the *ItemList* and keep $isk[item] := \gamma_{item}$ secret as the *item*-based secret key.

$\mathsf{KeyGen}_R(1^\lambda, i)$

1: Select $y_i \xleftarrow{u} \mathbb{Z}_p$, set $rpk[i] := h^{y_i}$ and $rsk[i] := y_i$ as the rater's public and secret keys.

$\mathsf{Register}_{SM}(\mathrm{St}_{SM}, M_{SM}), \mathsf{Register}_{R}(\mathrm{St}_{R}, M_{R})$

R: The prospective rater sends his identity i to the system manager.

SM: **If** $reg[i]$ is already set **Then**
output `error` and exit.
Else run KeyGen_R to obtain the tuple $(rpk[i], rsk[i])$
set $reg[i] \coloneqq (i, rpk[i])$
send $(rpk[i], rsk[i])$ and a certificate for $rpk[i]$ to the rater i.

$\mathsf{Join}(\mathrm{St}_{R}, M_{R}), \mathsf{Issue}(\mathrm{St}_{P}, M_{P})$

R: Look up the public key for the used *item* $ipk[item] = (g_{1_{item}}, g_{2_{item}}, W_{item}) \in ItemList$ and send $(i, rpk[i])$ and the certificate for $rpk[i]$ to the provider.

P: **If** the certificate for $rpk[i]$ is invalid or $(rpk[i], \cdot) \in IL_{item}$ **Then**
output `error` and exit.
Else select $x_{i_{item}} \xleftarrow{u} \mathbb{Z}_p$ and compute $A_{i_{item}} \coloneqq (g_{1_{item}} \cdot rpk[i])^{\frac{1}{x_{i_{item}} + \gamma_{item}}}$
store $(rpk[i], rrsk[i, item] \coloneqq (A_{i_{item}}, x_{i_{item}}))$ in the identification list IL_{item} for *item*
hand $rrsk[i, item]$ to the rater i as his rating key for *item*

$\mathsf{Rate}(item, smpk, ipk[item], rrsk[i, item], rsk[i], M)$

1: compute $\hat{f} \in \mathbb{G}_2$ by $\hat{f} \coloneqq \mathcal{H}_1(item)$.

2: choose $\alpha, \beta, \mu \xleftarrow{u} \mathbb{Z}_p$, and compute

$$T_1 \coloneqq u^{\alpha} \qquad T_2 \coloneqq v^{\beta} \qquad T_3 \coloneqq A \cdot w^{\alpha+\beta} \qquad T_4 \coloneqq d^{\mu} \qquad T_5 \coloneqq \phi(\hat{f})^{\mu+y}$$
$$\delta_1 \coloneqq x \cdot \alpha \qquad \delta_2 \coloneqq x \cdot \beta.$$

3: choose $r_\alpha, r_\beta, r_x, r_y, r_\mu, r_{\delta_1}, r_{\delta_2} \xleftarrow{u} \mathbb{Z}_p$ and compute

$$R_1 \coloneqq u^{r_\alpha} \qquad R_2 \coloneqq v^{r_\beta}$$
$$R_3 \coloneqq \mathrm{e}(T_3, g_2)^{r_x} \cdot \mathrm{e}(w, W)^{-r_\alpha - r_\beta} \cdot \mathrm{e}(w, g_2)^{-r_{\delta_1} - r_{\delta_2}} \cdot \mathrm{e}(h, g_2)^{-r_y}$$
$$R_4 \coloneqq T_1^{r_x} \cdot u^{-r_{\delta_1}} \qquad R_5 \coloneqq T_2^{r_x} \cdot v^{-r_{\delta_2}}$$
$$R_6 \coloneqq d^{r_\mu} \qquad R_7 \coloneqq \phi(\hat{f})^{r_\mu + r_y}$$

4: compute a challenge value c using $\mathcal{H}$:

$$c \coloneqq \mathcal{H}(M, item, smpk, ipk[item], T_1, T_2, T_3, T_4, T_5, R_1, R_2, R_3, R_4, R_5, R_6, R_7).$$

5: compute

$$s_\alpha \coloneqq r_\alpha + c \cdot \alpha \qquad s_\beta \coloneqq r_\beta + c \cdot \beta$$
$$s_x \coloneqq r_x + c \cdot x \qquad s_y \coloneqq r_y + c \cdot y$$
$$s_\mu \coloneqq r_\mu + c \cdot \mu$$
$$s_{\delta_1} \coloneqq r_{\delta_1} + c \cdot \delta_1 \qquad s_{\delta_2} \coloneqq r_{\delta_2} + c \cdot \delta_2$$

6: output the rating $\sigma \coloneqq (item, T_1, T_2, T_3, T_4, T_5, c, s_\alpha, s_\beta, s_x, s_y, s_\mu, s_{\delta_1}, s_{\delta_2})$.

Verify$(item, smpk, ipk[item], \mathcal{RL}, M, \sigma)$

1: compute $\hat{f} \in \mathbb{G}_2$ by $\hat{f} \coloneqq \mathcal{H}_1(item)$.
2: compute

$$R_1 \coloneqq u^{s_\alpha} \cdot T_1^{-c} \qquad R_2 \coloneqq v^{s_\beta} \cdot T_2^{-c}$$
$$R_3 \coloneqq \frac{\mathrm{e}(T_3, g_2)^{s_x} \cdot \mathrm{e}(T_3, W)^c}{\mathrm{e}(w, W)^{s_\alpha + s_\beta} \cdot \mathrm{e}(w, g_2)^{s_{\delta_1} + s_{\delta_2}} \cdot \mathrm{e}(h, g_2)^{s_y} \cdot \mathrm{e}(g_1, g_2)^c}$$
$$R_4 \coloneqq T_1^{s_x} \cdot u^{-s_{\delta_1}} \qquad R_5 \coloneqq T_2^{s_x} \cdot v^{-s_{\delta_2}}$$
$$R_6 \coloneqq d^{s_\mu} \cdot T_4^{-c} \qquad R_7 \coloneqq \phi(\hat{f})^{s_\mu + s_y} \cdot T_5^{-c}$$

3: **If** $c \neq \mathcal{H}(M, item, smpk, ipk[item], T_1, T_2, T_3, T_4, T_5, R_1, R_2, R_3, R_4, R_5, R_6, R_7)$ **Then**
4: output 0.
5: **For Each** $D \in \mathcal{RL}$ **do**
6: **If** $\mathrm{e}(T_5, \hat{d}) = \mathrm{e}(D \cdot T_4, \hat{f})$ **Then**
7: output 0.
8: output 1.

Open$(smpk, smsk, M, \sigma)$

1: run Steps 1-4 of the Verify-algorithm
2: **If** Step 4 outputs 0 **Then**
3: output `failure`
4: compute $A_{i_{item}} \coloneqq T_3 \cdot T_1^{-\xi_1} \cdot T_2^{-\xi_2}$
5: **For Each** $e \in IL_{item}$ **do**
6: **If** $e = (rpk, (A_{i_{item}}, x_{i_{item}}))$ for some rpk and some $x_{i_{item}}$ **Then**
7: **For Each** $r \in reg$ **do**
8: **If** $r = (i, rpk)$ for some i **Then**
9: output i
10: output `failure`

Link$(item, smpk, ipk[item], (M', \sigma'), (M'', \sigma''))$

1: run Steps 1-4 of the Verify-algorithm for (M', σ') and (M'', σ'')
2: **If** Step 4 outputs 0 for at least one of (M', σ') and (M'', σ'') **Then**
3: output 0
4: compute $\hat{f} \in \mathbb{G}_2$ by $\hat{f} \coloneqq \mathcal{H}_1(item)$
5: **If** $\mathrm{e}(T_5' \cdot T_5''^{-1}, \hat{d}) = \mathrm{e}(T_4' \cdot T_4''^{-1}, \hat{f})$ **Then**
6: output 1
7: **Else**
8: output 0

Revoke$(smpk, smsk, i)$

1: look up $rpk[i]$ in $reg[i]$
2: compute $rrt[i] \coloneqq rpk[i]^{\frac{1}{\zeta}}$
3: set $\mathcal{RL} \coloneqq \mathcal{RL} \cup \{rrt[i]\}$

Remark: We assume the communication between raters and the system manager and between raters and the provider to take place via secure channels. Furthermore, the rater's public key $rpk[i]$ is certified by the system manager, such that the provider can verify the integrity of the public keys during the Join, Issue-protocol. Since we assume the system manager to be honest, we can let him choose the rater's public and secret keys.

We further stress that the provided construction does not consider concurrent executions of the interactive protocols, although the security model is designed for this purpose. To guarantee security in concurrent executions the technique of Damgård for constructing concurrent zero-knowledge protocols, Lemma 2.1, can be used. This technique is applied in the construction of a secure reputation system in Chapter 6. ¤

Correctness: The correctness of the reputation system can be shown as follows:

- Protocol 4.1 is correct, so every honestly created rating will be declared as valid.
- Revocation token are computed correctly.
- For honestly created ratings the system manager can always recover the identity of the rater, because of the correctness of the Linear Encryption.
- Two ratings for the same *item* by the same rater are declared as publicly linked.
- Every secret rating key $rrsk[i]$ created by the provider can be publicly verified by

$$\mathrm{e}(A_{i_{item}}, g_2)^{x_{i_{item}}} \cdot \mathrm{e}(A_{i_{item}}, W) \cdot \mathrm{e}(h, g_2)^{-y_i} = \mathrm{e}(g_1, g_2).$$

The security of the provided construction is shown by proving the following lemmas. Their formal proofs are given in Chapter 8.

Lemma 4.5:
If the Decision Linear Assumption holds for bilinear group generator BiGrGen, the reputation system defined in Section 4.2.2 is CPA-anonymous. ◇

The basic idea to prove anonymity is to guess the two identities the adversary will choose in the anonymity experiment, such that an instance of the Decision Linear Problem can be incorporated into the adversary's challenge. This is similar to the proofs of CPA-security for public-key encryption schemes, hence the name CPA-anonymity.

Lemma 4.6:
If the Strong Diffie-Hellman Assumption holds for bilinear group generator BiGrGen, the reputation system defined in Section 4.2.2 is publicly linkable. ◇

To prove this lemma we adapt a technique of Boneh and Boyen [BB04]. This technique transforms an instance of the Strong Diffie-Hellman Problem $\left(G_2^{\hat{\gamma}}, G_2^{(\hat{\gamma}^2)}, \ldots, G_2^{(\hat{\gamma}^{q(\lambda)})}\right)$ into values (g_1, g_2, W) and $\{(A_i, x_i)\}_{i=1}^{q(\lambda)-1}$, such that $g_1 = \phi(g_2), W = g_2^{\gamma}$, and for each $i \in \{1, \ldots, q(\lambda) - 1\}$ it holds $\mathrm{e}(A_i, g_2)^{x_i} \cdot \mathrm{e}(A_i, W) = \mathrm{e}(g_1, g_2)$, while γ and $\hat{\gamma}$ are unknown.

If an adversary is able to generate a new pair (A, x), it can be used to compute a solution to the given SDH instance. Our adaption of this technique will generate similar values (g_1, g_2, h, W) and $\{(A_i, x_i, y_i)\}_{i=1}^{q(\lambda)-1}$, such that (g_1, g_2, W) is used as an *item*-based public key, y_i is used as the secret rating key of rater i with corresponding public key h^{y_i}, and each (A_i, x_i) is used as the rating key of rater i for the *item* defined by (g_1, g_2, W).

To obtain a new triple (A, x, y) from an adversary $\mathcal{A}$ against the public linkability of the reputation system we apply the General Forking Lemma (Lemma 2.2). For this purpose we define an algorithm $\mathcal{B}$, that interacts with $\mathcal{A}$, such that the forking algorithm $\mathbf{F}_{\mathcal{B}}$ from the General Forking Lemma obtains two related accepting transcripts of the signature of knowledge used as a rating. Then, the extractor defined in Lemma 4.4 is invoked to compute a tuple (A, x, y). A similar technique is used by Boneh, Boyen, and Shacham [BBS04] to prove the security of a group signature scheme, but they apply the Forking Lemma of Pointcheval and Stern [PS00].

Lemma 4.7:
If the Strong Diffie-Hellman Assumption holds for bilinear group generator BiGrGen, the reputation system defined in Section 4.2.2 is traceable. ◇

The proof of traceability is similar to that of linkability, because the attack scenario is the same. The proofs only differ in some technical details, but are based on the same idea.

Lemma 4.8:
If the Discrete Logarithm Assumption holds for bilinear group generator BiGrGen, the reputation system defined in Section 4.2.2 is strongly exculpable. ◇

Similar to the proof of anonymity, we guess the identity the adversary will attack. This allows us to use an instance of the Discrete Logarithm Problem as the rater's public key. When we guessed the identity correctly, we apply the General Forking Lemma to obtain a second rating from the same rater. These two related ratings can then be used to solve the Discrete Logarithm Problem.

5 Universal Composability

As seen in the previous chapters, the security of cryptographic systems is often defined by considering the execution of a cryptographic task in isolation. On the one hand, this approach allows to construct efficient schemes achieving exactly the demanded properties. On the other hand, there are no security guarantees when running different schemes concurrently, as it is done in practical applications. Hence, to define security for more complex systems the execution environment must be considered. Furthermore, when different schemes are composed to realize more complex systems, the security of the composition is not implied by the security of the components. A general framework that combines the isolated view on a cryptographic task, the consideration of the execution environment, and allows for secure composition of different components is the *Universal Composability Framework*, introduced by Canetti [Can01].

In the Universal Composability Framework a cryptographic task is described as an *ideal functionality* $\mathcal{F}$ that acts as a trusted and incorruptable party. Every party involved in the cryptographic task hands its inputs securely to $\mathcal{F}$, which in turn computes the outcome and hands each party its prescribed output. To model protocols running concurrently to $\mathcal{F}$ an *environment* $\mathcal{Z}$ chooses the inputs of the parties and collects their outputs. For a practical application the ideal functionality $\mathcal{F}$ is replaced by a cryptographic protocol Π in which the parties communicate with each other to compute their outputs. The security of such a system is defined by considering $\mathcal{Z}$ as an interactive distinguisher that has to decide whether it interacts with the *ideal process*, the execution of the cryptographic task using $\mathcal{F}$, or the real-world protocol Π. Since only choosing inputs and receiving outputs does not appropriately model adversarial behavior, $\mathcal{Z}$ can communicate with a real-world adversary $\mathcal{A}$ that attacks Π. This adversary can corrupt parties, controls the messages they exchange, and tries to influence the protocol execution such that the environment $\mathcal{Z}$ obtains additional information that might help to distinguish $\mathcal{F}$ and Π. But in the ideal process the parties do not communicate and have no internal states. Hence, it would be easy to distinguish between Π and $\mathcal{F}$. Therefore, an *ideal process simulator* $\mathcal{S}$ is introduced that communicates with $\mathcal{F}$ and emulates a protocol execution of Π in the ideal process. If this emulation is indistinguishable from a real protocol execution, we say Π UC-realizes $\mathcal{F}$, which means that an interaction with Π and $\mathcal{A}$ leaks as much information as interacting with $\mathcal{F}$ and $\mathcal{S}$. By this security definition it is possible to prove a general *composition theorem*, which basically states that the composition of secure protocols is also secure.

5.1 Protocol Execution and Security

Similar to the work of Goldreich, Micali, and Wigderson [GMW87], security in the Universal Composability Framework is defined by comparing a real execution of a cryptographic protocol with an ideal process.

Every cryptographic protocol is executed with respect to a given environment $\mathcal{Z}$, which models all concurrently running protocols, including those protocols providing inputs to or obtaining outputs from the protocol under consideration. That means, the environment $\mathcal{Z}$ chooses the inputs for every party π_i executing a protocol Π and obtains its outputs. Furthermore, the environment can freely interact with an adversary $\mathcal{A}$ that attacks the protocol Π by controlling the communication between the parties π_i. In particular, $\mathcal{A}$ can interact with every party π_i, but is not able to access the inputs and outputs of π_i. The execution of a protocol Π with an adversary $\mathcal{A}$ and an environment $\mathcal{Z}$ is formally described by Canetti [Can01], modeling all parties as a special kind of interactive Turing machines. We provide a simplified version:

- $\mathcal{Z}$ only hands inputs to other parties. The first party $\mathcal{Z}$ hands inputs to is the adversary $\mathcal{A}$. All other parties $\mathcal{Z}$ hands inputs to are parties π_i executing protocol Π.
- The adversary $\mathcal{A}$ can send messages to every other party in the system, namely to $\mathcal{Z}$ and all parties π_i currently executing Π.
- All parties π_i can send messages to $\mathcal{A}$. When party π_i wants to send a message m to party π_j, it sends (i, j, m) to $\mathcal{A}$, because $\mathcal{A}$ controls the communication. $\mathcal{A}$ is not forced to deliver m to π_j.
- Every party π_i and the adversary $\mathcal{A}$ hand their respective outputs to $\mathcal{Z}$.

A special message $\mathcal{A}$ can send to some party π_i is a *corruption message*. When π_i receives such a message, it sends its internal state to $\mathcal{A}$ and becomes inactive. The internal state contains all inputs, all random bits used, and all messages the party exchanged with others. If the code of π_i is *erasing*, it depends on the protocol which information is sent to $\mathcal{A}$. Typically, in such cases at least the random bits are erased directly after their usage and hence are not available to $\mathcal{A}$ after corruption. In contrast to this *adaptive corruption*, a special case of corruption is that of *non-adaptive corruption*, where an adversary $\mathcal{A}$ has to declare which parties are corrupted *before* the protocol execution starts. Corruption messages during protocol execution are ignored in that case. Independent of the corruption model, a corrupted party is fully controlled by $\mathcal{A}$ and the term „corrupted party π_i“ is used synonymously for „adversary $\mathcal{A}$“. We note that other corruption models are realizable in the Universal Composability Framework, but those are not of interest in this work.

In the execution of an ideal process, also called *ideal protocol*, the parties π_i do not communicate to compute their outputs, but hand their inputs directly and reliably to an ideal functionality $\mathcal{F}$ which in turn securely computes the outputs for every party. When a party π_i obtains an output from $\mathcal{F}$ it is immediately handed to $\mathcal{Z}$. By that mechanism the parties π_i are *dummy parties* as they do not execute any code. By that definition a real cryptographic protocol Π is replaced in the ideal protocol by the ideal functionality $\mathcal{F}$. Analogously, the adversary $\mathcal{A}$ attacking Π is replaced by an *ideal adversary* $\mathcal{S}$, also called a *simulator*. Since corrupting the dummy parties in the ideal protocol and observing their

communication as in the real protocol execution does not make any sense, the simulator $\mathcal{S}$ interacts directly with $\mathcal{F}$. The ideal functionality $\mathcal{F}$ can use this interaction to provide $\mathcal{S}$ with additional information an adversary in a real protocol would obtain anyway. This mechanism models the leakage of information during protocol execution. In the reverse direction $\mathcal{F}$ receives the corruption messages from $\mathcal{S}$ and can react accordingly. That means, when $\mathcal{S}$ wants to corrupt a dummy party π_i, $\mathcal{F}$ marks π_i as corrupted, hands all inputs and outputs of π_i generated so far to $\mathcal{S}$, and forwards all future inputs and outputs to $\mathcal{S}$. For non-adaptive corruptions this implies that $\mathcal{F}$ ignores all corruption messages that were sent by $\mathcal{S}$ *after* the start of the protocol execution.

With the description of protocol execution of real and ideal protocols, we now turn to the definition of security in the Universal Composability Framework. The basic idea is to define security by stating that no environment can distinguish a real protocol execution from an ideal protocol execution. That means, the environment runs an interactive probabilistic process in which it provides inputs to different parties, with respect to some security parameter. At the end of this process the environment has to output its guess with which protocol it interacted.

As argued by Canetti [Can01] it is important to consider *balanced environments* in the definition of security. An environment is balanced when the overall length of all inputs given by the environment to the parties executing a protocol is at most λ times the length of the input given to the adversary, where λ is the security parameter. Without balanced environments it would be possible to execute protocols in a way such that an adversary is not able to deliver all messages sent, which is an unnatural situation.

Definition 5.1: Protocol Emulation - [Can01]
Let Π and Φ be protocols that are executable in probabilistic polynomial-time. We say that Π *UC-emulates* Φ, if for any probabilistic polynomial-time adversary $\mathcal{A}$ there exists a probabilistic polynomial-time algorithm $\mathcal{S}$ such that for any balanced probabilistic polynomial-time environment $\mathcal{Z}$ it holds

$$\left\{\mathrm{EXEC}_{\Phi,\mathcal{S},\mathcal{Z}}(\lambda,z)\right\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*} \stackrel{c}{\equiv} \left\{\mathrm{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}}(\lambda,z)\right\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*}$$

where $\mathrm{EXEC}_{\Phi,\mathcal{S},\mathcal{Z}}(\lambda,z)$ denotes the output of environment $\mathcal{Z}$ interacting with protocol Φ and adversary $\mathcal{S}$ on input the security parameter λ and auxiliary information z and $\mathrm{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}}(\lambda,z)$ denotes the output of environment $\mathcal{Z}$ interacting with protocol Π and adversary $\mathcal{A}$ on input the security parameter λ and auxiliary information z. △

In this definition the environment $\mathcal{Z}$ is treated as an *interactive distinguisher* and the algorithm $\mathcal{S}$ is a simulator that generates a view for $\mathcal{Z}$ such that the execution of Φ is indistinguishable from an execution of Π. So, to prove that one protocol emulates another it is necessary to provide an appropriate simulator $\mathcal{S}$. In the special case where Φ is an ideal protocol and Π UC-emulates Φ, we then say that Π UC-realizes Φ.

Definition 5.2: Realizing Ideal Functionalities - [Can01]
Let $\mathcal{F}$ be an ideal functionality and let Π a protocol. We say Π *UC-realizes* $\mathcal{F}$, if Π UC-emulates the ideal protocol for $\mathcal{F}$. △

By that definition, an ideal functionality $\mathcal{F}$ defines both *correctness* and *security*. Every protocol Π realizing $\mathcal{F}$ is correct because the outputs in both protocol executions on the same inputs must be indistinguishable. Moreover, the information an adversary obtains by interacting with Π must also be obtainable by interacting with $\mathcal{F}$, which guarantees security. Based on that it is possible to *compose protocols* without losing their correctness and security properties. To illustrate what a composed protocol is, consider three protocols Π, Φ, ρ, where ρ uses Φ as a subroutine and Π UC-emulates Φ. Then the composed protocol $\rho^{\Phi \to \Pi}$ is identical to ρ, but whenever a party would run Φ in the original protocol, it instead runs Π. Due to the UC-emulation of Φ, the protocol Π can be used as an alternative. This is formalized as the Universal Composition Theorem.

Theorem 5.1: Universal Composition - [Can01]
Let Π, Φ, and ρ be protocols that are executable in probabilistic polynomial-time and let Π UC-emulate Φ. Then the protocol $\rho^{\Phi \to \Pi}$ UC-emulates ρ. $\diamond$

This theorem allows to define and analyze complex tasks in a modular way. Therefor, a task T is split into several modules $t_1, t_2, \ldots, t_n$. Then, each module can be realized by an appropriate protocol. After that, a protocol for task T can be defined, that uses the ideal functionalities of the modules $t_1, \ldots, t_n$. Finally, the task T can be realized by applying the composition theorem to replace the ideal protocols for $t_1, \ldots, t_n$ with their respecting realizations.

5.2 Technical Details

In this section we will highlight some technical details that are important to prove security in the Universal Composability Framework.

Identifying Parties and Functionalities

Protocols in the UC Framework are executed concurrently to other protocols, which is modeled by an environment $\mathcal{Z}$ providing inputs to and obtaining outputs from the protocol under consideration. This also includes the execution of multiple copies of a single protocol Π. These copies are called *instances of* Π, that are executed by several parties π. To allow communication between those parties, a unique party identifier *pid* is associated to them. Furthermore, it is important that parties only communicate with others executing the same protocol instance. For this reason, also every instance of a protocol has an session identifier, called *sid*. Hence, each party can be identified with the tuple (pid, sid). Whenever a party π with identifier (pid, sid) receives a message with a session identifier $sid' \neq sid$, this message will be ignored, because it does not belong to the protocol instance currently executed by π. Analogously, received messages with recipient $pid' \neq pid$ are ignored. For simplicity we assume that the *pid* and *sid* are provided by the environment $\mathcal{Z}$ during the first activation of a party. This also holds for the ideal functionality $\mathcal{F}$: the first activation of $\mathcal{F}$ contains a session identifier, which will be used by the instance of $\mathcal{F}$ as global identifier for the current protocol execution.

Delayed Output

In many situations handing outputs from an ideal functionality $\mathcal{F}$ immediately to its recipients is not desirable. Therefore, a functionality $\mathcal{F}$ can use the mechanism of *delayed output*, where $\mathcal{F}$ informs the adversary that some output is available and waits for the permission of the adversary to hand it to its recipient. This allows to model delays in message delivery, which is natural in distributed protocols. For *public delayed output* a functionality $\mathcal{F}$ sends a special message to the adversary, which contains the output v, its designated recipient π_i, and a unique query identifier. When the adversary responds to this message by sending the query identifier back to $\mathcal{F}$, the output v is handed to π_i. For *private delayed output* the mechanism is the same, but $\mathcal{F}$ does not include the output v in its message to the adversary. It is also possible to send an output v to a set of recipients π by informing the adversary about (v, π) and handing the output to a party $\pi_i \in \pi$, whenever the adversary responds with the query identifier and a party identifier. This is basically the same as informing the adversary about (v, π_i) for all $\pi_i \in \pi$ separately.

Message Transmission

In the most basic form a party π_i sends the message m to party π_j by handing (sid, i, j, m) to $\mathcal{A}$ and expecting $\mathcal{A}$ to deliver this tuple to π_j. But $\mathcal{A}$ is not forced to deliver this message, since $\mathcal{A}$ models an asynchronous, unauthenticated, and unreliable network. $\mathcal{A}$ is allowed to hand a modified tuple (sid, i, j, m') to π_j, or even to not deliver a message at all. This communication model is called the *bare model*. Based on that, ideal functionalities for authenticated and secure message transmission can be used to strengthen the security properties of the communication channels. With ideally authenticated communication channels an adversary is able to read the contents of a message, but is forced to deliver the message unmodified to the determined recipient. Secure communication channels additionally ensure that the adversary is unable to read the message sent.

Hybrid Protocols and Models

The Universal Composition Theorem states that the protocol $\rho^{\Phi \to \Pi}$ UC-emulates the protocol ρ, when Π, Φ, and ρ are protocols executable in probabilistic polynomial-time and Π UC-emulates Φ. Since there is no further restriction on Φ, it is possible that Φ is an ideal protocol. In this case the protocol ρ is said to be a *hybrid protocol*, as it combines a real protocol execution with invocations of ideal functionalities. The ideal functionalities used in a hybrid protocol ρ are known to every party and have unique session identifiers *sid* to ensure that only parties executing ρ can interact with them. A peculiarity of hybrid protocols is that the environment $\mathcal{Z}$ also knows that dedicated subroutines of a hybrid protocol ρ are ideal functionalities. Thus, hybrid protocols are said to be executed in a *hybrid model*. In a formal proof that a hybrid protocol ρ UC-emulates some ideal functionality $\mathcal{F}$, a simulator $\mathcal{S}$ has to generate a view for the environment $\mathcal{Z}$ that is indistinguishable from a protocol execution of ρ in the hybrid model. Since the ideal functionalities ρ uses as subroutines do not exist in the ideal protocol for $\mathcal{F}$, they have to be simulated by $\mathcal{S}$.

Defining Simulators

The definition of UC-emulation allows to construct individual simulators $\mathcal{S}$ for different adversaries $\mathcal{A}$. But Canetti [Can01] also provides alternative formulations of UC-emulation that are equivalent to Definition 5.1. The most important ones are UC-emulation with respect to the *dummy adversary* and with respect to *black-box simulation*. Both formulations simplify the construction of simulators.

The dummy adversary is an adversary that only delivers messages between the environment $\mathcal{Z}$ and the parties executing a protocol; there is no further interaction of $\mathcal{A}$ and other parties in the system. This means that the environment has full control over the communication, which also includes corruption messages. As shown by Canetti [Can01], a simulator for the dummy adversary can be transformed into a simulator for an arbitrary adversary. Hence, to prove that a protocol Π UC-emulates a protocol Φ, it is sufficient to construct a single simulator $\mathcal{S}$ considering the dummy adversary.

Another alternative formulation of UC-emulation is that of black-box simulation. For this kind of simulation a single simulator $\hat{\mathcal{S}}$ with black-box access to an adversary $\mathcal{A}$ is considered. To be able to apply the model of protocol execution as described in the previous section, the black-box simulator $\hat{\mathcal{S}}$ and the adversary $\mathcal{A}$ are encapsulated into a *shell simulator* $\mathcal{S}$, as follows:

- The shell simulator $\mathcal{S}$ is given $\hat{\mathcal{S}}$ and $\mathcal{A}$, and internally executes an instance of both algorithms.
- Whenever $\mathcal{S}$ receives input from the environment $\mathcal{Z}$, it forwards this input to $\mathcal{A}$. Every outgoing message from $\mathcal{A}$ is given as input to $\hat{\mathcal{S}}$. If $\hat{\mathcal{S}}$ wants to deliver messages to some party, this is carried out by $\mathcal{S}$.
- Whenever $\mathcal{S}$ receives an incoming message from some party, it forwards this message to $\hat{\mathcal{S}}$. Every output of $\hat{\mathcal{S}}$ is given to $\mathcal{A}$ as an incoming message. If $\mathcal{A}$ generates outputs, these are handed to the environment $\mathcal{Z}$ by $\mathcal{S}$.

Since this shell simulator is fixed for all black-box simulators and all adversaries, in formal proofs of UC-emulation it is often implicitly assumed that $\mathcal{S}$ exists and only the black-box simulator $\hat{\mathcal{S}}$ is defined. Albeit having black-box access to an adversary $\mathcal{A}$, the simulator $\hat{\mathcal{S}}$ *can not rewind* $\mathcal{A}$. This is due to the fact that $\mathcal{A}$ may exchange messages with the environment and rewinding $\mathcal{A}$ would cause exchanging these messages again. Since this does not happen in the real protocol execution, an environment can easily distinguish between the ideal and real protocol executions. Hence, rewinding $\mathcal{A}$ is not possible.

Reputation Systems in the Universal Composability Framework

6

This chapter is based on our paper „Practical, Anonymous, and Publicly Linkable Universally-Composable Reputation Systems“, presented at the Cryptographers' Track of the RSA Conference 2018 [BEJ18].

The model presented in Chapter 4 has several drawbacks. The most important issue of the model is that raters and providers are disjoint sets, which makes this model unusable in many applications. But also traceability and linkability are considered separately, which does not appropriately model the intuition of linkability, because the provided security definitions do not imply that linkable ratings can be opened to the same identity. This problem is pointed out by Kaafarani and Katsumata [KK18], albeit the construction given in Chapter 4 ensures this property. However, such issues highlight that experiment-based security definitions might not cover all subtle security properties.

To resolve the disadvantages of the reputation system presented in Chapter 4, we provide a new model and a corresponding realization of a reputation system in this chapter. The new model enables users to rate each other, meaning that raters and providers are treated as users with different roles. Furthermore, and most importantly, we formalize this model in the Universal Composability Framework to ensure that realizations are composable with arbitrary applications. Since models in this framework describe an ideal functionality, also non-obvious security properties are covered.

6.1 An Ideal Functionality for Reputation Systems

In the first part of this section we give some intuition to our ideal functionality of a reputation system $\mathcal{F}_{\mathsf{RS}}$. The second part concerns the formal definition of $\mathcal{F}_{\mathsf{RS}}$ in the Universal Composability Framework [Can01]. We discuss the functionality and its security properties in the third part of the section.

6.1.1 Intuition

Similar to the model presented in Chapter 4, our new model focuses on the process of secure rating generation, meaning that the aggregation of ratings and evaluating an application specific reputation function is not considered to obtain a system that can be combined with arbitrary applications. We consider reputation systems where users within the system can

rate each others products. The term *product* refers to anything that can be used as a basis for ratings, for example traded goods or services. Each user in our system has to *register* once at a *System Manager*, before a product can be rated. This gives the System Manager the ability to punish misbehaving users. Therefor the system must prevent users to register with different identities. When users do not want to rate products at all, a registration is not necessary - publishing products and verifying ratings is independent of the registration, which increases trust in the system. Analogously to registering, a product must be *purchased* before it can be rated. To further increase trust in the reputation system, raters must be able to rate purchased products anonymously. Without anonymity raters may tend to rate dishonestly when they fear negative consequences from the provider. At the same time a provider must be protected against unjustified negative ratings. This is achieved by giving the System Manager the ability to revoke the anonymity of a rater. Of course, the System Manager must not be able to accuse an honest user having misbehaved. The negative side-effects of anonymity are that self-ratings, meaning ratings for a product from the provider, are hard to prevent and that a single rater, who purchased a product, could rate this product multiple times. Therefore, we require a reputation system to explicitly forbid self-ratings and to provide *linkable ratings*, where everybody, even outsiders of the system, must be able to detect multiple ratings from the same user for the same product.

In the Universal Composability Framework every activation of a party corresponds to a specific task the party has to execute. Hence, to define a model for reputation systems in that framework, we have to provide multiple activations that build a reputation system. To give some intuition to these activations we briefly describe them before heading to the formal definition of $\mathcal{F}_{\mathsf{RS}}$.

The KeyGen activation is used by the System Manager P_{SM} to generate and publish public parameters pp. These parameters enable the System Manager to *open* ratings to obtain the identity of the rating's author.

The Register activation is an interactive protocol, initiated by a user P_i to register with the System Manager P_{SM}. Being registered is a prerequisite to generate ratings.

With the NewProduct activation a user P_i can generate public information ppk that combines a product identifier $prod$ an the user's identity. The value ppk is used to assign ratings to a dedicated product from a specific user. Hence, two values ppk_1 and ppk_2 must be different, when they are generated by different users, even when the used product $prod$ is the same.

A user P_i executes the interactive Purchase protocol with another user P_j to purchase a product $prod$ with associated public information ppk from P_j. Besides user registration, this is the second prerequisite to rate the purchased product. It is important that P_i and P_j are different parties to ensure that users cannot rate their own products.

The Rate activation is used by a user P_i, acting as a rater, to anonymously generate and publish a rating σ with respect to the rating-message m assigned to a dedicated product. As mentioned above, the user P_i must be registered and must have purchased the specified product beforehand.

With the Verify activation every user can verify whether or not a given rating σ for a rating-message m and a specified product is *valid*. A rating is valid, when the anonymous author of the rating is registered and purchased the specified product.

By executing the Link activation with two ratings for the same product any user can determine whether or not both ratings were generated by the same user. In any case the identity of the author(s) is kept secret.

To determine the author of a rating the System Manager P_{SM} can use the Open activation. When he also wants to provide publicly verifiable information that the claimed user is really the author of a rating, he can generate an *opening-proof* τ with the OProof activation. Opening-proofs can be verified with the Judge activation by any user.

Besides these activations, the formal definition of $\mathcal{F}_{\mathsf{RS}}$ contains the *internal* activations VfyProd, VfyRtg, LinkRtgs, and RebLDB. They are only introduced to provide reusable descriptions of often executed computations. VfyProd is used to verify that a party P_j and a product *prod* are properly combined in *ppk*. VfyRtg comprises all steps to verify a rating, whereas LinkRtgs comprises all steps to link ratings. With RebLDB the consistency of the linking information is guaranteed. All these internal activations are described in more detail in the following two sections.

6.1.2 The Formal Definition

Our ideal functionality for reputation systems $\mathcal{F}_{\mathsf{RS}}$ is influenced by the ideal functionalities for digital signatures $\mathcal{F}_{\mathsf{SIG}}$ [Can04], public-key encryption $\mathcal{F}_{\mathsf{PKE}}$ [Can01] and group signatures [Ate+05]. It interacts with a simulator $\mathcal{S}$ and the parties $P_{\mathrm{SM}}, P_1, P_2, \ldots, P_n$. The party P_{SM} acts as the System Manager, whereas the parties P_i correspond to the users within the reputation system. Furthermore, $\mathcal{F}_{\mathsf{RS}}$ manages the lists $\mathfrak{Params}, \mathfrak{Reg}, \mathfrak{Prods}, \mathfrak{Purch}, \mathfrak{Ratings}$, and $\mathfrak{Open}$ to store important information. Before giving the formal definition of $\mathcal{F}_{\mathsf{RS}}$, we explain how these lists are used. We also introduce the notation needed in the definition of $\mathcal{F}_{\mathsf{RS}}$.

$\mathfrak{Params}$: This list stores all pairs of the form (P_{SM}, pp) containing *public parameters* the simulator $\mathcal{S}$ gives to $\mathcal{F}_{\mathsf{RS}}$ during KeyGen-requests. The first component of a pair is fixed to P_{SM}, whereas the second component represents the actual parameters given by $\mathcal{S}$.

$\mathfrak{Reg}$: The list $\mathfrak{Reg}$ stores pairs of the form (pp, P_i) containing *registration information*. The first component stores the public parameters the registrated party used in the Register-protocol, whereas the second component is the registrated party.

$\mathfrak{Prods}$: All *products* that are used within the reputation system are stored as 4-tuples of the form $(P_i, prod, ppk, b)$ in the list $\mathfrak{Prods}$. The first component of a tuple declares the product owner, the second is a product identifier (a bitstring chosen by the environment), the third specifies the corresponding *product-public key* and the fourth component is a validity bit. There can exist different products with the same product identifier, but only for different product owners. The validity bit indicates whether the product-public key matches the given product owner and the product identifier.

$\mathfrak{Purch}$: When some party successfully purchased a product, this information is stored as 4-tuple $(P_i, P_j, prod, ppk)$ in the list $\mathfrak{Purch}$. For every tuple in the list the first component represents the purchaser, whereas the other components determine the product that was purchased (the product owner, the product identifier and the product-public key).

