Privacy-Preserving Cryptography: Attribute-Based Signatures and Updatable Credentials

Fabian Eidens, M.Sc.

September 28, 2022

A dissertation submitted to the
Department of Computer Science
Paderborn University

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

Fabian Eidens
Place of birth: Arnsberg, NRW, Germany
Author's contact information:
fabianeidens@gmail.com

Thesis Advisor:	**Prof. Dr. rer. nat. Johannes Blömer** Head of Codes and Cryptography Paderborn University, Paderborn, Germany
Second Examiner:	**Prof. Dr.-Ing. Tibor Jager** Head of IT-Security and Cryptography Bergische Universität Wuppertal, Wuppertal, Germany
Third Examiner:	**Univ.-Prof. Dr.-Ing. Mark Manulis** Head of Privacy and Applied Cryptography Lab Research Institute CODE, Universität der Bundeswehr München, München, Germany
Thesis submitted:	September 28, 2022
Thesis defense:	December 15, 2022
Last revision:	January 17, 2023

Acknowledgements

Here I want to thank everyone who was a part of the journey to my Ph.D. and supported me during my time at Paderborn University and beyond.

First, I want to thank my supervisor Johannes Blömer for giving me the opportunity to work on such an interesting topic and providing valuable feedback. I also want to express my gratitude to the Paderborn University, the CRC 901 project, SICP, and the KMU project for making this journey possible, enjoyable, and full of experiences that taught me a bunch of new skills. Further, I want to thank the reviewers Johannes, Tibor and Mark of this work and the committee members Friedhelm and Patricia for their time, questions, and feedback.

The whole journey was also enjoyable because of the support and companionship of the working group Codes and Cryptography. You supported me not only on a technical and topical level, guiding my research and sharpening the senses for formalisms, you also helped me during hard times and hard work. The good thing is that the working group consists of many different characters that provided support in all kinds of life situations. Hence, thank you Claudia for heading the organization of everything, talking to me about life, and being the social anchor. Many thanks to Jakob, Gennadij, Peter, Sascha, and Kathrin for welcoming me so well in the group and for fostering an amazing exchange of research insights. I also want express my thankfulness for having Jakob and Jan beside me for major parts of my journey. From exchanging knowledge about Yharnam, drawing maps and figures (for UC security) and discussing life and simulation-extractability, it was all a pleasure that I will miss deeply. Also, I am grateful to Laurens, Henrik, and Paul for mixing things up, giving the working group a new direction, and taking care of teaching, among other things, such that I had more focussed time to work on my thesis.

During my studies I worked with many students and this work really helped me staying connected and was a great avenue to work on other related topics. Many thanks to all students, student assistants, and especially the coauthors of our software library Cryptimeleon. During my bachelor's and master's studies I was lucky to have a fun group of fellow students. Many thanks to Arthur, Sergio, Anna, Martin, and especially to Jan M. and Alex for the action in our shared apartment. Also, many thanks to my math and programming teacher who, among others, set me on this path and bolstered my confidence to the point that I (and my family) decided to take the first steps of this journey.

Regarding this work, I want to thank Denis, Anna, Sascha, and Henrik for their

valuable feedback and keeping up with all the formalisms.

I also want to thank my coauthors for the great opportunities and smooth process of writing papers together. Furthermore, I want to thank the previous members of the IT-Security group, especially Peter and Saqib for all the jokes, magic tricks, and jigsaw puzzles. Seriously, the jigsaw puzzles were a big part of our breaks for one year and were next to darts one of the best counterbalances to our research work. On the subject of balance, I have to thank my friends Denis, Lea, Marina, Peter, Gennadij who always had a sympathetic ear, invited me for dinner (Marina and Peter, I especially mean you), lifted me up when needed, prepared countless Spaghetti alla Carbonara with me, and with whom I had so much fun.

Last but not least, I am deeply grateful to my family for all the happy hours, delicious meals, board game nights, vacations, and just our time together. Undoubtedly, I thank my parents for enabling this journey at all. Especially my mother, who is a technology enthusiast herself, supported my interest in computer technology (and games) and discusses with me every new development in this area. Thank you so much for all your love and support. Further, I want to thank my brother Dominik, Christina, Charlotte and Marlene. You did so much for me over the years. Even if you do not know what exactly, be sure that the time with you always made the next deadline rush much easier for me. Charlotte and Marlene you both came into my life at the right moments and remind me every time of the non-academic parts of life. Thank you so much for all the joy that you sparked. To stay in touch with family and friends was sometimes hard because of the distance. In this respect, I am very grateful to my wife's family Ljuba, Anatoli, and Nikolaj who have welcomed me warmly and gave me a second home and family close by. I would love to return to our board game events, watching movies, and enjoying borscht together.

Anna, my wife, I am so grateful for you being part of my life, the finished journey, and all the adventures ahead. Thank you so much for all the noticed and unnoticed support (whereby to be honest I think the latter predominates lately) and your love. Because of this journey we had to forgo some things along the way, but we as close friends and co-op partners have mastered all well. Already during our studies you supported me, discussed all matters of life (and cryptography) and gave me valuable insights. Thank you for your time, patience and your love. I am also deeply grateful for our son Gregor. You, your energy and joy inspire me every day. I have looked forward to every lunch break with you and your mom to recharge my batteries and to play with you for a moment. You also deserve a special award for being the best child during this challenging time with having our first child, then working from home, and writing a thesis. To be honest, you never made this challenging only I, who wanted to play with you more than I wanted to write this thesis. I love you so much and thank you for enriching our lives. I dedicate this work to all my loved ones.

My Contribution

The publications that are part of this thesis include joint work. In the following I state individual contributions where possible.

Publications in this thesis. The following publications are part of this thesis including their full versions.

[BBDE19a] Johannes Blömer, Jan Bobolz, Denis Diemert, and Fabian Eidens. "Updatable Anonymous Credentials and Applications to Incentive Systems." In: *ACM CCS 2019: 26th Conference on Computer and Communications Security*. Ed. by Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz. ACM Press, Nov. 2019, pp. 1671–1685. DOI: 10.1145/3319535.3354223.

[BBDE19b] Johannes Blömer, Jan Bobolz, Denis Diemert, and Fabian Eidens. *Updatable Anonymous Credentials and Applications to Incentive Systems*. Cryptology ePrint Archive, Report 2019/169. https://eprint.iacr.org/2019/169. 2019.

[BEJ18a] Johannes Blömer, Fabian Eidens, and Jakob Juhnke. *Enhanced Security of Attribute-Based Signatures*. Cryptology ePrint Archive, Report 2018/874. https://eprint.iacr.org/2018/874. 2018.

[BEJ18b] Johannes Blömer, Fabian Eidens, and Jakob Juhnke. "Enhanced Security of Attribute-Based Signatures." In: *CANS 18: 17th International Conference on Cryptology and Network Security*. Ed. by Jan Camenisch and Panos Papadimitratos. Vol. 11124. Lecture Notes in Computer Science. Springer, Heidelberg, 2018, pp. 235–255. DOI: 10.1007/978-3-030-00434-7_12.

In Chapters 3 and 4 (Part I) of this thesis I present and further extend the results the work published at CANS'18 [BEJ18b] and its full version [BEJ18a]. Johannes Blömer, Jakob Juhnke and I jointly discussed the details of the universal composability framework (UC) by Canetti [Can01] and the security notion of universal composability. The insightful discussions informed UC specific modeling details of the ideal functionality for attribute-based signature (ABS) present in Chapter 4. Besides this, the content and results presented in Part I of this thesis are my contribution. Chapter 5 (Part II) is based on the work published at CCS'19 [BBDE19a] and its full version [BBDE19b]. Johannes Blömer, Jan Bobolz and I

discussed possible extensions to anonymous credentials. Then, Jan Bobolz and I jointly worked on updatable anonymous credentials, the generic construction and a concrete instantiation. Together we also devised the incentive system presented in [BBDE19a; BBDE19b] that is not part of this thesis. Jan Bobolz then focussed on the soundness definition and soundness proof, where I focussed on the anonymity definition and the corresponding proof. Denis Diemert implemented the concrete incentive system presented in [BBDE19a; BBDE19b] and analyzed its performance. Furthermore, in Section 3.5 I reference the master's thesis by Pascal Bemmann [Bem17] that I co-supervised.

Further publications. The following publications are not part of this thesis.

[Bob+21] Jan Bobolz, Fabian Eidens, Stephan Krenn, Sebastian Ramacher, and Kai Samelin. "Issuer-Hiding Attribute-Based Credentials." In: *Cryptology and Network Security - 20th International Conference, CANS 2021, Vienna, Austria, December 13-15, 2021, Proceedings*. Ed. by Mauro Conti, Marc Stevens, and Stephan Krenn. Vol. 13099. Lecture Notes in Computer Science. Springer, 2021, pp. 158–178. DOI: 10.1007/978-3-030-92548-2_9.

[BEHF21] Jan Bobolz, Fabian Eidens, Raphael Heitjohann, and Jeremy Fell. *Cryptimeleon: A Library for Fast Prototyping of Privacy-Preserving Cryptographic Schemes*. Cryptology ePrint Archive, Report 2021/961. https://eprint.iacr.org/2021/961. 2021.

[Bob+20] Jan Bobolz, Fabian Eidens, Stephan Krenn, Daniel Slamanig, and Christoph Striecks. "Privacy-Preserving Incentive Systems with Highly Efficient Point-Collection." In: *ASIACCS 20: 15th ACM Symposium on Information, Computer and Communications Security*. Ed. by Hung-Min Sun, Shiuh-Pyng Shieh, Guofei Gu, and Giuseppe Ateniese. ACM Press, Oct. 2020, pp. 319–333. DOI: 10.1145/3320269.3384769.

[Bem+18] Kai Bemmann, Johannes Blömer, Jan Bobolz, Henrik Bröcher, Denis Diemert, Fabian Eidens, Lukas Eilers, Jan Haltermann, Jakob Juhnke, Burhan Otour, Laurens Porzenheim, Simon Pukrop, Erik Schilling, Michael Schlichtig, and Marcel Stienemeier. "Fully-Featured Anonymous Credentials with Reputation System." In: *Proceedings of the 13th International Conference on Availability, Reliability and Security, ARES 2018, Hamburg, Germany, August 27-30, 2018*. Ed. by Sebastian Doerr, Mathias Fischer, Sebastian Schrittwieser, and Dominik Herrmann. ACM, 2018, 42:1–42:10. DOI: 10.1145/3230833.3234517.

[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. Springer, Heidelberg, Apr. 2018, pp. 470–490. DOI: 10.1007/978-3-319-76953-0_25.

Abstract

In this thesis we explore the security models, features, and construction of two privacy-preserving attribute-based cryptographic primitives: attribute-based signatures (ABS) and updatable anonymous credentials (UAC). The core of ABS and UAC is the principle that authentication is based on personal information encoded as certified *attributes*. Authentication is then a process that shows that the attributes satisfy a *policy* without revealing the attributes. This gives users sovereignty over their attributes (personal information) and provides expressive authentication by the inclusion of policies.