$\mathfrak{Ratings}$: The list $\mathfrak{Ratings}$ stores the most complex information as 10-tuples of the form $(pp, P_i, P_j, prod, ppk, m, \sigma, b, lid, oid)$. The components of each tuple represent the following information:

1. pp - the public parameters a rating is generated for,
2. P_i - the identity of the rater ((pp, P_i) should match an entry in $\mathfrak{Reg}$),
3. P_j - the product owner of the product the rating is generated for,
4. $prod$ - the product identifier of the product the rating is generated for,
5. ppk - the product-public key of the product the rating is generated for (the tuple $(P_i, P_j, prod, ppk)$ should match an entry in $\mathfrak{Purch}$),
6. m - rating message (a placeholder for high-level applications),
7. σ - the rating,
8. b - the validity bit (indicating whether the rating is *valid*),
9. lid - the *linking-class identifier*, which is managed by the algorithm RebLDB,
10. oid - the *opening-proof identifier*.

The linking-class identifier is needed to model the linkability property: two ratings with the same linking-class identifier have the same author. The opening-class identifier binds a list of opening-proofs to a specific rating. Whenever a new rating is added to the list $\mathfrak{Ratings}$, $\mathcal{F}_{\mathsf{RS}}$ uses the current value of a global counter *LIDC* as the linking-class identifier and increments the counter. The subsequent execution of RebLDB ensures that the rating is put into the correct linking-class, according to the linkability-relation. A more detailed explanation of this behavior and the *oid*-mechanism is given in the discussion of the security properties of $\mathcal{F}_{\mathsf{RS}}$.

$\mathfrak{Open}$: This list stores all *opening-proofs* as 4-tuples of the form (oid, τ, b, P). The first component is an opening-proof identifier that binds a tuple to a specific rating with the same identifier. The second component is the actual opening-proof. The third component is a validity bit indicating whether the proof is *valid* and the fourth component is the claimed party that shall be the author of the associated rating.

The value $oid = \perp$ within a rating expresses that the rating was not opened yet and hence no opening-proof exists. To uniquely bind opening-proofs to ratings a global counter *OIDC* is used and incremented whenever a new opening-proof is bound to an unopened rating.

To manipulate the described lists, we introduce two operations:

- adding a tuple v to a list L is expressed by $L.\mathsf{Add}(v)$, and
- replacing a tuple v_{old} with a tuple v_{new} is expressed by $L.\mathsf{Rep}(v_{\text{old}}, v_{\text{new}})$.

Replacing a tuple v_{old} means that this tuple is removed from the list, while the tuple v_{new} is added to the list.

The classical notation to address components of tuples is using indices, i.e. $v = (v_1, \ldots, v_n)$, where v_i is the i'th component of tuple v. We deviate from this notation to prevent confusion with different variables and address the i'th component of a tuple v by $v[i]$.

Whenever the functionality $\mathcal{F}_{\mathsf{RS}}$ misses some information, the symbol $\perp$ is used to highlight this fact. Also the simulator $\mathcal{S}$ can output this symbol to indicate that it is not able to respond to a request. Depending on the situation, this is not necessarily a failure.

With these prerequisites we now give the formal definition of $\mathcal{F}_{\mathsf{RS}}$.

$\mathcal{F}_{\mathsf{RS}}$

$\mathcal{F}_{\mathsf{RS}}$ interacts with parties $P_{\text{SM}}, P_1, \ldots, P_n$, and the ideal adversary (simulator) $\mathcal{S}$. Further it manages the lists $\mathfrak{Params}, \mathfrak{Reg}, \mathfrak{Prods}, \mathfrak{Purch}, \mathfrak{Ratings}$, and $\mathfrak{Open}$, which are initially empty, and the counters $LIDC$, $OIDC$, which are initialized with 0. All outputs from $\mathcal{F}_{\mathsf{RS}}$ to some party P are public delayed outputs.

Registry Key Generation: On input (KeyGen, sid) from P_{SM}

1: Check that $sid = (P_{\text{SM}}, sid')$ for some sid'. If not, ignore the request.
2: Send (KeyGen, sid) to $\mathcal{S}$ and receive $(\mathsf{KeyGen}, sid, pp)$ from $\mathcal{S}$.
3: Set $\mathfrak{Params}.\mathsf{Add}(P_{\text{SM}}, pp)$ and send $(\mathsf{KeyGen}, sid, pp)$ to P_{SM}.

User Registration: On input $(\mathsf{Register}, sid, pp')$ from P_i

1: Check that $sid = (P_{\text{SM}}, sid')$ for some sid'. If not, ignore the request.
2: Send $(\mathsf{Register}, sid, pp', P_i)$ to $\mathcal{S}$ and receive $(\mathsf{Register}, sid, pp', P_i, b)$ from $\mathcal{S}$.
3: **If** P_{SM} and P_i are honest $\wedge\, (P_{\text{SM}}, pp') \in \mathfrak{Params} \wedge (P_i, pp') \notin \mathfrak{Reg}$ **Then** $f := 1$.
4: **Else If** P_{SM} is honest $\wedge\, (P_{\text{SM}}, pp') \notin \mathfrak{Params}$ **Then** $f := 0$.
5: **Else** $f := b$.
6: **If** $f = 1$ **Then** $\mathfrak{Reg}.\mathsf{Add}(pp', P_i)$.
7: Send $(\mathsf{Register}, sid, pp', P_i, f)$ to P_i and P_{SM}.

Product Addition: On input $(\mathsf{NewProduct}, sid, prod)$ from P_i

1: Check that $sid = (P_{\text{SM}}, sid')$ for some sid'. If not, ignore the request.
2: Send $(\mathsf{NewProduct}, sid, P_i, prod)$ to $\mathcal{S}$ and receive $(\mathsf{NewProduct}, sid, P_i, prod, ppk)$ from $\mathcal{S}$.
3: **If** $(P', prod', ppk, 1) \in \mathfrak{Prods}$, where $(P', prod') \neq (P_i, prod)$ **Then**
output `error` and halt.
4: **Else** $\mathfrak{Prods}.\mathsf{Add}(P_i, prod, ppk, 1)$ and send $(\mathsf{NewProduct}, sid, prod, ppk)$ to P_i.

Purchase: On input $(\mathsf{Purchase}, sid, P_j, prod, ppk)$ from P_i

1: Check that $sid = (P_{\mathrm{SM}}, sid')$ for some sid'. If not, ignore the request.
2: **If** $P_i = P_j \vee \mathsf{VfyProd}(sid, P_j, prod, ppk) = 0$ **Then** ignore the request.
3: Send $(\mathsf{Purchase}, sid, P_i, P_j, prod, ppk)$ to $\mathcal{S}$ and receive $(\mathsf{Purchase}, sid, P_i, P_j, prod, ppk, b)$ from $\mathcal{S}$.
4: **If** P_i and P_j are honest **Then** $f \coloneqq 1$.
5: **Else** $f \coloneqq b$.
6: **If** $f = 1$ **Then** $\mathfrak{Purch}.\mathsf{Add}(P_i, P_j, prod, ppk)$.
7: Send $(\mathsf{Purchase}, sid, P_i, P_j, prod, ppk, f)$ to P_i and P_j.

VfyProd: On internal input $(sid, P_j, prod, ppk)$

1: Send $(\mathsf{VfyProd}, sid, P_j, prod, ppk)$ to $\mathcal{S}$ and receive $(\mathsf{VfyProd}, sid, P_j, prod, ppk, b)$ from $\mathcal{S}$.
2: **If** $(P_j, prod, ppk, f') \in \mathfrak{Prods}$ **Then** $f \coloneqq f'$.
3: **Else If** $b = 1 \wedge P_j$ is honest **Then** output `error` and halt.
4: **Else If** $(P', prod', ppk, 1) \in \mathfrak{Prods}$, where $(P', prod') \neq (P_i, prod)$ **Then**
Set $\mathfrak{Prods}.\mathsf{Add}(P_i, prod, ppk, 0)$ and $f \coloneqq 0$.
5: **Else** Set $\mathfrak{Prods}.\mathsf{Add}(P_i, prod, ppk, b)$ and $f \coloneqq b$.
6: Return f.

Rate a Product: On input $(\mathsf{Rate}, sid, pp, P_j, prod, ppk, m)$ from P_i

1: Check that $sid = (P_{\mathrm{SM}}, sid')$ for some sid'. If not, ignore the request.
2: **If** $(pp, P_i) \notin \mathfrak{Reg}$
$\vee\, (P_i, P_j, prod, ppk) \notin \mathfrak{Purch}$
$\vee\, (pp, P_i, P_j, prod, ppk, m', \sigma', 1, lid, oid) \in \mathfrak{Ratings}$ for some m', σ', lid **Then** ignore the request.
3: **If** P_{SM} is honest **Then**
Send $(\mathsf{Rate}, sid, pp, P_j, prod, ppk, m)$ to $\mathcal{S}$ and receive $(\mathsf{Rate}, sid, pp, P_j, prod, ppk, m, \sigma)$ from $\mathcal{S}$.
4: **Else** Send $(\mathsf{Rate}, sid, pp, P_i, P_j, prod, ppk, m)$ to $\mathcal{S}$ and receive $(\mathsf{Rate}, sid, pp, P_i, P_j, prod, ppk, m, \sigma)$ from $\mathcal{S}$.
5: **If** $(pp, P', P_j, prod, ppk, m, \sigma, 0, lid, oid) \in \mathfrak{Ratings}$ for some P', lid, oid **Then**
Output `error` and halt.
6: Set $r \coloneqq (pp, P_i, P_j, prod, ppk, m, \sigma, 1, LIDC, \perp)$ and $LIDC \coloneqq LIDC + 1$.
7: Set $\mathfrak{Ratings}.\mathsf{Add}(r)$ and run $\mathsf{RebLDB}(sid, r, \perp)$.
8: Send $(\mathsf{Rate}, sid, pp, P_j, prod, ppk, m, \sigma)$ to P_i.

Verifying a Rating: On input $(\mathsf{Verify}, sid, pp, P_j, prod, ppk, m, \sigma)$ from P_i (or P_{SM})

1: Check that $sid = (P_{\mathrm{SM}}, sid')$ for some sid'. If not, ignore the request.
2: Set $(X, f, oid) \coloneqq \mathsf{VfyRtg}(sid, pp, P_j, prod, ppk, m, \sigma)$.
3: Send $(\mathsf{Verify}, sid, pp, P_j, prod, ppk, m, \sigma, f)$ to P_i (or P_{SM}).

VfyRtg: On internal input $(sid, pp, P_j, prod, ppk, m, \sigma)$

1: **If** $\mathsf{VfyProd}(sid, P_j, prod, ppk) = 0$ **Then** ignore the request.
2: Send $(\mathsf{Verify}, sid, pp, P_j, prod, ppk, m, \sigma)$ to $\mathcal{S}$ and receive $(\mathsf{Verify}, sid, pp, P_j, prod, ppk, m, \sigma, b, P)$ from $\mathcal{S}$.
3: **If** $(pp, X', P_j, prod, ppk, m, \sigma, f', lid', oid') \in \mathfrak{Ratings}$ for some X', f', lid' and oid' **Then** $X := X'$, $f := f'$, $oid := oid'$.
4: **Else If** $b = 0 \vee P = P_j$ **Then**
Set $\mathfrak{Ratings}.\mathsf{Add}(pp, \perp, P_j, prod, ppk, m, \sigma, 0, \perp, \perp)$, $X{:=}\perp$, $f{:=}0$, and $oid{:=}\perp$.
5: **Else If** P_j is honest $\wedge\ P \neq \perp \wedge (P, P_j, prod, ppk) \notin \mathfrak{Purch}$ **Then**
Output `error` and halt.
6: **Else If** $P \neq \perp \wedge P$ is honest **Then** output `error` and halt.
7: **Else If** $P = \perp \wedge P_{\mathrm{SM}}$ is honest **Then** output `error` and halt.
8: **Else** Set $r := (pp, P, P_j, prod, ppk, m, \sigma, 1, LIDC, \perp)$, $X := P$, $f := 1$, and $oid := \perp$.
Set $LIDC := LIDC + 1$, $\mathfrak{Ratings}.\mathsf{Add}(r)$, and run $\mathsf{RebLDB}(sid, r, \perp)$.
9: Return (X, f, oid).

Linking Ratings: On input $(\mathsf{Link}, sid, pp, P_j, prod, ppk, m_0, \sigma_0, m_1, \sigma_1)$ from P_i (or P_{SM})

1: Check that $sid = (P_{\mathrm{SM}}, sid')$ for some sid'. If not, ignore the request.
2: Set $b := \mathsf{LinkRtgs}(sid, pp, P_j, prod, ppk, m_0, \sigma_0, m_1, \sigma_1)$.
3: Send $(\mathsf{Link}, sid, pp, P_j, prod, ppk, m_0, \sigma_0, m_1, \sigma_1, b)$ to P_i (or P_{SM}).

LinkRtgs: On internal input $(sid, pp, P_j, prod, ppk, m_0, \sigma_0, m_1, \sigma_1)$

1: Set $(X_k, f_k, oid_k) := \mathsf{VfyRtg}(sid, pp, P_j, prod, ppk, m_k, \sigma_k)$ for $k \in \{0, 1\}$.
2: Send $(\mathsf{Link}, sid, pp, P_j, prod, ppk, m_0, \sigma_0, m_1, \sigma_1)$ to $\mathcal{S}$, receive $(\mathsf{Link}, sid, pp, P_j, prod, ppk, m_0, \sigma_0, m_1, \sigma_1, b)$ from $\mathcal{S}$, and set $f := 0$.
3: **If** $f_0 = f_1 = 1$ **Then**
4: Let $r_k \in \mathfrak{Ratings}$ be $r_k := (pp, X_k, P_j, prod, ppk, m_k, \sigma_k, 1, lid_k, oid_k)$, for $k \in \{0, 1\}$.
5: **If** $lid_0 = lid_1$ **Then** $f := 1$.
6: **Else If** $X_0 = X_1 \wedge X_0 = \perp \wedge X_1 = \perp$ **Then** $f := b$.
7: **Else If** $X_0 \neq X_1 \wedge X_0 \neq \perp \wedge X_1 \neq \perp$ **Then** $f := 0$.
8: **Else If** $(X_k = \perp \wedge X_{1-k} \neq \perp \wedge X_{1-k}$ is honest) for $k = 0 \vee k = 1$ **Then** $f := 0$.
9: **Else If** $(X_k = \perp \wedge X_{1-k} \neq \perp \wedge X_{1-k}$ is corrupted) for $k = 0 \vee k = 1$ **Then** $f := b$.
10: **If** $f = 1$ **Then** run $\mathsf{RebLDB}(sid, r_0, r_1)$.
11: Return f.

RebLDB: On internal input (sid, r, s)

1: Parse r as $(pp, X_0, P_j, prod, ppk, m_0, \sigma_0, 1, lid_0, oid_0)$.
2: **If** $s = \perp \wedge X_1 \neq \perp$ **Then**
3: Set $\mathcal{L} := \{\ell | \ell \in \mathfrak{Ratings} \wedge \ell[1] = pp \wedge \ell[2] = X_0 \wedge \ell[3] = P_j \wedge \ell[4] = prod \wedge \ell[5] = ppk \wedge \ell[8] = 1\}$ and $lid := \min\{\ell[9] | \ell \in \mathcal{L}\}$.
4: **For Each** $\ell \in \mathcal{L}$ **do**
Set $\ell' := \ell$, $\ell'[9] := lid$, and $\mathfrak{Ratings}.\mathsf{Rep}(\ell, \ell')$.

5: **If** $s \neq \perp$ **Then**
6: Parse s as $(pp, X_1, P_j, prod, ppk, m_1, \sigma_1, 1, lid_1, oid_1)$
7: **If** $X_0 = \perp \wedge X_1 \neq \perp$ **Then** Set $X := X_1$.
8: **Else** Set $X := X_0$.
9: Set $\mathcal{L} := \{\ell | \ell \in \mathfrak{Ratings} \wedge \ell[1] = pp \wedge \ell[3] = P_j \wedge \ell[4] = prod \wedge \ell[5] = ppk \wedge \ell[8] = 1 \ \wedge (\ell[9] = lid_0 \vee \ell[9] = lid_1)\}$ and $lid := \min\{lid_0, lid_1\}$.
10: **For Each** $\ell \in \mathcal{L}$ **do** Set $\ell' := \ell$, $\ell'[2] := X$, $\ell'[9] := lid$, and $\mathfrak{Ratings}.\mathsf{Rep}(\ell, \ell')$.
11: Set $\mathcal{P} := \{p | p \in \mathfrak{Purch} \wedge p[2] = P_j \wedge p[3] = prod \wedge p[4] = ppk\}$.
12: Set $\mathcal{L} := \{\ell | \ell \in \mathfrak{Ratings} \wedge \ell[1] = pp \wedge \ell[3] = P_j \wedge \ell[4] = prod \wedge \ell[5] = ppk \wedge \ell[8] = 1\}$.
13: **If** P_{SM} is corrupted, P_j is honest and $|\mathcal{P}| < |\{\ell[9] | \ell \in \mathcal{L}\}|$ **Then**
14: **For Each** $(\ell, \ell') \in \mathcal{L}^2$ **do**
15: Run $\mathsf{LinkRtgs}(sid, \ell[1], \ell[3], \ell[4], \ell[5], \ell[6], \ell[7], \ell'[6], \ell'[7])$ ignoring the output of $\mathsf{LinkRtgs}$.
16: Set $\mathcal{P} := \{p | p \in \mathfrak{Purch} \wedge p[2] = P_j \wedge p[3] = prod \wedge p[4] = ppk\}$.
17: Set $\mathcal{L} := \{\ell | \ell \in \mathfrak{Ratings} \wedge \ell[1] = pp \wedge \ell[3] = P_j \wedge \ell[4] = prod \wedge \ell[5] = ppk \wedge \ell[8] = 1\}$.
18: **If** P_j is honest and $|\mathcal{P}| < |\{\ell[9] | \ell \in \mathcal{L}\}|$ **Then** Output `error` and halt.

Determine Raters Identity: On input $(\mathsf{Open}, sid, pp, P_j, prod, ppk, m, \sigma)$ from P_{SM}
1: Check that $sid = (P_{\text{SM}}, sid')$ for some sid'. If not, ignore the request.
2: **If** $(P_{\text{SM}}, pp) \notin \mathfrak{Params}$ **Then** ignore the request.
3: Set $(X, f, oid) := \mathsf{VfyRtg}(sid, pp, P_j, prod, ppk, m, \sigma)$.
4: **If** $f = 1$ **Then**
5: Let $r \in \mathfrak{Ratings}$ be $r := (pp, X, P_j, prod, ppk, m, \sigma, 1, lid', oid)$ for some lid'.
6: **If** $oid = \perp$ **Then** $r' := r$, $r'[10] := OIDC$, $\mathfrak{Ratings}.\mathsf{Rep}(r, r')$ and $OIDC := OIDC + 1$.
7: Send $(\mathsf{Open}, sid, pp, P_j, prod, ppk, m, \sigma, X)$ to P_{SM}.
8: **Else** Send $(\mathsf{Open}, sid, pp, P_j, prod, ppk, m, \sigma, \perp)$ to P_{SM}.

Generate Opening Proofs: On input $(\mathsf{OProof}, sid, pp, P_j, prod, ppk, m, \sigma, P)$ from P_{SM}
1: Check that $sid = (P_{\text{SM}}, sid')$ for some sid'. If not, ignore the request.
2: **If** $(P_{\text{SM}}, pp) \notin \mathfrak{Params}$ **Then** ignore the request.
3: Set $(X, f, oid) := \mathsf{VfyRtg}(sid, pp, P_j, prod, ppk, m, \sigma)$.
4: Send $(\mathsf{OProof}, sid, pp, P_j, prod, ppk, m, \sigma, P)$ to $\mathcal{S}$ and receive $(\mathsf{OProof}, sid, pp, P_j, prod, ppk, m, \sigma, P, \tau)$ from $\mathcal{S}$.
5: **If** $f \neq 1 \vee X \neq P \vee oid = \perp$ **Then**
Send $(\mathsf{OProof}, sid, pp, P_j, prod, ppk, m, \sigma, P, \perp)$ to P_{SM}.
6: **Else**
7: **If** $\tau = \perp \vee (oid, \tau, 0, P) \in \mathfrak{Open}$ **Then** output `error` and halt.
8: $\mathfrak{Open}.\mathsf{Add}(oid, \tau, 1, P)$ and send $(\mathsf{OProof}, sid, pp, P_j, prod, ppk, m, \sigma, P, \tau)$ to P_{SM}.

Verify Opening-Proof: On input $(\mathsf{Judge}, sid, pp, P_j, prod, ppk, m, \sigma, P, \tau)$ from P_i (or P_{SM})

1: Check that $sid = (P_{\mathrm{SM}}, sid')$ for some sid'. If not, ignore the request.
2: Set $(X, f, oid) := \mathsf{VfyRtg}(sid, pp, P_j, prod, ppk, m, \sigma)$.
3: Send $(\mathsf{Judge}, sid, pp, P_j, prod, ppk, m, \sigma, P, \tau)$ to $\mathcal{S}$ and receive $(\mathsf{Judge}, sid, pp, P_j, prod, ppk, m, \sigma, P, \tau, b)$ from $\mathcal{S}$.
4: Set $v := b$.
5: **If** $f = 0 \vee P = \perp \vee \tau = \perp$ **Then**
Send $(\mathsf{Judge}, sid, pp, P_j, prod, ppk, m, \sigma, P, \tau, 0)$ to P_i (or P_{SM}).
6: **Else If** $X \neq \perp$ **Then**
7: Let $r \in \mathfrak{Ratings}$ be $r := (pp, X, P_j, prod, ppk, m, \sigma, 1, lid', oid)$ for some lid'.
8: Set $r' := r$.
9: **If** $X = P \wedge (oid, \tau, 1, P) \in \mathfrak{Open}$ **Then** $v := 1$.
10: **Else If** $X \neq P \vee (oid, \tau, 0, P) \in \mathfrak{Open}$ **Then** $v := 0$.
11: **Else If** P_{SM} and P are honest and $b = 1$ **Then** output `error` and halt.
12: **Else**
13: Let $r \in \mathfrak{Ratings}$ be $r := (pp, \perp, P_j, prod, ppk, m, \sigma, 1, lid', oid)$ for some lid'.
14: Set $r' := r$.
15: **If** $(oid, \tau, 0, P) \in \mathfrak{Open}$ **Then** $v := 0$.
16: **Else If** $b = 1 \wedge P$ is honest **Then** output `error` and halt.
17: **Else If** $b = 1$ **Then**
Set $v{:=}1$, $r'[2]{:=}P$, $\mathfrak{Ratings}.\mathsf{Rep}(r, r')$, $r{:=}r'$, and run $\mathsf{RebLDB}(sid, r', \perp)$.
18: **If** $oid = \perp$ **Then**
$r'[10] := OIDC$, $\mathfrak{Ratings}.\mathsf{Rep}(r, r')$, $\mathfrak{Open}.\mathsf{Add}(OIDC, \tau, v, P)$, $OIDC := OIDC + 1$.
19: **Else** $\mathfrak{Open}.\mathsf{Add}(oid, \tau, v, P)$.
20: Send $(\mathsf{Judge}, sid, pp, P_j, prod, ppk, m, \sigma, P, \tau, v)$ to P_i (or P_{SM}).

Functionality 1: Reputation System

6.1.3 Security Properties

As many other ideal functionalities in the UC framework, we define $\mathcal{F}_{\mathsf{RS}}$ to work as a „registry service“ to store parameters, ratings, and opening-proofs. Using the right parameters, every party is able to check whether ratings and opening-proofs are stored by $\mathcal{F}_{\mathsf{RS}}$. In all activations, $\mathcal{F}_{\mathsf{RS}}$ lets the simulator $\mathcal{S}$ choose the values needed to respond to the activation. The requirements on these values are defined as restrictions for each activation. In the following, we discuss these restrictions and the implied security properties.

Registry Key Generation: Similar to the Signature Functionality $\mathcal{F}_{\mathsf{SIG}}$ [Can04] and the Public-Key Encryption Functionality $\mathcal{F}_{\mathsf{PKE}}$ [Can01], we do not make any security relevant requirements on the *public parameters pp*.

User Registration: Being registered is a prerequisite to rate a product and covers the first step to prevent Sybil attacks, whitewashing attacks, bad mouthing attacks, and ballot stuffing attacks. The user registration models an interactive protocol between P_{SM} and some party P_i. In general, $\mathcal{F}_{\mathsf{RS}}$ lets the simulator $\mathcal{S}$ decide whether party P_i successfully registered, with the following two restrictions: non-registered honest parties communicating with an honest P_{SM} using the right public parameters will always be registered after the protocol execution ($b = 1$) and an honest P_{SM} will reject a party from registering when wrong parameters are used ($b = 0$).

Product Addition and VfyProd: The NewProduct-activation is used by party P_i to publish a new *product-public key ppk* for a given product $prod \in \{0,1\}^*$. The value *ppk* is bound to the bitstring *prod* and to the party requesting it, such that every party can validate the ownership of a product. Formally this means that a product-public key is only *valid* for one specific pair $(P, prod)$. This is a very important requirement, because it models *unforgeability* of product-public keys. Without this property any corrupted party P_j could „copy“ some *ppk* that was generated by an honest party P_i and declare foreign ratings as own ratings. Then all valid ratings for $(P_i, prod, ppk)$ would also be valid for $(P_j, prod', ppk')$. Since we want to have a reliable, trustworthy and fair system such attacks must be prevented. We emphasize that VfyProd is modeled as an internal subroutine within $\mathcal{F}_{\mathsf{RS}}$ and is implicitly used in other activations.

Purchase: Another prerequisite to rate a product is to purchase it. This is necessary to prevent value imbalance attacks. The purchasing protocol is an interactive protocol between two parties: the seller P_j and the purchaser P_i. Naturally, before purchasing a product its corresponding product-public key is verified. Only if this is valid the protocol will be executed. For two honest parties the purchasing process will successfully finish, whereas the simulator $\mathcal{S}$ determines the outcome of the protocol execution in any other case.

Rating a Product: When party P_i wants to rate the product *prod* with public key *ppk* owned by party P_j, P_i must be registered, must have purchased the specified product, and must not have rated the product before. Being registered is necessary to *open* ratings, whereas having purchased the product enables rating verifiers to detect self-ratings, bad mouthing attacks and ballot stuffing attacks. In the case that P_{SM} is honest, $\mathcal{F}_{\mathsf{RS}}$ guarantees *anonymity of raters*: the simulator $\mathcal{S}$ is asked to output a *rating* σ, that is valid for the specified product without knowing the rating party. Hence, the output rating cannot depend on the rater's identity. In the case that P_{SM} is corrupted, the simulator $\mathcal{S}$ obtains the identity of the rater, because in this case anonymity cannot be achieved.

Rating Verification and Determining the Rater's Identity: Given the right parameters, every rating can be verified. Note that ratings are only verified, if the specified product is valid. A *valid rating* guarantees the following properties, even for maliciously generated ratings:

- Non-Self-Rating: the rater is not the owner of the product.
- Linkability: the rater purchased the product (will be discussed later in detail).
- Traceability: the rater is registered and can be identified.

Every single property is crucial for trustworthy reputation. If self-ratings would not be prevented, ballot stuffing attacks were possible. The same holds for linkability, but this will be discussed later in detail. Being able to open ratings is also very important in practical applications, because otherwise misbehaving parties can not be identified and punished. Hence, it must be guaranteed that honest parties are not blamed having rated some product, when they did not. This property is called *non-frameability* and is discussed later in detail.

$\mathcal{F}_{\mathsf{RS}}$ not only asks the simulator $\mathcal{S}$ to validate a rating, but also to determine the rater's identity. This models the ability of P_{SM} to open *every* rating, not only those for which an Open-request occurs. Furthermore, it simplifies the definition of $\mathcal{F}_{\mathsf{RS}}$ without weakening the security properties, because VfyRtg encapsulates all important characteristics of a valid rating in a single and reusable procedure.

Linking Ratings and RebLDB: For every party using a reputation system it is important to know whether two valid ratings for the same product are generated by the same party. If this is true, the rater behaved dishonestly. We call this property *linkability*, which prevents bad mouthing attacks and ballot stuffing attacks, and formally defines an equivalence relation on ratings:

Reflexivity: $\mathsf{Link}(x, x) = 1$

Symmetry: $\mathsf{Link}(x, y) = \mathsf{Link}(y, x)$

Transitivity: $\mathsf{Link}(x, y) = 1 \wedge \mathsf{Link}(y, z) = 1 \Rightarrow \mathsf{Link}(x, z) = 1$.

The value *lid* stored by $\mathcal{F}_{\mathsf{RS}}$ for every rating represents the equivalence class the rating belongs to. Initially, *lid* is set to the current value of a global counter *LIDC*. The linking-class identifiers are updated by the RebLDB algorithm whenever a new rating is added to the list $\mathfrak{Ratings}$ (via Rate and Verify) or new linking information is obtained (via Link and Judge). This algorithm is only for internal use and not callable by any party. The RebLDB-algorithm merges two equivalence classes in the following cases:

- Step 2 covers calls to the algorithm from Rate, Verify, and Judge ($s = \perp$), where P_{SM} is not corrupted and/or X_1 is an uncorrupted rater ($X_1 \neq \perp$). In these cases RebLDB selects all valid ratings for the specified product from the same rater X_1 (the set $\mathcal{L}$) and sets the value *lid* ($\ell[9]$ for $\ell \in \mathcal{L}$) for all ratings in $\mathcal{L}$ to the minimal value within the selected ratings.

- Step 5 handles requests from Link where either the identity of the rater is not known but the simulator $\mathcal{S}$ tells $\mathcal{F}_{\mathsf{RS}}$ that these ratings are linkable (Step 6 of Link), or the identity of some corrupted party can be updated for some rating, because it is linkable to another rating $\mathcal{F}_{\mathsf{RS}}$ already knows the identity of (Step 9 in Link). According to the transitivity of the linkability relation, RebLDB merges the two equivalence classes into one class by selecting all ratings within the two classes (Step 9) and setting *lid* to be the smaller of both values. Additionally, if a party identity is given in X_1 or X_2 this value will be set for all ratings within the equivalence class (Step 10).

- In Steps 11–18 RebLDB verifies that there do not exist more equivalence classes for an honestly generated product than the party owning the product sold. This ensures that it is only possible to rate a product once (without being linkable) after purchasing.

  When P_{SM} is corrupted, it is possible that no linking information is available to $\mathcal{F}_{\mathsf{RS}}$. In this case $\mathcal{F}_{\mathsf{RS}}$ asks the simulator $\mathcal{S}$ to link all ratings for the product in question. Without this step a simple attack is possible:

  – $\mathcal{Z}$ lets the real-world adversary $\mathcal{A}$ corrupt P_{SM} and some party P_i, lets P_i purchase some product from an honest party P_j, generates multiple valid ratings for this product and verifies them.

  – In this scenario $\mathcal{F}_{\mathsf{RS}}$ adds the ratings to $\mathfrak{Ratings}$ during the Verify-protocol, which in turn calls RebLDB. Since no linking information is available to $\mathcal{F}_{\mathsf{RS}}$, without Step 13 $\mathcal{F}_{\mathsf{RS}}$ outputs `error`, even when all ratings are linkable. Hence, no protocol can realize $\mathcal{F}_{\mathsf{RS}}$.

  If after Step 13 there are still more equivalence classes than purchases, this violates the security requirements of $\mathcal{F}_{\mathsf{RS}}$.

Summarizing, the handling of equivalence classes is modeled by the RebLDB-algorithm which uses linking information obtained from the algorithms Rate, Verify, Link, and Judge.

Generating and Verifying Opening-Proofs: Opening-proofs are values that enable every party to verify that a blamed party is really the author of a given rating. This covers the property of *non-frameability*: no honest party can be accused being the author of a given rating, when it is not. $\mathcal{F}_{\mathsf{RS}}$ asks the simulator $\mathcal{S}$ to output *valid* opening-proofs and ignores the output of $\mathcal{S}$, if the given rating is invalid, a wrong identity is given or the rating has not been opened yet. Since there can be more than one valid opening-proof, the value *oid* is used to connect a rating with its list of opening-proofs. This mechanism ensures that an opening-proof cannot be used to determine a raters identity for other ratings.

6.2 Realizing Reputation Systems

In this section we provide a protocol that UC-realizes $\mathcal{F}_{\mathsf{RS}}$. To give some intuition to this protocol we describe the used building blocks and how they are combined in Section 6.2.1. Afterwards, we present the Protocol Π_{RS} in Section 6.2.2, that UC-realizes $\mathcal{F}_{\mathsf{RS}}$ in a hybrid model. The proof of UC-security is presented in Section 8.2.

6.2.1 Building Blocks and Intuition

The basic idea for a protocol realizing $\mathcal{F}_{\mathsf{RS}}$ is as follows. At first, a user chooses a personal secret key and registers it at a certification authority to bind the chosen key to his identity. Then, the user runs the Register-protocol to interactively prove knowledge of his secret key to the System Manager. When the System Manager accepts the interaction, he hands a

registration token to the user. A similar interactive proof is used in the Purchase-protocol, but a user obtains a *rating token* from a product provider when the interaction is accepted. To rate a purchased product a user can then non-interactively prove knowledge of a registration token and a rating token. The used non-interactive proof must ensure that ratings are linkable and can be opened by the System Manager. To enable the System Manager to generate opening-proofs, every user has to provide an *opening token* during the Register-protocol. By proving knowledge of this token non-interactively, the System Manager can convince anybody that a rating was generated by a claimed author.

For our protocol Π_{RS} we combine Σ-protocols, trapdoor commitments, digital signatures, public-key encryption, and the ideal functionalities $\mathcal{F}_{\mathsf{CRS}}$ for common reference strings, $\mathcal{F}_{\mathsf{CA}}$ for a certification authority, and $\mathcal{F}_{\mathsf{RO}}$ for a random oracle to provide a protocol that realizes $\mathcal{F}_{\mathsf{RS}}$. The Σ-protocols are used as the basis for interactive and non-interactive proofs. To apply Damgård's technique, Lemma 2.1, which provides a transformation of Σ-protocols into concurrent black-box zero-knowledge arguments of knowledge, we need a trapdoor commitment scheme. Since universally composable commitment schemes do not exist in the plain model, as shown by Canetti and Fischlin [CF01], we use a common reference string provided by $\mathcal{F}_{\mathsf{CRS}}$ to realize the commitments. The functionality $\mathcal{F}_{\mathsf{RO}}$ is needed for the Fiat-Shamir Transformation, Definition 2.25, that transforms Σ-protocols into non-interactive arguments. Indeed, we use Signatures of Knowledge based on Σ-protocols. The digital signature schemes we incorporate in our protocol are used to provide the registration tokens and the rating tokens a user needs to rate a product. By integrating a public-key encryption scheme users can securely transmit opening tokens to the System Manager.

Now let us describe how these building blocks are combined, starting with the formal definition of the ideal functionality $\mathcal{F}_{\mathsf{CRS}}$.

$\mathcal{F}_{\mathsf{CRS}}$

$\mathcal{F}_{\mathsf{CRS}}$ operates on distribution $\mathcal{D}$ and controls the value *crs*, which is initialized to $\perp$.

Retrieving the CRS: On input (*sid*) from P or $\mathcal{S}$

1: **If** $crs = \perp$ **Then** set $crs \leftarrow \mathcal{D}$.

2: Send (*sid*, *crs*) to the activating party (P or $\mathcal{S}$).

Functionality 2: Common Reference String

For our protocol the common reference string will consist of a group description $\mathbb{GD}$ of Type-3 bilinear groups, the public key of the Trapdoor Pedersen Commitment scheme and the description of three collision-resistant hash functions. The Trapdoor Pedersen Commitment scheme is defined as follows.

Definition 6.1: Trapdoor Pedersen Commitments PD **- [Ped92]**
Let $\mathbb{GD} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathrm{e}, g_1, g_2)$ be a bilinear group setting of Type-3, with generators $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$. The Trapdoor Pedersen Commitment Scheme for messages $m \in \mathbb{Z}_p$ is defined as follows:

KeyGen($\mathbb{GD}$)

Choose $td \xleftarrow{u} \mathbb{Z}_p$, $u \xleftarrow{u} \mathbb{G}_1$, set $v := u^{td}$, and output $pk := (u, v)$.

$\mathsf{Commit}(pk, m)$
Choose $r \leftarrow\!\!\!\!\!\!\$\ \mathbb{Z}_p$ and output the commitment $c := u^m \cdot v^r$ and the decommitment $d := (r, m)$.

$\mathsf{Reveal}(pk, c, d = (r, m))$
If $\mathsf{Commit}(pk, m; r) = c$ **Then** output m.
Else output $\perp$

$\mathsf{TCommit}(pk, td)$
Choose $r^*, m^* \leftarrow\!\!\!\!\!\!\$\ \mathbb{Z}_p$, compute the equivocal commitment $\hat{c} := \mathsf{Commit}(pk, m^*; r^*)$ and the equivocation key $ek := (r^*, m^*, td)$, and output $(\hat{c}, ek)$.

$\mathsf{TReveal}(pk, \hat{c}, ek = (r^*, m^*, td), m)$
Compute $r := (m^* - m + td \cdot r^*) \cdot td^{-1}$ and output the decommitment $d := (r, m)$. $\triangle$

A peculiarity of the Trapdoor Pedersen Commitment is that equivocal commitments are generated exactly as normal commitments without using the trapdoor. Hence, it is possible to reveal arbitrary messages for any given commitment, as long as the trapdoor is known.

To ensure that users cannot register at the System Manager multiple times with different identities we require every user P_i to choose a personal *user secret key* usk_i and a corresponding public key M_i, which must be registered at the ideal functionality for certification $\mathcal{F}_{\mathsf{CA}}$ [Can04]. Once registered at $\mathcal{F}_{\mathsf{CA}}$ a value M_i cannot be modified.

$\mathcal{F}_{\mathsf{CA}}$

Registering Values: On input $(\mathsf{Register}, sid, v)$ from P_i
1: Send $(\mathsf{Register}, sid)$ to $\mathcal{S}$ and receive $(\mathsf{Register}, sid, \mathtt{ok})$ from $\mathcal{S}$.
2: **If** $sid = P_i$ and this is the first request **Then** record (P_i, v).

Retrieving registered values: On input $(\mathsf{Retrieve}, sid)$ from P_j
1: Send $(\mathsf{Retrieve}, sid, P_j)$ to $\mathcal{S}$ and receive $(\mathsf{Retrieve}, sid, P_j, \mathtt{ok})$ from $\mathcal{S}$.
2: **If** there is a tuple (sid, v) recorded **Then** send $(\mathsf{Retrieve}, sid, v)$ to P_j.
3: **Else** send $(\mathsf{Retrieve}, sid, \perp)$ to P_j.