The first part of this thesis focusses on ABS. An ABS scheme allows users to authenticate *messages* in a privacy-preserving manner while proving that their personal information encoded as attributes in their secret keys satisfy a policy. Formally, the experiment-based security model of ABS in the literature includes *privacy* of attributes and secret keys and *unforgeability* of signatures. We enhance the security model of ABS schemes in this thesis in three ways. First we present a strengthened experiment-based definition and second we show an ideal ABS functionality secure in the universal composability framework (UC). The third enhancement is that we show that our strengthened experiment-based definition of ABS is equivalent to our ideal ABS functionality in UC. In detail, we strengthen the existing experiment-based security definitions for privacy in the form of simulation privacy that is stronger than the privacy definitions from the literature with respect to general policies. Further, we show that two major existing generic ABS constructions are simulation private and are therefore UC secure. Our enhanced experiment-based security model also results in more general experiment-based definitions that do not make implicit assumptions on the supported policies. Additionally, we strengthen the experiment-based soundness definition of ABS in the form of simulation-extractability and show a generic construction of ABS from simulation-extractable signatures of knowledge. We then present an instantiation based on succinct signatures of knowledge which is, to the best of our knowledge, the first adaptively secure ABS scheme simultaneously achieving constant-size signatures (three group elements) and arithmetic circuits as policies.

The second part of this thesis considers updatable anonymous credentials (UAC) that allow *users* to authenticate to a service via *anonymous credentials* that encode and certify personal information as *attributes*. We present UAC as a practical extension to the features of anonymous credentials as defined in the literature. In UAC users can get privacy-preserving updates on their attributes by the issuer

of the corresponding anonymous credential, i.e., the updates are hidden from the issuer. Additionally, we apply the same techniques used to realize the privacy-preserving updates to allow the issuing of anonymous credentials on hidden attributes. Both features enable new and modern applications with regular updates to users' attributes that are not possible with anonymous credential systems from the literature. We solve the challenges that hidden updates and attributes mean for the experiment-security definitions of UAC with a simulation-based anonymity definition and a specialized extractability definition. Furthermore, we present a generic construction of UAC from blind signatures on commitments and argument systems. To complement this, we provide an efficient instantiations in the random oracle model (ROM) and a variant in the common reference string (CRS) model. Finally, we combine the two parts of this thesis and discuss the connection of UAC and ABS by presenting an instantiation of UAC from ABS with minimal requirements on the ABS.

Zusammenfassung

In dieser Dissertation untersuchen wir die Sicherheitsmodelle, die Eigenschaften und die Konstruktion von zwei die Privatsphäre wahrenden attributbasierten kryptographischen Primitiven: Attribute-based Signatures (ABS) und Updatable Anonymous Credentials (UAC). Der Kern von ABS und UAC ist das Prinzip, dass die Authentifizierung auf persönlichen Informationen basiert, die als zertifizierte *Attribute* kodiert sind. Die Authentifizierung ist ein Prozess, der zeigt, dass die Attribute eine *Policy* erfüllen, ohne die Attribute offen zu legen. Dies gibt den Benutzern die Hoheit über ihre Attribute (persönlichen Informationen) und ermöglicht eine aussagekräftige Authentifizierung durch die Einbeziehung von Authentifizierungsregeln.

Der erste Teil dieser Dissertation befasst sich mit ABS. Ein ABS Schema ermöglicht es Benutzern, *Nachrichten* unter Wahrung der Privatsphäre zu authentifizieren und gleichzeitig zu beweisen, dass ihre persönlichen Informationen, die als Attribute in ihren geheimen Schlüsseln kodiert sind, eine Authentifizierungsregel erfüllen. Formal umfasst das experimentbasierte Sicherheitsmodell von ABS in der Literatur *Privatsphäre* (Privacy) von Attributen und geheimen Schlüsseln und *Unfälschbarkeit* (Unforgeability) von Signaturen. In dieser Dissertation erweitern wir das Sicherheitsmodell von ABS Verfahren auf drei Arten. Erstens präsentieren wir eine stärkere experimentbasierte Definition und zweitens zeigen wir eine ideale ABS Funktionalität, die in dem Universal Composability Framework (UC) sicher ist. Die dritte Erweiterung ist, dass wir zeigen, dass unsere stärkere experimentbasierte Definition von ABS äquivalent zu unserer idealen ABS Funktionalität in UC ist. Im Detail verstärken wir die bestehenden experimentbasierten Sicherheitsdefinitionen für Privacy in Form von Simulation Privacy, die stärker ist als die Privacy-Definitionen aus der Literatur in Bezug auf allgemeine Authentifizierungsregeln. Darüber hinaus zeigen wir, dass zwei wichtige existierende generische ABS Konstruktionen Simulation Privacy erfüllen und daher UC sicher sind. Unser erweitertes experimentbasiertes Sicherheitsmodell führt auch zu allgemeineren experimentbasierten Definitionen, die keine impliziten Annahmen über die unterstützten Authentifizierungsregeln machen. Zusätzlich verstärken wir die experimentbasierte Soundness-Definition von ABS in Form von Simulation-Extractability und zeigen eine generische Konstruktion von ABS aus simulations-extrahierbaren Signatures of Knowledge. Anschließend stellen wir eine Instanziierung vor, die auf kurzen Signatures of Knowledge basiert und unseres Wissens nach das erste adaptiv sichere ABS Verfahren ist, das gleichzeitig Signaturen konstanter Größe (drei Gruppenele-

mente) und arithmetische Schaltungen als Authentifizierungsregeln erreicht.

Der zweite Teil dieser Dissertation befasst sich mit Updatable Anonymous Credentials, die es *Benutzern* ermöglichen, sich bei einem Dienst mittels *Anonymous Credentials* zu authentifizieren, die persönliche Informationen als *Attribute* kodieren und zertifizieren. Wir präsentieren UAC als eine praktische Erweiterung der in der Literatur definierten Funktionen von Anonymous Credentials vor. In UAC können Benutzer vom Aussteller des entsprechenden Anonymous Credentials Aktualisierungen ihrer Attribute unter Wahrung der Privatsphäre erhalten, d.h. Aktualisierungen sind für den Aussteller verborgen. Darüber hinaus wenden wir dieselben Techniken an, die für die Realisierung der Aktualisierungen unter der Wahrung der Privatsphäre verwendet werden, um die Ausstellung von Anonymous Credentials für versteckte Attribute zu ermöglichen. Beide Funktionen ermöglichen neue und moderne Anwendungen mit regelmäßigen Aktualisierungen der Attribute der Benutzer, die mit Anonymous Credential Systemen aus der Literatur nicht möglich sind. Die Herausforderungen, die versteckte Aktualisierungen und Attribute für die experimentbasierten Sicherheitsdefinitionen von UAC bedeuten, lösen wir mit einer simulationsbasierten Anonymitätsdefinition und einer speziellen Extractability-Definition. Außerdem präsentieren wir eine generische Konstruktion von UAC aus Blind Signatures auf Commitments und Argument Systems. Ergänzend dazu zeigen wir eine effiziente Instanziierung im Random Oracle Modell (ROM) und eine Variante im Common Reference String (CRS) Modell. Abschließend kombinieren wir die beiden Teile dieser Dissertation und diskutieren die Verbindung von UAC und ABS, indem wir eine Instanziierung von UAC aus ABS mit minimalen Anforderungen an die ABS vorstellen.

Contents

Acronyms

ABE	attribute-based encryption
ABS	attribute-based signature
AC	anonymous credential
BS	blind signature
CNF	conjunctive normal form
CRS	common reference string
dlog	discrete logarithm
dpt	deterministic polynomial-time
IO	indistinguishability obfuscation
ITI	interactive Turing machine instance
ITM	interactive Turing machine
MSP	monotone span program
NIWI	non-interactive witness indistinguishable argument of knowledge
NIZK	non-interactive zero-knowledge argument of knowledge
O-SNARK	SNARK in the presence of oracles
PBS	policy-based signature
ppt	probabilistic polynomial-time
ROM	random oracle model
SAT	boolean satisfiability problem
SE-NIZK	simulation-extractable non-interactive zero-knowledge argument of knowledge
SE-SNARK	succinct simulation-extractable non-interactive zero-knowledge argument of knowledge
SNARK	succinct non-interactive argument of knowledge
SoK	signature of knowledge
SPS	structure-preserving signature
SPS-EQ	structure-preserving signatures on equivalence classes
UAC	updatable anonymous credential
UC	universal composability framework
WH	witness hiding
WI	witness indistinguishable
ZK	zero-knowledge

1 Introduction

On a high level, authentication can be viewed as a process proving that some statement is true. For example, if a user authenticates a document, the authentication proves that the user is the original author of the document, or that the user approves the content of the document similar to a handwritten signature. Similarly users can authenticate themselves towards a third party by showing their passport or other identifying information. Hence, classical authentication also always means identification. Usually in digital identity-based authentication at a service provider (e.g., to get access) users also provide identifying information (e.g., mail address, user name) and a secret (e.g., password). The service provider then stores this information in a database together with other information (attributes) of the user, e.g., affiliation, role, purchase history, favorite item, name, address, or access rights. This allows the service provider to construct comprehensive profiles of users. Furthermore, if multiple service providers collude, they can track users across several services. This is a serious threat to the privacy of users. An additional threat is a data breach. If an adversary gets access to the database it gets the private information of users (e.g., name and purchase history of users, role and access rights of employees) that can be harmful to users and to organizations alike.

A similar concern exists for authentication of documents instead of users. Usually a document that is signed via a digital signature scheme is associated to and verified with a public key of the user that signed the document. For example an employee in a sales department is allowed to sign documents such as contracts with her secret key. On verification under her public key it is publicly known who signed the document and who is responsible. However, if such documents are leaked, then not only the identities of the signers are leaked, but also the internal processes, i.e., who is allowed to sign what kind of documents. Therefore, it is desirable to sign documents in the name of the sales department and specific roles. This means, attribute-based instead of identity-based.

In this thesis we consider privacy-preserving attribute-based cryptography that enables authentication without identification via a practical feature set combined with strong security models. Motivated by the observation that privacy-preserving cryptography and its techniques are becoming more relevant for real-world applications we focus on practical features that are attribute-based and security

notions that capture real-world threats. Constructing privacy-preserving schemes and defining their security is a delicate task that has to balance strong security definitions including privacy, the efficiency of schemes and the support of practical features.

The two privacy-preserving attribute-based schemes that we are focussing on in this thesis are attribute-based signatures (ABS) and updatable anonymous credentials (UAC). They solve the above problems and more by loosening the connection between authentication and identification, providing privacy and practical features. At the core of ABS and UAC schemes is the principle that authentication is based on personal information encoded as certified *attributes*. Attribute-based authentication is then a process that shows that the attributes satisfy a *policy* without revealing the attributes. This gives users sovereignty over their attributes and the inclusion of policies leads to expressive authentication.