Functionality 3: Certification Authority

Before a user P_i can be registered by the System Manager, the public parameters pp have to be set up by P_{SM}. These parameters are the public keys of the Pointcheval-Sanders Signature scheme and the Cramer-Shoup Encryption scheme.

Definition 6.2: Pointcheval-Sanders Signatures PS - [PS16]
Let $\mathbb{GD} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathrm{e}, g_1, g_2)$ be a bilinear group setting of Type-3, with generators $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$. The Pointcheval-Sanders Signature Scheme for messages $m \in \mathbb{Z}_p$ is defined as follows:

$\mathsf{KeyGen}(\mathbb{GD})$

1: Choose $\xi_1, \xi_2 \leftarrow_u \mathbb{Z}_p$, and $\tilde{g} \leftarrow_u \mathbb{G}_2$.

2: Set $(\tilde{X}, \tilde{Y}) := (\tilde{g}^{\xi_1}, \tilde{g}^{\xi_2})$, $sk := (\xi_1, \xi_2)$, $pk := (\tilde{g}, \tilde{X}, \tilde{Y})$, and output (pk, sk).

$\mathsf{Sign}(sk, m)$

Choose $s \leftarrow_u \mathbb{G}_1$, set $\sigma := (\sigma_1, \sigma_2) := (s, s^{\xi_1 + \xi_2 \cdot m})$ and output σ as signature on m.

$\mathsf{Verify}(pk, m, \sigma)$

If $\sigma_1 \neq 1_{\mathbb{G}_1} \wedge \mathrm{e}(\sigma_1, \tilde{X} \cdot \tilde{Y}^m) = \mathrm{e}(\sigma_2, \tilde{g})$ **Then** output 1

Else output 0. △

To sign committed messages $M = g^m \in \mathbb{G}_1$ a modified signing algorithm can be used:

$\mathsf{Sign}(sk, M)$

Choose $\alpha \leftarrow_u \mathbb{Z}_p$, set $\sigma = (\sigma_1, \sigma_2) := \left(g_1^{\alpha}, (g_1^{\xi_1} \cdot M^{\xi_2})^{\alpha}\right)$, and output σ as signature on m.

Definition 6.3: Cramer-Shoup Encryption CS - [CS98]
Let $\mathbb{GD} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathrm{e}, g_1, g_2)$ be a bilinear group setting of Type-3, with generators $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$, and let $\mathcal{H} : \{0,1\}^* \to \mathbb{Z}_p$ be a collision resistant hash function. The Cramer-Shoup Encryption Scheme for messages $m \in \mathbb{G}_2$ is defined as follows:

$\mathsf{KeyGen}(\mathbb{GD})$

1: Choose $\tilde{h} \leftarrow_u \mathbb{G}_2$ and $\zeta_1, \zeta_2, \zeta_3, \zeta_4, \zeta_5 \leftarrow_u \mathbb{Z}_p$.

2: Set $\tilde{b} := g_2^{\zeta_1} \cdot \tilde{h}^{\zeta_2}$, $\tilde{d} := g_2^{\zeta_3} \cdot \tilde{h}^{\zeta_4}$, $\tilde{f} := g_2^{\zeta_5}$, $sk := (\zeta_1, \zeta_2, \zeta_3, \zeta_4, \zeta_5)$, and $pk := (g_2, \tilde{h}, \tilde{b}, \tilde{d}, \tilde{f}, \mathcal{H})$.

3: Output the key pair (sk, pk).

$\mathsf{Enc}(pk, m)$

1: Choose $\beta \leftarrow_u \mathbb{Z}_p$

2: Set $ct_1 := g_2^{\beta}$, $ct_2 := \tilde{h}^{\beta}$, $ct_3 := m \cdot \tilde{f}^{\beta}$, $\omega := \mathcal{H}(ct_1, ct_2, ct_3)$, $ct_4 := (\tilde{b} \cdot \tilde{d}^{\omega})^{\beta}$.

3: Output the cipher text $ct := (ct_1, ct_2, ct_3, ct_4)$.

$\mathsf{Dec}(sk, ct)$

If $ct_4 = ct_1^{\zeta_1} \cdot ct_2^{\zeta_2} \cdot (ct_1^{\zeta_3} \cdot ct_2^{\zeta_4})^{\omega}$, where $\omega := \mathcal{H}(ct_1, ct_2, ct_3)$ **Then** output $m := ct_3 \cdot ct_1^{-\zeta_5}$

Else output $\perp$ △

Using the common reference string provided by $\mathcal{F}_{\mathsf{CRS}}$, the public parameters pp provided by P_{SM}, and the user's secret and public key (usk_i, M_i) a user P_i can register at the System Manager by running a concurrent black-box zero-knowledge argument of knowledge to prove that P_i *knows a value* usk_i such that M_i is the public key corresponding to usk_i. When P_{SM} is convinced, he computes a *registration token* σ_i, which is a Pointcheval-Sanders Signature on the message usk_i, that is computed as a signature on the committed message M_i. During the registration the user P_i also has to provide a value $\hat{Y}_i \in \mathbb{G}_2$, the *opening token*, that depends on usk_i and will be used by the System Manager to open ratings.

Hence, this value must be sent securely to P_{SM} to preserve anonymity. For this purpose the Cramer-Shoup Encryption scheme is used in $\mathbb{G}_2$.

To provide a rateable product *prod* a user P_i has to publish a *product public-key ppk*. Therefor the user generates a new public key $\mathsf{PS}.pk_{i,prod}$ of the Pointcheval-Sanders Signature scheme, where the generator $\tilde{g} \in \mathbb{G}_2$ is not chosen uniformly at random but determined by the hash value $\mathcal{H}_2(i, prod)$. Then, the user binds $\mathsf{PS}.pk_{i,prod}$ to his identity with a Signature of Knowledge. This Signature of Knowledge is a Σ-protocol that was transformed using the Fiat-Shamir Transformation, Definition 2.25, hence they are only secure in the random oracle model. The random oracle is provided by the ideal functionality $\mathcal{F}_{\mathsf{RO}}$ [HM04].

$\mathcal{F}_{\mathsf{RO}}$

$\mathcal{F}_{\mathsf{RO}}$ operates on security parameter λ and manages the list L_{RO} of pairs of bitstrings, which is initially empty.

Retrieving values: On input (sid, m) from P or $\mathcal{S}$

1: **If** $(m, v) \in L_{RO}$ for some $v \in \{0,1\}^\lambda$ **Then** set $h := v$
2: **Else** choose $h \xleftarrow{u} \{0,1\}^\lambda$ and store (m, h) in L_{RO}
3: Send (sid, m, h) to the activating party (P or $\mathcal{S}$).

Functionality 4: Random Oracle

A user P_i can purchase a product *prod*, managed by user P_j, by running the same zero-knowledge protocol that is used for registration. After a successful protocol execution the user P_i obtains a *rating token*, which is a Pointcheval-Sanders Signature on usk_i, valid under the public key $\mathsf{PS}.pk_{i,prod}$.

To rate a product a user P_i has to non-interactively prove knowledge of the registration token σ_i, the rating token $\sigma_{i,j,prod}$, and its personal user secret key usk_i, for which the tokens were generated. Similar to the product public key, ratings are Signatures of Knowledge obtained from Σ-protocols by applying the Fiat-Shamir Transformation.

To determine the rater's identity, the System Manager can use the opening token that was provided by the user P_i during registration. By proving non-interactively that an opening token and a rating both depend on the same unknown user secret key usk_i the System Manager can create an *opening-proof*, which are again Signatures of Knowledge. It is important not to publish the opening tokens, because they allow to open any rating.

6.2.2 The Protocol

In our construction the output of $\mathcal{F}_{\mathsf{CRS}}$ is $(\mathbb{GD}, \mathsf{PD}.pk, \mathcal{H}, \mathcal{H}_1, \mathcal{H}_2)$, where $\mathbb{GD}$ is the output of the bilinear group generator $\mathsf{BiGrGen}(1^\lambda)$, $\mathsf{PD}.pk = (u, v) \in \mathbb{G}_1^2$ is the public key of the Trapdoor Pedersen Commitment scheme, and $\mathcal{H}\colon \{0,1\}^* \to \mathbb{Z}_p$, $\mathcal{H}_1\colon \{0,1\}^* \to \mathbb{G}_1$, and $\mathcal{H}_2\colon \{0,1\}^* \to \mathbb{G}_2$ are collision-resistant hash functions. We assume that every party obtains the common-reference string during its first activation.

We write $y := \mathcal{F}_{\mathsf{RO}}(x)$ to indicate a call to $\mathcal{F}_{\mathsf{RO}}$ on input (sid, x) and outputting y to the calling party.

We assume to communicate via authenticated channels between two parties. This implies that the identities of communicating parties are known to each other and that the adversary cannot modify the message's content.

Π_{RS}

All parties except P_{SM}: On the first activation of P_i

1: Choose a value $usk_i \leftarrow\!\!\shortmid \mathbb{Z}_p$ and compute $M_i := g_1^{usk_i}$, where $g_1 \in \mathbb{G}_1$ is given by $\mathcal{F}_{\mathsf{CRS}}$.
2: Send $(\mathsf{Register}, P_i, M_i)$ to $\mathcal{F}_{\mathsf{CA}}$, and store the user-secret-key usk_i.

Registry Key Generation: When P_{SM} receives (KeyGen, sid) from $\mathcal{Z}$

1: Run $\mathsf{PS.KeyGen}(\mathbb{GD})$ to obtain $\mathsf{PS}.pk := (\tilde{g}, \tilde{X}, \tilde{Y})$ and $\mathsf{PS}.sk := (\xi_1, \xi_2)$.
2: Run $\mathsf{CS.Setup}(\mathbb{GD})$ to obtain $\mathsf{CS}.pk := (g_2, \tilde{h}, \tilde{b}, \tilde{d}, \tilde{f}, \mathcal{H})$ and $\mathsf{CS}.sk := (\zeta_1, \zeta_2, \zeta_3, \zeta_4, \zeta_5)$.
3: Set $pp := (\mathsf{PS}.pk, \mathsf{CS}.pk)$ and $psk := (\mathsf{PS}.sk, \mathsf{CS}.sk)$.
4: Set $\mathfrak{Params}.\mathsf{Add}(pp)$ and $\mathfrak{Params}_s.\mathsf{Add}(pp, psk)$.
5: Output $(\mathsf{KeyGen}, sid, pp)$.

User Registration: When P_i receives $(\mathsf{Register}, sid, pp')$ from $\mathcal{Z}$

P_i: 1: Choose $\alpha, r \leftarrow\!\!\shortmid \mathbb{Z}_p$, compute $T := g_1^{\alpha}$, $R := u^{\mathcal{H}(T)} \cdot v^r$ and send (pp', R) to P_{SM}.
P_{SM}: 2: Obtain M_i from $\mathcal{F}_{\mathsf{CA}}(\mathsf{Retrieve}, P_i)$.
3: **If** $\mathcal{F}_{\mathsf{CA}}$ returned $(\mathsf{Retrieve}, P_i, \bot) \vee pp' \notin \mathfrak{Params} \vee (P_i, pp', M', Y', \sigma') \in \mathfrak{Reg}$ for some M', Y', σ' **Then** send `abort` to P_i and output $(\mathsf{Register}, sid, pp', P_i, 0)$.
4: **Else** Choose $ch \leftarrow\!\!\shortmid \mathbb{Z}_p$ and send ch to P_i.
P_i: 5: **If** P_{SM} sent `abort` **Then** output $(\mathsf{Register}, sid, pp', P_i, 0)$.
6: **Else** Compute $s_\alpha := \alpha + ch \cdot usk_i$, $ct \leftarrow \mathsf{CS.Enc}(\mathsf{CS}.pk, \tilde{Y}^{usk_i})$ and send (s_α, T, r, ct) to P_{SM}.
P_{SM}: 7: Compute $\tilde{Y}_i := \mathsf{CS.Dec}(\mathsf{CS}.sk, ct)$.
8: **If** decrypting ct failed $\vee$ $M_i^{ch} \cdot T \neq g_1^{s_\alpha} \vee R \neq u^{\mathcal{H}(T)} \cdot v^r \vee \mathrm{e}(M_i, \tilde{Y}) \neq \mathrm{e}(g_1, \tilde{Y}_i)$ **Then** send `abort` to P_i and output $(\mathsf{Register}, sid, pp', P_i, 0)$.
9: **Else** compute $\sigma_i \leftarrow \mathsf{PS.Sign}(\mathsf{PS}.sk, M_i)$, set $\mathfrak{Reg}.\mathsf{Add}(P_i, pp', M_i, \tilde{Y}_i, \sigma_i)$, send σ_i to P_i, and output $(\mathsf{Register}, sid, pp', P_i, 1)$.
P_i: 10: **If** P_{SM} sent `abort` **Then** output $(\mathsf{Register}, sid, pp', P_i, 0)$.
11: **Else If** $\mathsf{PS.Verify}(pp', usk_i, \sigma_i) = 1$ **Then**
12: store (pp', usk_i, σ_i) and output $(\mathsf{Register}, sid, pp', P_i, 1)$.
13: **Else** output $(\mathsf{Register}, sid, pp', P_i, 0)$.

Product Addition: When P_i receives $(\mathsf{NewProduct}, sid, prod)$ from $\mathcal{Z}$

1: Compute $\tilde{g}_{i,prod} := \mathcal{H}_2(i, prod)$ and run $\mathsf{PS.KeyGen}(\mathbb{GD})$ with $\tilde{g}_{i,prod}$ as generator of $\mathbb{G}_2$ to obtain $\mathsf{PS}.pk_{i,prod} := (\tilde{g}_{i,prod}, \tilde{X}_{i,prod}, \tilde{Y}_{i,prod})$ and $\mathsf{PS}.sk_{i,prod} := (\xi_{1_{i,prod}}, \xi_{2_{i,prod}})$.
2: Compute $M_{i,prod} := \mathcal{H}_1(i, prod)^{usk_i}$.
3: Choose $r \leftarrow\!\!\shortmid \mathbb{Z}_p$ and compute $R_1 := \mathcal{H}_1(i, prod)^r$ and $R_2 := g_1^r$.
4: Set $ch_{i,prod} := \mathcal{F}_{\mathsf{RO}}(\mathsf{PS}.pk_{i,prod}, M_i, M_{i,prod}, R_1, R_2)$ and $s_{i,prod} := r + ch_{i,prod} \cdot usk_i$.
5: Set $ppk_{i,prod} := (M_i, M_{i,prod}, ch_{i,prod}, s_{i,prod}, \mathsf{PS}.pk_{i,prod})$.

6: Set $\mathfrak{Prods}_i.\mathsf{Add}(prod, ppk_{i,prod})$.
7: Output $(\mathsf{NewProduct}, sid, prod, ppk_{i,prod})$.

Purchase: When P_i receives $(\mathsf{Purchase}, sid, P_j, prod, ppk)$ from $\mathcal{Z}$

P_i: 1: **If** $\mathsf{VfyProd}(P_j, prod, ppk) = 0 \vee P_i = P_j$ **Then** ignore the request.
2: **Else** choose $\alpha, r \xleftarrow{u} \mathbb{Z}_p$, compute $T := g_1^{\alpha}$, $R := u^{\mathcal{H}(T)} \cdot v^r$ and send $(prod, ppk, R)$ to P_j.
P_j: 3: Obtain M_i from $\mathcal{F}_{\mathsf{CA}}(\mathsf{Retrieve}, P_i)$.
4: **If** $\mathcal{F}_{\mathsf{CA}}$ returned $(\mathsf{Retrieve}, P_i, \perp) \vee (prod, ppk) \notin \mathfrak{Prods}_j$ **Then**
5: send `abort` to P_i and output $(\mathsf{Purchase}, sid, P_i, P_j, prod, ppk, 0)$.
6: **Else** choose $ch \xleftarrow{u} \mathbb{Z}_p$ and send ch to P_i.
P_i: 7: **If** P_j sent `abort` **Then** output $(\mathsf{Purchase}, sid, P_i, P_j, prod, ppk, 0)$.
8: **Else** compute $s_\alpha := \alpha + ch \cdot usk_i$ and send (s_α, T, r) to P_j.
P_j: 9: **If** $M_i^{ch} \cdot T \neq g_1^{s_\alpha} \vee R \neq u^{\mathcal{H}(T)} \cdot v^r$ **Then**
10: send `abort` to P_i and output $(\mathsf{Purchase}, sid, P_i, P_j, prod, ppk, 0)$.
11: **Else** compute $\sigma_{i,j,prod} \leftarrow \mathsf{PS.Sign}(\mathsf{PS}.sk_{i,prod}, M_i)$
12: set $\mathfrak{Purch}_j.\mathsf{Add}(P_i, prod, \sigma_{i,j,prod})$.
13: Send $\sigma_{i,j,prod}$ to P_i.
P_i: 14: **If** P_j sent `abort` **Then** output $(\mathsf{Purchase}, sid, P_i, P_j, prod, ppk, 0)$.
15: **Else If** $\mathsf{PS.Verify}(\mathsf{PS}.pk_{i,prod}, usk_i, \sigma_{i,j,prod}) = 1$ **Then**
16: store $\sigma_{i,j,prod}$, and output $(\mathsf{Purchase}, sid, P_i, P_j, prod, ppk, 1)$.
17: **Else** output $(\mathsf{Purchase}, sid, P_i, P_j, prod, ppk, 0)$.

VfyProd: On local input $(P_j, prod, ppk)$
1: Obtain M_j from $\mathcal{F}_{\mathsf{CA}}(\mathsf{Retrieve}, P_j)$
2: **If** $\mathcal{F}_{\mathsf{CA}}$ returned $(\mathsf{Retrieve}, P_j, \perp)$ **Then** return 0.
3: **Else** parse ppk as $(M'_j, M_{j,prod}, ch_{j,prod}, s_{j,prod}, \mathsf{PS}.pk_{j,prod})$.
4: Set $R_1 := \mathcal{H}_1(j, prod)^{s_{j,prod}} \cdot M_{j,prod}^{-ch_{j,prod}}$ and $R_2 := g_1^{s_{j,prod}} \cdot M_j^{-ch_{j,prod}}$.
5: **If** $M_j \neq M'_j \vee ch_{j,prod} \neq \mathcal{F}_{\mathsf{RO}}(\mathsf{PS}.pk_{j,prod}, M_j, M_{j,prod}, R_1, R_2)$ **Then** return 0.
6: **Else** return 1.

Rate Product: When P_i receives $(\mathsf{Rate}, sid, pp, P_j, prod, ppk, m)$ from $\mathcal{Z}$
1: **If** no tuple (pp, usk_i, σ_i) is stored such that $\mathsf{PS.Verify}(pp, usk_i, \sigma_i) = 1$
$\vee$ no $\sigma_{i,j,prod}$ is stored such that $\mathsf{PS.Verify}(\mathsf{PS}.pk_{i,prod}, usk_i, \sigma_{i,j,prod}) = 1$
$\vee$ a tuple $(m', \sigma) = (m', T_1, T_2, T_3, T_4, T_5, ch, s)$ is stored such that $(\mathsf{Verify}, sid, pp, P_j, prod, ppk, m', \sigma) = 1$ **Then** ignore the request.
2: Choose $t_1, t_2, k \xleftarrow{u} \mathbb{Z}_p$.
3: Compute $T_1 := \sigma_{i,1}^{t_1}$, $T_2 := \sigma_{i,2}^{t_1}$, $T_3 := \sigma_{i,j,prod,1}^{t_2}$, $T_4 := \sigma_{i,j,prod,2}^{t_2}$, $T_5 := \mathcal{H}_1(j, prod)^{usk_i}$.
4: Compute $R_1 := \mathrm{e}(T_1, \tilde{Y})^k$, $R_2 := \mathrm{e}(T_3, \tilde{Y}_{j,prod})^k$, $R_3 := \mathcal{H}_1(j, prod)^k$.
5: Set $ch := \mathcal{F}_{\mathsf{RO}}(T_1, T_2, T_3, T_4, T_5, R_1, R_2, R_3, pp, prod, ppk, m)$, and $s := k + ch \cdot usk_i$.
6: Set $\sigma := (T_1, T_2, T_3, T_4, T_5, ch, s)$ and store (m, σ).
7: Output $(\mathsf{Rate}, sid, pp, P_j, prod, ppk, m, \sigma)$.

Verify Rating: When P_i receives $(\mathsf{Verify}, sid, pp, P_j, prod, ppk, m, \sigma)$ from $\mathcal{Z}$

1: **If** $\mathsf{VfyProd}(P_j, prod, ppk) = 0$ **Then** ignore the request.
2: Parse σ as $(T_1, T_2, T_3, T_4, T_5, ch, s)$.
3: Set $R_1' \coloneqq \mathrm{e}(T_1, \tilde{X})^{ch} \cdot \mathrm{e}(T_2, \tilde{g})^{-ch} \cdot \mathrm{e}(T_1, \tilde{Y})^{s}$, $R_3' \coloneqq T_5^{-ch} \cdot \mathcal{H}_1(j, prod)^{s}$, $R_2' \coloneqq \mathrm{e}(T_3, \tilde{X}_{j,prod})^{ch} \cdot \mathrm{e}(T_4, \tilde{g}_{j,prod})^{-ch} \cdot \mathrm{e}(T_3, \tilde{Y}_{j,prod})^{s}$.
4: Set $f \coloneqq \left[T_5 \neq M_{j,prod} \wedge ch = \mathcal{F}_{\mathsf{RO}}(T_1, T_2, T_3, T_4, T_5, R_1', R_2', R_3', pp, prod, ppk, m)\right]$
5: Output $(\mathsf{Verify}, sid, pp, P_j, prod, ppk, m, \sigma, f)$.

Link Ratings: When P_i receives $(\mathsf{Link}, sid, pp, P_j, prod, ppk, m_0, \sigma_0, m_1, \sigma_1)$ from $\mathcal{Z}$

1: **If** $(\mathsf{Verify}, sid, pp, P_j, prod, ppk, m_k, \sigma_k) = 1$, for $k \in \{0, 1\}$ **Then**
2: Parse σ_0 as $(T_1, T_2, T_3, T_4, T_5, ch, s)$, and σ_1 as $(T_1', T_2', T_3', T_4', T_5', ch', s')$.
3: Output $(\mathsf{Link}, sid, pp, P_j, prod, ppk, m_0, \sigma_0, m_1, \sigma_1, (T_5 = T_5'))$.
4: **Else** Output $(\mathsf{Link}, sid, pp, P_j, prod, ppk, m_0, \sigma_0, m_1, \sigma_1, 0)$.

Identify Raters: When P_{SM} receives $(\mathsf{Open}, sid, pp, P_j, prod, ppk, m, \sigma)$ from $\mathcal{Z}$

1: **If** $pp \notin \mathfrak{Params}$ **Then** ignore the request.
2: Set $f \coloneqq (\mathsf{Verify}, sid, pp, P_j, prod, ppk, m, \sigma)$.
3: **If** $f = 1$ **Then** parse σ as $(T_1, T_2, T_3, T_4, T_5, ch, s)$ and iterate through $\mathfrak{Reg}$ to find a tuple $(P_i, pp, M_i, \tilde{Y}_i, \sigma_i)$ such that $\mathrm{e}(T_5, \tilde{Y}) = \mathrm{e}(\mathcal{H}_1(j, prod), \tilde{Y}_i)$.
4: **If** $f = 0 \vee$ no such tuple exists **Then** output $(\mathsf{Open}, sid, pp, P_j, prod, ppk, m, \sigma, \perp)$.
5: **Else** Set $\mathfrak{Open}.\mathsf{Add}(pp, P_j, prod, ppk, m, \sigma, P_i)$.
6: Output $(\mathsf{Open}, sid, pp, P_j, prod, ppk, m, \sigma, P_i)$.

Generate Opening-Proof: When P_{SM} receives $(\mathsf{OProof}, sid, pp, P_j, prod, ppk, m, \sigma, P)$ from $\mathcal{Z}$

1: **If** $pp \notin \mathfrak{Params}$ **Then** ignore the request.
2: Set $f \coloneqq (\mathsf{Verify}, sid, pp, P_j, prod, ppk, m, \sigma)$.
3: **If** $f = 0 \vee (pp, P_j, prod, ppk, m, \sigma, P) \notin \mathfrak{Open}$ **Then**
4: output $(\mathsf{OProof}, sid, pp, P_j, prod, ppk, m, \sigma, P, \perp)$.
5: **Else** Parse σ as $(T_1, T_2, T_3, T_4, T_5, ch, s)$.
6: Select the tuple $(P, pp, M_i, \tilde{Y}_i, \sigma_i)$ such that $\mathrm{e}(T_5, \tilde{Y}) = \mathrm{e}(\mathcal{H}_1(j, prod), \tilde{Y}_i)$.
7: Choose $\beta \xleftarrow{\$} \mathbb{Z}_p$ and compute $ct \coloneqq (ct_1, ct_2, ct_3, ct_4) \leftarrow \mathsf{CS.Enc}(\mathsf{CS}.pk, \tilde{Y}_i; \beta)$.
8: Choose $r \xleftarrow{\$} \mathbb{Z}_p$ and compute $R_1 \coloneqq g_2^r$, $R_2 \coloneqq \tilde{h}^r$, $R_3 \coloneqq \mathrm{e}(\mathcal{H}_1(j, prod), \tilde{f})^r$.
9: Compute $\omega \coloneqq \mathcal{H}(ct_1, ct_2, ct_3)$, $R_4 \coloneqq (\tilde{b} \cdot \tilde{d}^{\omega})^r$, $R_5 \coloneqq \mathrm{e}(g_1, \tilde{f})^r$.
10: Set $\hat{ch} \coloneqq \mathcal{F}_{\mathsf{RO}}(ct, R_1, R_2, R_3, R_4, R_5, \sigma, i, M_i)$, $\hat{s} \coloneqq r + \hat{ch} \cdot \beta$, and $\tau \coloneqq (P_i, ct, \hat{ch}, \hat{s})$.
11: Set $\mathfrak{Open}.\mathsf{Add}(pp, P_j, prod, ppk, m, \sigma, P, \tau)$.
12: Output $(\mathsf{OProof}, sid, pp, P_j, prod, ppk, m, \sigma, P, \tau)$.

Verify Opening-Proof: When P_i receives $(\mathsf{Judge}, sid, pp, P_j, prod, ppk, m, \sigma, P, \tau)$ from $\mathcal{Z}$

1: Set $f \coloneqq (\mathsf{Verify}, sid, pp, P_j, prod, ppk, m, \sigma)$.
2: Obtain M from $\mathcal{F}_{\mathsf{CA}}(\mathsf{Retrieve}, P)$.
3: **If** $\mathcal{F}_{\mathsf{CA}}$ returned $(\mathsf{Retrieve}, P, \perp) \vee f = 0 \vee P = \perp \vee \tau = \perp$ **Then**

4: output $(\mathsf{Judge}, sid, pp, P_j, prod, ppk, m, \sigma, P, \tau, 0)$.
5: **Else** Parse τ as $(P_i, (ct_1, ct_2, ct_3, ct_4), \hat{ch}, \hat{s})$.
6: Compute $R_1 \coloneqq ct_1^{-\hat{ch}} \cdot g_2^{\hat{s}}$, $R_2 \coloneqq ct_2^{-\hat{ch}} \cdot \tilde{h}^{\hat{s}}$.
7: Compute $R_3 \coloneqq \mathrm{e}(\mathcal{H}_1(j, prod), ct_3)^{-\hat{ch}} \cdot \mathrm{e}(T_5, \tilde{Y})^{\hat{ch}} \cdot \mathrm{e}(\mathcal{H}_1(j, prod), \tilde{f})^{\hat{s}}$
8: Compute $\omega \coloneqq \mathcal{H}(ct_1, ct_2, ct_3)$.
9: Compute $R_4 \coloneqq ct_4^{-\hat{ch}} \cdot (\tilde{b} \cdot \tilde{d}^{\omega})^{\hat{s}}$, $R_5 \coloneqq \mathrm{e}(g_1, ct_3)^{-\hat{ch}} \cdot \mathrm{e}(M, \tilde{Y})^{\hat{ch}} \cdot \mathrm{e}(g_1, \tilde{f})^{\hat{s}}$.
10: Set $f \coloneqq (P = P_i \wedge \hat{ch} = \mathcal{F}_{\mathsf{RO}}(ct, R_1, R_2, R_3, R_4, R_5, \sigma, i, M))$.
11: Output $(\mathsf{Judge}, sid, pp, P_j, prod, ppk, m, \sigma, P, \tau, f)$

Protocol 1: Protocol for $\mathcal{F}_{\mathsf{RS}}$

The Protocol Π_{RS} is defined in the $(\mathcal{F}_{\mathsf{RO}}, \mathcal{F}_{\mathsf{CRS}}, \mathcal{F}_{\mathsf{CA}})$-hybrid model assuming authenticated channels for communication. That means, for a proof of security we have to define a simulator $\mathcal{S}$ that fully controls the ideal functionalities $\mathcal{F}_{\mathsf{RO}}$, $\mathcal{F}_{\mathsf{CRS}}$, and $\mathcal{F}_{\mathsf{CA}}$. Hence, the simulator determines the public key of the Trapdoor Pedersen Commitment scheme and knows the corresponding trapdoor, which is needed to simulate the interactive proofs during the Register and Purchase-protocols. Furthermore, $\mathcal{F}_{\mathsf{RO}}$ can be patched to generate simulated Signatures of Knowledge. With these prerequisites we are able to prove the following theorem.

Theorem 6.1:
If the SXDH Assumption and the Pointcheval-Sanders Assumption hold for bilinear group generator BiGrGen, the hash functions $\mathcal{H}, \mathcal{H}_1$, and $\mathcal{H}_2$ are collision-resistant, and the communication channels between interacting parties are authenticated, Protocol Π_{RS} UC-realizes $\mathcal{F}_{\mathsf{RS}}$ in the $(\mathcal{F}_{\mathsf{RO}}, \mathcal{F}_{\mathsf{CRS}}, \mathcal{F}_{\mathsf{CA}})$-hybrid model, in the presence of static adversaries. $\diamond$

The full proof is given in Section 8.2.

7 Further Extensions and Future Research

This chapter is based on our paper „Practical, Anonymous, and Publicly Linkable Universally-Composable Reputation Systems“, presented at the Cryptographers' Track of the RSA Conference 2018 [BEJ18].

In this chapter we briefly discuss further extensions of the ideal functionality $\mathcal{F}_{\mathsf{RS}}$ and its realizing protocol Π_{RS} and point out interesting aspects of anonymous reputation systems that need further research.

7.1 Considering Adaptive Adversaries against $\mathcal{F}_{\mathsf{RS}}$

Theorem 6.1 only claims security against static adversaries, because anonymity and linkability are conflicting security properties, which impede the construction of UC-secure protocols in the presence of adaptive adversaries. To illustrate that, consider the following corruption with an adaptive adversary $\mathcal{A}$:

- The System Manager is honest.
- After registering and purchasing some product an honest user P_i is asked to rate the product.
- The user P_i is corrupted by $\mathcal{A}$ directly after outputting the rating.
- Now the adversary $\mathcal{A}$ generates a second rating for the same product.

According to the definition of $\mathcal{F}_{\mathsf{RS}}$ the two ratings must be linkable. In the real protocol this is true because the user P_i used the same secret keys the adversary obtains during corruption. But in the ideal protocol the simulator $\mathcal{S}$ does not obtain any identifying information about P_i during a Rate-request and has to simulate a rating. Hence, the simulator $\mathcal{S}$ has to guess which identity to use during the rating simulation. This means that ratings are not linkable in the ideal protocol with high probability such that it is easy to distinguish between the ideal and the real protocol. Therefore, it seems unlikely that $\mathcal{F}_{\mathsf{RS}}$ is UC-realizable in the presence of adaptive adversaries. This problem needs further investigation in future research.

7.2 Incorporating Revocation into $\mathcal{F}_{\mathsf{RS}}$ and Π_{RS}

Comparing the model for reputation systems from Chapter 4 with $\mathcal{F}_{\mathsf{RS}}$ it stands out that revocation of users is not considered in $\mathcal{F}_{\mathsf{RS}}$. The opening-proof mechanism $\mathcal{F}_{\mathsf{RS}}$ provides is a revocation technique that rescinds anonymity of the author of a single rating. But more extensive notions of revocation exist:

- Revoke a user completely such that the user cannot purchase products anymore, all existing ratings become invalid, and all future ratings will be invalid.
- Revoke all or selected existing ratings of a user while preserving the ability to rate.
- Preserve existing ratings of a user but prohibit future ratings.

Which revocation technique to use depends on the higher-level application. Because of that, we do not integrate revocation in the definition of $\mathcal{F}_{\mathsf{RS}}$. Nevertheless, Protocol Π_{RS} can be easily extended to support verifier-local revocation, which revokes a user completely. To revoke the user P_i the System Manager P_{SM}, or even P_i himself, publishes the opening token $\tilde{Y}_i$ as the user's revocation token rt_i on a revocation-list $\mathcal{RL}$. Then any verifier can check whether the author of a given rating $\sigma = (T_1, T_2, T_3, T_4, T_5, ch, s)$ is revoked by testing if the equation $\mathrm{e}(T_5, \tilde{Y}) = \mathrm{e}(\mathcal{H}_1(j, prod), rt)$ holds for an entry $rt \in \mathcal{RL}$. Since this is the same check the System Manager performs when opening a rating, the anonymity of a revoked user is also rescinded. Analogously, during Purchase-requests the product owner can test whether $\mathrm{e}(M_i, \tilde{Y}) = \mathrm{e}(g_1, rt)$ holds to detect a revoked user P_i.

In contrast to that, a simple way to revoke selected ratings is given by the public linkability. To revoke a single rating its author can generate a second, and hence linkable, rating for the same product. The higher-level application can then treat linkable ratings as revoked ones. A drawback of this solution is that the only party with the ability to selectively revoke ratings is the rating's author, since he is the only party knowing the user secret key usk_i.

7.3 Attribute-based Ratings

In the reputation systems defined in Chapter 4 and Chapter 6 ratings ensure that an anonymous user is registered and purchased the rated product. But it also might be useful to have ratings expressing different properties about the rater. To give an example this could be „the author of this rating is at least 30 years old and is either a mathematician or a magician“. Such statements can be expressed with *attribute-based signatures*.

Attribute-based signatures, introduced by Maji, Prabhakaran, and Rosulek [MPR11], extend the functionality of group signatures. But instead of managing a single group of signers based on their identity, in an attribute-based signature scheme signers obtain secret keys based on their attributes. These keys allow to sign messages with respect to a given policy, if and only if the encoded attributes satisfy the policy. The resulting signature hides the used attributes and any identifying information about the signer. Therefore, it should be possible to combine reputation systems with attribute-based signatures without affecting anonymity. Indeed, it is known how to construct universally composable attribute-based signatures [BEJ18a], so the only problem needing further investigation is how to make them compatible with public linkability.

8 Security Proofs

The first section of this chapter is based on our paper „Anonymous and Publicly Linkable Reputation Systems“, presented at the International Conference on Financial Cryptography and Data Security 2015 [BJK15], whereas the second section of this chapter is based on our paper „Practical, Anonymous, and Publicly Linkable Universally-Composable Reputation Systems“, presented at the Cryptographers' Track of the RSA Conference 2018 [BEJ18].

In this chapter we present the proofs of anonymity, public linkability, traceability and strong exculpability for the reputation system from Section 4.2.2 and the proof that the protocol Π_{RS} UC-realizes the ideal functionality $\mathcal{F}_{\mathsf{RS}}$.

8.1 Experiment-based Security

In this section we give the proofs of security for the construction defined in Section 4.2.2. In all proofs we model the hash functions $\mathcal{H}$ and $\mathcal{H}_1$ as random oracles and assume, without loss of generality, that an adversary $\mathcal{A}$ never queries an oracle twice with the same input. We further estimate the number of oracle queries from $\mathcal{A}$, which is polynomially bounded since $\mathcal{A}$ is a probabilistic polynomial-time algorithm.

For the proof of anonymity we need the following lemma.

Lemma 8.1:
Let $\mathbb{G}$ be a multiplicative group of prime order p and let $G \coloneqq (u, v, w, u^a, v^b, w^c) \in \mathbb{G}^6$ be an instance of the Decision Linear Problem in $\mathbb{G}$. Then we can construct another instance $H \coloneqq (r, s, t, r^d, s^e, t^f) \in \mathbb{G}^6$ that is independent of G but has the same probability distribution as G. ◇

Proof. Choose the values $\alpha, \beta, \gamma, \delta, \varepsilon, \varphi \xleftarrow{\$} \mathbb{Z}_p$ and set

$$r \coloneqq u^\alpha \quad s \coloneqq v^\beta \quad t \coloneqq w^\gamma \quad r^d \coloneqq (u^a)^{\alpha\cdot\varphi} \cdot u^{\alpha\cdot\delta} \quad s^e \coloneqq (v^b)^{\beta\cdot\varphi} \cdot v^{\beta\cdot\varepsilon} \quad t^f \coloneqq (w^c)^{\gamma\cdot\varphi} \cdot w^{\gamma\cdot(\delta+\varepsilon)}$$

With $\alpha, \beta, \gamma, \delta, \varepsilon \xleftarrow{\$} \mathbb{Z}_p$ the first five components of H are distributed uniformly at random. For the exponents d, e and f it holds

$$d = a \cdot \varphi + \delta \qquad e = b \cdot \varphi + \varepsilon \qquad f = c \cdot \varphi + \delta + \varepsilon,$$

hence

$$f = d + e \quad \Leftrightarrow \quad c \cdot \varphi + \delta + \varepsilon = a \cdot \varphi + \delta + b \cdot \varphi + \varepsilon \quad \Leftrightarrow \quad c \cdot \varphi = (a + b) \cdot \varphi,$$

so H is distributed exactly as G. □

The proofs of public linkability, traceability, and strong exculpability use the General Forking Lemma, so we restate it here.