Attribute-based signatures extend digital signatures such that the link between a signature and a user's public key is removed. In ABS a user's secret key that encodes her attributes is issued by an authority. A user can sign message-policy pairs with her secret key if the encoded attributes satisfy the policy. Here, a signature is not verified according to a user's public key instead it is verified with respect to public parameters that are applicable to all users. If the signature is valid, a verifier knows that the signer of the message-policy pair has a secret key that encodes satisfying attributes with respect to the policy without revealing the secret key or the attributes. Intuitively, the user's attributes are hidden with respect to the policy and all users that could have signed the message-policy pair, i.e., all users with attributes (encoded in their secret keys) that satisfy the policy. This security notion is called privacy in ABS.

An updatable anonymous credential (UAC) system is a privacy-preserving authentication system for users that replaces the usual way of authentication where users provide identifying information (e.g., mail address, user name) and a secret (e.g., password). In UAC users receive *credentials* that certify *attributes* from an *issuer*. Users can then anonymously authenticate to service providers (*verifiers*) by proving possession of a credential without revealing anything about their attributes except that they satisfy the access polices. Hence, in a UAC system users authenticate *themselves* towards verifiers and in ABS users authenticate messages (documents). The difference is that ABS schemes are signature schemes and UAC systems are authentication systems for users. Besides anonymous authentication, updatable anonymous credential (UAC) makes use of privacy-preserving techniques to allow users to get credentials on their attributes without revealing them to the issuer. Furthermore, UAC adds to the feature set of anonymous credential (AC) a novel and practical feature such that users can get their attributes (certified in a credential) *updated* by the issuer of the corresponding credential without revealing the updates, their credentials, or their attributes. This enables practical applications with regular updates of users' attributes such as updating a subscription status, points, or number of semesters.

Constructing such privacy-preserving attribute-based schemes comes with addi-

tional challenges compared to normal identity-based schemes. We not only have to consider the privacy of users, but also a meaningful soundness notion for service providers (verifiers). Soundness means here that a service provider can be sure that users cannot authenticate without having the necessary credentials or secret keys. Since the identity and attributes of users that generate signatures or authenticate via credentials are hidden, we also have to consider security (soundness) against colluding users that try to pool their attributes to satisfy policies. Besides security, we also have to consider the efficiency of schemes (bandwidth, processing resources), the supported attributes, and the expressiveness of the supported policies. These considerations then lead to our concrete goals for ABS and UAC: security with respect to a strong security model while supporting expressive policies combined with practical features and efficiency.

1.1 Contribution

We show that ABS and UAC have security models and features that make them ready to be used in real-world applications to protect user's privacy.

In the first part of this thesis we enhance the security model of ABS by presenting a strengthened experiment-based definition and ABS in the universal composability framework (UC). By this, we show that ABS can be securely composed with other schemes to form larger practical systems. In detail, we show the equivalence of our strengthened experiment-based definition and our ABS model in UC. To demonstrate the applicability of this result we show that two major generic ABS construction from the literature are secure with respect to our strengthened experiment-definition and therefore are secure in UC. Additionally, we present a generic ABS construction from signatures of knowledge and a concrete instantiation that features expressive policies combined with constant-size signatures.

UAC are the focus of the second part of this thesis. We present UAC as a practical extension of AC that allows users to update their attributes encoded in credentials in a privacy-preserving way. For this we present a formal definition of UAC and experiment-based security definitions. We then show a generic construction of UAC from building blocks that are commonly used to construct AC in the literature. Furthermore, we present an efficient instantiation of the generic UAC construction. Finally, we discuss the connection of UAC and ABS and present how to instantiate UAC from ABS.

1.2 Outline

In Chapter 2 we present preliminaries including general notation and definitions of building blocks that we use in this thesis. The main body of this thesis is presented in two parts. In Part I we present enhanced security models for ABS and in Part II we present UAC an extension of anonymous credentials with privacy-preserving updates. Part I is comprised of Chapters 3 and 4. In Chapter 3 we focus on the

experiment-based security model of ABS and our generic ABS construction. Then, in Chapter 4 we present our UC result that shows the equivalence between ABS in UC and our experiment-based security model. After that we move to Part II of this thesis and focus on UAC in Chapter 5. There, we present the formal definition, experiment-based security model, and a generic construction of UAC. Finally, we conclude this thesis with a summary of the contributions in Chapter 6.

2 Preliminaries

In this chapter we present general notation that we use throughout this thesis and introduce cryptographic building blocks in the form of assumptions, schemes, and security definitions. For the security definitions in this work we use experiment-based definitions and definitions based on ideal functionalities. For the definition of ideal functionalities we rely on the universal composability framework (UC) by Canetti [Can01] and introduce the framework only conceptually in this chapter. The formal details of the framework are presented in Chapter 3 where we also show the technical execution in form of an ideal functionality for attribute-based signatures. This way of structuring the preliminaries, allows us to focus first on the experiment-based definitions, which are more common in cryptography, and then move towards ideal functionalities. Therefore, we start the preliminaries with some general notational definitions, proceed with definitions of groups, relations, cryptographic assumptions, definitions of schemes that we use as building blocks in this thesis, and then describe the concept of the universal composability framework.

2.1 General Notation

In the following we introduce general notation that is used in this thesis, e.g., for the security parameter, adversaries, sets, oracles, and schemes.

Security parameter. We refer to $\lambda \in \mathbb{N}$ as the security parameter and if the context is clear we omit that it is an element of $\mathbb{N}$. Additionally, we refer to the security parameter λ in unary representation as 1^λ.

Algorithms. The algorithms considered in this thesis are (mostly) probabilistic polynomial-time (ppt). A ppt algorithm $\mathcal{A}$ uses internal randomness r and its running time is polynomially bounded in its input length. We also consider deterministic polynomial-time (dpt). The running time of a dpt algorithm is also polynomially bounded in its input length, but it does not use internal randomness. We also refer to a ppt (and dpt) algorithm as an efficient algorithm. We denote with $\mathcal{A}(x;r)$ that algorithm $\mathcal{A}$ runs with fixed randomness r on input x. Also, for an algorithm $\mathcal{A}$ let $trans_{\mathcal{A}}$ denote its transcript containing all inputs, outputs and

randomness of algorithm $\mathcal{A}$. If not stated otherwise, we use the terms *adversary*, *distinguisher*, *forger* and *ppt algorithm* synonymously. For a stateful algorithm $\mathcal{A}$ we denote the state by st and by $\mathcal{A}(x, st)$ we mean algorithm $\mathcal{A}$ runs on input x and state st. The (current) state of the algorithm is also part of its output. If the state is not explicitly given we assume, for cleaner notation, that a stateful algorithm $\mathcal{A}$ implicitly gets as input the (initially empty) state from its last run and that it implicitly includes its state in the final output. Generally, we consider algorithms to be stateless and we explicitly emphasize it if they are not.

Basics. In the following we introduce basic notation. For a set S we denote by $y \leftarrow S$ that y is chosen uniformly at random from set S. For an algorithm $\mathcal{A}$ we write $y \leftarrow \mathcal{A}(x)$ and mean that y is the output of algorithm $\mathcal{A}$ on input x. If algorithm $\mathcal{A}$ is a ppt, then the output y is also determined by the randomness of $\mathcal{A}$, therefore y is a random variable. For algorithm $\mathcal{A}$ we write $y \in \mathcal{A}(x)$ to denote that y is in the set of all possible outputs of $\mathcal{A}$ on input x. We refer to an interactive protocol as two ppt algorithms Alg' and Alg'' that interact with each other by exchanging messages. We denoted an interactive protocol as $\mathsf{Alg}'(c, u) \leftrightarrow \mathsf{Alg}''(c, v)$ with common input c, u as the private input for algorithm Alg', and v as the private input for algorithm Alg''. We denote oracle access to $\mathcal{O}^{\mathsf{name}}_{par}(input_1, \ldots, input_l)$ of an algorithm $\mathcal{A}$ with $\mathcal{A}^{\mathcal{O}^{\mathsf{name}}_{par}}$ where the oracle $\mathcal{O}^{\mathsf{name}}_{par}$ has access to fixed parameter list par and algorithm $\mathcal{A}$ determines the inputs $input_1, \ldots, input_l$ for each oracle query. To simplify notation, we also refer to an oracle as $\mathcal{O}^{\mathsf{name}}$ if the parameter list is clear from the context. An oracle $\mathcal{O}^{\mathsf{name}}$ is interactive if it runs an interactive protocol $(\mathsf{Alg}'(c, v) \leftrightarrow \mathsf{Alg}''(c, w))$ with algorithm $\mathcal{A}^{\mathcal{O}^{\mathsf{name}}}$, where the oracle $\mathcal{O}^{\mathsf{name}}$ executes one of the ppt algorithms of the interactive protocol and algorithm $\mathcal{A}$ replaces the other ppt algorithm of the interactive protocol. We denote this as $\mathsf{Alg}'(c, v) \leftrightarrow \mathcal{A}$ or $\mathcal{A} \leftrightarrow \mathsf{Alg}''; (c, w)$ respectively. For readability, for a list L we use the operator $\cup$ to append an element to the list and we use $L[i]$ to access the i-th element of the list.

Scheme notation. For a scheme X with algorithm K, we write X.K to denote which schemes and algorithms we refer to. For example, in case of a signature scheme $\mathsf{Sig} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ we use $\mathsf{Sig.Sign}$ to refer to the signing algorithm of Sig.

Schemes. In this work, we adopt the schemes to support a common setup algorithm that outputs some public parameters. In case of a signature scheme, this means that the key generation algorithm gets as input the public parameters instead of the security parameter. For example, public parameters can be a group description generated by a group generator algorithm with respect to the security parameter. Then, the common setup is the group generator.

Advantage. Throughout this thesis, we denote the advantage of an adversary $\mathcal{A}$ as $\mathrm{Adv}^{\mathsf{name}}_{\mathcal{A},\mathsf{S}}$, where name refers to the name of the security goal and experiment, and S refers to the scheme for which the advantage is defined. Similarly we denote security experiments as $\mathrm{Exp}^{\mathsf{name}}_{\mathcal{A},\mathsf{S}}$.

To define security of a scheme S we need a notion that the advantage of an adversary $\mathcal{A}$ is "very small". To formally define this notion we define negligible functions.

Definition 2.1.1 (Negligible Function)
A function $f\colon \mathbb{N} \to \mathbb{R}^{\geq 0}$ is negligible, if for every $c \in \mathbb{N}$ there exists a $\lambda_0 \in \mathbb{N}$, such that for all $\lambda \geq \lambda_0$ it holds that $f(\lambda) \leq 1/\lambda^c$. ◇

For readability, we denote a negligible function in security parameter λ by $\mathsf{negl}(\lambda)$.

2.2 Groups, Relations, and Assumptions

In the following we introduce some basic definitions of prime order groups, and bilinear groups since we use them as a building blocks in this thesis. Furthermore, we define relations and basic assumptions.