Lemma 2.2: General Forking Lemma - [BN06]
Fix $q \in \mathbb{N}$ and a set H of size $h := |H| \geq 2$. Let **IG** be a probabilistic algorithm, called the *input generator*, and let **A** be a probabilistic algorithm that on input $(x, h_1, \ldots, h_q)$ returns a tuple (J, St), where x is the output of **IG**, $h_1, \ldots, h_q \in H$, $J \in \{0, \ldots, q\}$ and St is some state information. Further let

$$\varepsilon_{\mathbf{A}} := \Pr[x \leftarrow \mathbf{IG}, h_1, \ldots, h_q \xleftarrow{u} H, (J, \mathrm{St}) \leftarrow \mathbf{A}(x, h_1, \ldots, h_q) : J \geq 1]$$

and the *forking algorithm* $\mathbf{F}_{\mathbf{A}}$ associated to **A** be the probabilistic algorithm that takes x as input and proceeds as follows:

$\mathbf{F}_{\mathbf{A}}(x)$
1: Pick random bits ω for **A**
2: $h_1, \ldots, h_q \xleftarrow{u} H$
3: $(I, \mathrm{St}) \leftarrow \mathbf{A}(x, h_1, \ldots, h_q; \omega)$
4: **If** $I = 0$ **Then** return $(0, \perp, \perp)$
5: $h'_I, \ldots, h'_q \xleftarrow{u} H$
6: $(I', \mathrm{St}') \leftarrow \mathbf{A}(x, h_1, \ldots, h_{I-1}, h'_I, \ldots, h'_q; \omega)$
7: **If** $I = I' \wedge h_I \neq h'_I$ **Then** return $(1, \mathrm{St}, \mathrm{St}')$
8: **Else** return $(0, \perp, \perp)$

Define the success probability of $\mathbf{F}_{\mathbf{A}}$ as

$$\varepsilon_{\mathbf{F}_{\mathbf{A}}} := \Pr[x \leftarrow \mathbf{IG}, (b, \mathrm{St}, \mathrm{St}') \leftarrow \mathbf{F}_{\mathbf{A}}(x) \colon b = 1].$$

Then it holds

$$\varepsilon_{\mathbf{F}_{\mathbf{A}}} \geq \varepsilon_{\mathbf{A}} \cdot \left(\frac{\varepsilon_{\mathbf{A}}}{q} - \frac{1}{h} \right). \qquad \diamond$$

8.1.1 Proof of Anonymity

As mentioned in Section 4.1.2 the anonymity experiment as defined in Figure 4.5 can be relaxed to CPA-anonymity. We will prove security in this slightly weaker model.

Lemma 4.5:
If the Decision Linear Assumption holds for bilinear group generator BiGrGen, the reputation system defined in Section 4.2.2 is CPA-anonymous. ◇

Proof of Lemma 4.5. Algorithm $\mathcal{B}$ is given a group description of Type-2 bilinear groups $\mathbb{GD} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathrm{e}, \phi, g_1, g_2)$ generated by a bilinear group generator $\mathsf{BiGrGen}$ on input 1^λ, where $|p| = \lambda$, and an instance of the Decision Linear Problem $(\hat{u}, \hat{v}, \hat{w}, \hat{u}^{\hat{a}}, \hat{v}^{\hat{b}}, \hat{w}^{\hat{c}}) \in \mathbb{G}_2^6$. Algorithm $\mathcal{B}$ has to decide whether $\hat{c} = \hat{a} + \hat{b}$ holds.

Suppose $\mathcal{A}$ is an adversary that wins the anonymity experiment with probability $\frac{1}{2} + \varepsilon$, queries the random oracle $\mathcal{H}$ at most $q_{\mathcal{H}}$ times, the random oracle $\mathcal{H}_1$ at most $q_{\mathcal{H}_1}$ times, and the oracle AddR at most q_{AddR} times. Then algorithm $\mathcal{B}$ can interact with $\mathcal{A}$ to decide the Decision Linear Problem with advantage $\varepsilon \cdot \left(\frac{1}{q_{\mathsf{AddR}} \cdot q_{\mathcal{H}_1}} - \frac{q_{\mathcal{H}}^2}{p}\right)$.

At first, $\mathcal{B}$ sets $\mathcal{HR} := \emptyset$, $\mathcal{CR} := \emptyset$, $\mathcal{RL} := \emptyset$, $\mathit{ItemList} := \emptyset$, $\mathit{reg} := \emptyset$, $\mathcal{RR} := \emptyset$, $\mathcal{JIR} := \emptyset$, $\mathcal{QR} := \emptyset$, and computes a second, independent instance of the Decision Linear Problem $(\hat{h}, \hat{d}, \hat{f}, \hat{h}^{y^*}, \hat{d}^{\mu^*}, \hat{f}^{r^*})$ by applying Lemma 8.1. To compute the system manager's public key $\mathcal{B}$ sets $\mathit{smpk} := (u := \phi(\hat{u}), v := \phi(\hat{v}), w := \phi(\hat{w}), h := \phi(\hat{h}), d := \phi(\hat{d}), \hat{d})$ and gives $\mathbb{GD}$ and smpk to $\mathcal{A}$.

Albeit $\mathcal{A}$ has to output two identities of honest raters in his anonymity challenge, it suffices for $\mathcal{B}$ to guess one identity, namely the one that is used in the challenge rating. Identities for honest raters are created using the AddR oracle, and there are at most q_{AddR} queries to this oracle. Hence, $\mathcal{B}$ chooses $\ell_1 \xleftarrow{u} \{1, \ldots, q_{\mathsf{AddR}}\}$ as his guess which identity will be used for $\mathcal{A}$'s challenge. Analogously, $\mathcal{B}$ has to guess for which *item* $\mathcal{A}$ wants to be challenged. For every *item* the hash value $\mathcal{H}_1(\mathit{item})$ is needed to create a rating, and there are at most $q_{\mathcal{H}_1}$ (different) queries to $\mathcal{H}_1$. Hence, $\mathcal{B}$ chooses $\ell_2 \xleftarrow{u} \{1, \ldots, q_{\mathcal{H}_1}\}$ as his guess that the ℓ_2'th query to $\mathcal{H}_1$ is for the *item* that $\mathcal{A}$ wants to be challenged on.

Then $\mathcal{A}$ starts to interact with $\mathcal{B}$ via the oracles. Algorithm $\mathcal{B}$ responds to oracle queries by running exactly the defined oracles from Figure 4.3 and Figure 4.4, except to queries to $\mathsf{AddR}, \mathsf{RSK}, \mathsf{RevR}, \mathsf{GRate}$ and the random oracles $\mathcal{H}$ and $\mathcal{H}_1$. These oracles are realized as follows:

$\mathcal{H}(\cdot)$: $\mathcal{B}$ chooses $c \xleftarrow{u} \mathbb{Z}_p$, gives c to $\mathcal{A}$, and ensures to respond identically to repeated queries.

$\mathcal{H}_1(\mathit{item})$: For the j'th query, $j \neq \ell_2$, $\mathcal{B}$ chooses $r_j \xleftarrow{u} \mathbb{Z}_p$, sets $f' := \hat{h}^{r_j}$ and gives f' to $\mathcal{A}$. To the ℓ_2'th query $\mathcal{B}$ responds by patching the oracle at *item* to match $\hat{f}$ by setting $\mathcal{H}_1(\mathit{item}) := \hat{f}$. Furthermore, $\mathcal{B}$ ensures to respond identically to repeated queries.

$\mathsf{AddR}(i)$: For the j'th query, $j \neq \ell_1$, $\mathcal{B}$ follows the oracle definition of Figure 4.3. To the ℓ_1'th query $\mathcal{B}$ responds by setting $i^* := i$, $\mathit{rpk}[i^*] := \phi(\hat{h}^{y^*})$ and returning $\mathit{rpk}[i^*]$.

$\mathsf{RSK}(i)$: If $i \neq i^*$, $\mathcal{B}$ responds as defined in Figure 4.3. If $i = i^*$, $\mathcal{B}$ cannot respond as $\mathit{rsk}[i^*] = y^*$ is not known. Hence, $\mathcal{B}$ declares failure and exits.

$\mathsf{RevR}(i)$: If $i \neq i^*$, $\mathcal{B}$ responds as defined in Figure 4.3. If $i = i^*$, $\mathcal{B}$ cannot respond as $\mathit{rsk}[i^*] = y^*$ is not known and d^{y^*} can not be computed. Hence, $\mathcal{B}$ declares failure and exits.

$\mathsf{GRate}(i, \mathit{item}, M)$: Algorithm $\mathcal{B}$ has to handle three different cases:

- If $i \neq i^*$, $\mathcal{B}$ responds as defined in Figure 4.4.
- If $i = i^* \wedge \mathcal{H}_1(\mathit{item}) = \hat{f}$, $\mathcal{B}$ declares failure and exits.

- If $i = i^* \wedge \mathcal{H}_1(item) \neq \hat{f}$, $\mathcal{B}$ simulates the rating using the simulator of Lemma 4.3. To do this, $\mathcal{B}$ checks that there exists a public key for the given *item* in the *ItemList*. If not, $\mathcal{B}$ returns an empty string (as defined in the GRate oracle). If rater i does not own a personal rating key $rrsk[i, item] \in IL_{item}$, $\mathcal{B}$ creates one by running the join-issue protocol. Then, $\mathcal{B}$ chooses $\alpha, \beta, \mu \xleftarrow{u} \mathbb{Z}_p$ and computes

$$T_1 := u^{\alpha} \quad T_2 := v^{\beta} \quad T_3 := A_{i^*_{item}} \cdot w^{\alpha+\beta} \quad T_4 := d^{\mu} \quad T_5 := \phi\left((\hat{h}^{y^*})^{r_j} \cdot \hat{h}^{r_j \cdot \mu}\right).$$

  Then $\mathcal{B}$ chooses $c, s_\alpha, s_\beta, s_x, s_{y'}, s_\mu, s_{\delta_1}, s_{\delta_2} \xleftarrow{u} \mathbb{Z}_p$, computes $R_1, \ldots, R_7$ according to the simulator of Lemma 4.3, and patches the random oracle $\mathcal{H}$ by setting $\mathcal{H}(M, item, smpk, ipk[item], T_1, T_2, T_3, T_4, T_5, R_1, R_2, R_3, R_4, R_5, R_6, R_7) := c$. If this causes a collision, $\mathcal{B}$ declares failure and exits. Finally, $\mathcal{B}$ hands the simulated rating $\sigma := (item, T_1, T_2, T_3, T_4, T_5, c, s_\alpha, s_\beta, s_x, s_y, s_\mu, s_{\delta_1}, s_{\delta_2})$ to $\mathcal{A}$.

Eventually, $\mathcal{A}$ outputs a tuple $(i_0, i_1, item, M, \mathrm{St})$. If i_0 or i_1 are not honest raters or the specified *item* does not exist in the *ItemList*, $\mathcal{B}$ declares failure and exits. If $\mathcal{A}$ never queried $\mathcal{H}_1(item)$, $\mathcal{B}$ sets $\mathcal{H}_1(item) := \hat{f}$ and ignores ℓ_2 for future queries. Then $\mathcal{B}$ chooses a bit $b \xleftarrow{u} \{0, 1\}$. If $i_b \neq i^*$ or $\mathcal{H}_1(item) \neq \hat{f}$, $\mathcal{B}$ declares failure and exits. Otherwise, to compute the challenge rating $\mathcal{B}$ sets

$$T_1 := \phi(\hat{u}^{\hat{a}}) \qquad T_2 := \phi(\hat{v}^{\hat{b}}) \qquad T_3 := A_{i^*_{item}} \cdot \phi(\hat{w}^{\hat{c}}) \qquad T_4 := \phi(\hat{d}^{\mu^*}) \qquad T_5 := \phi(\hat{f}^{r^*})$$

and simulates the values $(c, s_\alpha, s_\beta, s_x, s_y, s_\mu, s_{\delta_1}, s_{\delta_2})$ as described in the GRate oracle. If this causes a collision when patching the random oracle $\mathcal{H}$ to equal c, $\mathcal{B}$ declares failure and exits. Otherwise, the challenge rating is given to $\mathcal{A}$.

In the guess-phase $\mathcal{B}$ responds to $\mathcal{A}$'s oracle queries as before. When $\mathcal{A}$ outputs its guess $b' \in \{0, 1\}$, $\mathcal{B}$ outputs 1 as his guess for his Decision Linear challenge, if and only if $i_{b'} = i^*$.

Algorithm $\mathcal{B}$ can compute responses to $\mathcal{A}$'s oracle queries in probabilistic polynomial-time, as required for an adversary against Decision Linear. Now we analyze the advantage of $\mathcal{B}$ in deciding Decision Linear in $\mathbb{G}_2$.

Suppose the Decision Linear instance given to $\mathcal{B}$ is a real Decision Linear tuple, which means that $\hat{c} = \hat{a} + \hat{b}$. In this case, all keys given to $\mathcal{A}$ and all responses to $\mathcal{A}$'s queries are properly distributed. Especially, by Lemma 8.1 it holds $r^* = y^* + \mu^*$, so the challenge rating is a properly distributed and valid rating of user i_b. Hence, $\mathcal{A}$ guesses b correctly with probability $\frac{1}{2} + \varepsilon$.

Suppose the Decision Linear instance given to $\mathcal{B}$ is a random instance, which means that $\hat{c} \xleftarrow{u} \mathbb{Z}_p$. In this case, all keys given to $\mathcal{A}$ and all responses to $\mathcal{A}$'s queries are properly distributed. However, by Lemma 8.1 it holds $r^* \xleftarrow{u} \mathbb{Z}_p$, so the challenge rating is completely independent of i_b. Hence, $\mathcal{A}$ guesses b correctly with probability $\frac{1}{2}$.

If patching the random oracles never fails, $\mathcal{B}$ guesses the correct identity i^* and the correct *item* for the challenge rating, $\mathcal{B}$ will not abort. Patching the random oracles fails with a probability of at most $\frac{q_{\mathcal{H}}^2}{p^{10}}$, since $\mathcal{H}$ takes as input 10 uniformly at random chosen values of $\mathbb{G}_1$ and $\mathbb{Z}_p$ besides additional fixed values. Guessing the correct identity i^* happens with a probability of at least $\frac{1}{q_{\mathsf{AddR}}}$. Analogously, guessing the correct *item* happens with a

probability of at least $\frac{1}{q_{\mathcal{H}_1}}$. Hence, $\mathcal{B}$ outputs a guess for his Decision Linear challenge with a probability of at least $\frac{1}{q_{\mathsf{AddR}} \cdot q_{\mathcal{H}_1}} - \frac{q_{\mathcal{H}}^2}{p^{10}}$. Therefore, it holds

$$\left| \begin{array}{l} \Pr\left[\mathcal{B}(\mathbb{GD}, \hat{u}, \hat{v}, \hat{w}, \hat{u}^{\hat{a}}, \hat{v}^{\hat{b}}, \hat{w}^{\hat{a}+\hat{b}}) = 1\right] \\ -\Pr\left[\mathcal{B}(\mathbb{GD}, \hat{u}, \hat{v}, \hat{w}, \hat{u}^{\hat{a}}, \hat{v}^{\hat{b}}, \hat{w}^{\hat{c}}) = 1\right] \end{array} \right| \geq \left| \begin{array}{r} \left(\frac{1}{q_{\mathsf{AddR}} \cdot q_{\mathcal{H}_1}} - \frac{q_{\mathcal{H}}^2}{p^{10}}\right) \cdot \left(\frac{1}{2} + \varepsilon\right) \\ -\left(\frac{1}{q_{\mathsf{AddR}} \cdot q_{\mathcal{H}_1}} - \frac{q_{\mathcal{H}}^2}{p^{10}}\right) \cdot \frac{1}{2} \end{array} \right|$$
$$= \varepsilon \cdot \left(\frac{1}{q_{\mathsf{AddR}} \cdot q_{\mathcal{H}_1}} - \frac{q_{\mathcal{H}}^2}{p^{10}}\right).$$

Since we assume that the Decision Linear Assumption holds for bilinear group generator $\mathsf{BiGrGen}$, this advantage must be negligible, which implies that ε is negligible. Therefore the reputation system is CPA-anonymous. □

8.1.2 Proof of Public Linkability

Lemma 4.6:
If the Strong Diffie-Hellman Assumption holds for bilinear group generator $\mathsf{BiGrGen}$, the reputation system defined in Section 4.2.2 is publicly linkable. ◇

Proof of Lemma 4.6. At first, we will define the algorithm $\mathcal{B}$ that interacts with an adversary $\mathcal{A}$ against the public linkability experiment defined in Figure 4.5. Then we show how the forking algorithm $\mathbf{F}_{\mathcal{B}}$ associated to $\mathcal{B}$ is invoked to solve the Strong Diffie-Hellman Problem.

Algorithm $\mathcal{B}$ is given a group description $\mathbb{GD} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathrm{e}, \phi, G_1, G_2)$ of Type-2 bilinear groups generated by a bilinear group generator $\mathsf{BiGrGen}$ on input 1^λ, where $|p| = \lambda$, an instance of the SDH Problem $\left(G_2^{\hat{\gamma}}, G_2^{(\hat{\gamma}^2)}, \ldots, G_2^{(\hat{\gamma}^{q(\lambda)})}\right)$, for some function $q\colon \mathbb{Z} \to \mathbb{Z}$, and $q_{\mathcal{H}}$ values $h_1, \ldots, h_{q_{\mathcal{H}}} \in \mathbb{Z}_p$. Note that $q(\lambda)$ is a constant for a fixed security parameter λ.

Suppose $\mathcal{A}$ is an adversary that wins the public linkability experiment with probability ε, queries the random oracle $\mathcal{H}$ at most $q_{\mathcal{H}}$ times, and the oracle $\mathsf{AddItem}$ at most q_{AddItem} times. Without loss of generality we assume that $\mathcal{A}$ creates exactly $q(\lambda) - 1$ raters via the $\mathsf{SndToSM}$ oracle. Then algorithm $\mathcal{B}$ interacts with $\mathcal{A}$ as follows.

At first, $\mathcal{B}$ guesses the *item* for which $\mathcal{A}$ will output the solution to the public linkability experiment. To do so, $\mathcal{B}$ selects $\ell \xleftarrow{u} \{1, \ldots, q_{\mathsf{AddItem}}\}$ and uses the *item* of the ℓ'th query to the $\mathsf{AddItem}$ oracle, denoted by $item^*$.

Then, $\mathcal{B}$ prepares the *item*-based public key $ipk[item^*]$, by choosing $\alpha, x_j, y_j \xleftarrow{u} \mathbb{Z}_p$, for $j = 1, \ldots, q(\lambda) - 1$, $k \xleftarrow{u} \{1, \ldots, q(\lambda) - 1\}$, setting $\gamma := \hat{\gamma} - x_k$, which is unknown, and computing

$$g_2 := G_2^{\alpha \cdot \prod_{i=1}^{q-1} (\hat{\gamma} - x_k + x_i) - y_k \cdot \prod_{i=1, i \neq k}^{q-1} (\hat{\gamma} - x_k + x_i)} \qquad g_1 := \phi(g_2) \tag{8.1}$$

$$\hat{h} := G_2^{\prod_{i=1, i \neq k}^{q-1} (\hat{\gamma} - x_k + x_i)} \qquad h := \phi(\hat{h}) \tag{8.2}$$

$$W \coloneqq \left[G_2^{\alpha \cdot \prod_{i=1}^{q-1} (\hat{\gamma} - x_k + x_i) - y_k \cdot \prod_{i=1, i \neq k}^{q-1} (\hat{\gamma} - x_k + x_i)} \right]^{\hat{\gamma} - x_k} \tag{8.3}$$

$$A_j \coloneqq \phi \left[\left(G_2^{\alpha \cdot \prod_{i=1}^{q-1} (\hat{\gamma} - x_k + x_i) - y_k \cdot \prod_{i=1, i \neq k}^{q-1} (\hat{\gamma} - x_k + x_i)} \cdot G_2^{y_j \cdot \prod_{i=1, i \neq k}^{q-1} (\hat{\gamma} - x_k + x_i)} \right)^{\frac{1}{\gamma + x_j}} \right].$$

All products in the exponents are polynomials of $\hat{\gamma}$ of degree at most $q(\lambda)$. By expanding the products all the specified values can be computed using the given SDH instance. With $\gamma \coloneqq \hat{\gamma} - x_k$, it holds $W = g_2^{\gamma}$, while γ is unknown to $\mathcal{B}$.

To generate the system manager's public key, $\mathcal{B}$ selects $w \xleftarrow{u} \mathbb{G}_1, \zeta, \xi_1, \xi_2 \xleftarrow{u} \mathbb{Z}_p$ and computes

$$u \coloneqq w^{\frac{1}{\xi_1}} \qquad v \coloneqq w^{\frac{1}{\xi_2}} \qquad \hat{d} \coloneqq \hat{h}^{\frac{1}{\zeta}} \qquad d \coloneqq \phi(\hat{d}).$$

Now $\mathcal{B}$ sets $\mathcal{HR} \coloneqq \emptyset$, $\mathcal{CR} \coloneqq \emptyset$, $\mathcal{RL} \coloneqq \emptyset$, *ItemList* $\coloneqq \emptyset$, *reg* $\coloneqq \emptyset$, $\mathcal{RR} \coloneqq \emptyset$, $\mathcal{JIR} \coloneqq \emptyset$, $\mathcal{QR} \coloneqq \emptyset$ and $smpk \coloneqq (u, v, w, h, d, \hat{d})$, hands $\mathbb{GD}$ and $smpk$ to $\mathcal{A}$ and starts to interact with $\mathcal{A}$ via the oracles. $\mathcal{A}$'s queries are answered as follows:

- $\boldsymbol{\mathcal{H}(\cdot)}$: To the i'th query $\mathcal{B}$ responds with $h_i \in \mathbb{Z}_p$, that was given as input to $\mathcal{B}$, and ensures to respond identically to repeated queries.
- $\boldsymbol{\mathcal{H}_1(item)}$: $\mathcal{B}$ chooses $\hat{f} \xleftarrow{u} \mathbb{G}_2$, gives $\hat{f}$ to $\mathcal{A}$ and ensures to respond identically to repeated queries.
- **AddItem**$\boldsymbol{(item)}$: For the j'th query, $j \neq \ell$, $\mathcal{B}$ executes exactly the oracle defined in Figure 4.4. To the ℓ'th query $\mathcal{B}$ responds by setting $item^* \coloneqq item$ and $ipk[item^*] \coloneqq (g_1, g_2, W)$, adding $ipk[item^*]$ to the *ItemList* and returning $ipk[item^*]$.
- **SndToSM**$\boldsymbol{(i, M_{SM})}$: If the public and secret key of rater i are not set, $\mathcal{B}$ sets $rsk[i] \coloneqq y_j$ and $rpk[i] \coloneqq h^{y_j}$ for the j'th query. Then $\mathcal{B}$ executes the oracle defined in Figure 4.4.
- **SndToP**$\boldsymbol{(i, item, rpk[i], M_P)}$: $\mathcal{B}$ executes exactly the oracle defined in Figure 4.4. If $item = item^*$, prior to the oracle execution $\mathcal{B}$ sets $rrsk[i, item^*] \coloneqq (A_j, x_j)$, where A_j and x_j correspond to user i's secret key $rsk[i] = y_j$ defined in the SndToSM oracle for user i.

Eventually, $\mathcal{A}$ outputs an *item* and exactly $q(\lambda)$ ratings $((m_1, \sigma_1), \ldots, (m_{q(\lambda)}, \sigma_{q(\lambda)}))$. If $item \neq item^*$, at least one rating is invalid, or there are at least two publicly linkable ratings, $\mathcal{B}$ declares failure and exits. Otherwise, $\mathcal{B}$ computes the revocation tokens $rrt[i] = d^{y_i}$ for all $q(\lambda) - 1$ system members, adds them to the revocation list $\mathcal{RL}$ and runs the verification algorithm for every rating $(m_i, \sigma_i), i = 1, \ldots, q(\lambda)$. Since the ratings are not publicly linkable, there must be at least one rating (m_{i^*}, σ_{i^*}) such that σ_{i^*} is still a valid rating. Hence, $\mathcal{B}$ outputs (J, St), where J is the index of the query to $\mathcal{H}$ needed to compute σ_{i^*} and $\mathrm{St} \coloneqq \big(\alpha, \{A_i, x_i, y_i\}_{i=1}^{q(\lambda)-1}, k, smpk, item^*, ipk[item^*], (m_{i^*}, \sigma_{i^*})\big)$. If $\mathcal{A}$ never queried the random oracle for σ_{i^*}, $\mathcal{B}$ sets $J \coloneqq 0$ and $\mathrm{St} \coloneqq \perp$.

Since $\mathcal{B}$ perfectly simulates the public linkability experiment, $\mathcal{A}$ succeeds with probability ε. The probability that $\mathcal{A}$ guesses the hash value for σ_{i^*} correctly is $\frac{1}{p}$ and the probability that

$\mathcal{B}$ guesses the correct *item* for which $\mathcal{A}$ outputs its solution is $\frac{1}{q_{\mathsf{AddItem}}}$. Hence, $\mathcal{B}$ outputs (J, St), where $J \geq 1$, with a probability of at least $\frac{\varepsilon}{q_{\mathsf{AddItem}}} - \frac{1}{p}$.

The transformation of $\mathcal{B}$'s SDH instance into $q(\lambda) - 1$ tuples (A_j, x_j, y_j) and an *item*-based public key can be done in probabilistic polynomial-time. The same holds for the computations needed to respond to $\mathcal{A}$'s oracle queries and for the computation of (A, x) using the extracted values (A^*, x^*, y^*). Hence, $\mathcal{B}$ is a probabilistic polynomial-time algorithm.

Now we apply the General Forking Lemma by executing $\mathbf{F}_{\mathcal{B}}$. With probability $\boldsymbol{\varepsilon}_{\mathbf{F}_{\mathcal{B}}}$ it returns $(1, \mathrm{St}, \mathrm{St}')$, where $\mathrm{St} = (\alpha, \{A_i, x_i, y_i\}_{i=1}^{q(\lambda)-1}, k, \mathit{smpk}, \mathit{item}^*, \mathit{ipk}[\mathit{item}^*], (m_{i^*}, \sigma_{i^*}))$ and $\mathrm{St}' = (\alpha, \{A_i, x_i, y_i\}_{i=1}^{q(\lambda)-1}, k, \mathit{smpk}, \mathit{item}^*, \mathit{ipk}[\mathit{item}^*], (m'_{i^*}, \sigma'_{i^*}))$. Then we can execute the extractor of Lemma 4.4 on $(\mathbb{GD}, (m_{i^*}, \sigma_{i^*}), (m'_{i^*}, \sigma'_{i^*}))$ to obtain the eSDH tuple (A^*, x^*, y^*), which can be transformed into a solution to the given SDH problem. Observe that A^* must be of the form

$$
\begin{aligned}
A^* &= \phi\left[\left(G_2^{\alpha\cdot\prod_{i=1}^{q-1}(\hat{\gamma}-x_k+x_i)-y_k\cdot\prod_{i=1,i\neq k}^{q-1}(\hat{\gamma}-x_k+x_i)} \cdot G_2^{y^*\cdot\prod_{i=1,i\neq k}^{q-1}(\hat{\gamma}-x_k+x_i)}\right)^{\frac{1}{\gamma+x^*}}\right] \\
&= \phi\left[\left(G_2^{\alpha\cdot\prod_{i=1}^{q-1}(\hat{\gamma}-x_k+x_i)+(y^*-y_k)\cdot\prod_{i=1,i\neq k}^{q-1}(\hat{\gamma}-x_k+x_i)}\right)^{\frac{1}{\hat{\gamma}-x_k+x^*}}\right]. \qquad (8.4)
\end{aligned}
$$

Let $f(X) \coloneqq \prod_{i=1}^{q-1}(X - x_k + x_i)$ and $g(X) \coloneqq \prod_{i=1,i\neq k}^{q-1}(X - x_k + x_i)$. Then

$$
f(\hat{\gamma}) \cdot \frac{1}{\hat{\gamma} - x_k + x^*} = \tau(\hat{\gamma}) + \frac{\beta}{\hat{\gamma} - x_k + x^*}
$$

and

$$
g(\hat{\gamma}) \cdot \frac{1}{\hat{\gamma} - x_k + x^*} = \tau'(\hat{\gamma}) + \frac{\beta'}{\hat{\gamma} - x_k + x^*}
$$

holds for some polynomials τ and τ' of degree at most $q-2$, where β or β' equals 0, if and only if $(\hat{\gamma} - x_k + x^*)$ is a factor of $f(\hat{\gamma})$ or $g(\hat{\gamma})$. These cases will be discussed later. Using this representation we obtain

$$
A^* = \phi\left[G_2^{\alpha\cdot\left(\tau(\hat{\gamma})+\frac{\beta}{\hat{\gamma}-x_k+x^*}\right)+(y^*-y_k)\cdot\left(\tau'(\hat{\gamma})+\frac{\beta'}{\hat{\gamma}-x_k+x^*}\right)}\right]
$$

and we can define

$$
\begin{aligned}
A &\coloneqq \phi\left[\left(\frac{G_2^{\alpha\cdot\left(\tau(\hat{\gamma})+\frac{\beta}{\hat{\gamma}-x_k+x^*}\right)+(y^*-y_k)\cdot\left(\tau'(\hat{\gamma})+\frac{\beta'}{\hat{\gamma}-x_k+x^*}\right)}}{G_2^{\alpha\cdot\tau(\hat{\gamma})+(y^*-y_k)\cdot\tau'(\hat{\gamma})}}\right)^{\frac{1}{\alpha\cdot\beta+(y^*-y_k)\cdot\beta'}}\right] \\
&= \phi\left[\left(\frac{G_2^{\alpha\cdot\tau(\hat{\gamma})+\alpha\cdot\frac{\beta}{\hat{\gamma}-x_k+x^*}+(y^*-y_k)\cdot\tau'(\hat{\gamma})+(y^*-y_k)\cdot\frac{\beta'}{\hat{\gamma}-x_k+x^*}}}{G_2^{\alpha\cdot\tau(\hat{\gamma})+(y^*-y_k)\cdot\tau'(\hat{\gamma})}}\right)^{\frac{1}{\alpha\cdot\beta+(y^*-y_k)\cdot\beta'}}\right]
\end{aligned}
$$

$$
\begin{aligned}
&= \phi\left[\left(G_2^{\alpha\cdot\frac{\beta}{\hat{\gamma}-x_k+x^*}+(y^*-y_k)\cdot\frac{\beta'}{\hat{\gamma}-x_k+x^*}}\right)^{\frac{1}{\alpha\cdot\beta+(y^*-y_k)\cdot\beta'}}\right] \\
&= \phi\left[\left(G_2^{\frac{1}{\hat{\gamma}-x_k+x^*}\cdot(\alpha\cdot\beta+(y^*-y_k)\cdot\beta')}\right)^{\frac{1}{\alpha\cdot\beta+(y^*-y_k)\cdot\beta'}}\right] \\
&= \phi\left[G_2^{\frac{1}{\hat{\gamma}-x_k+x^*}}\right].
\end{aligned}
$$

Hence, $(A, x^* - x_k)$ is a solution to the SDH problem instance that can be computed because all necessary information is included in St and St′ from $\mathcal{B}$.

Now we discuss the different cases that can occur during the described transformation.

Case 1: $(A^*, x^*, y^*) \in \{(A_j, x_j, y_j)\}_{j=1}^{q-1}$: Obviously, if (A^*, x^*, y^*) is one of the triples $\mathcal{B}$ generated himself, no new information is obtained from (A^*, x^*, y^*). Hence, $\mathcal{B}$ cannot compute A and has to abort.

Case 2: $x^* \notin \{x_j\}_{j=1}^{q-1}$: In this case the values β and β' are not equal to 0. Hence, the value A can be computed as described above and $(A, x^* - x_k)$ is a solution to $\mathcal{B}$'s SDH problem.

Case 3: $x^* \in \{x_j\}_{j=1}^{q-1}$: This case has to be divided into two different subcases:

a) $x^* \neq x_k$: Since x^* is equal to x_j for some $j \neq k$, $(\hat{\gamma} - x_k + x_j)$ is a factor of both polynomials $f(\hat{\gamma})$ and $g(\hat{\gamma})$. Hence, it holds $\beta = \beta' = 0$, A cannot be computed and $\mathcal{B}$ has to abort.

b) $x^* = x_k$: In this case $(\hat{\gamma} - x_k + x^*) = \hat{\gamma}$ is a factor of $f(\hat{\gamma})$, but not one of $g(\hat{\gamma})$. Hence, $\beta = 0$ and $\beta' \neq 0$ holds. Also $y^* \neq y_k$ (because otherwise A^* would be equal to A_k) and $(y^* - y_k) \cdot \beta' \neq 0$ holds, so $(A, 0)$ is a solution to $\mathcal{B}$'s SDH problem.

Because $\mathcal{A}$ was successful in generating unlinkable ratings it holds $y^* \notin \{y_i\}_{i=1}^{q-1}$. Hence, Case 1 does not occur. If Case 2 occurs, we can compute a solution to the SDH problem with probability 1. For Case 3 it holds $\Pr[x^* = x_k] = \frac{1}{q(\lambda)-1}$ and either Case 2 or Case 3 occurs with a probability of at least $\frac{1}{2}$.

Putting all together, assuming the more pessimistic scenario of Case 3, we can compute a solution to the SDH problem with a probability ε' of at least

$$
\varepsilon' \geq \boldsymbol{\varepsilon}_{\mathbf{F}_\mathcal{B}} \cdot \frac{1}{q(\lambda)-1} \cdot \frac{1}{2} \geq \left[\left(\frac{\varepsilon}{q_{\mathsf{AddItem}}} - \frac{1}{p}\right) \cdot \left(\frac{\frac{\varepsilon}{q_{\mathsf{AddItem}}} - \frac{1}{p}}{q_\mathcal{H}} - \frac{1}{p}\right)\right] \cdot \frac{1}{q(\lambda)-1} \cdot \frac{1}{2}.
$$

Since we assume that the Strong Diffie-Hellman Assumption holds for bilinear group generator $\mathsf{BiGrGen}$, the probability ε' must be negligible which implies that $\boldsymbol{\varepsilon}_{\mathbf{F}_\mathcal{B}}$ is negligible and hence also ε. Therefore the reputation system is publicly linkable. □

8.1.3 Proof of Traceability

Lemma 4.7:
If the Strong Diffie-Hellman Assumption holds for bilinear group generator $\mathsf{BiGrGen}$, the reputation system defined in Section 4.2.2 is traceable. ◇

Proof of Lemma 4.7. Analogously to the proof of public linkability, we will apply the General Forking Lemma to compute a solution to a given Strong Diffie-Hellman problem instance. For this purpose we define an algorithm $\mathcal{B}$, that interacts with an adversary $\mathcal{A}$ against traceability, which will be executed by the forking algorithm $\mathbf{F}_{\mathcal{B}}$ to obtain two related ratings.

Algorithm $\mathcal{B}$ is given a group description $\mathbb{GD} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathrm{e}, \phi, G_1, G_2)$ of Type-2 bilinear groups generated by a bilinear group generator $\mathsf{BiGrGen}$ on input 1^λ, where $|p| = \lambda$, an instance of the SDH Problem $\left(G_2^{\hat{\gamma}}, G_2^{(\hat{\gamma}^2)}, \ldots, G_2^{(\hat{\gamma}^{q(\lambda)})}\right)$, for some function $q\colon \mathbb{Z} \to \mathbb{Z}$, and $q_{\mathcal{H}}$ values $h_1, \ldots, h_{q_{\mathcal{H}}} \in \mathbb{Z}_p$. Note that $q(\lambda)$ is a constant for a fixed security parameter λ.

Suppose $\mathcal{A}$ is an adversary that wins the traceability experiment with probability ε, queries the random oracle $\mathcal{H}$ at most $q_{\mathcal{H}}$ times, the oracle $\mathsf{AddItem}$ at most q_{AddItem} times, and the oracle AddR at most q_{AddR} times. Without loss of generality we assume that $\mathcal{A}$ creates exactly $q(\lambda) - 1$ raters via the AddR and $\mathsf{SndToSM}$ oracles. Then algorithm $\mathcal{B}$ can interact with $\mathcal{A}$ as follows.

Analogously to [BBS04] we have to distinguish between two different forger types: the *Type-I forger* outputs a valid rating (m, σ), for some *item* of his choice, such that the Open algorithm outputs `failure`; the *Type-II forger* outputs a valid rating (m, σ), for some *item* of his choice, that can be traced back to an honest rater. Hence, $\mathcal{B}$ guesses the forger type by choosing $b \xleftarrow{u} \{I, II\}$ and behaves slightly different in the two cases.

At first, $\mathcal{B}$ transforms the SDH problem instance into tuples (A_j, x_j, y_j), for $j = 1, \ldots, q(\lambda) - 1$, and values $(g_1, g_2, \hat{h}, h, W)$ using the same technique as in the proof of public linkability (Lemma 4.6). Then $\mathcal{B}$ guesses an *item** as the *item* for which $\mathcal{A}$ will output the rating as its solution to the traceability experiment by choosing $\ell \xleftarrow{u} \{1, \ldots, q_{\mathsf{AddItem}}\}$ and handling the ℓ'th query to the $\mathsf{AddItem}$ oracle appropriately. In the case that $b = II$, $\mathcal{B}$ selects $A_q \xleftarrow{u} \mathbb{G}_1$ and $y_q \xleftarrow{u} \mathbb{Z}_p$, sets $x_q := \star$ and guesses the honest user i^* for which $\mathcal{A}$ will output the forged rating σ by choosing $\ell_1 \xleftarrow{u} \{1, \ldots, q_{\mathsf{AddR}}\}$ and handling the ℓ_1'th query to the AddR oracle appropriately.

To generate the system manager's public key, $\mathcal{B}$ selects $w \xleftarrow{u} \mathbb{G}_1, \zeta, \xi_1, \xi_2 \xleftarrow{u} \mathbb{Z}_p$ and computes

$$u := w^{\frac{1}{\xi_1}} \qquad v := w^{\frac{1}{\xi_2}} \qquad \hat{d} := \hat{h}^{\frac{1}{\zeta}} \qquad d := \phi(\hat{d}).$$

Now $\mathcal{B}$ sets $\mathcal{HR} := \emptyset$, $\mathcal{CR} := \emptyset$, $\mathcal{RL} := \emptyset$, *ItemList* $:= \emptyset$, *reg* $:= \emptyset$, $\mathcal{RR} := \emptyset$, $\mathcal{JIR} := \emptyset$, $\mathcal{QR} := \emptyset$ and $\mathit{smpk} := (u, v, w, h, d, \hat{d})$, hands the group description $\mathbb{GD}$ and the system manager's public key *smpk* to $\mathcal{A}$ and starts to interact with $\mathcal{A}$ via the oracles. The number of already registered raters is counted using the variable $\hat{j}$, which is initially set to 0. $\mathcal{A}$'s queries are answered as follows:

$\mathcal{H}(\cdot)$: To the i'th query $\mathcal{B}$ responds with $h_i \in \mathbb{Z}_p$, that was given as input to $\mathcal{B}$, and ensures to respond identically to repeated queries.