Definition 2.2.1 (Group Generator - [KL14])
A group generator GrGen is a ppt algorithm that on input security parameter 1^λ outputs a group description $\mathbb{GD} = (\mathbb{G}, p, g)$, where $\mathbb{G}$ is cyclic group of prime order p, with bit-length $|p| = \lambda$, and $g \in \mathbb{G}$ is a generator of group $\mathbb{G}$. ◇

Definition 2.2.2 (Bilinear Groups - [BLS04])
A bilinear group is a tuple $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, e, g_1, g_2)$ such that $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ are cyclic groups, of prime order p, and g_1 and g_2 are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively. We call $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ a bilinear map or pairing if it is efficiently computable and fulfills the following conditions:

Bilinear $\forall (g_1, g_2) \in \mathbb{G}_1 \times \mathbb{G}_2, \forall (a, b) \in \mathbb{Z}_p^2\colon e(g_1^a, g_2^b) = e(g_1, g_2)^{a \cdot b}$

Non-degenerate $e(g_1, g_2) \neq 1_{\mathbb{G}_T}$, where $1_{\mathbb{G}_T}$ is the identity element of $\mathbb{G}_T$.

◇

To generate a bilinear group to be used in cryptographic schemes we rely on a bilinear group generator.

Definition 2.2.3 (Bilinear Group Generator)
We call a group generator GrGen bilinear group generator if it on input security parameter 1^λ outputs a bilinear group description $\mathbb{GD} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, e, g_1, g_2)$, such that the bit-length of prime p is $|p| = \lambda$. ◇

To formally define signatures of knowledge and non-interactive arguments in the following we need the notion of a relation. Intuitively, a relation R determines some setup parameters and which instances x are valid. For valid instances there exists a witness w such that $(x, w) \in R$ holds. In the following, we formally define relations.

Definition 2.2.4 (Relation - [BS20; CFQ19])
Let P be a ppt algorithm that on input security parameter 1^λ, outputs system parameter $\Lambda \in \{0,1\}^{\mathsf{poly}(\lambda)}$, where the length is polynomial in λ. A collection $\{R_{\lambda,\Lambda}\}_{\lambda\in\mathbb{N},\Lambda\in P(1^\lambda)}$ of finite sets $R_{\lambda,\Lambda}$ of relations $R_i \subseteq \mathcal{I}_i \times \mathcal{W}_i$, is called a family of relations if:

- $\mathcal{I}_i$, and $\mathcal{W}_i$ are finite sets,
- for every instance-witness pair $(x, w) \in R_i$, it holds that $|w| \leq \mathsf{poly}(|x|)$, i.e., the size of a witness w is polynomial in the size of the instance x
- there is a description of R_i which size is polynomial in λ,
- and there exists a ppt algorithm that on input security parameter λ, system parameter Λ, description of R_i, instance x, and witness w outputs 1 if $(x, w) \in R_i$ and 0 otherwise.

$\diamond$

As in [Gro16] we assume that the security parameter λ and system parameter Λ are implicitly included in the description of a relation R.

Definition 2.2.5 (Relation Generator - [GM17])
A relation generator RGen for family of relations $\{R_{\lambda,\Lambda}\}_{\lambda,\Lambda}$ is a ppt algorithm that on input security parameter 1^λ outputs description of relation R from $R_{\lambda,\Lambda}$. Denoted as $R \leftarrow \mathsf{RGen}(1^\lambda)$. $\diamond$

We use a relation generator in this thesis as an algorithm that internally uses a setup algorithm (e.g., a group generator) to determine the system parameters and outputs a description of a relation.

Let us consider the following example of a relation based on computing discrete logarithms. Here, the description of relation R specifies a group, the instance consists of two elements of that group (u, h) and the witness for this instance is then an exponent a such that $u^a = h$ holds. Next, we give a formal definition of a cryptographic assumption called the discrete logarithm assumption that corresponds to the above example.

Definition 2.2.6 (Discrete Logarithm Assumption - [KL14])
Let GrGen be a group generator and $\mathcal{A}$ an adversary. We define the advantage in

$\underline{\mathrm{Exp}^{\mathsf{dlog}}_{\mathcal{A},\mathsf{GrGen}}(\lambda)}$

1 : $(\mathbb{G}, p, g) \leftarrow \mathsf{GrGen}(1^\lambda)$
2 : $(u, h) \leftarrow \mathbb{G}^2$
3 : $a \leftarrow \mathcal{A}(\mathbb{G}, p, g, u, h)$
4 : **if** $u^a = h$ **return** 1
5 : **else return** 0

Experiment 2.1: Discrete logarithm experiment for a group generator.

the experiment $\mathrm{Exp}^{\mathsf{dlog}}_{\mathcal{A},\mathsf{GrGen}}(\lambda)$ (Experiment 2.1) as

$$\mathrm{Adv}^{\mathsf{dlog}}_{\mathcal{A},\mathsf{GrGen}}(\lambda) \coloneqq \Pr\left[\mathrm{Exp}^{\mathsf{dlog}}_{\mathcal{A},\mathsf{GrGen}}(\lambda) = 1\right]$$

and say that the discrete logarithm (dlog) assumption holds with respect to group generator GrGen if for all ppt adversaries $\mathcal{A}$ it holds that $\mathrm{Adv}^{\mathsf{dlog}}_{\mathcal{A},\mathsf{GrGen}}(\lambda) \leq \mathsf{negl}(\lambda)$. ◇

2.3 Schemes and Security Definitions

In this section we formally define important building blocks, that we use for our constructions in this thesis, and we define their security. Schemes that we use as building blocks range from hash functions and signatures to more involved schemes like non-interactive zero-knowledge arguments.

2.3.1 Hash Functions

Hash functions in cryptography are typically used to compress inputs in such a way that collisions are hard to find for an adversary. A collision is a pair of two inputs m_0 and m_1, where $m_0 \neq m_1$, that under the hash function map to the same value. Naturally, compressing requires that the domain of the function is larger than the range, otherwise the identity function is good enough. Hash functions that are interesting for cryptography require some form of collision-resistance as a security guarantee. Let us first define hash functions and then their security in form of collision-resistance and target collision-resistance.

Definition 2.3.1 (Hash Function - [KL14])
A hash function Hash with key length $l_K(\lambda)$ and output length $l_h(\lambda)$ consists of ppt algorithms $(\mathsf{KeyGen}, \mathsf{H})$ where:

- $hk \leftarrow \mathsf{KeyGen}(1^\lambda)$ is a ppt key generation algorithm that on input security parameter 1^λ, outputs a hash key hk of length $l_K(\lambda)$.

- $h \leftarrow \mathsf{H}(hk, m)$ is a ppt hash algorithm that on input $m \in \{0,1\}^*$ outputs a hash value of length $l_h(\lambda)$.

Alternatively, a key generation algorithm KeyGen can take as input public parameters pp, where we assume that they implicitly encode the security parameter. If a hash function only takes inputs $m \in \{0,1\}^{l(\lambda)}$ we require that the input length is greater than the output length, $l(\lambda) > l_h(\lambda)$, such that the hash function is compressing. ◇

In the following security definitions for hash functions we let KeyGen run on input 1^λ. We omit explicit definitions for the alternative variant with input public parameters pp generated by some setup algorithm, e.g., group generator. Let us define collision-resistance for hash functions in the following. Intuitively, it should be infeasible for an adversary on input the hash key to output two distinct messages that have the same hash value.

Definition 2.3.2 (Collision-Resistant Hash Function - [GM17])
We call a hash function $\mathsf{Hash} = (\mathsf{KeyGen}, \mathsf{H})$ a *collision-resistant* hash function, if for all ppt adversaries $\mathcal{A}$, there exists a negligible function negl such that

$$\Pr\left[hk \leftarrow \mathsf{KeyGen}(1^\lambda);\ m_0, m_1 \leftarrow \mathcal{A}(hk) : \right.$$
$$\left. m_0 \neq m_1 \wedge \mathsf{H}(hk, m_0) = \mathsf{H}(hk, m_1)\right] = \mathsf{negl}(\lambda) \ .$$

◇

A weaker form of collision-resistance is the so called target collision-resistance. Here, it should be infeasible for an adversary to output one input, the target, before the generation of the hash key and then given the hash key find a distinct second input that collides.

Definition 2.3.3 (Target Collision-Resistant Hash Function - [GM17])
We call a hash function $\mathsf{Hash} = (\mathsf{KeyGen}, \mathsf{H})$ *target collision-resistant* hash function, if for all ppt adversaries $\mathcal{A}$, there exists a negligible function negl such that

$$\Pr\left[m_0, st \leftarrow \mathcal{A}(1^\lambda);\ hk \leftarrow \mathsf{KeyGen}(1^\lambda);\ m_1 \leftarrow \mathcal{A}(hk, st) : \right.$$
$$\left. m_0 \neq m_1 \wedge \mathsf{H}(hk, m_0) = \mathsf{H}(hk, m_1)\right] = \mathsf{negl}(\lambda) \ .$$

◇

2.3.2 Digital Signatures

Digital signatures are used to authenticate messages in an identity-based way. This means, a public key of a signature scheme is associated to a user and a user signs a message with its corresponding secret key. If a signature verifies as valid under a public key, the verifier can be sure that the user associated to the public key

signed the message. Hence, digital signatures provide authenticity, integrity, and non-repudiation. We base the notion of digital signature schemes on the definition by Goldwasser, Micali and Rivest [GMR88] and adapt it accommodate a setup algorithm.

Definition 2.3.4 (Signature Scheme)
A (digital) signature scheme Sig consists of the algorithms $\mathsf{Setup}, \mathsf{KeyGen}$, Sign, and Verify.

- $pp \leftarrow \mathsf{Setup}(1^\lambda)$ is a ppt setup algorithm that on input security parameter 1^λ outputs public parameters pp.
- $(pk, sk) \leftarrow \mathsf{KeyGen}(pp)$ is a ppt key generation algorithm that on input public parameters pp outputs a key pair (pk, sk). We assume that the public key pk includes the public parameters pp and public key pk defines the message space $\mathcal{M}$.
- $\sigma \leftarrow \mathsf{Sign}(pk, sk, m)$ is a ppt signing algorithm that on input a public key pk, a secret key sk, and a message $m \in \mathcal{M}$ outputs a signature σ.
- $0/1 \leftarrow \mathsf{Verify}(pk, m, \sigma)$ is a dpt verification algorithm that on input a public key pk, a message $m \in \mathcal{M}$, and a signature σ outputs a bit $b \in \{0, 1\}$, where $b = 1$ means valid and $b = 0$ means invalid.

$\diamond$

We require that a signature scheme is correct which intuitively means that, under an honest setup and honestly generated keys, all honestly generated signatures are declared as valid by the verification algorithm.

Definition 2.3.5 (Correctness)
A signature scheme Sig is *correct*, if for all $\lambda \in \mathbb{N}$, all $pp \in \mathsf{Setup}(1^\lambda)$, all $(pk, sk) \in \mathsf{KeyGen}(pp)$ and all $m \in \mathcal{M}$ it holds that $\mathsf{Verify}(pk, m, \mathsf{Sign}(pk, sk, m)) = 1$. $\diamond$

Let us next define the security of a digital signature scheme, called unforgeability. For an adversary, also called forger, it should be infeasible to output a valid signature on a message that was never queried to the provided signing oracle.

Definition 2.3.6 (EUF-CMA)
Let Sig be a signature scheme and $\mathcal{A}$ an adversary. We define the advantage in the experiment $\mathrm{Exp}^{\mathsf{euf\text{-}cma}}_{\mathcal{A},\mathsf{Sig}}(\lambda)$ (Experiment 2.2) as

$$\mathrm{Adv}^{\mathsf{euf\text{-}cma}}_{\mathcal{A},\mathsf{Sig}}(\lambda) \coloneqq \Pr\Big[\mathrm{Exp}^{\mathsf{euf\text{-}cma}}_{\mathcal{A},\mathsf{Sig}}(\lambda) = 1\Big]$$

and call Sig *existentially unforgeable against adaptively chosen message attacks* (EUF-CMA) if for all ppt adversaries $\mathcal{A}$, it holds that $\mathrm{Adv}^{\mathsf{euf\text{-}cma}}_{\mathcal{A},\mathsf{Sig}}(\lambda) = \mathsf{negl}(\lambda)$. $\diamond$

$\underline{\mathrm{Exp}^{\mathsf{euf\text{-}cma}}_{\mathcal{A},\mathsf{Sig}}(\lambda)}$

1 : $Q \coloneqq \emptyset$
2 : $pp \leftarrow \mathsf{Setup}(1^\lambda)$
3 : $(pk, sk) \leftarrow \mathsf{KeyGen}(pp)$
4 : $(m, \sigma) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Sign}}_{pk,sk}}(pk)$
5 : **return** 1 **if**
6 : $\quad \mathsf{Verify}(pk, m, \sigma) = 1 \wedge$
7 : $\quad m \notin Q$
8 : **else return** 0

$\underline{\mathcal{O}^{\mathsf{Sign}}_{pk,sk}(m_i)}$

1 : $\sigma_i \leftarrow \mathsf{Sign}(pk, sk, m_i)$
2 : $Q \coloneqq Q \cup \{m_i\}$
3 : **return** σ_i

Experiment 2.2: Unforgeability experiment for a digital signature scheme.

The above definition of unforgeability is also called *weak*, since we only require that the output message of the adversary is new, i.e., $m \notin Q$. To define the *strong* version we have to require that the output message-signature pair was never queried by the adversary, i.e., $(m, \sigma) \notin Q$.

2.3.3 Blind Signatures

In the following we formalize blind signature (BS) schemes which capture that a user should be able to get a message signed by another party, called signer, holding a secret key without revealing the message to the signer. The concept of blind signatures were first introduced in [Cha82]. For a detailed discussion of the field of blind signatures we refer to [FHKS16; FPS20]. Constructions that use blind signatures, e.g., group signatures and anonymous credentials, provide further semantics that explain why a signer should blindly sign a message that it does not know. For example, the user provides a proof that shows that a predetermined statement about the message is true, e.g., that the message corresponds to the user's unique identity. Here, the user's goal is to get a signature on her identity without revealing her identity to the signer. We consider in this thesis blind signatures that follow a commit-then-sign structure like "user commits to the message and then a signature is jointly computed". To this end, we explicitly define the commitment step separately from the interactive signing protocol. Hence, the signing protocol is of the form "signing a committed value". Anticipatory, this allows us in a credential system to use an argument system to prove properties of the commitment before signing it, see Chapter 5.

Definition 2.3.7 (Blind Signature Scheme)
A blind signature scheme BS consists of the algorithms $\mathsf{Setup}, \mathsf{KeyGen}$, Commit, $\mathsf{BlindSign}$, $\mathsf{BlindRcv}$, and Verify.

- $pp \leftarrow \mathsf{Setup}(1^\lambda)$ is a ppt setup algorithm that on input security parameter 1^λ outputs public parameters pp. We assume that the public parameters pp implicitly define parameters for the message space.
- $(pk, sk) \leftarrow \mathsf{KeyGen}(pp, 1^n)$ is a ppt key generation algorithm that on input public parameters pp outputs a key pair (pk, sk). We assume that the public key pk defines the message space $\mathcal{M}^n$ and commitment randomness space $\mathcal{R}^{\mathsf{Com}}$.
- $com \leftarrow \mathsf{Commit}(pp, pk, m, r)$ is a dpt committing algorithm that on input the public parameters pp, public key pk, message $m \in \mathcal{M}^n$, and commitment randomness $r \in \mathcal{R}^{\mathsf{Com}}$ outputs a commitment com.
- $\sigma \leftarrow \mathsf{BlindRcv}(pp, pk, m, r) \leftrightarrow \mathsf{BlindSign}(pp, pk, sk, com)$ is an interactive protocol with common input public parameters pp and public key pk between receiver's ppt algorithm $\mathsf{BlindRcv}$ and signer's ppt algorithm $\mathsf{BlindSign}$. $\mathsf{BlindSign}$ gets as additional input a secret key sk and a commitment com. $\mathsf{BlindRcv}$ with additional input a message $m \in \mathcal{M}^n$ and commitment randomness $r \in \mathcal{R}^{\mathsf{Com}}$ outputs a signature σ or failure symbol $\perp$ (if the interaction was not successful).
- $0/1 \leftarrow \mathsf{Verify}(pp, pk, m, \sigma)$ is a dpt verification algorithm that on input public parameters pp, public key pk, a message $m \in \mathcal{M}^n$, and a signature σ outputs a bit $b \in \{0, 1\}$, where $b = 1$ means valid and $b = 0$ means invalid.

$\diamond$

An example of the parameters for the message space is that the public parameters pp contain a group description which fixes the structure of the messages, e.g., messages contain elements from $\mathbb{Z}_p$. Then, the public key pk defines the length of the supported messages, e.g., $\mathcal{M}^n := \mathbb{Z}_p^n$.

In the following, we call a message-randomness pair (m, r) corresponding to a commitment $com \leftarrow \mathsf{Commit}(pp, pk, m, r)$ an opening of the commitment com. Further, we require that an BS scheme is correct, unforgeable, and message private as defined in the following.

Definition 2.3.8 (Correctness)
A blind signature scheme BS is *correct*, if for all $\lambda, n \in \mathbb{N}$, all $pp \in \mathsf{Setup}(1^\lambda)$, all $(pk, sk) \in \mathsf{KeyGen}(pp, 1^n)$, all $m \in \mathcal{M}^n$, and all commitment randomness $r \in \mathcal{R}^{\mathsf{Com}}$ it holds that

$$\Pr[\sigma \leftarrow \mathsf{BlindRcv}(pp, pk, m, r) \leftrightarrow \mathsf{BlindSign}(pp, pk, sk, \mathsf{Commit}(pp, pk, m, r)) : \mathsf{Verify}(pp, pk, m, \sigma) = 1] = 1 \ .$$

$\diamond$

$\underline{\mathrm{Exp}^{\mathsf{euf\text{-}cma\text{-}bs}}_{\mathcal{A},\mathsf{BS}}(\lambda)}$

1 : $Q \coloneqq \emptyset$
2 : $pp \leftarrow \mathsf{Setup}(1^\lambda)$
3 : $(1^n, st) \leftarrow \mathcal{A}(pp)$
4 : $(pk, sk) \leftarrow \mathsf{KeyGen}(pp, 1^n)$
5 : $(m, \sigma) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{BlindSign}}_{pp,pk,sk}}(pk, st)$
6 : **return** 1 **if**
7 : $\quad \mathsf{Verify}(pp, pk, m, \sigma) = 1 \wedge$
8 : $\quad m \notin Q$
9 : **else return** 0

$\underline{\mathcal{O}^{\mathsf{BlindSign}}_{pp,pk,sk}(m_i, r_i)}$

1 : $com_i = \mathsf{Commit}(pp, pk, m_i, r_i)$
2 : $\mathcal{A} \leftrightarrow \mathsf{BlindSign}(pp, pk, sk, com_i)$
3 : $Q \coloneqq Q \cup \{m_i\}$

Experiment 2.3: Unforgeability experiment for blind signature schemes.

The following unforgeability notion for blind signatures captures, besides the unforgeability of the signatures, also the (computational) binding of the commitment. This means an adversary should not be able to open a commitment in two different ways.

Definition 2.3.9 (Unforgeability)
Let BS be a blind signature scheme and $\mathcal{A}$ an adversary. We define the advantage in the experiment $\mathrm{Exp}^{\mathsf{euf\text{-}cma\text{-}bs}}_{\mathcal{A},\mathsf{BS}}(\lambda)$ (Experiment 2.3) as

$$\mathrm{Adv}^{\mathsf{euf\text{-}cma\text{-}bs}}_{\mathcal{A},\mathsf{BS}}(\lambda) \coloneqq \Pr\left[\mathrm{Exp}^{\mathsf{euf\text{-}cma\text{-}bs}}_{\mathcal{A},\mathsf{BS}}(\lambda) = 1\right]$$

and call BS *existentially unforgeable against adaptively chosen message attacks* (short unforgeable) if for all ppt adversaries $\mathcal{A}$, it holds that $\mathrm{Adv}^{\mathsf{euf\text{-}cma\text{-}bs}}_{\mathcal{A},\mathsf{BS}}(\lambda) = \mathsf{negl}(\lambda)$. ◇

Next, we define the privacy for the receiver of a blind signature. Intuitively, we want that the commitment and the interaction with the signer does not leak any information about the message to be signed. This means, the following definition captures that the committed message is hidden from the signer and that the interaction with the signer is independent of the message, i.e., the interaction with the signer does not leak any information about the receiver's message.

Definition 2.3.10 (Message Privacy)
Let BS be a blind signature scheme and $\mathcal{A}$ an adversary. We define the advantage in the experiment $\mathrm{Exp}^{\mathsf{msg\text{-}priv\text{-}bs}}_{\mathcal{A},\mathsf{BS}}(\lambda)$ (Experiment 2.4) as

$$\mathrm{Adv}^{\mathsf{msg\text{-}priv\text{-}bs}}_{\mathcal{A},\mathsf{BS}}(\lambda) \coloneqq \left|2 \cdot \Pr\left[\mathrm{Exp}^{\mathsf{msg\text{-}priv\text{-}bs}}_{\mathcal{A},\mathsf{BS}}(\lambda) = 1\right] - 1\right|$$

$\underline{\mathrm{Exp}^{\mathsf{msg\text{-}priv\text{-}bs}}_{\mathcal{A},\mathsf{BS}}(\lambda)}$

1 : $L \coloneqq \emptyset$
2 : $pp \leftarrow \mathsf{Setup}(1^\lambda)$
3 : $(pk, st) \leftarrow \mathcal{A}(pp)$
4 : $b \leftarrow \{0,1\}$
5 : $b' \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Commit}}_{pp,pk,b}, \mathcal{O}^{\mathsf{BlindRcv}}_{pp,pk}(j)}(st)$
6 : **return** 1 **if**
7 : $\quad b = b'$
8 : **else return** 0

$\underline{\mathcal{O}^{\mathsf{Commit}}_{pp,pk,b}(m_i)}$

1 : $r_i \leftarrow \mathcal{R}^{\mathsf{Com}}$
2 : **if** $b = 0$ **then**
3 : $\quad m \leftarrow \mathcal{M}^n$
4 : **else**
5 : $\quad m \coloneqq m_i$
6 : $com_i \leftarrow \mathsf{Commit}(pp, pk, m, r_i)$
7 : $L \coloneqq L \cup (m, r_i)$
8 : **return** com_i

$\underline{\mathcal{O}^{\mathsf{BlindRcv}}_{pp,pk}(j)}$

1 : **parse** $L[j] = (m_j, r_j)$
2 : **if** $(m_j, r_j) = (\bot, \bot)$ **then**
3 : $\quad$ **ignore**
4 : $\sigma \leftarrow \mathsf{BlindRcv}(pp, pk, m_j, r_j) \leftrightarrow \mathcal{A}$

Experiment 2.4: Message privacy experiment for blind signature schemes.

and call BS perfectly *message private*, if for all (unbounded) adversaries $\mathcal{A}$ it holds that $\mathrm{Adv}^{\mathsf{msg\text{-}priv\text{-}bs}}_{\mathcal{A},\mathsf{BS}}(\lambda) = 0$.