$\mathcal{H}_1(item)$: $\mathcal{B}$ chooses $\hat{f} \leftarrow_u \mathbb{G}_2$, gives $\hat{f}$ to $\mathcal{A}$ and ensures to respond identically to repeated queries.

AddItem($item$): For the j'th query, $j \neq \ell$, $\mathcal{B}$ executes exactly the oracle as defined in Figure 4.4. To the ℓ'th query $\mathcal{B}$ responds by setting $item^* := item$ and $ipk[item^*] := (g_1, g_2, W)$, adding $ipk[item^*]$ to the *ItemList* and returning $ipk[item^*]$.

AddR(i): If $b = I$, $\mathcal{B}$ sets $\hat{j} := \hat{j} + 1$ and executes the oracle defined in Figure 4.3, using $y_{\hat{j}}$ as the secret key of rater i. The same is done in case $b = II$, except for the ℓ_1'th query, where $\mathcal{B}$ sets $i^* := i$, $rsk[i^*] := y_q$, $rpk[i^*] := h^{y_q}$, $reg[i^*] := (i^*, rpk[i^*])$ and returns $rpk[i^*]$.

SndToSM(i, M_{SM}): $\mathcal{B}$ sets $\hat{j} := \hat{j} + 1$ and executes the oracle defined in Figure 4.4, using $y_{\hat{j}}$ as the secret key of rater i.

RSK(i): If $b = II \wedge i = i^*$, $\mathcal{B}$ declares failure and exits. Otherwise, $\mathcal{B}$ responds by running exactly the defined oracle from Figure 4.3.

RRSK($i, item$): $\mathcal{B}$ responds by running exactly the defined oracle from Figure 4.3.

RevR(i): $\mathcal{B}$ responds by running exactly the defined oracle from Figure 4.3.

SndToP($i, item, rpk[i], M_P$): $\mathcal{B}$ responds by running the oracle defined in Figure 4.4. In the case that $item = item^*$, $\mathcal{B}$ runs the oracle using $rrsk[i, item^*] := (A_j, x_j)$, where A_j and x_j correspond to rater i's secret key $rsk[i] = y_j$ defined in the AddR or SndToSM oracle for rater i (note that in case $b = II$ for $i = i^*$ the tuple $(A_q, x_q = \star)$ is used).

Open($item, m, \sigma$): $\mathcal{B}$ responds by running the oracle defined in Figure 4.3.

GRate($i, item, m$): $\mathcal{B}$ has to handle two different cases:

- If $b = II \wedge i = i^* \wedge item = item^*$, $\mathcal{B}$ simulates the rating using the simulator of Lemma 4.3: $\mathcal{B}$ chooses $\alpha, \beta, \mu \leftarrow_u \mathbb{Z}_p$ and computes

  $$T_1 := u^\alpha \quad T_2 := v^\beta \quad T_3 := A_{i^*_{item^*}} \cdot w^{\alpha+\beta} \quad T_4 := d^\mu \quad T_5 := \phi(\mathcal{H}_1(item^*))^{\mu + y_{i^*}}.$$

  Then $\mathcal{B}$ chooses $c, s_\alpha, s_\beta, s_x, s_y, s_\mu, s_{\delta_1}, s_{\delta_2} \leftarrow_u \mathbb{Z}_p$, computes $R_1, \ldots, R_7$ according to the simulator of Lemma 4.3, and patches the random oracle $\mathcal{H}$ by setting $\mathcal{H}(M, item^*, smpk, ipk[item^*], T_1, T_2, T_3, T_4, T_5, R_1, R_2, R_3, R_4, R_5, R_6, R_7) := c$. When this causes a collision, $\mathcal{B}$ declares failure and exits. Finally, $\mathcal{B}$ hands the simulated rating $\sigma := (item^*, T_1, T_2, T_3, T_4, T_5, c, s_\alpha, s_\beta, s_x, s_y, s_\mu, s_{\delta_1}, s_{\delta_2})$ to $\mathcal{A}$.

- In any other case $\mathcal{B}$ responds as defined in Figure 4.4. If for an honest rater i the rating key $rrsk[i, item]$ is not set, such a key can be generated before the signature is created, as defined in the SndToP oracle. If $item = item^*$ the tuple (A_j, x_j, y_j) corresponding to $rsk[i] = y_j$ is used as secret rating key.

Eventually, $\mathcal{A}$ outputs a triple $(item, m, \sigma)$. If $item \neq item^*$ or the rating is invalid, verified with an empty revocation list $\mathcal{RL}$, $\mathcal{B}$ declares failure and exits. Otherwise, depending on $b \in \{I, II\}$, $\mathcal{B}$ has to distinguish two different cases:

- $b = I$: If σ opens to some $A^* \in \{A_j\}_{j=1}^{q(\lambda)-1}$, $\mathcal{B}$ declares failure and exits. Otherwise, $\mathcal{A}$ successfully forged a rating for a non-existing rater.

- $b = II$: If σ opens to some $A^* \neq A_q$, $\mathcal{B}$ declares failure and exits. Otherwise, $\mathcal{A}$ successfully forged a rating for rater i^*.

For a *Type-I forger* the environment is simulated perfectly, because $\mathcal{B}$ knows all secret and public keys. Hence, $\mathcal{B}$ is always able to compute correct and properly distributed responses to $\mathcal{A}$'s queries and $\mathcal{A}$ outputs a valid forgery $(item^*, m^*, \sigma^*)$, for the guessed $item^*$, with a probability of at least $\frac{\varepsilon}{q_{\mathsf{AddItem}}}$. With a probability of $\frac{1}{p}$ adversary $\mathcal{A}$ guesses the hash value for σ^* correctly. Hence, $\mathcal{B}$ is successful with a probability $\boldsymbol{\varepsilon}_{\mathcal{B}}$ of at least $\frac{\varepsilon}{q_{\mathsf{AddItem}}} - \frac{1}{p}$.

For a *Type-II forger* the environment is simulated perfectly unless $\mathcal{A}$ queries the RSK oracle for user i^* or a collision occurs while simulating a rating. Hence, $\mathcal{A}$ outputs a valid forgery $(item^*, m^*, \sigma^*)$, for the guessed $item^*$, that traces to i^* with a probability of at least $\frac{\varepsilon}{q_{\mathsf{AddItem}} \cdot q_{\mathsf{AddR}}} - \frac{q_{\mathcal{H}}^2}{p^{10}} - \frac{1}{p}$.

For both forger types $\mathcal{B}$ outputs (J, St), where J is the index of the query to $\mathcal{H}$ needed to compute σ^* and $\mathrm{St} \coloneqq \big(\alpha, \{A_i, x_i, y_i\}_{i=1}^{q(\lambda)-1}, k, smpk, item^*, ipk[item^*], (m_{i^*}, \sigma_{i^*})\big)$. If $\mathcal{A}$ never queried the random oracle for σ^*, $\mathcal{B}$ sets $J \coloneqq 0$ and $\mathrm{St} \coloneqq \perp$.

As shown in Lemma 4.6, the transformation of $\mathcal{B}$'s SDH instance and the computations needed to respond to $\mathcal{A}$'s oracle queries can be done in probabilistic polynomial-time. Hence, $\mathcal{B}$ is a probabilistic polynomial-time algorithm.

Now we apply the General Forking Lemma to obtain a second solution to the traceability experiment which can be used to solve SDH. The needed technique is exactly the same as in the proof of public linkability in Lemma 4.6. That means, with $\boldsymbol{\varepsilon}_{\mathcal{B}} = \frac{\varepsilon}{q_{\mathsf{AddItem}}} - \frac{1}{p}$ for a *Type-I forger* and $\boldsymbol{\varepsilon}_{\mathcal{B}} = \frac{\varepsilon}{q_{\mathsf{AddItem}} \cdot q_{\mathsf{AddR}}} - \frac{q_{\mathcal{H}}^2}{p^{10}} - \frac{1}{p}$ for a *Type-II forger*, we obtain an SDH tuple (A^*, x^*, y^*) with probability $\boldsymbol{\varepsilon}_{\mathbf{F}_{\mathcal{B}}}$ of at least $\boldsymbol{\varepsilon}_{\mathcal{B}} \cdot \left(\frac{\boldsymbol{\varepsilon}_{\mathcal{B}}}{q_{\mathcal{H}}} - \frac{1}{p}\right)$. This tuple is not one of the tuples $\mathcal{B}$ created, because otherwise the forger $\mathcal{A}$ would not be successful. Hence, either Case 2 or Case 3 described in Lemma 4.6 occurs with a probability of at least $\frac{1}{2}$. Moreover, $\mathcal{B}$ guesses the correct forger type with a probability of at least $\frac{1}{2}$ and we can compute a solution (A, x) to the SDH problem with a probability ε' of at least

$$\begin{aligned}
\varepsilon' &\geq \boldsymbol{\varepsilon}_{\mathbf{F}\mathcal{B}} \cdot \frac{1}{q(\lambda) - 1} \cdot \frac{1}{2} \cdot \frac{1}{2} \geq \boldsymbol{\varepsilon}_{\mathcal{B}} \cdot \left(\frac{\boldsymbol{\varepsilon}_{\mathcal{B}}}{q_{\mathcal{H}}} - \frac{1}{p}\right) \cdot \frac{1}{4 \cdot q(\lambda) - 4} \\
&= \left[\left(\frac{\varepsilon}{q_{\mathsf{AddItem}} \cdot q_{\mathsf{AddR}}} - \frac{q_{\mathcal{H}}^2}{p^{10}} - \frac{1}{p}\right) \cdot \left(\frac{\frac{\varepsilon}{q_{\mathsf{AddItem}} \cdot q_{\mathsf{AddR}}} - \frac{q_{\mathcal{H}}^2}{p^{10}} - \frac{1}{p}}{q_{\mathcal{H}}} - \frac{1}{p}\right)\right] \cdot \frac{1}{4 \cdot q(\lambda) - 4}
\end{aligned}$$

assuming the more pessimistic scenario of Case 3 and a *Type-II forger*.

Since we assume that the Strong Diffie-Hellman Assumption holds for bilinear group generator $\mathsf{BiGrGen}$, the probability ε' must be negligible which implies that ε is negligible. Therefore, the reputation system is traceable. □

8.1.4 Proof of Strong Exculpability

Lemma 4.8:
If the Discrete Logarithm Assumption holds for bilinear group generator $\mathsf{BiGrGen}$, the reputation system defined in Section 4.2.2 is strongly exculpable. ◇

Proof of Lemma 4.8. As in the proofs of public linkability and traceability, we will apply the General Forking Lemma, but for strong exculpability we will compute a solution to the Discrete Logarithm Problem. For this purpose we define an algorithm $\mathcal{B}$, that interacts with an adversary $\mathcal{A}$ against strong exculpability, which will be executed by the forking algorithm $\mathbf{F}_{\mathcal{B}}$ to obtain two related ratings.

Algorithm $\mathcal{B}$ is given a group description $\mathbb{GD} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \mathrm{e}, \phi, G_1, G_2)$ of Type-2 bilinear groups generated by a bilinear group generator $\mathsf{BiGrGen}$ on input 1^λ, where $|p| = \lambda$, an instance of the Discrete Logarithm Problem $(\hat{h}, \mathcal{D}) \in \mathbb{G}_2$, and $q_{\mathcal{H}}$ values $h_1, \ldots, h_{q_{\mathcal{H}}} \in \mathbb{Z}_p$.

Suppose $\mathcal{A}$ is an adversary that wins the strong exculpability experiment with probability ε, queries the random oracle $\mathcal{H}$ at most $q_{\mathcal{H}}$ times and the oracle AddR at most q_{AddR} times. Then algorithm $\mathcal{B}$ can interact with $\mathcal{A}$ as follows.

$\mathcal{B}$ sets $\mathcal{HR} \coloneqq \emptyset$, $\mathcal{CR} \coloneqq \emptyset$, $\mathcal{RL} \coloneqq \emptyset$, $\mathit{ItemList} \coloneqq \emptyset$, $\mathit{reg} \coloneqq \emptyset$, $\mathcal{RR} \coloneqq \emptyset$, $\mathcal{JIR} \coloneqq \emptyset$, $\mathcal{QR} \coloneqq \emptyset$ and guesses the rater i^* for which the adversary $\mathcal{A}$ will output a rating σ as its solution to the strong exculpability experiment by choosing $\ell \xleftarrow{\$} \{1, \ldots, q_{\mathsf{AddR}}\}$ and handling the ℓ'th query to the AddR oracle appropriately.

To generate the system manager's public key, $\mathcal{B}$ selects $w \xleftarrow{\$} \mathbb{G}_1$, $\xi_1, \xi_2, \zeta \xleftarrow{\$} \mathbb{Z}_p$ and computes

$$u \coloneqq w^{\frac{1}{\xi_1}} \qquad v \coloneqq w^{\frac{1}{\xi_2}} \qquad h \coloneqq \phi(\hat{h}) \qquad \hat{d} \coloneqq \hat{h}^{\frac{1}{\zeta}} \qquad d \coloneqq \phi(\hat{d}).$$

Now $\mathcal{B}$ sets $\mathit{smpk} \coloneqq (u, v, w, h, d, \hat{d})$ and $\mathit{smsk} \coloneqq (\xi_1, \xi_2, \zeta)$ as the system manager's public and secret keys, hands the group description $\mathbb{GD}$, and the system manager's public key smpk to $\mathcal{A}$, and starts to interact with $\mathcal{A}$ via the oracles. Algorithm $\mathcal{B}$ responds to oracle queries by running exactly the defined oracles from Figure 4.3 and Figure 4.4, except to queries to $\mathsf{AddR}, \mathsf{RSK}, \mathsf{RevR}, \mathsf{GRate}$, and the random oracles $\mathcal{H}$ and $\mathcal{H}_1$. $\mathcal{A}$' queries are answered as follows:

- $\boldsymbol{\mathcal{H}(\cdot)}$: To the i'th query $\mathcal{B}$ responds with $h_i \in \mathbb{Z}_p$, that was given as input to $\mathcal{B}$, and ensures to respond identically to repeated queries.
- $\boldsymbol{\mathcal{H}_1(item)}$: $\mathcal{B}$ chooses $r_{item} \xleftarrow{\$} \mathbb{Z}_p$, gives $\hat{f} \coloneqq \hat{h}^{r_{item}}$ to $\mathcal{A}$, and ensures to respond identically to repeated queries.
- $\boldsymbol{\mathsf{AddR}(i)}$: For the j'th query, $j \neq \ell$, $\mathcal{B}$ chooses $y_i \xleftarrow{\$} \mathbb{Z}_p$, sets $\mathit{rsk}[i] \coloneqq y_i, \mathit{rpk}[i] \coloneqq h^{y_i}$, and hands $\mathit{rpk}[i]$ to $\mathcal{A}$. To the ℓ'th query, $\mathcal{B}$ responds by setting $i^* \coloneqq i, \mathit{rsk}[i^*] \coloneqq \star$, indicating that $\mathit{rsk}[i^*]$ is unknown, $\mathit{rpk}[i^*] \coloneqq \phi(\mathcal{D})$, and returning $\mathit{rpk}[i]$.
- $\boldsymbol{\mathsf{RSK}(i)}$: $\mathcal{B}$ executes the oracle defined in Figure 4.3, except when $i = i^*$. In that case $\mathcal{B}$ declares failure and exits.
- $\boldsymbol{\mathsf{RevR}(i)}$: $\mathcal{B}$ executes the oracle defined in Figure 4.3, except when $i = i^*$. In that case $\mathcal{B}$ sets $\mathit{rrt}[i^*] \coloneqq \phi(\mathcal{D})^{\frac{1}{\zeta}}$ and gives it to $\mathcal{A}$.

GRate(i, $item$, m): $\mathcal{B}$ executes the oracle defined in Figure 4.4. Only if $i = i^*$ $\mathcal{B}$ has to simulate a rating, since y_{i^*} is unknown. Therefore, $\mathcal{B}$ chooses $\alpha, \beta, \mu \leftarrow_{\$} \mathbb{Z}_p$ and computes

$$T_1 := u^{\alpha} \quad T_2 := v^{\beta} \quad T_3 := A_{i^*_{item}} \cdot w^{\alpha+\beta} \quad T_4 := d^{\mu} \quad T_5 := \phi(\hat{h}^{r_{item} \cdot \mu} \cdot \mathcal{D}^{r_{item}}).$$

The value T_5 is correct, since

$$T_5 = \phi(\hat{h}^{r_{item} \cdot \mu} \cdot \mathcal{D}^{r_{item}}) = \phi(\hat{h}^{r_{item} \cdot \mu} \cdot \hat{h}^{y_{i^*} \cdot r_{item}}) = \phi(\hat{h}^{r_{item}})^{\mu + y_{i^*}} = \phi(\hat{f})^{\mu + y_{i^*}}.$$

Now, $\mathcal{B}$ runs the simulator from Lemma 4.3 on input $(T_1, T_2, T_3, T_4, T_5)$ to obtain a transcript $(T_1, T_2, T_3, T_4, T_5, R_1, R_2, R_3, R_4, R_5, R_6, R_7, c, s_\alpha, s_\beta, s_x, s_{\hat{y}}, s_\mu, s_{\delta_1}, s_{\delta_2})$, and patches $\mathcal{H}$ by setting $\mathcal{H}(M, item, smpk, ipk[item], T_1, T_2, T_3, T_4, T_5, R_1, R_2, R_3, R_4, R_5, R_6, R_7) := c$. If this causes a collision, $\mathcal{B}$ declares failure and exits. Finally, $\mathcal{B}$ hands the simulated rating $\sigma := (item, T_1, T_2, T_3, T_4, T_5, c, s_\alpha, s_\beta, s_x, s_{\hat{y}}, s_\mu, s_{\delta_1}, s_{\delta_2})$ to $\mathcal{A}$.

Eventually, $\mathcal{A}$ outputs $(item^*, m^*, \sigma^*)$. If the rating is invalid, cannot be opened or traces to a corrupted rater, $\mathcal{B}$ declares failure and exits. Since the strong exculpability experiment is simulated perfectly unless $\mathcal{A}$ queries queries $\mathsf{RSK}(i^*)$ or patching the random oracle fails, $\mathcal{B}$ obtains a rating that traces to i^* with a probability of at least $\frac{\varepsilon}{q_{\mathsf{AddR}}} - \frac{q_{\mathcal{H}}^2}{p^{10}} - \frac{1}{p}$ and $\mathcal{B}$ can output (J, St), where J is the index of the query to $\mathcal{H}$ needed to compute σ^* and $\mathrm{St} := \big(smpk, item^*, ipk[item^*], (m_{i^*}, \sigma_{i^*})\big)$. If $\mathcal{A}$ never queried the random oracle for σ^*, $\mathcal{B}$ sets $J := 0$ and $\mathrm{St} := \bot$.

Now we apply the General Forking Lemma to obtain a second rating, which can be used to compute the solution for the discrete logarithm problem. The technique is exactly the same as in the proof of public linkability (Lemma 4.6), so we obtain two valid ratings that trace to rater i^* with probability $\boldsymbol{\varepsilon}_{\mathbf{F}_\mathcal{B}}$ of at least

$$\boldsymbol{\varepsilon}_{\mathbf{F}_\mathcal{B}} \geq \left(\frac{\varepsilon}{q_{\mathsf{AddR}}} - \frac{q_{\mathcal{H}}^2}{p^{10}} - \frac{1}{p} \right) \cdot \left(\frac{\frac{\varepsilon}{q_{\mathsf{AddR}}} - \frac{q_{\mathcal{H}}^2}{p^{10}} - \frac{1}{p}}{q_{\mathcal{H}}} - \frac{1}{p} \right)$$

Using the extractor from Lemma 4.4, we obtain a triple (A, x, y), where y is the secret rating key corresponding to i^*'s public key $rpk[i^*] = \phi(\mathcal{D})$. Hence $y = \log_h(\phi(\mathcal{D}))$, and $y = \log_{\hat{h}}(\mathcal{D})$, as required.

Since we assume that the Discrete Logarithm Assumption holds for bilinear group generator $\mathsf{BiGrGen}$, the probability $\boldsymbol{\varepsilon}_{\mathbf{F}_\mathcal{B}}$ must be negligible which implies that ε is negligible. Therefore the reputation system is strongly exculpable. □

8.2 UC-Security

Theorem 6.1:
If the SXDH Assumption and the Pointcheval-Sanders Assumption hold for bilinear group generator $\mathsf{BiGrGen}$, the hash functions $\mathcal{H}, \mathcal{H}_1$, and $\mathcal{H}_2$ are collision-resistant, and the communication channels between interacting parties are authenticated, Protocol Π_{RS} UC-realizes $\mathcal{F}_{\mathsf{RS}}$ in the $(\mathcal{F}_{\mathsf{RO}}, \mathcal{F}_{\mathsf{CRS}}, \mathcal{F}_{\mathsf{CA}})$-hybrid model, in the presence of static adversaries. ◇

To prove Theorem 6.1 we have to show that for any probabilistic polynomial-time real-world adversary $\mathcal{A}$ there exists a probabilistic polynomial-time ideal-world adversary $\mathcal{S}$ such that for any probabilistic polynomial-time environment $\mathcal{Z}$ it holds:

$$\left\{\mathrm{EXEC}_{\mathcal{F}_{\mathsf{RS}},\mathcal{S}^{\mathcal{A}},\mathcal{Z}}(\lambda,z)\right\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*} \stackrel{c}{\equiv} \left\{\mathrm{EXEC}^{\mathcal{F}_{\mathsf{RO}},\mathcal{F}_{\mathsf{CRS}},\mathcal{F}_{\mathsf{CA}}}_{\Pi_{\mathsf{RS}},\mathcal{A},\mathcal{Z}}(\lambda,z)\right\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*} .$$

We divide the proof of this statement into three parts. In the first part we define the simulator $\mathcal{S}$ that interacts with $\mathcal{F}_{\mathsf{RS}}$ and simulates the cryptographic computations. Note that during Rate-requests $\mathcal{S}$ does not obtain any identifying information of the rater. Hence, $\mathcal{S}$ uses the zero-knowledge simulator for the Signature of Knowledge that represents a rating. Analogously, opening-proofs are represented by a Signature of Knowledge. Therefore, $\mathcal{S}$ uses the corresponding zero-knowledge simulator to generate opening-proofs.

In the second part of the proof we define a hybrid game $\mathcal{G}$ and a corresponding simulator $\mathcal{S}_1$ for which we prove that no environment $\mathcal{Z}$ can distinguish whether it interacts with $(\mathcal{F}_{\mathsf{RS}},\mathcal{S})$ or $(\mathcal{G},\mathcal{S}_1)$. In this game $\mathcal{S}_1$ obtains all identifying information during Rate-requests and therefore can execute the computations as defined in Protocol Π_{RS}. Also opening-proofs can be generated by $\mathcal{S}_1$ as in Protocol Π_{RS}. Hence, an environment $\mathcal{Z}$ is only able to distinguish $(\mathcal{F}_{\mathsf{RS}},\mathcal{S})$ and $(\mathcal{G},\mathcal{S}_1)$, if it can distinguish between simulated and real ratings and opening-proofs. Under the *SXDH-Assumption* this is not possible.

In the third part of the proof we show that $\mathcal{S}_1$ executes exactly the same computations as Protocol Π_{RS}. This implies that any environment $\mathcal{Z}$ that distinguishes between $(\mathcal{G},\mathcal{S}_1)$ and $(\Pi_{\mathsf{RS}},\mathcal{A})$ is able to let $\mathcal{F}_{\mathsf{RS}}$ output `error`, whereas the Protocol Π_{RS} outputs some value, or $\mathcal{F}_{\mathsf{RS}}$ outputs 0, whereas Protocol Π_{RS} outputs 1 (or vice versa). Using different reductions (provided in Section 8.2.4) to the Pointcheval-Sanders-Problem and to the CCA2-security of the Cramer-Shoup encryption scheme we show that such environments cannot exist. Hence, Π_{RS} UC-realizes $\mathcal{F}_{\mathsf{RS}}$ in the ($\mathcal{F}_{\mathsf{RO}}$, $\mathcal{F}_{\mathsf{CRS}}$, $\mathcal{F}_{\mathsf{CA}}$)-hybrid model.

Formally, we prove the following: for every $\mathcal{A}$ and every $\mathcal{Z}$

$$\left\{\mathrm{EXEC}_{\mathcal{F}_{\mathsf{RS}},\mathcal{S}^{\mathcal{A}},\mathcal{Z}}(\lambda,z)\right\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*} \stackrel{c}{\equiv} \left\{\mathrm{EXEC}^{\mathcal{F}_{\mathsf{RO}},\mathcal{F}_{\mathsf{CRS}},\mathcal{F}_{\mathsf{CA}}}_{\Pi_{\mathsf{RS}},\mathcal{A},\mathcal{Z}}(\lambda,z)\right\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*}$$

by introducing a hybrid game $\mathcal{G}$ and proving the two relations

$$\left\{\mathrm{EXEC}_{\mathcal{F}_{\mathsf{RS}},\mathcal{S}^{\mathcal{A}},\mathcal{Z}}(\lambda,z)\right\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*} \stackrel{c}{\equiv} \left\{\mathcal{G}_{\mathcal{F}_{\mathsf{RS}},\mathcal{S}_1^{\mathcal{A}},\mathcal{Z}}(\lambda,z)\right\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*}$$

and

$$\left\{\mathcal{G}_{\mathcal{F}_{\mathsf{RS}},\mathcal{S}_1^{\mathcal{A}},\mathcal{Z}}(\lambda,z)\right\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*} \stackrel{c}{\equiv} \left\{\mathrm{EXEC}^{\mathcal{F}_{\mathsf{RO}},\mathcal{F}_{\mathsf{CRS}},\mathcal{F}_{\mathsf{CA}}}_{\Pi_{\mathsf{RS}},\mathcal{A},\mathcal{Z}}(\lambda,z)\right\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*} .$$

As abbreviations we set

$$\mathrm{IDEAL} \coloneqq \left\{\mathrm{EXEC}_{\mathcal{F}_{\mathsf{RS}},\mathcal{S}^{\mathcal{A}},\mathcal{Z}}(\lambda,z)\right\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*}$$

and

$$\mathrm{HYBRID} \coloneqq \left\{\mathrm{EXEC}^{\mathcal{F}_{\mathsf{RO}},\mathcal{F}_{\mathsf{CRS}},\mathcal{F}_{\mathsf{CA}}}_{\Pi_{\mathsf{RS}},\mathcal{A},\mathcal{Z}}(\lambda,z)\right\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*} .$$

8.2.1 Foundations

The protocols $\Pi_{\mathsf{RS}}.\mathsf{Register}$, $\Pi_{\mathsf{RS}}.\mathsf{NewProduct}$, $\Pi_{\mathsf{RS}}.\mathsf{Purchase}$, $\Pi_{\mathsf{RS}}.\mathsf{Rate}$, and $\Pi_{\mathsf{RS}}.\mathsf{OProof}$ are based on zero-knowledge proofs of knowledge. In this section we prove these properties because they are used in the proof of Theorem 6.1. To formalize the statements to prove we use the notation introduced by Camenisch and Stadler [CS97].

Lemma 8.2:
In the protocols $\Pi_{\mathsf{RS}}.\mathsf{Register}$ and $\Pi_{\mathsf{RS}}.\mathsf{Purchase}$ party P_i proves the statement $\mathsf{ZKPK}\{(usk_i) : M_i = g_1^{usk_i}\}$ to P_{SM} and P_j, respectively. ◇

Proof.

- Completeness: An honest prover will generate an accepting transcript as the verification equations hold: $M_i^{ch} \cdot T = (g_1^{usk_i})^{ch} \cdot g_1^{\alpha} = g_1^{\alpha + ch \cdot usk_i} = g_1^{s_\alpha}$ and $R = u^{\mathcal{H}(T)} \cdot v^r$.
- Interactive Simulator: In order to simulate transcripts of the protocol, the simulator has to set up the Trapdoor Pedersen Commitment (Definition 6.1). By running the KeyGen algorithm the simulator knows the trapdoor td. With this value the simulation works as follows:

  Choose $\alpha', r' \xleftarrow{u} \mathbb{Z}_p$, compute $T' \coloneqq g_1^{\alpha'}$, $R \coloneqq u^{\mathcal{H}(T')} \cdot v^{r'}$ and send the commitment R to the verifier. On receiving a challenge $ch \in \mathbb{Z}_p$ the simulator chooses $s_\alpha \xleftarrow{u} \mathbb{Z}_p$ and sets $T \coloneqq g_1^{s_\alpha} \cdot M_i^{-ch}$. Finally, by using the commitment trapdoor td the simulator computes $r \coloneqq (\mathcal{H}(T') - \mathcal{H}(T) + td \cdot r') \cdot td^{-1}$, according to the $\mathsf{TReveal}$ algorithm, and outputs (s_α, T, r) to the verifier. The resulting transcripts are identically distributed as real transcripts.
- Extractor: Given two accepting transcripts (R, ch, s_α, T, r) and $(R, ch', s'_\alpha, T, r)$ the extractor computes $usk_i \coloneqq (s_\alpha - s'_\alpha)/(ch - ch')$, which is the discrete logarithm of M_i to base g_1: $M_i^{ch} \cdot T = g_1^{s_\alpha} \wedge M_i^{ch'} \cdot T = g_1^{s'_\alpha} \Longrightarrow M_i^{ch - ch'} = g_1^{s_\alpha - s'_\alpha} \Longrightarrow M_i = g_1^{(s_\alpha - s'_\alpha)/(ch - ch')}$. □

Lemma 8.3:
The value $ppk_{i,prod}$ output in Protocol $\Pi_{\mathsf{RS}}.\mathsf{NewProduct}$ is a Signature of Knowledge on message $\mathsf{PS}.pk_{i,prod}$ proving the statement $\mathsf{SoK}\{(usk_i) : M_i = g_1^{usk_i} \wedge M_{i,prod} = \mathcal{H}_1(i, prod)^{usk_i}\}$. ◇

Proof.

- Completeness: An honest prover will generate an accepting transcript as the verification equations hold:

$$\begin{aligned} \mathcal{H}_1(i, prod)^{s_{i,prod}} \cdot M_{i,prod}^{-ch_{i,prod}} &= \mathcal{H}_1(i, prod)^{r + ch_{i,prod} \cdot usk_i} \cdot \mathcal{H}_1(i, prod)^{-usk_i \cdot ch_{i,prod}} \\ &= \mathcal{H}_1(i, prod)^r = R_1 \qquad (8.5) \\ g_1^{s_{i,prod}} \cdot M_i^{-ch_{i,prod}} &= g_1^{r + ch_{i,prod} \cdot usk_i} \cdot g_1^{-usk_i \cdot ch_{i,prod}} = g_1^r = R_2 \qquad (8.6) \end{aligned}$$

and hence

$$ch_{i,prod} = \mathcal{F}_{\mathsf{RO}}(\mathsf{PS}.pk_{i,prod}, M_i, M_{i,prod}, R_1, R_2). \tag{8.7}$$

- Simulator: Given $M_i, M_{i,prod}$, and $\mathsf{PS}.pk_{i,prod}$ as input and using the random oracle $\mathcal{F}_{\mathsf{RO}}$, transcripts can be simulated, as follows:

  Choose $ch_{i,prod}, s_{i,prod} \xleftarrow{u} \mathbb{Z}_p$, set $R_1 := \mathcal{H}_1(i, prod)^{s_{i,prod}} \cdot M_{i,prod}^{-ch_{i,prod}}$ and $R_2 := g_1^{s_{i,prod}} \cdot M_i^{-ch_{i,prod}}$, and patch $\mathcal{F}_{\mathsf{RO}}(\mathsf{PS}.pk_{i,prod}, M_i, M_{i,prod}, R_1, R_2) := c_{i,prod}$. The resulting transcripts are identically distributed as real transcripts.

- Extractor: Given two accepting transcripts, as Signatures of Knowledge on message $\mathsf{PS}.pk_{i,prod}$, $(M_i, M_{i,prod}, ch_{i,prod}, s_{i,prod})$ and $(M_i, M_{i,prod}, ch'_{i,prod}, s'_{i,prod})$ the extractor computes $usk_i := (s_{i,prod} - s'_{i,prod})/(ch_{i,prod} - ch'_{i,prod})$, which is the discrete logarithm of M_i to base g_1: $M_i^{ch_{i,prod}} \cdot R_1 = g_1^{s_{i,prod}} \wedge M_i^{ch'_{i,prod}} \cdot R_1 = g_1^{s'_{i,prod}} \implies M_i^{ch_{i,prod} - ch'_{i,prod}} = g_1^{s_{i,prod} - s'_{i,prod}} \implies M_i = g_1^{(s_{i,prod} - s'_{i,prod})/(ch_{i,prod} - ch'_{i,prod})}$. Analogously one can argue for the discrete logarithm of $M_{i,prod}$ to base $\mathcal{H}_1(i, prod)$. □

Lemma 8.4:
The value σ output in Protocol $\Pi_{\mathsf{RS}}.\mathsf{Rate}$ is a Signature of Knowledge on message m proving the following statement:

$$\begin{aligned}\mathsf{SoK}\{(usk_i, \sigma_i, \sigma_{i,j,prod}) : \ & \mathsf{Verify}(\mathsf{PS}.pk, usk_i, (T_1, T_2)) = 1 \\ & \wedge \mathsf{Verify}(\mathsf{PS}.pk_{j,prod}, usk_i, (T_3, T_4)) = 1 \\ & \wedge T_5 = \mathcal{H}_1(j, prod)^{usk_i}\}.\end{aligned}$$

◇

Proof.

- Completeness: An honest prover will generate an accepting transcript as the verification equations hold:

$$\begin{aligned}&\mathrm{e}(T_1, \tilde{X})^{ch \cdot x}\, \mathrm{e}(T_2, \tilde{g})^{-ch} \cdot \mathrm{e}(T_1, \tilde{Y})^{s} \\ &\quad = \mathrm{e}(T_1, \tilde{g}^{\xi_1})^{ch} \cdot \mathrm{e}(T_2, \tilde{g})^{-ch} \cdot \mathrm{e}(T_1, \tilde{g}^{\xi_2})^{k + ch \cdot usk_i} \\ &\quad = \mathrm{e}(T_1, \tilde{g}^{\xi_1})^{ch} \cdot \mathrm{e}(T_1, \tilde{g}^{\xi_2 \cdot usk_i})^{ch} \cdot \mathrm{e}(T_2, \tilde{g})^{-ch} \cdot \mathrm{e}(T_1, \tilde{g}^{\xi_2})^{k} \\ &\quad = \mathrm{e}(T_1, \tilde{g}^{\xi_1 + \xi_2 \cdot usk_i})^{ch} \cdot \mathrm{e}(T_2, \tilde{g})^{-ch} \cdot \mathrm{e}(T_1, \tilde{g}^{\xi_2})^{k} \\ &\quad = \mathrm{e}(T_1, \tilde{Y})^{k} = R_1 \qquad (8.8) \\ &\iff \mathsf{PS.Verify}(\mathsf{PS}.pk, usk_i, (T_1, T_2)) = 1 \qquad (8.9)\end{aligned}$$

$$\begin{aligned}&\mathrm{e}(T_3, \tilde{X}_{j,prod})^{ch} \cdot \mathrm{e}(T_4, \tilde{g}_{j,prod})^{-ch} \cdot \mathrm{e}(T_3, \tilde{Y}_{j,prod})^{s} \\ &\quad = \mathrm{e}(T_3, \tilde{g}_{j,prod}^{\xi_{1_{j,prod}}})^{ch} \cdot \mathrm{e}(T_4, \tilde{g}_{j,prod})^{-ch} \cdot \mathrm{e}(T_3, \tilde{g}_{j,prod}^{\xi_{2_{j,prod}}})^{k + ch \cdot usk_i} \\ &\quad = \mathrm{e}(T_3, \tilde{g}_{j,prod}^{\xi_{1_{j,prod}}})^{ch} \cdot \mathrm{e}(T_3, \tilde{g}_{j,prod}^{\xi_{2_{j,prod}} \cdot usk_i})^{ch} \cdot \mathrm{e}(T_4, \tilde{g}_{j,prod})^{-ch} \cdot \mathrm{e}(T_3, \tilde{g}_{j,prod}^{\xi_{2_{j,prod}}})^{k}\end{aligned}$$

$$= \mathrm{e}(T_3, \tilde{g}_{j,prod}^{\xi_{1_{j,prod}}+\xi_{2_{j,prod}}\cdot usk_i})^{ch} \cdot \mathrm{e}(T_4, \tilde{g}_{j,prod})^{-ch} \cdot \mathrm{e}(T_3, \tilde{g}_{j,prod}^{\xi_{2_{j,prod}}})^{k}$$
$$= \mathrm{e}(T_3, \tilde{Y}_{j,prod})^{k} = R_2 \tag{8.10}$$
$$\iff \mathsf{PS.Verify}(\mathsf{PS}.pk_{j,prod}, usk_i, (T_3, T_4)) = 1 \tag{8.11}$$

$$T_5^{-ch} \cdot \mathcal{H}_1(j, prod)^s = \mathcal{H}_1(j, prod)^{-usk_i \cdot ch} \cdot \mathcal{H}_1(j, prod)^{k+usk_i \cdot ch} = \mathcal{H}_1(j, prod)^k = R_3 \tag{8.12}$$

and hence

$$ch = \mathcal{F}_{\mathsf{RO}}(T_1, T_2, T_3, T_4, T_5, R_1, R_2, R_3, pp, prod, ppk, m). \tag{8.13}$$

- Simulator: Given $pp, j, prod, ppk$ and m as input and using the random oracle $\mathcal{F}_{\mathsf{RO}}$, transcripts can be simulated, as follows:

  Choose $ch, s \leftarrow\!\!\shortmid \mathbb{Z}_p$ and $T_1, T_2, T_3, T_4, T_5 \leftarrow\!\!\shortmid \mathbb{G}_1$, set $R_1 \coloneqq \mathrm{e}(T_1, \tilde{X})^{ch} \cdot \mathrm{e}(T_2, \tilde{g})^{-ch} \cdot \mathrm{e}(T_1, \tilde{Y})^{s}$, $R_2 \coloneqq \mathrm{e}(T_3, \tilde{X}_{j,prod})^{ch} \cdot \mathrm{e}(T_4, \tilde{g}_{j,prod})^{-ch} \cdot \mathrm{e}(T_3, \tilde{Y}_{j,prod})^{s}$, $R_3 \coloneqq T_5^{-ch} \cdot \mathcal{H}_1(j, prod)^s$, and patch $\mathcal{F}_{\mathsf{RO}}(T_1, T_2, T_3, T_4, T_5, R_1, R_2, R_3, pp, prod, ppk, m) \coloneqq ch$. Under the assumption that the Decisional Diffie-Hellman Problem is hard in $\mathbb{G}_1$ the tuples (T_1, T_2), and (T_3, T_4) are indistinguishable from real signatures on message usk_i, under the respective public keys $\mathsf{PS}.pk$ and $\mathsf{PS}.pk_{j,prod}$. Futhermore, the value T_5 is chosen uniformly at random. Hence, the tuple $(g_1, M_i, \mathcal{H}_1(j, prod), T_5)$ is indistinguishable from real transcripts (all values M_i are given by $\mathcal{F}_{\mathsf{CA}}$ and hence known to a verifier). The remainder of the transcript is simulated perfectly.