◇

2.3.4 Signatures of Knowledge

Signatures of knowledge (SoKs) were first formalized by Chase and Lysyanskaya [CL06]. Before that, there where folklore constructions from the Fiat-Shamir transformation applied to non-interactive zero-knowledge proof of knowledge systems (mostly Sigma protocols [Dam10; FKMV12]) known. Intuitively, a SoK lets a signer generate a signature on an instance of a publicly known relation if the signer knows a witness for the instance. Here, the expressiveness of the signature of knowledge depends on the relation and the message. In general, SoKs are a generalization of

signature schemes.

The following definitions for a SoK scheme are originally presented by Groth and Maller [GM17]. We use their model and a concrete construction of a SoK to show and instantiate a generic attribute-based signature construction in Section 3.3.

Definition 2.3.11 (Signature of Knowledge)
A signature of knowledge scheme SoK for relation generator RGen consists of the algorithms $\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{SimSetup}$, and $\mathsf{SimSign}$.

- $pp \leftarrow \mathsf{Setup}(R)$ is a ppt setup algorithm that on input a relation $R \in \mathsf{RGen}(1^\lambda)$ returns public parameters pp. We assume that the public parameters pp define the message space $\mathcal{M}$.
- $\sigma \leftarrow \mathsf{Sign}(pp, x, w, m)$ is a ppt signing algorithm that on input pp, instance-witness pair (x, w) and a message $m \in \mathcal{M}$ returns a signature σ.
- $0/1 \leftarrow \mathsf{Verify}(pp, x, m, \sigma)$ is a dpt verification algorithm that on input pp, an instance x, a message $m \in \mathcal{M}$, and a signature σ outputs either 0 or 1.
- $(pp, td_{sim}) \leftarrow \mathsf{SimSetup}(R)$ is a ppt simulation setup algorithm that on input a relation $R \in \mathsf{RGen}(1^\lambda)$ returns public parameters pp and a trapdoor td_{sim}.
- $\sigma \leftarrow \mathsf{SimSign}(pp, td_{sim}, x, m)$ is a ppt signing algorithm that on input pp, a simulation trapdoor td_{sim}, an instance x, and a message $m \in \mathcal{M}$ returns a signature σ.

$\diamond$

We require correctness from a SoK scheme which guarantees that honestly generated signatures under an honest setup and keys are declared as valid by the verification algorithm, i.e., algorithm Verify outputs 1.

Definition 2.3.12 (Correctness)
A signature of knowledge scheme SoK for relation generator RGen is *correct*, if for all $\lambda \in \mathbb{N}$, all $R \in \mathsf{RGen}(1^\lambda)$, all $(x, w) \in R$, all $pp \in \mathsf{Setup}(R)$, all messages $m \in \mathcal{M}$, and all $\sigma \in \mathsf{Sign}(pp, x, w, m)$ it holds that $\mathsf{Verify}(pp, x, m, \sigma) = 1$ with certainty. $\diamond$

Simulatability. The verifier of a SoK should obtain no information about the witness w from an instance before seeing a signature. We model this property by defining a simulation signing algorithm that can generate signatures without using a witness. The simulation algorithm works under possibly different setup parameters, however the simulated parameters and simulated signatures are required to be indistinguishable from the real counterparts.

$\underline{\mathrm{Exp}^{\mathsf{sim}}_{\mathcal{A},\mathsf{SoK}}(\lambda)}$

1 : $R \leftarrow \mathsf{RGen}(1^\lambda)$
2 : $pp_1 \leftarrow \mathsf{Setup}(R)$
3 : $(pp_0, td_{sim}) \leftarrow \mathsf{SimSetup}(R)$
4 : $b \leftarrow \{0,1\}$
5 : $b' \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Sign}_b}_{pp_b, td_{sim}}}(pp_b)$
6 : **return** 1 **if**
7 : $\quad b = b'$
8 : **else return** 0

$\underline{\mathcal{O}^{\mathsf{Sign}_1}_{pp_0, td_{sim}}(x_i, w_i, m_i)}$

1 : **if** $(x_i, w_i) \notin R \vee m_i \notin \mathcal{M}$ **then**
2 : $\quad$ **ignore**
3 : $\sigma_i \leftarrow \mathsf{Sign}(pp_0, x_i, w_i, m)$
4 : **return** σ_i

$\underline{\mathcal{O}^{\mathsf{Sign}_0}_{pp_1, td_{sim}}(x_i, w_i, m_i)}$

1 : **if** $(x_i, w_i) \notin R \vee m_i \notin \mathcal{M}$ **then**
2 : $\quad$ **ignore**
3 : $\sigma_i \leftarrow \mathsf{SimSign}(pp_1, td_{sim}, x_i, m)$
4 : **return** σ_i

Experiment 2.5: Simulatability experiment for SoK.

Definition 2.3.13 (Simulatability)
Let SoK be a signature of knowledge scheme and let $\mathcal{A}$ be an adversary. We define the advantage in the experiment $\mathrm{Exp}^{\mathsf{sim}}_{\mathcal{A},\mathsf{SoK}}(\lambda)$ (Experiment 2.5) as

$$\mathrm{Adv}^{\mathsf{sim}}_{\mathcal{A},\mathsf{SoK}}(\lambda) \coloneqq \left| 2 \cdot \Pr\left[\mathrm{Exp}^{\mathsf{sim}}_{\mathcal{A},\mathsf{SoK}}(\lambda) = 1\right] - 1 \right|$$

and call SoK perfectly *simulatable*, if for all ppt adversaries $\mathcal{A}$ it holds that

$$\mathrm{Adv}^{\mathsf{sim}}_{\mathcal{A},\mathsf{SoK}}(\lambda) = 0 \ .$$

$\diamond$

Simulation-extractability. Since simulatability is considered, the adversary $\mathcal{A}$ has access to simulated signatures on false instances x, i.e., $\forall w \in \mathcal{W}\colon (x,w) \notin R$. Even then adversary $\mathcal{A}$ should not be able to issue valid signatures without *knowing* a witness. *Knowing* is modeled by requiring that there exists an extractor, that extracts a witness from a given proof. The following definition from [GM17] captures this. We adapt the naming of the definitions based on the weak and strong simulation-extractability notions defined in [BKSV21; Kos+15].

Definition 2.3.14 (White-Box Strong Simulation-Extractability)
Let SoK be a signature of knowledge scheme and let $\mathcal{A}$ be an adversary. We define the advantage in the experiment $\mathrm{Exp}^{\mathsf{wb\text{-}sok\text{-}ext}}_{\mathcal{A},\mathsf{Extr}_\mathcal{A},\mathsf{SoK}}(\lambda)$ (Experiment 2.6) as

$$\mathrm{Adv}^{\mathsf{wb\text{-}sok\text{-}ext}}_{\mathcal{A},\mathsf{Extr}_\mathcal{A},\mathsf{SoK}}(\lambda) \coloneqq \Pr\left[\mathrm{Exp}^{\mathsf{wb\text{-}sok\text{-}ext}}_{\mathcal{A},\mathsf{Extr}_\mathcal{A},\mathsf{SoK}}(\lambda) = 1\right]$$

$\mathrm{Exp}^{\mathsf{wb\text{-}sok\text{-}ext}}_{\mathcal{A},\mathsf{Extr}_{\mathcal{A}},\mathsf{SoK}}(\lambda)$

$R \leftarrow \mathsf{RGen}(1^\lambda); Q = \emptyset$
$(pp, td_{sim}) \leftarrow \mathsf{SimSetup}(R)$
$(x, m, \sigma) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathrm{S}}_{pp,td_{sim}}}(pp)$
$w \leftarrow \mathsf{Extr}_{\mathcal{A}}(trans_{\mathcal{A}})$
return 1 **if**
  $\mathsf{Verify}(pp, x, m, \sigma) = 1 \wedge$
  $(x, m, \sigma) \notin Q \ \wedge (x, w) \notin R$
else return 0

$\mathcal{O}^{\mathrm{S}}_{pp,td_{sim}}(x_i, m_i)$

$\sigma_i \leftarrow \mathsf{SimSign}(pp, td_{sim}, x_i, m_i)$
$Q = Q \cup \{(x_i, m_i, \sigma_i)\}$
return σ_i

Experiment 2.6: White-box strong simulation-extractability experiment for SoK.

and we call a simulatable SoK white-box strong *simulation-extractable* if for all ppt adversaries $\mathcal{A}$ there exists a ppt extractor $\mathsf{Extr}_{\mathcal{A}}$ such that $\mathrm{Adv}^{\mathsf{wb\text{-}sok\text{-}ext}}_{\mathcal{A},\mathsf{Extr}_{\mathcal{A}},\mathsf{SoK}}(\lambda) = \mathsf{negl}(\lambda)$. ◇

We call it *strong* simuation-extractable because the winning condition in Experiment 2.6 checks if the instance, message, and signature output by the adversary $\mathcal{A}$ was not queried before, i.e., $(x, m, \sigma) \notin Q$. A *weak* variant only requires that the signature σ was not queried before. This is similar to strong and weak EUF-CMA signatures.

Intuitively, the above definition says that the better the adversary is in breaking the simulation-extractability the better the extractor has to perform. To illustrate this, take any adversary and an extractor that outputs $\perp$ on every input, where $\perp$ is a failure symbol assumed not to be an element of any relation. Clearly, $\perp$ is then not a valid witness for the instance x that the adversary produced, i.e., $(x, \perp) \notin \mathbb{R}$ holds. Hence, the last part of the winning condition is already satisfied. The rest of the winning condition is then only up to the ability of the adversary. Overall, to break simulation-extractability an adversary has to accomplish that no extractor can succeed in extracting a valid witness for the instance even if it has access to the adversary's code, input, outputs, and random coins. Additionally, the instance, message, and signature tuple that the adversary outputs has to be valid *and* at the same time it has to be new, i.e., not queried before.

Definition 2.3.15 (Succinctness for SoK - [GM17])
Let λ be a security parameter and let x be an instance of some relation R. A signature of knowledge scheme SoK is called *succinct*, if the verifier's runtime is polynomial in $\lambda + |x|$ and the signature size is polynomial in λ. ◇

The white-box definition of simulation-extractability comes with technical challenges for concrete schemes, because it requires the existence of a white-box ex-

tractor and the existence relies on non-falsifiable assumptions, so called knowledge assumptions [BCCT12; GW11]. If one further requires very short signatures that are independent of the size of the witness (succinctness) one cannot hope that the witness is somehow encoded in the signature and easily extractable for the extractor without having access to the transcript of the adversary. Therefore, white-box definitions are typically used for applications where succinctness is important.