- Extractor: Given two accepting transcripts $(T_1, T_2, T_3, T_4, T_5, ch, s)$ and $(T_1, T_2, T_3, T_4, T_5, ch', s')$ the extractor computes $usk_i \coloneqq (s - s')/(ch - ch')$, which is the discrete logarithm of T_5 to base $\mathcal{H}_1(j, prod)$: $T_5^{-ch} \cdot \mathcal{H}_1(j, prod)^s = R_3 \wedge T_5^{-ch'} \cdot \mathcal{H}_1(j, prod)^{s'} = R_3 \implies T_5^{ch-ch'} = \mathcal{H}_1(j, prod)^{s-s'} \implies T_5 = \mathcal{H}_1(j, prod)^{(s-s')/(ch-ch')}$. The tuples (T_1, T_2) and (T_3, T_4) are valid signatures on message usk_i and do not need further extraction. □

Lemma 8.5:
The value τ output in Protocol $\Pi_{\mathsf{RS}}.\mathsf{OProof}$ is a Signature of Knowledge on message (σ, i, M_i) proving the following statement:

$$\mathsf{SoK}\{(\beta, \tilde{Y}_i) \colon ct = \mathsf{CS.Enc}(\mathsf{CS}.pk, \tilde{Y}_i; \beta) \wedge \mathrm{e}(T_5, \tilde{Y}) = \mathrm{e}(\mathcal{H}_1(j, prod), \tilde{Y}_i).$$

◇

Proof.

- Completeness: An honest prover will generate an accepting transcript as the verification equations hold:

$$ct_1^{-\hat{ch}} \cdot g_2^{\hat{s}} = g_2^{-\beta \cdot \hat{ch}} \cdot g_2^{r+\hat{ch}\cdot\beta} = g_2^r = R_1 \tag{8.14}$$
$$ct_2^{-\hat{ch}} \cdot \tilde{h}^{\hat{s}} = \tilde{h}^{-\beta \cdot \hat{ch}} \cdot \tilde{h}^{r+\hat{ch}\cdot\beta} = \tilde{h}^r = R_2 \tag{8.15}$$

$$
\begin{aligned}
&\mathrm{e}(\mathcal{H}_1(j, prod), ct_3)^{-\hat{ch}} \cdot \mathrm{e}(T_5, \tilde{Y})^{\hat{ch}} \cdot \mathrm{e}(\mathcal{H}_1(j, prod), \tilde{f})^{\hat{s}} \\
&= \mathrm{e}(\mathcal{H}_1(j, prod), \tilde{Y}_i \cdot \tilde{f}^{\beta})^{-\hat{ch}} \cdot \mathrm{e}(\mathcal{H}_1(j, prod)^{usk_i}, \tilde{Y})^{\hat{ch}} \cdot \mathrm{e}(\mathcal{H}_1(j, prod), \tilde{f})^{r+\hat{ch}\cdot\beta} \\
&= \mathrm{e}(\mathcal{H}_1(j, prod), \tilde{Y}_i^{-1} \cdot \tilde{f}^{-\beta})^{\hat{ch}} \cdot \mathrm{e}(\mathcal{H}_1(j, prod), \tilde{Y}^{usk_i})^{\hat{ch}} \cdot \mathrm{e}(\mathcal{H}_1(j, prod), \tilde{f})^{r+\hat{ch}\cdot\beta} \\
&= \mathrm{e}(\mathcal{H}_1(j, prod), \tilde{Y}_i^{-1} \cdot \tilde{f}^{-\beta} \cdot \tilde{Y}^{usk_i})^{\hat{ch}} \cdot \mathrm{e}(\mathcal{H}_1(j, prod), \tilde{f}^{\beta})^{\hat{ch}} \cdot \mathrm{e}(\mathcal{H}_1(j, prod), \tilde{f})^{r} \\
&= \mathrm{e}(\mathcal{H}_1(j, prod), \tilde{Y}_i^{-1} \cdot \tilde{f}^{-\beta} \cdot \tilde{Y}^{usk_i} \cdot \tilde{f}^{\beta})^{\hat{ch}} \cdot \mathrm{e}(\mathcal{H}_1(j, prod), \tilde{f})^{r} \\
&= \mathrm{e}(\mathcal{H}_1(j, prod), 1_{\mathbb{G}_2})^{\hat{ch}} \cdot \mathrm{e}(\mathcal{H}_1(j, prod), \tilde{f})^{r} = \mathrm{e}(\mathcal{H}_1(j, prod), \tilde{f})^{r} = R_3
\end{aligned} \tag{8.16}
$$

$$
\begin{aligned}
&\omega \coloneqq \mathcal{H}(ct_1, ct_2, ct_3) \\
&\quad ct_4^{-\hat{ch}} \cdot (\tilde{b} \cdot \tilde{d}^{\omega})^{\hat{s}} = (\tilde{b} \cdot \tilde{d}^{\omega})^{-\beta\cdot\hat{ch}} \cdot (\tilde{b} \cdot \tilde{d}^{\omega})^{r+\hat{ch}\cdot\beta} = (\tilde{b} \cdot \tilde{d}^{\omega})^{r} = R_4
\end{aligned} \tag{8.17}
$$

$$
\begin{aligned}
&\mathrm{e}(g_1, ct_3)^{-\hat{ch}} \cdot \mathrm{e}(M, \tilde{Y})^{\hat{ch}} \cdot \mathrm{e}(g_1, \tilde{f})^{\hat{s}} \\
&\qquad = \mathrm{e}(g_1, \tilde{Y}_i \cdot \tilde{f}^{\beta})^{-\hat{ch}} \cdot \mathrm{e}(g_1^{usk_i}, \tilde{Y})^{\hat{ch}} \cdot \mathrm{e}(g_1, \tilde{f})^{r+\hat{ch}\cdot\beta} \\
&\qquad = \mathrm{e}(g_1, \tilde{Y}_i^{-1} \cdot \tilde{f}^{-\beta})^{\hat{ch}} \cdot \mathrm{e}(g_1, \tilde{Y}^{usk_i})^{\hat{ch}} \cdot \mathrm{e}(g_1, \tilde{f})^{r+\hat{ch}\cdot\beta} \\
&\qquad = \mathrm{e}(g_1, \tilde{Y}_i^{-1} \cdot \tilde{f}^{-\beta} \cdot \tilde{Y}^{usk_i})^{\hat{ch}} \cdot \mathrm{e}(g_1, \tilde{f})^{r+\hat{ch}\cdot\beta} \\
&\qquad = \mathrm{e}(g_1, \tilde{Y}_i^{-1} \cdot \tilde{f}^{-\beta} \cdot \tilde{Y}^{usk_i})^{\hat{ch}} \cdot \mathrm{e}(g_1, \tilde{f}^{\beta})^{\hat{ch}} \cdot \mathrm{e}(g_1, \tilde{f})^{r} \\
&\qquad = \mathrm{e}(g_1, \tilde{Y}_i^{-1} \cdot \tilde{f}^{-\beta} \cdot \tilde{Y}^{usk_i} \cdot \tilde{f}^{\beta})^{\hat{ch}} \cdot \mathrm{e}(g_1, \tilde{f})^{r} \\
&\qquad = \mathrm{e}(g_1, 1_{\mathbb{G}_2})^{\hat{ch}} \cdot \mathrm{e}(g_1, \tilde{f})^{r} = \mathrm{e}(g_1, \tilde{f})^{r} = R_5,
\end{aligned} \tag{8.18}
$$

where Equation 8.16 holds, if and only if $\mathrm{dlog}_{\tilde{Y}}(\tilde{Y}_i) = \mathrm{dlog}_{\mathcal{H}_1(j,prod)}(T_5)$ and Equation 8.18 holds, if and only if $\mathrm{dlog}_{g_1}(M) = \mathrm{dlog}_{\tilde{Y}}(\tilde{Y}_i)$.

- Simulator: Given $j, prod, ppk$, and a rating (m, σ) as input and using the random oracle $\mathcal{F}_{\mathsf{RO}}$, transcripts can be simulated, as follows:

  Choose $\hat{ch}, \hat{s} \xleftarrow{\pi} \mathbb{Z}_p$ and $ct_1, ct_2, ct_3, ct_4 \xleftarrow{\pi} \mathbb{G}_2$, compute $R_1 \coloneqq ct_1^{-\hat{ch}} \cdot g_2^{\hat{s}}$, $R_2 \coloneqq ct_2^{-\hat{ch}} \cdot \tilde{h}^{\hat{s}}$,

  $R_3 \coloneqq \mathrm{e}(\mathcal{H}_1(j, prod), ct_3)^{-\hat{ch}} \cdot \mathrm{e}(T_5, \tilde{Y})^{\hat{ch}} \cdot \mathrm{e}(\mathcal{H}_1(j, prod), \tilde{f})^{\hat{s}}$, $\omega \coloneqq \mathcal{H}(ct_1, ct_2, ct_3)$,

  $R_4 \coloneqq ct_4^{-\hat{ch}} \cdot (\tilde{b} \cdot \tilde{d}^{\omega})^{\hat{s}}$, $R_5 \coloneqq \mathrm{e}(g_1, c_3)^{-\hat{ch}} \cdot \mathrm{e}(M, \tilde{Y})^{\hat{ch}} \cdot \mathrm{e}(g_1, \tilde{f})^{\hat{s}}$,

  and patch $\mathcal{F}_{\mathsf{RO}}(ct_1, ct_2, ct_3, ct_4, R_1, R_2, R_3, R_4, R_5, \sigma, i, M) \coloneqq \hat{ch}$. The ciphertext $ct = (ct_1, ct_2, ct_3, ct_4)$ is indistinguishable from real ciphertexts, assuming the Decisional Diffie-Hellman Problem is hard in $\mathbb{G}_1$ and the remainder of the transcript is simulated perfectly.

- Extractor: Given two accepting transcripts $(ct_1, ct_2, ct_3, ct_4, \hat{ch}, \hat{s})$ and $(ct_1, ct_2, ct_3, ct_4, \hat{ch}', \hat{s}')$ the extractor computes $\beta \coloneqq (\hat{s} - \hat{s}')/(\hat{ch} - \hat{ch}')$, which is the discrete logarithm of ct_1 to base g_2: $ct_1^{-\hat{ch}} \cdot g_2^{\hat{s}} = R_1 \wedge ct_1^{-\hat{ch}'} \cdot g_2^{\hat{s}'} = R_1 \implies g_2^{\hat{s}-\hat{s}'} = ct_1^{\hat{ch}-\hat{ch}'} \implies g_2^{(\hat{s}-\hat{s}')/(\hat{ch}-\hat{ch}')} = ct_1$. Analogously, one can argue for R_2 and R_4. Further, the

extractor computes $\tilde{Y}_i \coloneqq ct_3^{-\beta}$, which is the encrypted value: dividing two instances of Equation 8.18 gives

$$\begin{aligned} & \mathrm{e}(M,\tilde{Y})^{\hat{ch}-\hat{ch'}} \cdot \mathrm{e}(g_1,\tilde{f})^{\hat{s}-\hat{s}'} = \mathrm{e}(g_1,ct_3)^{\hat{ch}-\hat{ch'}} \\ \iff & \mathrm{e}(M,\tilde{Y}) \cdot \mathrm{e}(g_1,\tilde{f})^{(\hat{s}-\hat{s}')/(\hat{ch}-\hat{ch'})} = \mathrm{e}(g_1,ct_3) \\ \iff & \mathrm{e}(M,\tilde{Y}) \cdot \mathrm{e}(g_1,\tilde{f})^{\beta} = \mathrm{e}(g_1,ct_3) \\ \iff & \mathrm{e}(M,\tilde{Y}) = \mathrm{e}(g_1,ct_3) \cdot \mathrm{e}(g_1,\tilde{f})^{-\beta} = \mathrm{e}(g_1,ct_3 \cdot \tilde{f}^{-\beta}), \end{aligned}$$

which means that ct_3 encrypts an element $\tilde{Y}' \in \mathbb{G}_2$ such that $\mathrm{e}(M,\tilde{Y}) = \mathrm{e}(g_1,\tilde{Y}')$. The only element $\tilde{Y}'$ with this property is $\tilde{Y}_i = \tilde{Y}^{usk_i}$. □

8.2.2 Definition of the Simulator

$\mathcal{S}$ manages the same lists as $\mathcal{F}_{\mathsf{RS}}$, namely $\mathfrak{Params}$, $\mathfrak{Reg}$, $\mathfrak{Prods}$, $\mathfrak{Purch}$, $\mathfrak{Ratings}$, and $\mathfrak{Open}$ which are initially empty. Lists indexed with an „s“ additionally store secret key material.

Simulation of $\mathcal{F}_{\mathsf{RO}}$: $\mathcal{S}$ manages the list L_{RO} and answers to requests exactly the same way as $\mathcal{F}_{\mathsf{RO}}$. At some points during the simulation - while generating anonymous ratings and opening-proofs - $\mathcal{S}$ will have to patch the list L_{RO} because $\mathcal{S}$ simulates Σ-protocols that were transformed into non-interactive zero-knowledge proofs using the Fiat-Shamir heuristic. How $\mathcal{S}$ handles this cases is described later in detail.

Simulation of $\mathcal{F}_{\mathsf{CRS}}$: $\mathcal{S}$ chooses $(\mathbb{GD}, \mathsf{PD}.pk, \mathcal{H}, \mathcal{H}_1, \mathcal{H}_2)$ according to the definition of $\mathcal{F}_{\mathsf{CRS}}$ and hands $crs \coloneqq (\mathbb{GD}, \mathsf{PD}.pk, \mathcal{H}, \mathcal{H}_1, \mathcal{H}_2)$ to every party requesting it.

Simulation of $\mathcal{F}_{\mathsf{CA}}$: $\mathcal{S}$ manages the lists L_{CA} and L_{sCA}. Whenever an honest party P_i is activated for the first time, $\mathcal{S}$ chooses $usk_i \xleftarrow{u} \mathbb{Z}_p$, computes $M_i \coloneqq g_1^{usk_i}$ and sets $L_{CA}.\mathsf{Add}(P_i, M_i)$ and $L_{sCA}\mathsf{Add}(P_i, M_i, usk_i)$. When $\mathcal{S}$ receives $(\mathsf{Register}, P_i, v)$ from some (corrupted) party P_i for the first time, the tuple (P_i, v) is added to the list L_{CA}. All later $\mathsf{Register}$-requests from the same party are ignored. When $\mathcal{S}$ receives $(\mathsf{Retrieve}, P_i)$ from some party P and a tuple (P_i, v) is stored in L_{CA}, $\mathcal{S}$ sends $(\mathsf{Retrieve}, P_i, v)$ to P. If no such tuple could be found in L_{CA}, $\mathcal{S}$ sends $(\mathsf{Retrieve}, P_i, \perp)$ to P.

Simulation of Registry Key Generation: When $\mathcal{S}$ receives (KeyGen, sid) from $\mathcal{F}_{\mathsf{RS}}$, $\mathcal{S}$ executes the protocol in behalf of P_{SM}, sets $\mathfrak{Params}.\mathsf{Add}(pp)$, $\mathfrak{Params}_s.\mathsf{Add}(pp, psk)$ and sends $(\mathsf{KeyGen}, sid, pp)$ to $\mathcal{F}_{\mathsf{RS}}$.

Simulation of User Registration:

- P_{SM} and P_i honest: When $\mathcal{S}$ receives $(\mathsf{Register}, sid, pp', P_i)$ from $\mathcal{F}_{\mathsf{RS}}$, $\mathcal{S}$ executes the registration protocol in behalf of P_i and P_{SM}. $\mathcal{S}$ can do this by using $(P_i, M_i, usk_i) \in L_{sCA}$ and $(pp', psk') \in \mathfrak{Params}_s$.
- P_{SM} honest and P_i corrupted: When $\mathcal{S}$ receives (pp', R) from $\mathcal{A}$ as it intends to send from P_i to P_{SM}, $\mathcal{S}$ sends $(\mathsf{Register}, sid, pp')$ in behalf of P_i to $\mathcal{F}_{\mathsf{RS}}$, receives $(\mathsf{Register}, sid, pp', P_i)$ from $\mathcal{F}_{\mathsf{RS}}$, and executes the protocol in behalf of P_{SM} using $(pp', psk') \in \mathfrak{Params}_s$.

When $\mathcal{F}_{\mathsf{RS}}$ outputs $(\mathsf{Register}, sid, pp', P_i, f)$ to P_i, $\mathcal{S}$ does not deliver this message because a corrupted P_i does not expect to receive this message from $\mathcal{F}_{\mathsf{RS}}$.

- P_{SM} corrupted and P_i honest: When $\mathcal{S}$ receives $(\mathsf{Register}, sid, pp', P_i)$ from $\mathcal{F}_{\mathsf{RS}}$, $\mathcal{S}$ executes the protocol in behalf of P_i by using $(P_i, M_i, usk_i) \in L_{sCA}$. If $\mathcal{S}$ receives a value σ_i from P_{SM} and $\mathsf{PS.Verify}(pp', usk_i, \sigma_i) = 1$, set $\mathfrak{Reg}.\mathsf{Add}(P_i, pp', M_i, \tilde{Y}_i, \sigma_i)$ and output $(\mathsf{Register}, sid, pp', P_i, 1)$ to $\mathcal{F}_{\mathsf{RS}}$.

  When $\mathcal{F}_{\mathsf{RS}}$ outputs $(\mathsf{Register}, sid, pp', P_i, f)$ to P_{SM}, $\mathcal{S}$ does not deliver this message because a corrupted P_{SM} does not expect to receive this message from $\mathcal{F}_{\mathsf{RS}}$.

Simulation of Product Addition: When $\mathcal{S}$ receives $(\mathsf{NewProduct}, sid, P_i, prod)$ from $\mathcal{F}_{\mathsf{RS}}$, $\mathcal{S}$ executes the protocol in behalf of P_i using $(P_i, M_i, usk_i) \in L_{sCA}$. The request to $\mathcal{F}_{\mathsf{RO}}$ does not require any special handling. After completing the protocol, $\mathcal{S}$ sets $\mathfrak{Prods}.\mathsf{Add}(P_i, prod, ppk_{i,prod})$, $\mathfrak{Prods}_s.\mathsf{Add}(P_i, prod, ppk_{i,prod}, \mathsf{PS}.sk_{i,prod})$ and outputs $(\mathsf{NewProduct}, sid, P_i, prod, ppk_{i,prod})$ to $\mathcal{F}_{\mathsf{RS}}$.

Simulation of Purchasing a Product:

- P_i and P_j honest: When $\mathcal{S}$ receives $(\mathsf{Purchase}, sid, P_i, P_j, prod, ppk)$ from $\mathcal{F}_{\mathsf{RS}}$, $\mathcal{S}$ executes the purchasing protocol in behalf of P_i and P_j. $\mathcal{S}$ can do this by using $(P_i, M_i, usk_i), (P_j, M_j, usk_j) \in L_{sCA}$ and $(P_j, prod, ppk_{j,prod}, \mathsf{PS}.sk_{j,prod}) \in \mathfrak{Prods}_s$.

- P_i honest and P_j corrupted: When $\mathcal{S}$ receives $(\mathsf{Purchase}, sid, P_i, P_j, prod, ppk)$ from $\mathcal{F}_{\mathsf{RS}}$, $\mathcal{S}$ executes the purchasing protocol in behalf of P_i, including the request to $\mathsf{VfyProd}$. $\mathcal{S}$ can do this by using $(P_i, M_i, usk_i) \in L_{sCA}$.

  When $\mathcal{F}_{\mathsf{RS}}$ outputs $(\mathsf{Purchase}, sid, P_i, P_j, prod, ppk, f)$ to P_j, $\mathcal{S}$ does not deliver this message because a corrupted P_j does not expect to receive this message from $\mathcal{F}_{\mathsf{RS}}$.

- P_i corrupted and P_j honest: When $\mathcal{S}$ receives $(prod, ppk, R)$ from $\mathcal{A}$ as it intends to send from P_i to P_j, $\mathcal{S}$ sends $(\mathsf{Purchase}, sid, P_j, prod, ppk)$ in behalf of P_i to $\mathcal{F}_{\mathsf{RS}}$, receives $(\mathsf{Purchase}, sid, P_i, P_j, prod, ppk)$ from $\mathcal{F}_{\mathsf{RS}}$, and executes the purchasing protocol in behalf of P_j. $\mathcal{S}$ can do this by using $(P_j, M_j, usk_j) \in L_{sCA}$ and $(P_j, prod, ppk_{j,prod}, \mathsf{PS}.sk_{j,prod}) \in \mathfrak{Prods}_s$.

  When $\mathcal{F}_{\mathsf{RS}}$ outputs $(\mathsf{Purchase}, sid, P_i, P_j, prod, ppk, f)$ to P_i, $\mathcal{S}$ does not deliver this message because a corrupted P_i does not expect to receive this message from $\mathcal{F}_{\mathsf{RS}}$.

Simulation of VfyProd: When $\mathcal{S}$ receives $(\mathsf{VfyProd}, sid, P_j, prod, ppk)$ from $\mathcal{F}_{\mathsf{RS}}$, $\mathcal{S}$ executes $\mathsf{VfyProd}(P_j, prod, ppk)$ as a local algorithm and responds as defined in the protocol. If $(P_j, M_j) \notin L_{CA}$, $\mathcal{S}$ does not respond to this request, which means that $\mathcal{F}_{\mathsf{RS}}$ waits infinitely for a response.

Remark: This product-verification is done whenever a rating is verified. It implies that all subsequent requests based on the product-verification are ignored, whenever this verification cannot be executed due to missing parameters $(P_j, M_j) \notin L_{CA}$.

Simulation of Rating a Product:

- P_{SM} is honest: In this case $\mathcal{S}$ cannot execute the protocol because the identity of the rater is not known. Hence, $\mathcal{S}$ computes an accepting transcript of the underlying Σ-protocol, as follows: When receiving $(\mathsf{Rate}, sid, pp, P_j, prod, ppk, m)$ from $\mathcal{F}_{\mathsf{RS}}$, $\mathcal{S}$ uses the Zero-Knowledge simulator given in the proof of Lemma 8.4. During the simulation, $\mathcal{S}$ tries to patch the random oracle. If there is some value v such that $\big[(T_1, T_2, T_3, T_4, T_5, R_1, R_2, R_3, prod, ppk, m), v\big] \in L_{RO}$ then $\mathcal{S}$ outputs `error` and halts. Otherwise, $\mathcal{S}$ sets $L_{RO}.\mathsf{Add}\big[(T_1, T_2, T_3, T_4, T_5, R_1, R_2, R_3, prod, ppk, m), ch\big]$, $\sigma := (T_1, T_2, T_3, T_4, T_5, ch, s)$, $\mathfrak{Ratings}.\mathsf{Add}(pp, \bot, P_j, prod, ppk, m, \sigma)$ and outputs $(\mathsf{Rate}, sid, pp, P_j, prod, ppk, m, \sigma)$ to $\mathcal{F}_{\mathsf{RS}}$.
- P_{SM} is corrupted: When receiving $(\mathsf{Rate}, sid, pp, P_i, P_j, prod, ppk, m)$ from $\mathcal{F}_{\mathsf{RS}}$, $\mathcal{S}$ generates an accepting transcript of the underlying Σ-protocol as a rating using $(P_i, M_i, usk_i) \in L_{sCA}$, $(P_i, pp, M_i, \tilde{Y}_i, \sigma_i) \in \mathfrak{Reg}$, and $(P_i, P_j, prod, ppk, \sigma_{i,j,prod})$. Finally, $\mathcal{S}$ sets $\mathfrak{Ratings}.\mathsf{Add}(pp, P_i, P_j, prod, ppk, m, \sigma)$ and outputs $(\mathsf{Rate}, sid, pp, P_i, P_j, prod, ppk, m, \sigma)$ to $\mathcal{F}_{\mathsf{RS}}$.

Simulation of Rating Verification: When $\mathcal{S}$ receives (Verify, sid, pp, P_j, $prod$, ppk, m, σ) from $\mathcal{F}_{\mathsf{RS}}$, $\mathcal{S}$ executes the verification protocol. Once the value f is obtained from the protocol (which implies that $\mathsf{VfyProd}$ returned 1), $\mathcal{S}$ tries to determine the author of the rating:

Parse σ as $(T_1, T_2, T_3, T_4, T_5, ch, s)$.

If $f = 0$ **Then** $\mathcal{S}$ outputs $(\mathsf{Verify}, sid, pp, P_j, prod, ppk, m, \sigma, 0, \bot)$ to $\mathcal{F}_{\mathsf{RS}}$. As defined in $\mathcal{F}_{\mathsf{RS}}$, invalid ratings will never be opened.

Else If $(pp, \bot, P_j, prod, ppk, m, \sigma) \in \mathfrak{Ratings}$ **Then** output (Verify, sid, pp, P_j, $prod$, ppk, m, σ, 1, $\bot$) to $\mathcal{F}_{\mathsf{RS}}$. In this case, P_{SM} and the author of the rating P_i are honest - $\mathcal{F}_{\mathsf{RS}}$ will include the correct identity for the rating.

Else If P_{SM} is honest and there exists a tuple $(P_k, pp, M_k, \tilde{Y}_k, \sigma_k) \in \mathfrak{Reg}$ such that $\mathrm{e}(T_5, \tilde{Y}) = \mathrm{e}(\mathcal{H}_1(j, prod), \tilde{Y}_k)$ **Then** Output (Verify, sid, pp, P_j, $prod$, ppk, m, σ, 1, P_k). This covers the case that a rating was created by a corrupted party, while P_{SM} is honest. Hence, $\mathcal{S}$ can use the registration information to determine the raters identity.

Else Output $(\mathsf{Verify}, sid, pp, P_j, prod, ppk, m, \sigma, 1, \bot)$ to $\mathcal{F}_{\mathsf{RS}}$. In the case that P_{SM} is corrupted, $\mathcal{S}$ is not requested to output the correct identity.

Simulation of Linking Ratings: When $\mathcal{S}$ receives (Link, sid, pp, P_j, $prod$, ppk, m_1, σ_1, m_2, σ_2) from $\mathcal{F}_{\mathsf{RS}}$, $\mathcal{S}$ executes the linking protocol as defined in Protocol $\Pi_{\mathsf{RS}}.\mathsf{Link}$ and outputs $(\mathsf{Link}, sid, pp, P_j, prod, ppk, m_1, \sigma_1, m_2, \sigma_2, b)$ to $\mathcal{F}_{\mathsf{RS}}$, where b is the bit computed during the protocol execution. This implies that $\mathsf{VfyProd}$ returned 1.

Simulation of Determining the Raters Identity: Since $\mathcal{F}_{\mathsf{RS}}$ already asked $\mathcal{S}$ for the raters identity during verification, $\mathcal{S}$ is not involved in this step. Hence, $\mathcal{S}$ does not need to simulate something.

Simulation of Generating Opening-Proofs: When $\mathcal{S}$ receives (OProof, sid, pp, P_j, $prod$, ppk, m, σ, P) from $\mathcal{F}_{\mathsf{RS}}$, $\mathcal{S}$ computes $f := (\mathsf{Verify}, sid, pp, P_j, prod, ppk, m, \sigma)$. If $f = 0$, $\mathcal{S}$ sends (OProof, sid, pp, P_j, $prod$, ppk, m, σ, P, $\bot$) to $\mathcal{F}_{\mathsf{RS}}$. Otherwise, it is

possible that $(pp, \bot, P_j, prod, ppk, m, \sigma) \in \mathfrak{Ratings}$ - meaning $\mathcal{S}$ simulated this rating for an honest but unknown party. In this case, $\mathcal{S}$ has to simulate an opening-proof such that P is accepted as the author of the rating. To do so, $\mathcal{S}$ uses the tuple $(P, M) \in L_{CA}$, which has to exist because the rating is valid ($f = 1$), and executes the Zero-Knowledge simulator given in the proof of Lemma 8.5. During the simulation, $\mathcal{S}$ tries to patch the random oracle. If there is some value v such that $\big[(ct_1,\ ct_2,\ ct_3,\ ct_4,\ R_1,\ R_2,\ R_3,\ R_4,\ R_5,\ \sigma,\ i,\ M),\ v\big] \in L_{RO}$ then $\mathcal{S}$ outputs `error` and halts. Otherwise, $\mathcal{S}$ sets $L_{RO}.\mathsf{Add}\big[(ct_1,\ ct_2,\ ct_3,\ ct_4,\ R_1,\ R_2,\ R_3,\ R_4,\ R_5,\ \sigma,\ i,\ M),\ \hat{ch}\big]$ and $\tau \coloneqq (P, ct_1, ct_2, ct_3, ct_4, \hat{ch}, \hat{s})$ and outputs $(\mathsf{OProof}, sid, pp, P_j, prod, ppk, m, \sigma, P, \tau)$ to $\mathcal{F}_{\mathsf{RS}}$.

In the case $(pp, \bot, P_j, prod, ppk, m, \sigma) \notin \mathfrak{Ratings}$ $\mathcal{S}$ did not simulate the rating. Hence, $\mathcal{S}$ executes the opening protocol according to $\Pi_{\mathsf{RS}}.\mathsf{OProof}$ and outputs (OProof, sid, pp, P_j, $prod$, ppk, m, σ, P, τ) to $\mathcal{F}_{\mathsf{RS}}$.

Note that $\mathcal{S}$ creates the proof for party P, even if the rating was not opened yet or P was not the author of the rating. In both cases, $\mathcal{F}_{\mathsf{RS}}$ will ignore the proof and output $(\mathsf{OProof}, sid, pp, P_j, prod, ppk, m, \sigma, P, \bot)$, as expected.

Simulation of Opening-Proof Verification: When $\mathcal{S}$ receives (Judge, sid, pp, P_j, $prod$, ppk, m, σ, P, τ) from $\mathcal{F}_{\mathsf{RS}}$, $\mathcal{S}$ executes the opening-proof verification protocol as defined in Protocol $\Pi_{\mathsf{RS}}.\mathsf{Judge}$ and outputs (Judge, sid, pp, P_j, $prod$, ppk, m, σ, P, τ, f) to $\mathcal{F}_{\mathsf{RS}}$, where f is the bit computed during the protocol execution.

8.2.3 Indistinguishability of the Ideal and Real Protocols

Hybrid game $\mathcal{G}$

In this game the ideal functionality always gives $\mathcal{S}_1$ the identifying information during rating requests, i.e. instead of sending (Rate, sid, pp, P_j, $prod$, ppk, m) when P_{SM} is honest, the ideal functionality sends (Rate, sid, pp, P_i, P_j, $prod$, ppk, m) to $\mathcal{S}_1$ both when P_{SM} is honest and corrupted. $\mathcal{S}_1$ works exactly as $\mathcal{S}$ except when simulating ratings for honest parties and when simulating opening-proofs. To simulate ratings - both when P_{SM} is honest and corrupted - $\mathcal{S}_1$ executes the same protocol as $\mathcal{S}$ does for corrupted P_{SM}, which is possible because $\mathcal{S}_1$ knows the identity of the honest rater and can use its key material to generate a rating. To simulate the opening-proof generation, $\mathcal{S}_1$ executes the same protocol as $\mathcal{S}$ does for the case $(pp, \bot, P_j, prod, ppk, m, \sigma) \notin \mathfrak{Ratings}$. Even for ratings that $\mathcal{S}_1$ created for honest parties correct opening-proofs can be generated, because $\mathcal{S}_1$ used the correct identifying information for these ratings.

Indistinguishability of IDEAL and $\mathcal{G}$

We need to show that ratings and opening-proofs generated by $\mathcal{S}$ and $\mathcal{S}_1$ are indistinguishable. These are the only differences of the algorithms.

The rating protocol $\mathcal{S}$ executes for honest P_{SM} uses the zero-knowledge simulator of the Σ-protocol that underlies the signature of knowledge for rating products in Protocol Π_{RS},

extended by patching $\mathcal{F}_{\mathsf{RO}}$ to generate valid ratings (see Lemma 8.4 for details). Assuming DDH is hard in $\mathbb{G}_1$ the tuples $(\sigma_{i,1}, \sigma_{i,2}, T_1, T_2)$, $(\sigma_{i,j,prod,1}, \sigma_{i,j,prod,2}, T_3, T_4)$, and $(g_1, M_i, \mathcal{H}_1(j, prod), T_5)$ generated by $\mathcal{S}$ are random DDH-instances, whereas such tuples generated by $\mathcal{S}_1$ are DDH-tuples. Given elements $(T_1, T_2, T_3, T_4, T_5) \in \mathbb{G}_1^5$ the Σ-protocol can be simulated perfectly and $\mathcal{S}$ outputs ratings that are indistinguishable from ratings $\mathcal{S}_1$ outputs, assuming patching $\mathcal{F}_{\mathsf{RO}}$ does not fail.

Analogously to the simulation of ratings, the protocol for generating opening-proofs executed by $\mathcal{S}$ uses the zero-knowledge simulator for the Σ-protocol that underlies the signature of knowledge for generating an opening-proof in Protocol Π_{RS}, extended by patching $\mathcal{F}_{\mathsf{RO}}$ to generate valid proofs (see Lemma 8.5 for details). The Cramer-Shoup encryption scheme is CCA2-secure under the DDH assumption, which means that no adversary can distinguish a valid encryption $ct = (ct_1, ct_2, ct_3, ct_4)$ from completely random tuples. Given $(ct_1, ct_2, ct_3, ct_4) \in \mathbb{G}_2^4$ the Σ-protocol can be simulated perfectly and $\mathcal{S}$ outputs opening-proofs that are indistinguishable from opening-proofs $\mathcal{S}_1$ outputs, assuming patching $\mathcal{F}_{\mathsf{RO}}$ does not fail.

Patching $\mathcal{F}_{\mathsf{RO}}$ only fails with negligible probability, because the challenge-values $ch, \hat{ch}$ used for the simulation of the Σ-protocols are chosen uniformly and independently at random. Since L_{RO} only contains polynomially many entries, choosing some values $ch, \hat{ch}$ that are already stored in L_{RO} as v only happens with negligible probability. Hence, IDEAL and $\mathcal{G}$ are indistinguishable.

Indistinguishability of $\mathcal{G}$ and HYBRID

By definition, $\mathcal{S}_1$ executes exactly the same operations as honest parties do when running Protocol Π_{RS}. Hence, the only way for $\mathcal{Z}$ to distinguish between $\mathcal{G}$ and HYBRID is to force $\mathcal{F}_{\mathsf{RS}}$ to output values that differ significantly from the values output by Π_{RS}. This only happens when $\mathcal{F}_{\mathsf{RS}}$ outputs `error` and halts, whereas the Protocol Π_{RS} outputs some value, or $\mathcal{F}_{\mathsf{RS}}$ outputs 0, whereas Protocol Π_{RS} outputs 1 (or vice versa). We show that every environment $\mathcal{Z}$ can do this only with negligible probability, which results in the indistinguishability of $\mathcal{G}$ and HYBRID.

Registry Key Generation: $\mathcal{F}_{\mathsf{RS}}$ always outputs the values obtained from $\mathcal{S}_1$. Since $\mathcal{S}_1$ behaves exactly as Protocol Π_{RS}, the outputs of $\mathcal{G}$ and HYBRID are indistinguishable.

User Registration: $\mathcal{F}_{\mathsf{RS}}$ enforces the outcome of this protocol only in Steps 3 and 4. For both conditions Protocol Π_{RS} generates exactly the same outputs as $\mathcal{F}_{\mathsf{RS}}$.

Hence, $\mathcal{G}$ and HYBRID are indistinguishable.

Product Addition: In Step 3, $\mathcal{F}_{\mathsf{RS}}$ outputs `error` and halts, when $\mathcal{S}_1$ outputs some $ppk_{i,prod} = (M_i, M_{i,prod}, ch_{i,prod}, s_{i,prod}, \tilde{g}_{i,prod}, \tilde{X}_{i,prod}, \tilde{Y}_{i,prod})$ that is already registered for some other party P_j or for another product $prod'$. If this happens, it must hold that $\mathcal{H}_2(i, prod) = \mathcal{H}_2(j, prod) \vee \mathcal{H}_2(i, prod) = \mathcal{H}_2(i, prod') \vee \mathcal{H}_2(i, prod) = \mathcal{H}_2(j, prod')$. In any case, this would be a collision in the collision-resistant hash function $\mathcal{H}_2$. Analogously we can argue for $\mathcal{H}_1$. Furthermore, $ppk_{i,prod}$ includes a proof of knowledge of the value usk_i, which is chosen uniformly and independently for honest parties. Hence, the probability that $ppk_{i,prod}$ is already registered for some

other party or some other product is negligible, implying that $\mathcal{G}$ and HYBRID are indistinguishable.

Purchase: The purchasing request will be ignored, according to Step 2 in $\mathcal{F}_{\mathsf{RS}}$, when $P_i = P_j$ or when VfyProd returned 0. The same is done in Protocol Π_{RS}. If P_j is corrupted and did not register (P_j, M_j) to $\mathcal{F}_{\mathsf{CA}}$, $\mathcal{S}_1$ is not able to execute the VfyProd-algorithm and ignores the request. This means that $\mathcal{F}_{\mathsf{RS}}$ gets no response from $\mathcal{S}_1$ and hence does not execute the purchasing protocol. That implies that $\mathcal{G}$ and HYBRID are indistinguishable.

Only in Step 4 the outcome of the purchasing protocol is fixed by $\mathcal{F}_{\mathsf{RS}}$. But in that case $\mathcal{S}_1$ knows all information needed to execute the protocol in behalf of P_i and P_j as defined in Protocol Π_{RS} and outputs 1. Hence, $\mathcal{G}$ and HYBRID are indistinguishable.

VfyProd: The product verification is only used as a subprotocol within the purchasing protocol and all rating verifications - there is no direct activation from $\mathcal{Z}$ for this. Whenever VfyProd returns 0 the calling protocol will ignore the request, both in $\mathcal{F}_{\mathsf{RS}}$ and Protocol Π_{RS}. $\mathcal{S}_1$ exploits this behavior by ignoring the VfyProd-request, when a corrupted party did not register at $\mathcal{F}_{\mathsf{CA}}$. This will in turn ignore the request to the calling protocol. Hence, $\mathcal{G}$ and HYBRID are indistinguishable in that case.

During the product verification $\mathcal{F}_{\mathsf{RS}}$ could differ in the output from Protocol Π_{RS} in Steps 2, 3, and 4.

Step 2 ensures consistency. Since VfyProd is a deterministic algorithm in Protocol Π_{RS}, consistency is guaranteed.

Step 3 covers the case that a maliciously generated *ppk* would be accepted as honestly generated (by an honest party). We can prove that this happens only with negligible probability, via a reduction to the PS1-Problem. This reduction is given in Section 8.2.4.

In Step 4 $\mathcal{F}_{\mathsf{RS}}$ ensures that every *ppk* is only valid for exactly one party P_i and one product *prod*. Analogously to Product Addition, if *ppk* would also be valid for some party P_j and/or a product *prod*$'$, this breaks the collision-resistance of the hash functions $\mathcal{H}_1$ and $\mathcal{H}_2$. Hence, $\mathcal{G}$ and HYBRID are indistinguishable.

Rate a Product: To generate valid ratings $\mathcal{F}_{\mathsf{RS}}$ ensures in Step 2 that the party P_i is registered, purchased the product to rate - which implies that the product is valid - and did not rate the specified product yet. The same is checked in Protocol Π_{RS}. Step 5 covers consistency: the tuple (pp, P', P_j, *prod*, *ppk*, m, σ, 0, *lid*, *oid*), for some P', *lid*, *oid*, only exists in the list $\mathfrak{Ratings}$, if $(pp, P_j, prod, ppk, m, \sigma)$ is verified by $\mathcal{S}_1$ as invalid and was verified before the rating request occurred. This is because the only possibility to store σ as invalid is Step 4 during the verification of ratings in $\mathcal{F}_{\mathsf{RS}}$. During rating requests Protocol Π_{RS}, and hence the simulator $\mathcal{S}_1$, only generates valid ratings, i.e. the deterministic verification algorithm will output 1. Hence, the rating σ output by $\mathcal{S}_1$ cannot exist in $\mathfrak{Ratings}$ or is verified as valid. In both cases the tuple $(pp, P', P_j, prod, ppk, m, \sigma, 0, lid, oid)$ is not stored in $\mathfrak{Ratings}$. Hence, $\mathcal{G}$ and HYBRID are indistinguishable.

As a remark, in Game $\mathcal{G}$ the Simulator $\mathcal{S}_1$ always gets the identity of a rater and can generate ratings as defined in Protocol Π_{RS}.

Verifying a Rating: To verify ratings the VfyRtg-protocol is used. This subprotocol is also an essential tool during the Link, Open, OProof, and Judge protocols, because these protocols are only meaningful for valid ratings. During the VfyRtg-protocol the specified product is verified by the VfyProd-protocol. Hence, whenever $\mathcal{S}_1$ does not respond to a VfyProd request, also VfyRtg and its calling protocols will not proceed, which exploits the request ignoring behavior.

Analogously to Step 5 of the Rate protocol, Step 3 covers consistency. Since the verification algorithm in Protocol Π_{RS} is deterministic, two verification requests with the same input will generate the same output, as required by $\mathcal{F}_{\mathsf{RS}}$. The Steps 4–7 can only occur for ratings that were not generated by honest parties using the Rate-protocol.

Step 4 handles invalid ratings and self-ratings. The verification protocol in Protocol Π_{RS} covers the same cases: invalid ratings are recognized by the test $ch = \mathcal{F}_{\mathsf{RO}}(T_1, T_2, T_3, T_4, T_5, R_1, R_2, R_3, prod, ppk, m)$ and self-ratings are recognized by $T_5 \neq M_{j,prod}$.

Obviously, when P_{SM} and P_j are corrupted, it is possible to generate arbitrary tuples $(usk'_j, \sigma'_j, \sigma_{j,j,prod})$ such that $\mathsf{PS.Verify}(\mathsf{PS}.pk, usk'_j, \sigma'_j) = 1$ and $\mathsf{PS.Verify}(\mathsf{PS}.pk_{j,prod}, usk'_j, \sigma_{j,j,prod}) = 1$, but $\mathcal{H}_1(j, prod)^{usk'_j} \neq M_{j,prod}$. With these values P_j can rate his own product, such that the verification algorithm cannot detect it. But in this case $\mathcal{S}_1$ is not requested to output the identity of the signer ($P = \perp$). Hence, $\mathcal{F}_{\mathsf{RS}}$ either outputs 0 because the rating is invalid ($b = 0$), or continues executing the verification protocol. Since P_{SM} and P_j are considered as corrupted in this case, the Steps 5–7 do not occur. Hence, $\mathcal{F}_{\mathsf{RS}}$ will output 1 in Step 9, as it would also happen in Protocol Π_{RS}. When P_{SM} is honest and a corrupted P_j generates a valid rating $(T_1, T_2, T_3, T_4, T_5, ch, s)$ for his own product, this means that he proved knowledge of a value usk, such that (T_1, T_2) is a valid signature for message $usk \neq usk_j$ under the public key $\mathsf{PS}.pk$ from P_{SM} and that $T_5 \neq M_{j,prod}$. Since P_{SM} is honest, $\mathcal{S}_1$ has to output the identity of the rater. If $usk = usk_i$ for some honest party P_i, $\mathcal{S}_1$ will find an entry in $\mathfrak{Reg}$ that falsely identifies P_i as the rater. If P_{SM} does not find an identifying entry in $\mathfrak{Reg}$, $\mathcal{S}_1$ returns $\perp$ to $\mathcal{F}_{\mathsf{RS}}$. Both cases are discussed in Steps 6 and 7.

Step 5 ensures that rating a product of an honest party P_j is only possible after purchasing it. Every rating includes a proof of knowledge of a valid signature for some message usk under the public key $\mathsf{PS}.pk_{j,prod}$ from party P_j. This signature is handed to the rater during the Purchase-protocol, but in this case some corrupted party proved knowledge of such a signature without executing the Purchase-protocol. That means, the signature must be a forgery, which contradicts the EUF-CMA security of the signature scheme in use. We will prove this in detail via a reduction to the PS1-Problem. Hence, under the PS1-Problem $\mathcal{G}$ and HYBRID are indistinguishable in that case.

Step 6 ensures strong exculpability, meaning that it is not feasible to produce valid ratings in behalf of honest parties. Impersonating an honest party requires to find the parties' secret key *usk*. Using a reduction to the PS1-Problem, provided in Section 8.2.4, we can prove that this is only possible with negligible probability.

Step 7 ensures traceability, meaning that the identity of every rater can be determined from valid ratings. Being able to create valid but untraceable ratings requires to forge a signature of the EUF-CMA secure signature scheme used by P_{SM}. Analogously to Step 6, we can prove that such attacks are infeasible under the PS1-Problem.

Linking Ratings: Whenever $\mathcal{S}_1$ has to respond to a Link-request, we know that the specified product is valid (VfyProd returned 1) and $\mathcal{F}_{\mathsf{RS}}$ stored the ratings to link in the list $\mathfrak{Ratings}$. Both the simulator $\mathcal{S}_1$ and $\mathcal{F}_{\mathsf{RS}}$ return 0 to the Link request when at least one of the ratings is invalid. Hence, for the analysis of Link we only consider valid ratings that passed all verification-tests.

Now we analyze Link-requests.

Step 4 claims that ratings are unique. This is ensured by the Verify-request because $\mathcal{F}_{\mathsf{RS}}$ adds a rating only if it is not present in the list $\mathfrak{Ratings}$.

In Step 5 $\mathcal{F}_{\mathsf{RS}}$ enforces consistency. If two ratings are linkable, RebLDB is used to store this information and subsequent Link-requests for ratings of the same equivalence class must be linkable, too. In Protocol Π_{RS} for every party and a given product the element T_5 of a rating $\sigma = (T_1, T_2, T_3, T_4, T_5, ch, s)$ is a fixed value. Hence, all ratings for a fixed product with identical values T_5 belong to the same equivalence class and are linkable, as expected.

In Step 6 $\mathcal{F}_{\mathsf{RS}}$ has no information that could be used to link the given ratings. This can only happen when P_{SM} is corrupted and the ratings are also generated by corrupted parties. So the value obtained from the simulator $\mathcal{S}_1$ is used as output. Hence, Protocol Π_{RS} and $\mathcal{F}_{\mathsf{RS}}$ generate the same output.

Step 7 expresses that it must be infeasible to generate ratings that can be opened to different parties but are linkable. Since every rating $\sigma = (T_1, T_2, T_3, T_4, T_5, ch, s)$ in Protocol Π_{RS} includes a zero-knowledge proof of knowledge of some value *usk* such that (T_1, T_2) and (T_3, T_4) are valid signatures for message *usk* and the discrete logarithm of T_5 to the base $\mathcal{H}_1(j, prod)$ is the same value *usk*, this requirement holds. Hence, in this case Protocol Π_{RS} and $\mathcal{F}_{\mathsf{RS}}$ generate the same output.

Step 8 covers the case that it must be infeasible to generate a valid rating in behalf of an honest user. This is analogue to VfyRtg-Step 6 for an corrupted P_{SM} and we can also prove via a reduction to the PS1-Problem, given in Section 8.2.4, that this event does not occur.

Step 9 is analogue to Step 6.

The RebLDB-Step 18 is analogue to VfyRtg-Step 5 for corrupted P_{SM}. With a reduction to the PS1-Problem we can prove that this event does not occur. The proof is given in Section 8.2.4.

Summarizing the analysis of Link and RebLDB, $\mathcal{G}$ and HYBRID are indistinguishable.

Determine Raters Identity: $\mathcal{S}_1$ outputs the raters' identity within the Verify-protocol. Since $\mathcal{S}$ can correctly output the identity, also $\mathcal{S}_1$ can do this.

Generate Opening-Proofs: This activation is only of interest for honest P_{SM}. $\mathcal{F}_{\mathsf{RS}}$ and Protocol Π_{RS} only generate an opening-proof for valid ratings that were opened to the given party. For invalid or unopened ratings and when the given rater identity is incorrect both protocols output $\perp$. Hence, for the analysis of OProof we only consider valid ratings that passed all verification-tests. Step 7 covers consistency: an opening-proof that was once invalid cannot be made valid. Since the Judge-protocol is deterministic in Protocol Π_{RS} and OProof only generates valid proofs, this cannot happen. Hence, $\mathcal{G}$ and HYBRID are indistinguishable.

Verifying Opening-Proofs: If the rating is invalid, no party identity or no opening-proof is given, $\mathcal{F}_{\mathsf{RS}}$ and Protocol Π_{RS} both output 0. For the analysis of Judge we only consider valid ratings that passed all verification-tests. Step 6 expresses that P_{SM} is honest, the rating was generated by an honest party, or Step 18 occurred previously for this specific rating. The Steps 9 and 10 cover consistency, which is ensured by Protocol Π_{RS} because the verification of opening-proofs is deterministic. Every valid opening-proof for the correct identity will be verified as valid by Protocol Π_{RS}; invalid proofs will detected as those. Opening-proofs to a wrong identity can be detected by Protocol Π_{RS} because the real identity the proof was generated for is a part of τ. Step 11 covers non-frameability. Maliciously generated but valid opening-proofs cause $\mathcal{F}_{\mathsf{RS}}$ to output `error` and halt, when P_{SM} and P are honest. Via a reduction to the CCA2-security of the Cramer-Shoup encryption scheme, provided in Section 8.2.4, we can prove that such opening-proofs cannot be generated. Step 12 expresses that P_{SM} is corrupted and the given rating was generated by a corrupted party. Otherwise, $\mathcal{F}_{\mathsf{RS}}$ would know the identity of the rater. In Step 15 consistency is guaranteed, as in Steps 9 and 10. In Step 16 an honest user is accepted as the author of the given rating. Since $\mathcal{F}_{\mathsf{RS}}$ does not know the identity of that rating ($X = \perp$) the rating must be maliciously generated in behalf of an honest party. This is impossible as we will prove via a reduction to PS1-Problem in Section 8.2.4.

In Step 17 the simulator accepts the opening-proof as valid for party P. Hence, the identity P is stored to ensure consistency for future verification requests. The Steps 18 and 19 store the verified opening-proof for the given rating to ensure consistency for future verification requests.

As discussed above, using reductions to the PS1-Problem and a reduction to the CCA2-security of the Cramer-Shoup encryption scheme, we can conclude that no environment $\mathcal{Z}$ can distinguish between $\mathcal{G}$ and HYBRID. The reductions complete the proof and are given in the next section.

8.2.4 The Reductions used within the Security Proof

To complete the proof of Theorem 6.1 we have to show that no environment can use the Steps VfyProd.3, VfyRtg.5/6/7, RebLDB.18, LinkRtgs.8 and Judge.11/16 to its advantage. We prove this with several reductions using a proof of knowledge extractor which needs rewinding. In UC the environment is treated as an interactive distinguisher, i.e. rewinding

an interactive machine is not possible. This is not contradicting because we use rewinding to prove the indistinguishability of hybrid games and not within the simulation. The same technique was used within other UC-based proofs [Lin11; Bla+13; Gro04].

All proofs have the same structure: assuming there exists an environment $\mathcal{Z}$ that can distinguish between $\mathcal{G}$ and HYBRID, and given either a PS1-instance or a CCA2-challenger for the Cramer-Shoup encryption scheme, we define a simulator interacting with $\mathcal{F}_{\mathsf{RS}}$ in game $\mathcal{G}$ we use to find a solution to the given problem instance. Since we assume the PS1-Problem and the SXDH-Assumption hold, no such environment can exist.

Lemma 8.6:
If the PS1-Problem holds for bilinear group generator BiGrGen, then no environment can distinguish between $\mathcal{G}$ and HYBRID at Steps VfyProd.3, VfyRtg.6, LinkRtgs.8, or Judge.16. ◇

Proof. Assume that there exists an environment $\mathcal{Z}$ interacting with game $\mathcal{G}$ that is able to let $\mathcal{F}_{\mathsf{RS}}$ output `error` and halt at the Steps VfyProd.3, VfyRtg.6, LinkRtgs.8, or Judge.16 with non-negligible probability.

We will use this environment to define a simulator $\mathcal{S}_2$ that we can use to compute a solution to the Pointcheval-Sanders-Problem with non-negligible probability. The hash function $\mathcal{H}_1$ is treated as a random oracle.

We are given $\mathbb{GD}$ as the output of BiGrGen, $(g, Y, \tilde{g}, \tilde{X}, \tilde{Y})$ and unlimited access to oracle $\mathcal{O}$ from our challenger and have to output a tuple $(m^*, s, s^{x+m^*\cdot y})$ such that $s \neq 1_{\mathbb{G}_1}$ and m^* was not asked to $\mathcal{O}$. In the first part of the proof, we will describe how $\mathcal{S}_2$ interacts with $\mathcal{F}_{\mathsf{RS}}$ and handles the interaction with $\mathcal{Z}$ and the real-world adversary $\mathcal{A}$. In the second part we analyze $\mathcal{S}_2$.

Simulator $\mathcal{S}_2$ works as $\mathcal{S}_1$, except in the following cases:

Calls to $\mathcal{F}_{\mathsf{RO}}$: $\mathcal{S}_2$'s answers are generated the same way as $\mathcal{S}_1$ does. We will point out the situations in which $\mathcal{S}_2$ deviates from this policy.

Calls to $\mathcal{F}_{\mathsf{CRS}}$: $\mathcal{S}_2$ runs $\mathsf{PD.KeyGen}(\mathbb{GD})$ to obtain $\mathsf{PD}.pk := (u, v)$ and $\mathsf{PD}.td := \mathrm{dlog}_u(v)$. The common reference string is set to $(\mathbb{GD}, \mathsf{PD}.pk, \mathcal{H}, \mathcal{H}_1, \mathcal{H}_2)$ according to the definition of $\mathcal{F}_{\mathsf{CRS}}$ in $\mathcal{G}$.

Calls to $\mathcal{H}_1$: $\mathcal{S}_2$ manages the list $L_{\mathcal{H}_1}$ to respond identically to repeated requests. When some x is queried for the first time ($\mathcal{H}_1(x)$ is called for some $x \in \{0,1\}^*$), $\mathcal{S}_2$ chooses $\alpha_x \leftarrow\!\!\shortmid \mathbb{Z}_p$, computes $\hat{g}_x := g_1^{\alpha_x}$, and stores $(x, \alpha_x, \hat{g}_x)$ in $L_{\mathcal{H}_1}$. Finally, $\mathcal{S}_2$ hands $\hat{g}_x$ to the caller, as it is also done for repeated queries $\mathcal{H}_1(x)$, i.e. $(x, \alpha_x, \hat{g}_x) \in L_{\mathcal{H}_1}$.

Calls to $\mathcal{F}_{\mathsf{CA}}$: Whenever an honest party P_i is activated for the first time, $\mathcal{S}_2$ chooses $u_i \leftarrow\!\!\shortmid \mathbb{Z}_p$, computes $M_i := Y^{u_i}$ and sets $L_{CA}.\mathsf{Add}(P_i, M_i)$. Note that the user-secret-key usk_i is implicitly set to be $y \cdot u_i$ for an unknown y. Calls from corrupted parties are handled as defined for $\mathcal{S}_1$.

Registry Key Generation: For an honest P_{SM} $\mathcal{S}_2$ handles the KeyGen-requests as defined for $\mathcal{S}_1$. A corrupted P_{SM} is managed by adversary $\mathcal{A}$.

User Registration: For honest P_{SM} $\mathcal{S}_2$ works exactly as $\mathcal{S}_1$. $\mathcal{S}_2$ simulates the computations for an honest party P_i using the interactive simulator given in the proof of Lemma 8.2.

Product Addition: For honest party P_i, $\mathcal{S}_2$ sets $M_{i,prod} \coloneqq Y^{u_i \cdot \alpha_{i,prod}}$ and computes $\tilde{g}_{i,prod} \coloneqq \mathcal{H}_2(i, prod)$, where $\alpha_{i,prod}$ is set during the request $\mathcal{H}_1(i, prod)$. Then, $\mathcal{S}_2$ runs the algorithm $\mathsf{PS.KeyGen}(\mathbb{GD})$ to obtain $\mathsf{PS}.pk_{i,prod} \coloneqq (\tilde{g}_{i,prod}, \tilde{X}_{i,prod}, \tilde{Y}_{i,prod})$ and $\mathsf{PS}.sk_{i,prod} \coloneqq (\xi_{1_{i,prod}}, \xi_{2_{i,prod}})$ and simulates the non-interactive zero-knowledge proof of knowledge using the simulator given in the proof of Lemma 8.3. With these values, $\mathcal{S}_2$ runs the remaining steps defined in Protocol Π_{RS}.

Purchase: In behalf of an honest seller P_j, $\mathcal{S}_2$ behaves as $\mathcal{S}_1$. For an honest purchaser P_i, $\mathcal{S}_2$ uses the same simulator as during the Register-protocol (see Lemma 8.2).

VfyProd: $\mathcal{S}_2$ works exactly as $\mathcal{S}_1$.

Rate a Product: To simulate ratings for an honest party P_i (note that $\mathcal{S}_2$ obtains the identity from $\mathcal{F}_{\mathsf{RS}}$), $\mathcal{S}_2$ uses the values σ_i from the Register-protocol with P_{SM}, $\sigma_{i,j,prod}$ from the Purchase-protocol with P_j, $\alpha_{j,prod}$ chosen by $\mathcal{H}_1$, u_i chosen by $\mathcal{F}_{\mathsf{CA}}$, and Y given by the PS1-instance: choose $t_1, t_2, k \leftarrow\!\!\$\ \mathbb{Z}_p$ and compute $T_1 \coloneqq \sigma_{i,1}^{t_1}$, $T_2 \coloneqq \sigma_{i,2}^{t_1}$, $T_3 \coloneqq \sigma_{i,j,prod,1}^{t_2}$, $T_4 \coloneqq \sigma_{i,j,prod,2}^{t_2}$, $T_5 \coloneqq Y^{\alpha_{j,prod} \cdot u_i}$. With these values $\mathcal{S}_2$ simulates the zero-knowledge proof of knowledge as given in the proof of Lemma 8.4. Then $\mathcal{S}$ patches $\mathcal{F}_{\mathsf{RO}}$ by setting $L_{RO}.\mathsf{Add}(T_1, T_2, T_3, T_4, T_5, R_1, R_2, R_3, pp, prod, ppk, m), ch)$, sets $\sigma \coloneqq (T_1, T_2, T_3, T_4, T_5, ch, s)$ and outputs σ as the rating.

For all remaining protocols (Verify, Link, Open, OProof, Judge) $\mathcal{S}_2$ works exactly as $\mathcal{S}_1$.

Now we show how $\mathcal{S}_2$ can be used to find a solution to the given PS1-instance.

When $\mathcal{F}_{\mathsf{RS}}$ outputs `error` in VfyProd.3, we know that the party P_j is honest, ppk fulfills the verification equations defined in VfyProd, and P_j did not use the NewProduct-protocol to generate ppk.

Especially, for $ppk = (M_j, M_{j,prod}, ch_{j,prod}, s_{j,prod}, \tilde{g}_{j,prod}, \tilde{X}_{j,prod}, \tilde{Y}_{j,prod})$ the non-interactive zero-knowledge proof of knowledge $(M_j, M_{j,prod}, ch_{j,prod}, s_{j,prod})$ is valid.

Now we rewind the game $\mathcal{G}$ up to the point where $\mathcal{F}_{\mathsf{RO}}$ outputs $ch_{j,prod}$ for the first time.

In the rewound game, $\mathcal{S}_2$ lets $\mathcal{F}_{\mathsf{RO}}$ output a new value $ch'_{j,prod} \neq ch_{j,prod}$.

Eventually, $\mathcal{S}_2$ obtains some ppk' for the same pair $(j, prod)$ as in the first run of the game, where the non-interactive zero-knowledge proof of knowledge $(M_j, M_{j,prod}, ch'_{j,prod}, s'_{j,prod})$ is valid, too. Using the extractor of Lemma 8.3 we obtain $usk_i = y \cdot u_i$, where u_i is chosen by $\mathcal{F}_{\mathsf{CA}}$.

Furthermore, we can compute $y \coloneqq u_i^{-1} \cdot usk_i$ and use it to find a solution to the PS1-Problem.

When $\mathcal{F}_{\mathsf{RS}}$ outputs `error` in VfyRtg.6, LinkRtgs.8, or Judge.16, the given rating $\sigma = (T_1, T_2, T_3, T_4, T_5, ch, s)$ is valid and must be maliciously generated in behalf of an honest user P, as discussed previously.

We rewind the game $\mathcal{G}$ up to the point where $\mathcal{F}_{\mathsf{RO}}$ outputs c for the first time.

In the rewound game, $\mathcal{S}_2$ lets $\mathcal{F}_{\mathsf{RO}}$ output a new value $ch' \neq ch$.

Eventually, $\mathcal{S}_2$ obtains another valid rating $\sigma' = (T_1, T_2, T_3, T_4, T_5, ch', s')$ for the same P_j, $prod$, ppk and m. Using the extractor of Lemma 8.4 we obtain $usk = y \cdot u_i$, where u_i is chosen by $\mathcal{F}_{\mathsf{CA}}$.

Analogously to VfyProd.3, we can compute $y := u_i^{-1} \cdot usk$ and use this value to find a solution to the PS1-Problem.

To compute a solution to the PS1-Problem given the value y, we choose $m \leftarrow \mathbb{Z}_p$, query the oracle $\mathcal{O}(m)$ and obtain a pair $(\mathcal{H}_1, H_2) := (h, h^{x+m\cdot y}) \in \mathbb{G}_1$ for some unknown $x \in \mathbb{Z}_p$.

Then we set $H_3 := H_2 \cdot \mathcal{H}_1^{-m\cdot y} = h^{x+m\cdot y} \cdot h^{-m\cdot y} = h^x$, choose $r, m^* \leftarrow \mathbb{Z}_p$ and output $(m^*, \mathcal{H}_1^r, H_3^r \cdot \mathcal{H}_1^{r\cdot m^* \cdot y})$.

All outputs from $\mathcal{S}_2$ are distributed identically to the outputs of $\mathcal{S}_1$, assuming patching the random oracles does not fail. As argued previously, this only happens with negligible probability.

Hence, when $\mathcal{Z}$ can distinguish between the game $\mathcal{G}$ and HYBRID at the Steps VfyProd.3, VfyRtg.6, LinkRtgs.8, or Judge.16 we can solve the PS1-Problem with non-negligible probability. □

Lemma 8.7:
If the SXDH-Assumption holds for bilinear group generator BiGrGen, and hence the Cramer-Shoup encryption scheme is CCA2-secure in $\mathbb{G}_2$, then no environment can distinguish between $\mathcal{G}$ and HYBRID at Step Judge.11. ◇

Proof. Assume that there exists an environment $\mathcal{Z}$ interacting with game $\mathcal{G}$ that is able to let $\mathcal{F}_{\mathsf{RS}}$ output `error` at the Step Judge.11 with non-negligible probability.

We will use this environment to define a simulator $\mathcal{S}_3$ which we use to break the CCA2-security of the Cramer-Shoup encryption scheme. We use the LR-formulation for CCA2-security [BR07], which is equivalent to the standard CCA2-notion.

We are given $\mathbb{GD}$ as the output of BiGrGen, CS.$pk = (g_2, \tilde{h}, \tilde{b}, \tilde{d}, \tilde{f}, \mathcal{H})$, access to an encryption-oracle $\mathcal{LR}$ and access to a decryption-oracle $\mathcal{D}$ from our challenger. We have to output a bit b as a guess whether the left ($b = 0$) or the right ($b = 1$) message given to oracle $\mathcal{LR}$ was encrypted, under the limitation not to query $\mathcal{D}$ to decrypt some ciphertext produced by $\mathcal{LR}$.

In the first part of the proof, we will describe how $\mathcal{S}_3$ interacts with $\mathcal{F}_{\mathsf{RS}}$ and handles the interaction with $\mathcal{Z}$ and the real-world adversary $\mathcal{A}$. In the second part we analyze $\mathcal{S}_3$. Note that Step Judge.11 is only of interest for honest P_{SM}. A corrupted P_{SM} is always able to generate opening-proofs, because it controls the encryption scheme. Hence, we assume $\mathcal{A}$ does not corrupt P_{SM}.

Simulator $\mathcal{S}_3$ works as $\mathcal{S}_1$, except in the following cases:

Registry Key Generation: $\mathcal{S}_3$ generates $\mathsf{PS}.pk$ and $\mathsf{PS}.sk$ as $\mathcal{S}_1$ does and sets the public key of the Cramer-Shoup encryption to be $\mathsf{CS}.pk$ given from the challenger.

User Registration: When the honest P_{SM} interacts with an corrupted party P_i, $\mathcal{S}_3$ works exactly as $\mathcal{S}_1$, except when decrypting the value $ct = (ct_1, ct_2, ct_3, ct_4)$ obtained from party P_i. Here, $\mathcal{S}_3$ queries its decryption oracle $\mathcal{D}(ct)$ to obtain the value $\tilde{Y}_i$. When P_{SM} interacts with an honest party P_i, $\mathcal{S}_3$ queries the $\mathcal{LR}$-oracle with $1_{\mathbb{G}_2}$ and $\tilde{Y}_i = \tilde{Y}^{usk_i}$ as $\mathcal{LR}(1_{\mathbb{G}_2}, \tilde{Y}_i)$ to obtain the ciphertext $ct = (ct_1, ct_2, ct_3, ct_4)$ and uses it during the protocol execution.

For the protocols NewProduct, Purchase, VfyProd, Rate, Verify, Link, and Open $\mathcal{S}_3$ works exactly as $\mathcal{S}_1$, because there is no encryption involved.

Generating Opening-Proofs: When P_{SM} has to generate an opening-proof for a corrupted party P_i, it executes exactly the same protocol as $\mathcal{S}_1$. When P_{SM} has to generate an opening-proof for an honest party P_i, $\mathcal{S}_3$ runs the same verification checks as $\mathcal{S}_1$ does, queries $\mathcal{LR}(1_{\mathbb{G}_2}, \tilde{Y}_i)$ to obtain the ciphertext $ct = (ct_1, ct_2, ct_3, ct_4)$ and simulates the opening-proof using the simulator of Lemma 8.5. Then $\mathcal{S}_3$ patches $\mathcal{F}_{\mathsf{RO}}$, sets $\tau \coloneqq (P_i, ct_1, ct_2, ct_3, ct_4, \hat{ch}, \hat{s})$ and outputs τ.

Opening-Proof Verification: $\mathcal{S}_3$ works exactly as $\mathcal{S}_1$.

Now we show how $\mathcal{S}_3$ can be used to break the CCA2-security of the Cramer-Shoup encryption scheme.

We are working with Type-3 pairings where no map from $\mathbb{G}_1$ to $\mathbb{G}_2$ exists. Therefore, elements from $\mathbb{G}_1$ cannot be used to compute elements in $\mathbb{G}_2$ and we can concentrate on the group $\mathbb{G}_2$ during the analysis.

When $\mathcal{F}_{\mathsf{RS}}$ outputs `error` in Step 11 of $(\mathsf{Judge}, sid, pp, P_j, prod, ppk, m, \sigma, P_i, \tau)$ we know from the validity of the non-interactive zero-knowledge proofs of knowledge σ and τ that

$$T_5 = \mathcal{H}_1(j, prod)^{usk_i} \qquad (T_5 \text{ is given by } \sigma,)$$
$$ct_3 = \tilde{Z} \cdot \tilde{f}^{\beta} \qquad (ct_3 \text{ is given by } \tau,\ \beta \text{ is unknown})$$

and

$$\mathrm{e}(\mathcal{H}_1(j, prod)^{usk_i}, \tilde{Y}) = \mathrm{e}(\mathcal{H}_1(j, prod), \tilde{Z}), \qquad (\text{since } \tau \text{ is valid})$$

which is only possible, when $\tilde{Z} = \tilde{Y}_i$. This in turn means that the opening-proof contains the correct value $\tilde{Y}_i$ for party P_i.

The ciphertexts of $\tilde{Y}_i$ that were generated during the Register-protocol and for other opening-proofs for the same party are the only values that depend on $\tilde{Y}_i$. But these ciphertexts contain $\tilde{Y}_i$ only if the $\mathcal{LR}$-oracle encrypts the message on the right-hand side of a call ($b = 1$). Hence, we output $b' = 1$ as our guess to the CCA2-challenger.

When $\mathcal{F}_{\mathsf{RS}}$ never outputs `error`, meaning that it was not possible to maliciously produce an opening-proof, we output $b' = 0$ as our guess to the CCA2-challenger, because we assume that $1_{\mathbb{G}_2}$ was encrypted using $\mathcal{LR}$. In this case all ciphertexts are independent of $\tilde{Y}_i$, which implies that computing $\tilde{Y}_i$ is not possible.

All outputs from $\mathcal{S}_3$ are distributed identically to the outputs of $\mathcal{S}_1$, assuming patching the random oracle does not fail. As argued previously, this only happens with negligible probability. Hence, when $\mathcal{Z}$ can distinguish between the game $\mathcal{G}$ and HYBRID at the Step Judge.11 we can break the CCA2-security of the Cramer-Shoup encryption scheme with non-negligible probability. □

Lemma 8.8:
If the PS1-Problem holds for bilinear group generator BiGrGen, then no environment can distinguish between $\mathcal{G}$ and HYBRID at Steps VfyRtg.5/7 and RebLDB.18. ◇

Proof. Assume that there exists an environment $\mathcal{Z}$ interacting with game $\mathcal{G}$ that is able to let $\mathcal{F}_{\mathsf{RS}}$ output `error` at the Steps VfyRtg.5, VfyRtg.7 or RebLDB.18 with non-negligible probability.

We will use this environment to define a simulator $\mathcal{S}_4$ that we can use to compute a solution to the Pointcheval-Sanders-Problem with non-negligible probability. The hash function $\mathcal{H}_2$ is treated as a random oracle.

We are given $\mathbb{GD}$ as the output of BiGrGen, $(g, Y, \tilde{g}, \tilde{X}, \tilde{Y})$ and unlimited access to oracle $\mathcal{O}$ from our challenger and have to output a tuple $(m^*, s, s^{x+m^*\cdot y})$ such that $s \neq 1_{\mathbb{G}_1}$ and m^* was not asked to $\mathcal{O}$. In the first part of the proof, we will describe how $\mathcal{S}_4$ interacts with $\mathcal{F}_{\mathsf{RS}}$ and handles the interaction with $\mathcal{Z}$ and the real-world adversary $\mathcal{A}$. In the second part we analyze $\mathcal{S}_4$

Simulator $\mathcal{S}_4$ works as $\mathcal{S}_1$, except in the following cases:

Calls to $\mathcal{F}_{\mathsf{RO}}$: $\mathcal{S}_4$'s answers are generated the same way as $\mathcal{S}_1$ does.

Calls to $\mathcal{F}_{\mathsf{CRS}}$: $\mathcal{S}_4$ runs $\mathsf{PD.KeyGen}(\mathbb{GD})$ to obtain $\mathsf{PD}.pk := (u, v)$ and $\mathsf{PD}.td := \mathrm{dlog}_u(v)$.

The common reference string is set to ($\mathbb{GD}$, $\mathsf{PD}.pk$, $\mathcal{H}$, $\mathcal{H}_1$, $\mathcal{H}_2$) according to the definition of $\mathcal{F}_{\mathsf{CRS}}$ in $\mathcal{G}$.

Calls to $\mathcal{F}_{\mathsf{CA}}$: $\mathcal{S}_4$ works exactly as $\mathcal{S}_1$.

Calls to $\mathcal{H}_2$: $\mathcal{S}_4$ manages the list $L_{\mathcal{H}_2}$ to respond identically to repeated requests. When some x is queried for the first time ($\mathcal{H}_2(x)$ is called for some $x \in \{0,1\}^*$), $\mathcal{S}_4$ chooses $\alpha_x \leftarrow_u \mathbb{Z}_p$, computes $\tilde{g}_x := \tilde{g}^{\alpha_x}$, and stores $(x, \alpha_x, \tilde{g}_x)$ in $L_{\mathcal{H}_2}$. Finally, $\mathcal{S}_4$ hands $\tilde{g}_x$ to the caller, as it is also done for repeated queries $\mathcal{H}_2(x)$, i.e. $(x, \alpha_x, \tilde{g}_x) \in L_{\mathcal{H}_1}$. Note that $\tilde{g}$ is used here, which is given by the PS1-Problem instance.

Registry Key Generation: For an honest P_{SM} $\mathcal{S}_4$ sets $\mathsf{PS}.pk := (\tilde{g}, \tilde{X}, \tilde{Y})$, runs the algorithm $\mathsf{CS.KeyGen}(\mathbb{GD})$ to obtain $\mathsf{CS}.pk$ and $\mathsf{CS}.sk$, sets $pp := (\mathsf{PS}.pk, \mathsf{CS}.pk)$ and outputs pp. A corrupted P_{SM} is managed by adversary $\mathcal{A}$.

User Registration: For an honest party P_i, $\mathcal{S}_4$ works exactly as $\mathcal{S}_1$.

For an honest P_{SM} interacting with an honest party P_i, $\mathcal{S}_4$ executes the operations defined in Protocol Π_{RS}, but instead of computing a signature (σ_1, σ_2) itself, $\mathcal{S}_4$ queries its oracle $\mathcal{O}(usk_i)$, with usk_i given by $\mathcal{F}_{\mathsf{CA}}$, to obtain a valid signature for the registering party.

For an honest P_{SM} interacting with a corrupted party P_i, $\mathcal{S}_4$ executes Protocol Π_{RS} up to the point where P_{SM} has to generate a signature for P_i. Now we rewind the adversary $\mathcal{A}$ up to the point where it sent its first message (pp', R) in behalf of P_i and respond with a new random challenge $ch' \neq ch$ (the same technique is used in [Bla+13]). Now, we extract usk_i, query $\mathcal{O}(usk_i)$ to obtain a valid signature for party P_i, and finalize the interaction according to Protocol Π_{RS}.

Product Addition: For honest party P_i, $\mathcal{S}_4$ chooses $\beta_{i,prod}, \gamma_{i,prod} \xleftarrow{R} \mathbb{Z}_p$ and sets $\tilde{g}_{i,prod} := \mathcal{H}_2(i, prod) = \tilde{g}^{\alpha_{i,prod}}$, according to the random oracle $\mathcal{H}_2$, $\tilde{X}_{i,prod} := \tilde{X}^{\alpha_{i,prod} \cdot \beta_{i,prod}}$, $\tilde{Y}_{i,prod} := \tilde{Y}^{\alpha_{i,prod} \cdot \beta_{i,prod} \cdot \gamma_{i,prod}}$, where $\tilde{g}, \tilde{X}, \tilde{Y}$ are given by the PS1-Problem instance. Then, $\mathcal{S}_4$ generates the non-interactive zero-knowledge proof of knowledge and outputs $ppk_{i,prod}$ as defined in Protocol Π_{RS}.

Purchase: For an honest party P_i, $\mathcal{S}_4$ works exactly as $\mathcal{S}_1$.