If succinctness is not a crucial point or falsifiable assumptions are favorable and to circumvent technical challenges there are also black-box simulation-extractable signatures of knowledge [BGPR20; CL06; FO11]. Constructing such schemes and working with them in security reductions is simpler since the extractor does not depend on any specific adversary. Here the black-box extractor (also called online extractor) works with an additional extraction trapdoor that enables the extraction of the witness given an instance, a message and a signature.

Definition 2.3.16 (Black-Box Strong Simulation-Extractability - [CL06])
Let SoK be a signature of knowledge scheme and let $\mathcal{A}$ be an adversary. We define the advantage in the experiment $\mathrm{Exp}^{\mathsf{bb\text{-}sok\text{-}ext}}_{\mathcal{A},\mathsf{Extr},\mathsf{SoK}}(\lambda)$ (Experiment 2.7) as

$$\mathrm{Adv}^{\mathsf{bb\text{-}sok\text{-}ext}}_{\mathcal{A},\mathsf{Extr},\mathsf{SoK}}(\lambda) \coloneqq \Pr\Big[\mathrm{Exp}^{\mathsf{bb\text{-}sok\text{-}ext}}_{\mathcal{A},\mathsf{Extr},\mathsf{SoK}}(\lambda) = 1\Big]$$

and call a simulatable SoK black-box strong *simulation-extractable* if (1) there exists a ppt algorithm $\mathsf{ExtrSetup}$ such that for all ppt adversaries $\mathcal{A}$ and for any $R \in \mathsf{RGen}(1^\lambda)$ it holds that

$$\Big|\Pr[(crs, td_{sim}) \leftarrow \mathsf{SimSetup}(R)\colon \mathcal{A}(crs) = 1] - \Pr[(crs, td_{sim}, td_{extr}) \leftarrow \mathsf{ExtrSetup}(R)\colon \mathcal{A}(crs) = 1]\Big|$$

is negligible and (2) there exists a ppt extractor Extr such that for all ppt adversaries $\mathcal{A}$ it holds $\mathrm{Adv}^{\mathsf{wb\text{-}sok\text{-}ext}}_{\mathcal{A},\mathsf{Extr}_{\mathcal{A}},\mathsf{SoK}}(\lambda) = \mathsf{negl}(\lambda)$. ◇

2.3.5 Non-Interactive Argument Systems

We present in the following two notions for argument systems; one in the common reference string (CRS) model, where a trusted party sets up a trusted (structured) string, and a second notion in the random oracle model (ROM). We opt to separate the notions to circumvent overly complex definitions and not to overload the notation which would be the case if we captured both notions in one set of definitions. Note, that in the literature there is also the term *proof system*. The difference is that security notions in proof systems are defined with respect to unbounded adversaries where in argument systems computationally bounded adversaries are considered, i.e., in argument systems they are ppt algorithms.

$\mathrm{Exp}^{\mathsf{bb\text{-}sok\text{-}ext}}_{\mathcal{A},\mathsf{Extr},\mathsf{SoK}}(\lambda)$

$R \leftarrow \mathsf{RGen}(1^\lambda); Q := \emptyset$
$(pp, td_{sim}, td_{extr}) \leftarrow \mathsf{ExtrSetup}(R)$
$(x, m, \sigma) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathrm{S}}_{pp,td_{sim}}}(pp)$
$w \leftarrow \mathsf{Extr}(pp, td_{extr}, x, m, \sigma, Q)$
return 1 **if**
  $\mathsf{Verify}(pp, x, m, \sigma) = 1 \wedge$
  $(x, m, \sigma) \notin Q \ \wedge (x, w) \notin R$
else return 0

$\mathcal{O}^{\mathrm{S}}_{pp,td_{sim}}(x_i, m_i)$

$\sigma_i \leftarrow \mathsf{SimSign}(pp, td_{sim}, x_i, m_i)$
$Q = Q \cup \{(x_i, m_i, \sigma_i)\}$
return σ_i

Experiment 2.7: Black-box strong simulation-extractability experiment for SoK.

Argument Systems with CRS

In the following, we use the definition from [GM17] for non-interactive zero-knowledge argument of knowledge (NIZK) in the CRS model. In this model the CRS is honestly generated by some trusted third party. Intuitively, a CRS determines public parameters that are available to all parties of a system, e.g., a group description and further public parameters of schemes that need a trusted setup. In real-world applications this is a drawback since a trusted third party is not always available. However, there are prominent example of systems that generated their CRS with a specialized multi-party computation that guarantees, as long as one party was honest, that the CRS was honestly and correctly generated, e.g., Zcash [COM22]. For further details on how to generate a CRS in real-world application we refer to [Ben+15; BGG19]. In the CRS model a so called trapdoor can usually be generated alongside the CRS that just exists for the modeling of security, i.e., it is used to define security and concretely to be able to prove the security of systems. For example, in an argument system a simulation trapdoor enables that an efficient simulation algorithm outputs simulated proofs that are indistinguishable from non-simulated proofs without using a witness. This notion is used to define a form of privacy, called zero-knowledge. Similarly, an extraction trapdoor for the CRS enables that an efficient extractor exists that can extract witnesses from non-simulated proofs. Intuitively, this notion is used to define a form of soundness for argument systems called extractability.

Definition 2.3.17 (Argument System)
An argument system Arg for relation generator RGen consists of algorithms Setup, $\mathsf{Prove}, \mathsf{Verify}$, and $\mathsf{SimProve}$.

- $(crs, td_{sim}) \leftarrow \mathsf{Setup}(R)$ is a ppt setup algorithm that on input a relation $R \in \mathsf{RGen}(1^\lambda)$ returns a common reference string crs and a simulation trapdoor td_{sim}.

- $\pi \leftarrow \mathsf{Prove}(crs, x, w)$ is a ppt proving algorithm that on input a common reference string crs and instance-witness pair $(x, w) \in R$ returns a proof π.
- $0/1 \leftarrow \mathsf{Verify}(crs, x, \pi)$ is a dpt verification algorithm that on input a common reference string crs, an instance x, and a proof π outputs either 0 or 1.
- $\pi \leftarrow \mathsf{SimProve}(crs, td_{sim}, x)$ is a ppt proof simulator algorithm that on input a common reference string crs, a simulation trapdoor td_{sim}, and an instance x returns a (simulated) proof π.

$\diamond$

Note, that in the above definition the CRS denoted as crs (and simulation trapdoor td_{sim}) depends on the relation R output by the relation generator RGen. This means that the simulation algorithm $\mathsf{SimProve}$ outputs simulated proofs for this specific relation R. For an argument system Arg we require that it is correct. Therefore, let us define what correctness means for an argument system Arg.

Definition 2.3.18 (Correctness)
An argument system Arg for relation generator RGen is perfectly *correct*, if for all $\lambda \in \mathbb{N}$, for all $R \in \mathsf{RGen}(1^\lambda)$, all $(x, w) \in R$, all $(crs, td_{sim}) \in \mathsf{Setup}(R)$, and all $\pi \in \mathsf{Prove}(crs, x, w)$ it holds that $\mathsf{Verify}(crs, x, \pi) = 1$. $\diamond$

Security notions for argument system include witness-indistinguishability, zero-knowledge, soundness, and simulation-extractability. Witness-indistinguishability captures that two proofs for the same instance generated using different witnesses should be indistinguishable for an adversary.

Definition 2.3.19 (Witness-Indistinguishability)
Let Arg be an argument system for relation generator RGen and let $\mathcal{A}$ be an adversary. We define the advantage in the experiment $\mathrm{Exp}^{\mathsf{wi}}_{\mathcal{A},\mathsf{Arg}}(\lambda)$ (Experiment 2.8) as

$$\mathrm{Adv}^{\mathsf{wi}}_{\mathcal{A},\mathsf{Arg}}(\lambda) \coloneqq \left| 2 \cdot \Pr\left[\mathrm{Exp}^{\mathsf{wi}}_{\mathcal{A},\mathsf{Arg}}(\lambda) = 1\right] - 1 \right|$$

and call argument system Arg a non-interactive *witness-indistinguishable* (WI) argument system if for all ppt adversaries $\mathcal{A}$, it holds that $\mathrm{Adv}^{\mathsf{wi}}_{\mathcal{A},\mathsf{Arg}}(\lambda) = 0$. $\diamond$

The term zero-knowledge is motivated by the intuitive description that what an adversary can compute before engaging with the prover should be the same compared to what the adversary can compute after engaging with the prover. This means the adversary gained no knowledge by interacting with the prover. In the security definition this is captured by requiring that it should be infeasible for an adversary to distinguish if it gets simulated proofs, generated without using the witness, or real proofs, generated with the witness for the instance. Intuitively, if there is an efficient algorithm that can simulate indistinguishable proofs without

$\underline{\mathrm{Exp}^{\mathsf{wi}}_{\mathcal{A},\mathsf{Arg}}(\lambda)}$

$R \leftarrow \mathsf{RGen}(1^\lambda)$
$(crs, td_{sim}) \leftarrow \mathsf{Setup}(R)$
$b \leftarrow \{0,1\}$
$b' \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Prove}_b}_{crs}}(crs)$
return 1 **if**
  $b = b'$
else return 0

$\underline{\mathcal{O}^{\mathsf{Prove}}_{crs}(x_i, w_{i,0}, w_{i,1})}$

if $(x_i, w_{i,0}) \notin R \vee (x_i, w_{i,1}) \notin R$ **then**
  ignore
$\pi_i \leftarrow \mathsf{Prove}(crs, x_i, w_{i,b})$
return π_i

Experiment 2.8: Witness-indistinguishability experiment for argument systems with a CRS.

$\underline{\mathrm{Exp}^{\mathsf{zk}}_{\mathcal{A},\mathsf{Arg}}(\lambda)}$

$R \leftarrow \mathsf{RGen}(1^\lambda)$
$(crs, td_{sim}) \leftarrow \mathsf{Setup}(R)$
$b \leftarrow \{0,1\}$
$b' \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Prove}_b}_{crs,td_{sim}}(crs)}$
return 1 **if**
  $b = b'$
else return 0

$\underline{\mathcal{O}^{\mathsf{Prove}_1}_{crs,td_{sim}}(x_i, w_i)}$

if $(x_i, w_i) \notin R$ **then**
  ignore
$\pi_i \leftarrow \mathsf{Prove}(crs, x_i, w_i)$
return π_i

$\underline{\mathcal{O}^{\mathsf{Prove}_0}_{crs,td_{sim}}(x_i, w_i)}$

if $(x_i, w_i) \notin R$ **then**
  ignore
$\pi_i \leftarrow \mathsf{SimProve}(crs, td_{sim}, x_i)$
return π_i

Experiment 2.9: Zero-knowledge experiment for argument systems with a CRS.

using a witness, then it must hold that real proofs do not leak anything about the witness.