For an honest party P_j interacting with an honest party P_i, $\mathcal{S}_4$ executes the operations defined in Protocol Π_{RS}, but instead of computing a signature $\sigma_{i,j,prod}$ itself, $\mathcal{S}_4$ queries its oracle $\mathcal{O}(\gamma_{j,prod} \cdot usk_i)$, with $\gamma_{j,prod}$ chosen during $\mathsf{NewProduct}$ and usk_i given by $\mathcal{F}_{\mathsf{CA}}$, to obtain a pair (σ'_1, σ'_2). Then $\mathcal{S}_4$ sets $\sigma_{i,j,prod} := (\sigma'_1, \sigma_2'^{\beta_{j,prod}})$ and finalizes Protocol Π_{RS}.

For an honest party P_j interacting with a corrupted party P_i, $\mathcal{S}_4$ executes Protocol Π_{RS} up to the point where P_j has to generate a signature for P_i. Now we rewind the adversary $\mathcal{A}$ up to the point where it sent its first message $(prod, ppk, R)$ in behalf of P_i and respond with new random challenge $ch' \neq ch$ (the same technique is used in [Bla+13]). Now, we extract usk_i, query $\mathcal{O}(\gamma_{j,prod} \cdot usk_i)$ to obtain a pair (σ'_1, σ'_2), set $\sigma_{i,j,prod} := (\sigma'_1, \sigma_2'^{\beta_{j,prod}})$, and finalize the interaction according to Protocol Π_{RS}.

For all remaining protocols ($\mathsf{VfyProd}$, Rate, Verify, Link, Open, OProof, Judge) $\mathcal{S}_4$ works exactly as $\mathcal{S}_1$.

Now we show how $\mathcal{S}_4$ can be used to find a solution to the given PS1-instance.

When $\mathcal{F}_{\mathsf{RS}}$ outputs `error` in VfyRtg.5 some registered party P_i generated a valid rating $\sigma = (T_1, T_2, T_3, T_4, T_5, ch, s)$ without purchasing the corresponding product. We now rewind the whole game $\mathcal{G}$ up to the point where the random oracle $\mathcal{F}_{\mathsf{RO}}$ output ch for the first time. In the rewound game, $\mathcal{S}_4$ lets $\mathcal{F}_{\mathsf{RO}}$ output a new value $ch' \neq ch$. Eventually, $\mathcal{S}_4$ obtains a second valid rating $\sigma' = (T_1, T_2, T_3, T_4, T_5, ch', s')$ and we can compute usk_i using the extractor of Lemma 8.4. Furthermore, since σ and σ' are valid, (T_3, T_4) must be a valid signature for message usk_i under the public key $(\tilde{g}_{j,prod}, \tilde{X}_{j,prod}, \tilde{Y}_{j,prod})$:

$$\mathrm{e}(T_3, \tilde{X}_{j,prod} \cdot \tilde{Y}_{j,prod}^{usk_i}) = \mathrm{e}(T_3, \tilde{g}_{j,prod}^{x \cdot \beta_{j,prod}} \cdot \tilde{g}_{j,prod}^{y \cdot \beta_{j,prod} \cdot \gamma_{j,prod} \cdot usk_i}) = \mathrm{e}(T_4, \tilde{g}_{j,prod})$$

$$\Longrightarrow T_3^{x\cdot\beta_{j,prod}+y\cdot\beta_{j,prod}\cdot\gamma_{j,prod}\cdot usk_i} = T_4.$$

And we can compute

$$T_4^{1/\beta_{j,prod}} = T_3^{x+y\cdot\gamma_{j,prod}\cdot usk_i},$$

which is a valid signature for message $m = \gamma_{j,prod} \cdot usk_i$. Since oracle $\mathcal{O}$ is only queried for usk_i during Register and was not called during Purchase, we can output $(m, T_3, T_4^{1/\beta_{j,prod}})$ as a solution to the given PS1-Problem instance. The probability that m was already queried is negligible, because all values γ_x are chosen uniformly and independently at random.

When $\mathcal{F}_{\mathsf{RS}}$ outputs `error` in VfyRtg.7, a valid rating $\sigma = (T_1, T_2, T_3, T_4, T_5, ch, s)$ could not be opened by the honest P_{SM}, which means that the rating was generated in behalf of an unregistered party. Furthermore, we know that $T_5 = \mathcal{H}_1(j, prod)^{usk}$ for some $usk \in \mathbb{Z}_p$ and (T_1, T_2) is a valid signature for the message usk under the public key $\mathsf{PS}.pk = (\tilde{g}, \tilde{X}, \tilde{Y})$, because σ is valid. As described above, we rewind the whole game $\mathcal{G}$ to extract usk. $\mathcal{S}_4$ queries $\mathcal{O}$ only during the Register-protocol and the Purchase-protocol. Since P_{SM} cannot open the rating, $\mathcal{O}(usk)$ was not queried in the Register-protocol. The probability that $\mathcal{S}_4$ queried $\mathcal{O}(usk)$ in the Purchase-protocol, meaning $usk = \gamma_x \cdot usk_i$ for some γ_x and some usk_i, is negligible, because all values γ_x are chosen uniformly and independently at random. Hence, we can output (usk, T_1, T_2) as the solution to the given PS1-Problem instance.

When $\mathcal{F}_{\mathsf{RS}}$ outputs `error` in RebLDB.18, then there exist to many valid, but non-linkable ratings for the given product. Since two ratings $\sigma' = (T_1', T_2', T_3', T_4', T_5', ch', s')$, $\sigma'' = (T_1'', T_2'', T_3'', T_4'', T_5'', ch'', s'')$ are linkable, if and only if $T_5' = T_5''$, there must exist at least one rating $\sigma = (T_1, T_2, T_3, T_4, T_5, ch, s)$, where $T_5 = \mathcal{H}_1(j, prod)^{usk}$ for some usk that was not extracted during the Purchase-protocol. Rewinding the game $\mathcal{G}$ and extracting usk, we can output $(\gamma_{j,prod} \cdot usk, T_3, T_4^{1/\beta_{j,prod}})$ as the solution to the PS1-Problem instance, as described above.

All outputs from $\mathcal{S}_4$ are distributed identically to the outputs of $\mathcal{S}_1$, assuming patching the random oracles does not fail. As argued previously, this only happens with negligible probability. Hence, when $\mathcal{Z}$ can distinguish between the game $\mathcal{G}$ and HYBRID at the Steps VfyRtg.5/7 or RebLDB.18 we can solve the PS1-Problem with non-negligible probability. □

Bibliography

[Abe+10] Masayuki Abe, Georg Fuchsbauer, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. „Structure-Preserving Signatures and Commitments to Group Elements“. In: *Advances in Cryptology – CRYPTO 2010*. Ed. by Tal Rabin. Vol. 6223. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2010, pp. 209–236.

[And+08] Elli Androulaki, Seung Geol Choi, Steven M. Bellovin, and Tal Malkin. „Reputation Systems for Anonymous Networks“. In: *Privacy Enhancing Technologies, 8th International Symposium, PETS 2008, Leuven, Belgium, July 23-25, 2008, Proceedings*. Ed. by Nikita Borisov and Ian Goldberg. Vol. 5134. Lecture Notes in Computer Science. Springer, 2008, pp. 202–218. DOI: 10.1007/978-3-540-70630-4_13.

[Ate+00] Giuseppe Ateniese, Jan Camenisch, Marc Joye, and Gene Tsudik. „A Practical and Provably Secure Coalition-Resistant Group Signature Scheme“. In: *Advances in Cryptology – CRYPTO 2000*. Ed. by Mihir Bellare. Vol. 1880. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2000, pp. 255–270.

[Ate+05] Giuseppe Ateniese, Jan Camenisch, Susan Hohenberger, and Breno de Medeiros. *Practical Group Signatures without Random Oracles*. Cryptology ePrint Archive, Report 2005/385. http://eprint.iacr.org/2005/385. 2005.

[BB04] Dan Boneh and Xavier Boyen. „Short Signatures Without Random Oracles“. In: *Advances in Cryptology – EUROCRYPT 2004*. Ed. by Christian Cachin and Jan Camenisch. Vol. 3027. Lecture Notes in Computer Science. Interlaken, Switzerland: Springer, Heidelberg, Germany, May 2004, pp. 56–73.

[BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. „Short Group Signatures“. In: *Advances in Cryptology – CRYPTO 2004*. Ed. by Matthew Franklin. Vol. 3152. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2004, pp. 41–55.

[BEJ18] Johannes Blömer, Fabian Eidens, and Jakob Juhnke. „Practical, Anonymous, and Publicly Linkable Universally-Composable Reputation Systems“. In: *Topics in Cryptology – CT-RSA 2018*. Ed. by Nigel P. Smart. Vol. 10808. Lecture Notes in Computer Science. San Francisco, CA, USA: Springer, Heidelberg, Germany, Apr. 2018, pp. 470–490. DOI: 10.1007/978-3-319-76953-0_25. Full Version: Cryptology ePrint Archive, Report 2018/029. https://eprint.iacr.org/2018/029. 2018.

[BEJ18a] Johannes Blömer, Fabian Eidens, and Jakob Juhnke. „Enhanced Security of Attribute-Based Signatures“. In: *Cryptology and Network Security - 17th International Conference, CANS 2018, Naples, Italy, September 30 - October 3, 2018, Proceedings*. Ed. by Jan Camenisch and Panos Papadimitratos. Vol. 11124. Lecture Notes in Computer Science. Springer, 2018, pp. 235–255. DOI: 10.1007/978-3-030-00434-7_12.

[Bel+98] Mihir Bellare, Anand Desai, David Pointcheval, and Phillip Rogaway. „Relations Among Notions of Security for Public-Key Encryption Schemes“. In: *Advances in Cryptology – CRYPTO'98*. Ed. by Hugo Krawczyk. Vol. 1462. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1998, pp. 26–45.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. „Non-Interactive Zero-Knowledge and Its Applications (Extended Abstract)“. In: *20th Annual ACM Symposium on Theory of Computing*. Chicago, IL, USA: ACM Press, May 1988, pp. 103–112.

[BG93] Mihir Bellare and Oded Goldreich. „On Defining Proofs of Knowledge“. In: *Advances in Cryptology – CRYPTO'92*. Ed. by Ernest F. Brickell. Vol. 740. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1993, pp. 390–420.

[BJK15] Johannes Blömer, Jakob Juhnke, and Christina Kolb. „Anonymous and Publicly Linkable Reputation Systems“. In: *FC 2015: 19th International Conference on Financial Cryptography and Data Security*. Ed. by Rainer Böhme and Tatsuaki Okamoto. Vol. 8975. Lecture Notes in Computer Science. San Juan, Puerto Rico: Springer, Heidelberg, Germany, Jan. 2015, pp. 478–488. DOI: 10.1007/978-3-662-47854-7_29. Full Version: Cryptology ePrint Archive, Report 2014/546. http://eprint.iacr.org/2014/546. 2014.

[Bla+13] Olivier Blazy, Céline Chevalier, David Pointcheval, and Damien Vergnaud. „Analysis and Improvement of Lindell's UC-Secure Commitment Schemes“. In: *ACNS 13: 11th International Conference on Applied Cryptography and Network Security*. Ed. by Michael J. Jacobson Jr., Michael E. Locasto, Payman Mohassel, and Reihaneh Safavi-Naini. Vol. 7954. Lecture Notes in Computer Science. Banff, AB, Canada: Springer, Heidelberg, Germany, June 2013, pp. 534–551. DOI: 10.1007/978-3-642-38980-1_34.

[BLS04] Dan Boneh, Ben Lynn, and Hovav Shacham. „Short Signatures from the Weil Pairing“. In: *Journal of Cryptology* 17.4 (Sept. 2004), pp. 297–319.

[BMW03] Mihir Bellare, Daniele Micciancio, and Bogdan Warinschi. „Foundations of Group Signatures: Formal Definitions, Simplified Requirements, and a Construction Based on General Assumptions“. In: *Advances in Cryptology – EUROCRYPT 2003*. Ed. by Eli Biham. Vol. 2656. Lecture Notes in Computer Science. Warsaw, Poland: Springer, Heidelberg, Germany, May 2003, pp. 614–629.

[BN06] Mihir Bellare and Gregory Neven. „Multi-signatures in the plain public-Key model and a general forking lemma“. In: *ACM CCS 06: 13th Conference on Computer and Communications Security*. Ed. by Ari Juels, Rebecca N. Wright,

and Sabrina De Capitani di Vimercati. Alexandria, Virginia, USA: ACM Press, Oct. 2006, pp. 390–399.

[Bon98] Dan Boneh. „The Decision Diffie-Hellman Problem“. In: *Algorithmic Number Theory, Third International Symposium, ANTS-III, Portland, Oregon, USA, June 21-25, 1998, Proceedings*. Ed. by Joe Buhler. Vol. 1423. Lecture Notes in Computer Science. Springer, 1998, pp. 48–63. DOI: 10.1007/BFb0054851.

[BPW12] David Bernhard, Olivier Pereira, and Bogdan Warinschi. „How Not to Prove Yourself: Pitfalls of the Fiat-Shamir Heuristic and Applications to Helios“. In: *Advances in Cryptology – ASIACRYPT 2012*. Ed. by Xiaoyun Wang and Kazue Sako. Vol. 7658. Lecture Notes in Computer Science. Beijing, China: Springer, Heidelberg, Germany, Dec. 2012, pp. 626–643. DOI: 10.1007/978-3-642-34961-4_38.

[BR07] Mihir Bellare and Phillip Rogaway. *Introduction to Modern Cryptography, Chapter 7 (course notes)*. Can be found at http://cseweb.ucsd.edu/~mihir/cse207/w-asym.pdf. 2007.

[BR93] Mihir Bellare and Phillip Rogaway. „Random Oracles are Practical: A Paradigm for Designing Efficient Protocols“. In: *ACM CCS 93: 1st Conference on Computer and Communications Security*. Ed. by V. Ashby. Fairfax, Virginia, USA: ACM Press, Nov. 1993, pp. 62–73.

[BS04] Dan Boneh and Hovav Shacham. „Group Signatures With Verifier-Local Revocation“. In: *ACM CCS 04: 11th Conference on Computer and Communications Security*. Ed. by Vijayalakshmi Atluri, Birgit Pfitzmann, and Patrick McDaniel. Washington D.C., USA: ACM Press, Oct. 2004, pp. 168–177.

[BSZ05] Mihir Bellare, Haixia Shi, and Chong Zhang. „Foundations of Group Signatures: The Case of Dynamic Groups“. In: *Topics in Cryptology – CT-RSA 2005*. Ed. by Alfred Menezes. Vol. 3376. Lecture Notes in Computer Science. San Francisco, CA, USA: Springer, Heidelberg, Germany, Feb. 2005, pp. 136–153.

[Can+07] Ran Canetti, Ronald L. Rivest, Madhu Sudan, Luca Trevisan, Salil P. Vadhan, and Hoeteck Wee. „Amplifying Collision Resistance: A Complexity-Theoretic Treatment“. In: *Advances in Cryptology – CRYPTO 2007*. Ed. by Alfred Menezes. Vol. 4622. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2007, pp. 264–283.

[Can01] Ran Canetti. „Universally Composable Security: A New Paradigm for Cryptographic Protocols“. In: *42nd Annual Symposium on Foundations of Computer Science*. Las Vegas, NV, USA: IEEE Computer Society Press, Oct. 2001, pp. 136–145. Also consider the updated versions: Cryptology ePrint Archive, Report 2000/067. http://eprint.iacr.org/2000/067. 2000.

[Can04] Ran Canetti. „Universally Composable Signature, Certification, and Authentication“. In: *17th IEEE Computer Security Foundations Workshop, (CSFW-17 2004), 28-30 June 2004, Pacific Grove, CA, USA*. IEEE Computer Society, 2004, p. 219. DOI: 10.1109/CSFW.2004.24.

[CF01] Ran Canetti and Marc Fischlin. „Universally Composable Commitments“. In: *Advances in Cryptology – CRYPTO 2001*. Ed. by Joe Kilian. Vol. 2139. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2001, pp. 19–40.

[CGH04] Ran Canetti, Oded Goldreich, and Shai Halevi. „The random oracle methodology, revisited“. In: *Journal of the ACM* 51.4 (2004), pp. 557–594. DOI: 10.1145/1008731.1008734.

[Che+10] Shenlong Chen, Yuqing Zhang, Peng Liu, and Jingyu Feng. „Coping with Traitor Attacks in Reputation Models for Wireless Sensor Networks“. In: *Proceedings of the Global Communications Conference, 2010. GLOBECOM 2010, 6-10 December 2010, Miami, Florida, USA*. IEEE, 2010, pp. 1–6. DOI: 10.1109/GLOCOM.2010.5684005.

[CL06] Melissa Chase and Anna Lysyanskaya. „On Signatures of Knowledge“. In: *Advances in Cryptology – CRYPTO 2006*. Ed. by Cynthia Dwork. Vol. 4117. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2006, pp. 78–96.

[CS97] Jan Camenisch and Markus Stadler. „Efficient Group Signature Schemes for Large Groups (Extended Abstract)“. In: *Advances in Cryptology – CRYPTO'97*. Ed. by Burton S. Kaliski Jr. Vol. 1294. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1997, pp. 410–424.

[CS98] Ronald Cramer and Victor Shoup. „A Practical Public Key Cryptosystem Provably Secure Against Adaptive Chosen Ciphertext Attack“. In: *Advances in Cryptology – CRYPTO'98*. Ed. by Hugo Krawczyk. Vol. 1462. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1998, pp. 13–25.

[CSK13] Sebastian Clauß, Stefan Schiffner, and Florian Kerschbaum. „k-anonymous reputation“. In: *ASIACCS 13: 8th ACM Symposium on Information, Computer and Communications Security*. Ed. by Kefei Chen, Qi Xie, Weidong Qiu, Ninghui Li, and Wen-Guey Tzeng. Hangzhou, China: ACM Press, May 2013, pp. 359–368.

[Cv91] David Chaum and Eugène van Heyst. „Group Signatures“. In: *Advances in Cryptology – EUROCRYPT'91*. Ed. by Donald W. Davies. Vol. 547. Lecture Notes in Computer Science. Brighton, UK: Springer, Heidelberg, Germany, Apr. 1991, pp. 257–265.

[Dam00] Ivan Damgård. „Efficient Concurrent Zero-Knowledge in the Auxiliary String Model“. In: *Advances in Cryptology – EUROCRYPT 2000*. Ed. by Bart Preneel. Vol. 1807. Lecture Notes in Computer Science. Bruges, Belgium: Springer, Heidelberg, Germany, May 2000, pp. 418–430.

[Dam02] Ivan Damgård. *On σ-protocols*. 2002. URL: http://www.daimi.au.dk/~ivan/Sigma.ps.

[Dam88] Ivan Damgård. „Collision Free Hash Functions and Public Key Signature Schemes“. In: *Advances in Cryptology – EUROCRYPT'87*. Ed. by David Chaum and Wyn L. Price. Vol. 304. Lecture Notes in Computer Science. Amsterdam, The Netherlands: Springer, Heidelberg, Germany, Apr. 1988, pp. 203–216.

[De +01] Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. „Robust Non-interactive Zero Knowledge“. In: *Advances in Cryptology – CRYPTO 2001*. Ed. by Joe Kilian. Vol. 2139. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2001, pp. 566–598.

[Del00] Chrysanthos Dellarocas. „Immunizing online reputation reporting systems against unfair ratings and discriminatory behavior“. In: *Proceedings of the 2'nd ACM Conference on Electronic Commerce*. 2000, pp. 150–157. DOI: 10.1145/352871.352889.

[DG03] Ivan Damgård and Jens Groth. „Non-interactive and reusable non-malleable commitment schemes“. In: *35th Annual ACM Symposium on Theory of Computing*. San Diego, CA, USA: ACM Press, June 2003, pp. 426–437.

[DNS04] Cynthia Dwork, Moni Naor, and Amit Sahai. „Concurrent zero-knowledge“. In: *Journal of the ACM* 51.6 (2004), pp. 851–898. DOI: 10.1145/1039488.1039489.

[DNS98] Cynthia Dwork, Moni Naor, and Amit Sahai. „Concurrent Zero-Knowledge“. In: *30th Annual ACM Symposium on Theory of Computing*. Dallas, TX, USA: ACM Press, May 1998, pp. 409–418.

[Dou02] John R. Douceur. „The Sybil Attack“. In: *Peer-to-Peer Systems, First International Workshop, IPTPS 2002, Cambridge, MA, USA, March 7-8, 2002, Revised Papers*. Ed. by Peter Druschel, M. Frans Kaashoek, and Antony I. T. Rowstron. Vol. 2429. Lecture Notes in Computer Science. Springer, 2002, pp. 251–260. DOI: 10.1007/3-540-45748-8_24.

[Eze38] Mordecai Ezekiel. „The Cobweb Theorem“. In: *The Quarterly Journal of Economics* 52.2 (1938), pp. 255–280. ISSN: 00335533, 15314650. URL: http://www.jstor.org/stable/1881734.

[Fig+17] Núria Busom Figueres, Ronald Petrlic, Francesc Sebé, Christoph Sorge, and Magda Valls. „A privacy-preserving reputation system with user rewards“. In: *Journal of Network and Computer Applications* 80 (2017), pp. 58–66. DOI: 10.1016/j.jnca.2016.12.023.

[Fis01] Marc Fischlin. „Trapdoor commitment schemes and their applications“. PhD thesis. Goethe University Frankfurt, Frankfurt am Main, Germany, 2001. URL: http://zaurak.tm.informatik.uni-frankfurt.de/diss/data/src/00000229/00000229.pdf.gz.

[FLS99] Uriel Feige, Dror Lapidot, and Adi Shamir. „Multiple NonInteractive Zero Knowledge Proofs Under General Assumptions“. In: *SIAM Journal on Computing* 29.1 (1999), pp. 1–28. DOI: 10.1137/S0097539792230010.

[FS07] Eiichiro Fujisaki and Koutarou Suzuki. „Traceable Ring Signature“. In: *PKC 2007: 10th International Conference on Theory and Practice of Public Key Cryptography*. Ed. by Tatsuaki Okamoto and Xiaoyun Wang. Vol. 4450. Lecture Notes in Computer Science. Beijing, China: Springer, Heidelberg, Germany, Apr. 2007, pp. 181–200.

[FS87] Amos Fiat and Adi Shamir. „How to Prove Yourself: Practical Solutions to Identification and Signature Problems“. In: *Advances in Cryptology – CRYPTO'86*. Ed. by Andrew M. Odlyzko. Vol. 263. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1987, pp. 186–194.

[GK11] Michael T. Goodrich and Florian Kerschbaum. „Privacy-enhanced reputation-feedback methods to reduce feedback extortion in online auctions“. In: *First ACM Conference on Data and Application Security and Privacy, CODASPY 2011, San Antonio, TX, USA, February 21-23, 2011, Proceedings*. Ed. by Ravi S. Sandhu and Elisa Bertino. ACM, 2011, pp. 273–282. DOI: 10.1145/1943513.1943550.

[GM17] Jens Groth and Mary Maller. „Snarky Signatures: Minimal Signatures of Knowledge from Simulation-Extractable SNARKs“. In: *Advances in Cryptology – CRYPTO 2017, Part II*. Ed. by Jonathan Katz and Hovav Shacham. Vol. 10402. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2017, pp. 581–612.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. „A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks“. In: *SIAM Journal on Computing* 17.2 (1988), pp. 281–308. DOI: 10.1137/0217017.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. „The Knowledge Complexity of Interactive Proof Systems“. In: *SIAM Journal on Computing* 18.1 (1989), pp. 186–208. DOI: 10.1137/0218012.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. „How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority“. In: *19th Annual ACM Symposium on Theory of Computing*. Ed. by Alfred Aho. New York City, NY, USA: ACM Press, May 1987, pp. 218–229.

[GOS06] Jens Groth, Rafail Ostrovsky, and Amit Sahai. „Perfect Non-interactive Zero Knowledge for NP“. In: *Advances in Cryptology – EUROCRYPT 2006*. Ed. by Serge Vaudenay. Vol. 4004. Lecture Notes in Computer Science. St. Petersburg, Russia: Springer, Heidelberg, Germany, May 2006, pp. 339–358.

[GPS08] Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. „Pairings for cryptographers“. In: *Discrete Applied Mathematics* 156.16 (2008), pp. 3113–3121. DOI: 10.1016/j.dam.2007.12.010.

[Gro04] Jens Groth. „Evaluating Security of Voting Schemes in the Universal Composability Framework“. In: *ACNS 04: 2nd International Conference on Applied Cryptography and Network Security*. Ed. by Markus Jakobsson, Moti Yung, and Jianying Zhou. Vol. 3089. Lecture Notes in Computer Science. Yellow Mountain, China: Springer, Heidelberg, Germany, June 2004, pp. 46–60.

[Gro06] Jens Groth. „Simulation-Sound NIZK Proofs for a Practical Language and Constant Size Group Signatures“. In: *Advances in Cryptology – ASIACRYPT 2006*. Ed. by Xuejia Lai and Kefei Chen. Vol. 4284. Lecture Notes in Computer Science. Shanghai, China: Springer, Heidelberg, Germany, Dec. 2006, pp. 444–459.

[Gro09] Jens Groth. *Homomorphic Trapdoor Commitments to Group Elements*. Cryptology ePrint Archive, Report 2009/007. `http://eprint.iacr.org/2009/007`. 2009.

[GSW10] Essam Ghadafi, Nigel P. Smart, and Bogdan Warinschi. „Groth-Sahai Proofs Revisited“. In: *PKC 2010: 13th International Conference on Theory and Practice of Public Key Cryptography*. Ed. by Phong Q. Nguyen and David Pointcheval. Vol. 6056. Lecture Notes in Computer Science. Paris, France: Springer, Heidelberg, Germany, May 2010, pp. 177–192.

[Has+13] Omar Hasan, Lionel Brunie, Elisa Bertino, and Ning Shang. „A Decentralized Privacy Preserving Reputation Protocol for the Malicious Adversarial Model“. In: *IEEE Transactions on Information Forensics and Security* 8.6 (2013), pp. 949–962. DOI: `10.1109/TIFS.2013.2258914`.

[HM04] Dennis Hofheinz and Jörn Müller-Quade. „Universally Composable Commitments Using Random Oracles“. In: *TCC 2004: 1st Theory of Cryptography Conference*. Ed. by Moni Naor. Vol. 2951. Lecture Notes in Computer Science. Cambridge, MA, USA: Springer, Heidelberg, Germany, Feb. 2004, pp. 58–76.

[HM96] Shai Halevi and Silvio Micali. „Practical and Provably-Secure Commitment Schemes from Collision-Free Hashing“. In: *Advances in Cryptology – CRYPTO'96*. Ed. by Neal Koblitz. Vol. 1109. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1996, pp. 201–215.

[HZN09] Kevin J. Hoffman, David Zage, and Cristina Nita-Rotaru. „A survey of attack and defense techniques for reputation systems“. In: *ACM Computing Surveys* 42.1 (2009), 1:1–1:31. DOI: `10.1145/1592451.1592452`.

[IJ02] Roslan Ismail and Audun Jøsang. „The Beta Reputation System“. In: *15th Bled eConference: eReality: Constructing the eEconomy, Bled, Slovenia, June 17-19, 2002*. 2002, p. 41. URL: `http://aisel.aisnet.org/bled2002/41`.

[JG09] Audun Jøsang and Jennifer Golbeck. „Challenges for robust trust and reputation systems“. In: *Proceedings of the 5th International Workshop on Security and Trust Management (SMT 2009), Saint Malo, France*. Citeseer. 2009, p. 52.

[Ker09] Florian Kerschbaum. „A verifiable, centralized, coercion-free reputation system“. In: *Proceedings of the 2009 ACM Workshop on Privacy in the Electronic Society, WPES 2009, Chicago, Illinois, USA, November 9, 2009*. Ed. by Ehab Al-Shaer and Stefano Paraboschi. ACM, 2009, pp. 61–70. DOI: `10.1145/1655188.1655197`.

[KK18] Ali El Kaafarani and Shuichi Katsumata. „Anonymous Reputation Systems Achieving Full Dynamicity from Lattices“. In: *To appear in the Proceedings of the 22nd International Conference on Financial Cryptography and Data Security (FC)*. 2018. URL: `https://fc18.ifca.ai/preproceedings/87.pdf`.

[KL07] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman and Hall/CRC Press, 2007. ISBN: 978-1-58488-551-1.

[KT12] Eleni Koutrouli and Aphrodite Tsalgatidou. „Taxonomy of attacks and defense mechanisms in P2P reputation systems - Lessons for reputation system designers“. In: *Computer Science Review* 6.2-3 (2012), pp. 47–70. DOI: 10.1016/j.cosrev.2012.01.002.

[KY04] Aggelos Kiayias and Moti Yung. *Group Signatures: Provable Security, Efficient Constructions and Anonymity from Trapdoor-Holders*. Cryptology ePrint Archive, Report 2004/076. http://eprint.iacr.org/2004/076. 2004.

[Lin11] Yehuda Lindell. „Highly-Efficient Universally-Composable Commitments Based on the DDH Assumption“. In: *Advances in Cryptology – EUROCRYPT 2011*. Ed. by Kenneth G. Paterson. Vol. 6632. Lecture Notes in Computer Science. Tallinn, Estonia: Springer, Heidelberg, Germany, May 2011, pp. 446–466.

[Liu+11] Siyuan Liu, Jie Zhang, Chunyan Miao, Yin Leng Theng, and Alex C. Kot. „iCLUB: an integrated clustering-based approach to improve the robustness of reputation systems“. In: *10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2011), Taipei, Taiwan, May 2-6, 2011, Volume 1-3*. Ed. by Liz Sonenberg, Peter Stone, Kagan Tumer, and Pinar Yolum. IFAAMAS, 2011, pp. 1151–1152. URL: http://portal.acm.org/citation.cfm?id=2034462%5C&CFID=69154334%5C&CFTOKEN=45298625.

[MAB09] Zaki Malik, Ihsan Akbar, and Athman Bouguettaya. „Web Services Reputation Assessment Using a Hidden Markov Model“. In: *Service-Oriented Computing, 7th International Joint Conference, ICSOC-ServiceWave 2009, Stockholm, Sweden, November 24-27, 2009. Proceedings*. Ed. by Luciano Baresi, Chi-Hung Chi, and Jun Suzuki. Vol. 5900. Lecture Notes in Computer Science. 2009, pp. 576–591. DOI: 10.1007/978-3-642-10383-4_42.

[MBD12] Stephen Marsh, Anirban Basu, and Natasha Dwyer. „Rendering unto Cæsar the Things That Are Cæsar's: Complex Trust Models and Human Understanding“. In: *Trust Management VI - 6th IFIP WG 11.11 International Conference, IFIPTM 2012, Surat, India, May 21-25, 2012. Proceedings*. Ed. by Theo Dimitrakos, Rajat Moona, Dhiren R. Patel, and D. Harrison McKnight. Vol. 374. IFIP Advances in Information and Communication Technology. Springer, 2012, pp. 191–200. DOI: 10.1007/978-3-642-29852-3_13.

[MK14] Antonis Michalas and Nikos Komninos. „The lord of the sense: A privacy preserving reputation system for participatory sensing applications“. In: *IEEE Symposium on Computers and Communications, ISCC 2014, Funchal, Madeira, Portugal, June 23-26, 2014*. IEEE Computer Society, 2014, pp. 1–6. DOI: 10.1109/ISCC.2014.6912480.

[MPR11] Hemanta K. Maji, Manoj Prabhakaran, and Mike Rosulek. „Attribute-Based Signatures“. In: *Topics in Cryptology – CT-RSA 2011*. Ed. by Aggelos Kiayias. Vol. 6558. Lecture Notes in Computer Science. San Francisco, CA, USA: Springer, Heidelberg, Germany, Feb. 2011, pp. 376–392.

[MY08] Daniele Micciancio and Scott Yilek. „The Round-Complexity of Black-Box Zero-Knowledge: A Combinatorial Characterization“. In: *TCC 2008: 5th Theory of Cryptography Conference*. Ed. by Ran Canetti. Vol. 4948. Lecture Notes in Computer Science. San Francisco, CA, USA: Springer, Heidelberg, Germany, Mar. 2008, pp. 535–552.

[NF06] Toru Nakanishi and Nobuo Funabiki. „A Short Verifier-Local Revocation Group Signature Scheme with Backward Unlinkability“. In: *Advances in Information and Computer Security, First International Workshop on Security, IWSEC 2006, Kyoto, Japan, October 23-24, 2006, Proceedings*. Ed. by Hiroshi Yoshiura, Kouichi Sakurai, Kai Rannenberg, Yuko Murayama, and Shin-ichi Kawamura. Vol. 4266. Lecture Notes in Computer Science. Springer, 2006, pp. 17–32. DOI: 10.1007/11908739_2.

[NY90] Moni Naor and Moti Yung. „Public-key Cryptosystems Provably Secure against Chosen Ciphertext Attacks“. In: *22nd Annual ACM Symposium on Theory of Computing*. Baltimore, MD, USA: ACM Press, May 1990, pp. 427–437.

[Ped92] Torben P. Pedersen. „Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing“. In: *Advances in Cryptology – CRYPTO'91*. Ed. by Joan Feigenbaum. Vol. 576. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1992, pp. 129–140.

[PLS14] Ronald Petrlic, Sascha Lutters, and Christoph Sorge. „Privacy-preserving reputation management“. In: *Symposium on Applied Computing, SAC 2014, Gyeongju, Republic of Korea - March 24 - 28, 2014*. Ed. by Yookun Cho, Sung Y. Shin, Sang-Wook Kim, Chih-Cheng Hung, and Jiman Hong. ACM, 2014, pp. 1712–1718. DOI: 10.1145/2554850.2554881.

[PS00] David Pointcheval and Jacques Stern. „Security Arguments for Digital Signatures and Blind Signatures“. In: *Journal of Cryptology* 13.3 (2000), pp. 361–396.

[PS16] David Pointcheval and Olivier Sanders. „Short Randomizable Signatures“. In: *Topics in Cryptology – CT-RSA 2016*. Ed. by Kazue Sako. Vol. 9610. Lecture Notes in Computer Science. San Francisco, CA, USA: Springer, Heidelberg, Germany, Feb. 2016, pp. 111–126. DOI: 10.1007/978-3-319-29485-8_7.

[Ros06] Alon Rosen. *Concurrent Zero-Knowledge - With Additional Background by Oded Goldreich*. Information Security and Cryptography. Springer, 2006. ISBN: 978-3-540-32938-1. DOI: 10.1007/3-540-32939-0.

[SL03] Aameek Singh and Ling Liu. „TrustMe: Anonymous Management of Trust Relationships in Decentralized P2P Systems“. In: *3rd International Conference on Peer-to-Peer Computing (P2P 2003), 1-3 September 2003, Linköping, Sweden*. Ed. by Nahid Shahmehri, Ross Lee Graham, and Germano Caronni. IEEE Computer Society, 2003, pp. 142–149. DOI: 10.1109/PTP.2003.1231514.

[SL12] Yan Sun and Yuhong Liu. „Security of Online Reputation Systems: The evolution of attacks and defenses“. In: *IEEE Signal Processing Magazine* 29.2 (2012), pp. 87–97. DOI: 10.1109/MSP.2011.942344.

[SP18] Johannes Sänger and Günther Pernul. „Interactive Reputation Systems - How to Cope with Malicious Behavior in Feedback Mechanisms“. In: *Business & Information Systems Engineering* 60.4 (2018), pp. 273–287. DOI: 10.1007/s12599-017-0493-1.

[Ste06] Sandra Steinbrecher. „Design Options for Privacy-Respecting Reputation Systems within Centralised Internet Communities“. In: *Security and Privacy in Dynamic Environments, Proceedings of the IFIP TC-11 21st International Information Security Conference (SEC 2006), 22-24 May 2006, Karlstad, Sweden*. Ed. by Simone Fischer-Hübner, Kai Rannenberg, Louise Yngström, and Stefan Lindskog. Vol. 201. IFIP. Springer, 2006, pp. 123–134. DOI: 10.1007/0-387-33406-8_11.

[Sun+06] Yan Lindsay Sun, Zhu Han, Wei Yu, and K. J. Ray Liu. „Attacks on Trust Evaluation in Distributed Networks“. In: *40th Annual Conference on Information Sciences and Systems*. Mar. 2006, pp. 1461–1466. DOI: 10.1109/CISS.2006.286695.

[SXL05] Mudhakar Srivatsa, Li Xiong, and Ling Liu. „TrustGuard: countering vulnerabilities in reputation management for decentralized overlay networks“. In: *Proceedings of the 14th international conference on World Wide Web, WWW 2005, Chiba, Japan, May 10-14, 2005*. Ed. by Allan Ellis and Tatsuya Hagino. ACM, 2005, pp. 422–431. DOI: 10.1145/1060745.1060808.

[Tea+06] W. T. Luke Teacy, Jigar Patel, Nicholas R. Jennings, and Michael Luck. „TRAVOS: Trust and Reputation in the Context of Inaccurate Information Sources“. In: *Autonomous Agents and Multi-Agent Systems* 12.2 (2006), pp. 183–198. DOI: 10.1007/s10458-006-5952-x.

[YS02] Bin Yu and Munindar P. Singh. „An evidential model of distributed reputation management“. In: *The First International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2002, July 15-19, 2002, Bologna, Italy, Proceedings*. ACM, 2002, pp. 294–301. DOI: 10.1145/544741.544809.

[Zha+12] Lizi Zhang, Siwei Jiang, Jie Zhang, and Wee Keong Ng. „Robustness of Trust Models and Combinations for Handling Unfair Ratings“. In: *Trust Management VI - 6th IFIP WG 11.11 International Conference, IFIPTM 2012, Surat, India, May 21-25, 2012. Proceedings*. Ed. by Theo Dimitrakos, Rajat Moona, Dhiren R. Patel, and D. Harrison McKnight. Vol. 374. IFIP Advances in Information and Communication Technology. Springer, 2012, pp. 36–51. DOI: 10.1007/978-3-642-29852-3_3.

[Zha+16] Ennan Zhai, David Isaac Wolinsky, Ruichuan Chen, Ewa Syta, Chao Teng, and Bryan Ford. „AnonRep: Towards Tracking-Resistant Anonymous Reputation“. In: *13th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2016, Santa Clara, CA, USA, March 16-18, 2016*. Ed. by Katerina J. Argyraki and Rebecca Isaacs. USENIX Association, 2016, pp. 583–596. URL: https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/zhai.