Definition 2.3.20 (Zero-Knowledge)
Let Arg be an argument system for relation generator RGen and let $\mathcal{A}$ be an adversary. We define the advantage in the experiment $\mathrm{Exp}^{\mathsf{zk}}_{\mathcal{A},\mathsf{Arg}}(\lambda)$ (Experiment 2.9) as

$$\mathrm{Adv}^{\mathsf{zk}}_{\mathcal{A},\mathsf{Arg}}(\lambda) \coloneqq \left|2 \cdot \Pr\left[\mathrm{Exp}^{\mathsf{zk}}_{\mathcal{A},\mathsf{Arg}}(\lambda) = 1\right] - 1\right|$$

and call argument system Arg a non-interactive (perfect) *zero-knowledge* argument system if for all ppt adversaries $\mathcal{A}$, it holds that $\mathrm{Adv}^{\mathsf{zk}}_{\mathcal{A},\mathsf{Arg}}(\lambda) = 0$. ◇

Next, let us define extractability for argument systems, where we require that

$\underline{\mathrm{Exp}^{\mathsf{bb\text{-}extr}}_{\mathcal{A},\mathsf{Arg}}(\lambda)}$

$R \leftarrow \mathsf{RGen}(1^\lambda)$
$(crs, td_{extr}) \leftarrow \mathsf{ExtrSetup}(R)$
$(x, \pi) \leftarrow \mathcal{A}(crs)$
$w \leftarrow \mathsf{Extr}(crs, td_{extr}, x, \pi)$
return 1 **if**
  $\mathsf{Verify}(crs, x, \pi) = 1 \wedge$
  $(x, w) \notin R$
else return 0

Experiment 2.10: Extractability experiment for argument systems with a CRS.

there is an efficient extraction algorithm that works for every adversary and extracts the witness of a given proof. This extractor is also called a black-box extractor.

Definition 2.3.21 (Black-Box Extractability)
Let Arg be an argument system for relation generator RGen and let $\mathcal{A}$ be an adversary. We define the advantage in the experiment $\mathrm{Exp}^{\mathsf{bb\text{-}extr}}_{\mathcal{A},\mathsf{Arg}}(\lambda)$ (Experiment 2.10) as

$$\mathrm{Adv}^{\mathsf{bb\text{-}extr}}_{\mathcal{A},\mathsf{Arg}}(\lambda) \coloneqq \Pr\left[\mathrm{Exp}^{\mathsf{bb\text{-}extr}}_{\mathcal{A},\mathsf{Arg}}(\lambda) = 1\right]$$

and call Arg computational black-box *extractable* or an (black-box) *argument of knowledge*, if (1) there exists a ppt algorithm $\mathsf{ExtrSetup}$ such that for all ppt adversaries $\mathcal{A}$ and for any $R \in \mathsf{RGen}(1^\lambda)$ it holds that

$$\Big|\Pr[(crs, td_{sim}) \leftarrow \mathsf{Setup}(R) \colon \mathcal{A}(crs) = 1] - \Pr[(crs, td_{extr}) \leftarrow \mathsf{ExtrSetup}(R) \colon \mathcal{A}(crs) = 1]\Big|$$

is negligible and (2) if there exists a ppt algorithm Extr such that for all ppt adversary $\mathcal{A}$ it holds that $\mathrm{Adv}^{\mathsf{bb\text{-}extr}}_{\mathcal{A},\mathsf{Arg}}(\lambda) = \mathsf{negl}(\lambda)$. If the advantage $\mathrm{Adv}^{\mathsf{bb\text{-}extr}}_{\mathcal{A},\mathsf{Arg}}(\lambda)$ is 0 for an (unbounded) adversary $\mathcal{A}$ then we call Arg perfect extractable. ◇

Definition 2.3.22 (NIWI)
We call a *witness-indistinguishable* and *extractable* argument system a non-interactive witness indistinguishable argument of knowledge (NIWI). ◇

An example of such a NIWI is the Groth-Sahai proof system [GS12].

Next, we define the notion of extractability via a white-box extractor that depends on the adversary and has access to the code and randomness of the adver-

$\mathrm{Exp}^{\mathsf{wb\text{-}extr}}_{\mathcal{A},\mathsf{Extr}_{\mathcal{A}},\mathsf{Arg}}(\lambda)$

$R \leftarrow \mathsf{RGen}(1^\lambda)$
$(crs, td_{sim}) \leftarrow \mathsf{Setup}(R)$
$(x, \pi) \leftarrow \mathcal{A}(crs)$
$w \leftarrow \mathsf{Extr}_{\mathcal{A}}(trans_{\mathcal{A}})$
return 1 **if**
  $\mathsf{Verify}(crs, x, \pi) = 1 \wedge$
  $(x, w) \notin R$
else return 0

Experiment 2.11: White-box extractability experiment for argument systems with a CRS.

sary. Hence, an argument system white-box extractable if from every algorithm outputting a valid proof it is possible to efficiently extract a valid witness.

Definition 2.3.23 (White-Box Extractability)
Let Arg be an argument system for relation generator RGen and let $\mathcal{A}$ be an adversary. We define the advantage in the experiment $\mathrm{Exp}^{\mathsf{wb\text{-}extr}}_{\mathcal{A},\mathsf{Extr}_{\mathcal{A}},\mathsf{Arg}}(\lambda)$ (Experiment 2.11) as

$$\mathrm{Adv}^{\mathsf{wb\text{-}extr}}_{\mathcal{A},\mathsf{Extr}_{\mathcal{A}},\mathsf{Arg}}(\lambda) \coloneqq \Pr\left[\mathrm{Exp}^{\mathsf{wb\text{-}extr}}_{\mathcal{A},\mathsf{Extr}_{\mathcal{A}},\mathsf{Arg}}(\lambda) = 1\right]$$

and call Arg white-box computational *extractable* or an (white-box) argument of knowledge if for all ppt adversary $\mathcal{A}$, there exists a ppt extractor $\mathsf{Extr}_{\mathcal{A}}$ such that $\mathrm{Adv}^{\mathsf{wb\text{-}extr}}_{\mathcal{A},\mathsf{Extr}_{\mathcal{A}},\mathsf{Arg}}(\lambda) = \mathsf{negl}(\lambda)$. ◇

Next, we define simulation-extractability for argument systems. Intuitively, this definition is a combination of the zero-knowledge property and extractability. Hence, it captures that, even if the adversary can query simulated proofs of its choice, it is infeasible for the adversary to produce a new proof such that no extractor can output the witness that the adversary used.

Definition 2.3.24 (Simulation-Extractability)
Let Arg be a non-interactive zero-knowledge argument system for relation generator RGen and let $\mathcal{A}$ be an adversary. We define the advantage in the experiment $\mathrm{Exp}^{\mathsf{sim\text{-}ext}}_{\mathcal{A},\mathsf{Extr}_{\mathcal{A}},\mathsf{Arg}}(\lambda)$ (Experiment 2.12) as

$$\mathrm{Adv}^{\mathsf{sim\text{-}ext}}_{\mathcal{A},\mathsf{Extr}_{\mathcal{A}},\mathsf{Arg}}(\lambda) \coloneqq \Pr[\mathrm{Exp}^{\mathsf{sim\text{-}ext}}_{\mathcal{A},\mathsf{Extr}_{\mathcal{A}},\mathsf{Arg}}(\lambda) = 1]$$

and call Arg (white-box) *simulation-extractable* (sim-ext) if for all ppt adversaries $\mathcal{A}$ there exists a ppt extractor $\mathsf{Extr}_{\mathcal{A}}$ such that $\mathrm{Adv}^{\mathsf{sim\text{-}ext}}_{,\mathcal{A},\mathsf{Extr}_{\mathcal{A}},\mathsf{Arg}}(\lambda) = \mathsf{negl}(\lambda)$. ◇

$\mathrm{Exp}^{\mathsf{sim\text{-}ext}}_{\mathcal{A},\mathsf{Extr}_{\mathcal{A}},\mathsf{Arg}}(\lambda)$

1 : $R \leftarrow \mathsf{RGen}(1^\lambda); Q = \emptyset$
2 : $(crs, td_{sim}) \leftarrow \mathsf{Setup}(R)$
3 : $(x, \pi) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{SimProve}}_{crs,td_{sim}}}(crs)$
4 : $w \leftarrow \mathsf{Extr}_{\mathcal{A}}(trans_{\mathcal{A}})$
5 : **return** 1 **if**
6 : $\quad \mathsf{Verify}(crs, x, \pi) = 1 \wedge$
7 : $\quad (x, \pi) \notin Q \wedge (x, w) \notin R$
8 : **else return** 0

$\mathcal{O}^{\mathsf{SimProve}}_{crs,td_{sim}}(x_i)$

1 : $\pi_i \leftarrow \mathsf{SimProve}(crs, td_{sim}, x_i)$
2 : $Q = Q \cup \{(x_i, \pi_i)\}$
3 : **return** π_i

Experiment 2.12: Simulation-extractability experiment for argument systems with a CRS.

Observe that simulation-extractability implies extractability where the adversary has no access to a simulation oracle. An important property of an argument system for real-world applications is the length of the proofs that it produces. Intuitively, argument systems with "very short" proofs (independent of the size of the witness) are called succinct.

Definition 2.3.25 (Succinctness for Argument Systems - [GM17])
A non-interactive argument system where the verifier runs in time polynomially in $\lambda + |x|$ and the proof size is polynomial in λ is called *succinct*. ◇

In the following we define some naming conventions of argument systems that we use in this thesis.

Definition 2.3.26 (SNARK, NIZK, SE-SNARK)
A *succinct non-interactive argument of knowledge* is a SNARK. A *succinct non-interactive zero-knowledge argument of knowledge* is called a succinct NIZK, and a *succinct simulation-extractable* NIZK argument is called SE-SNARK. ◇

Argument Systems in the ROM

In the following we define argument systems in the random oracle model (ROM) [BR93]. The ROM follows a paradigm that simplifies the design and security proofs of cryptographic schemes while capturing elements that are used in a real-world implementation. Intuitively a random oracle is black-box that is available to all parties. On a query that defines input x of the black-box, it checks whether it has already answered a query for x. If yes it uses the previous answer to the query as its output. If it has not yet answered a query for input x it chooses every bit of its output uniformly at random. Let us describe the paradigm of the ROM in more detail. In the ROM all parties have access to a random oracle RO which

technically is a random function mapping arbitrary bit strings to bit strings of a specific length. Then one constructs a cryptographic scheme in the ROM and proves its security with respect to the random oracle RO. Then one replaces the random oracle RO with a hash function that "behaves like a random oracle". In practice this means a random oracle is instantiated by SHA-2 or SHA-3. Note, the last step is only heuristic and informed by practical experience. For more details on the paradigm and formalism we refer to [BR93; MF21].

Compared to the CRS model there is no trusted party in the ROM that sets up a string or the random oracle. Hence, simulation (extraction) does not work with a trapdoor that is generated alongside the CRS rather the random oracle is used to simulate. Argument systems in the ROM can be online extractable (straight-line extractable) [Fis05] or extractable via rewinding [FS87].

The most well known transformations that give us practical argument systems without a CRS in the ROM are the Fischlin transformation [BFW15; Fis05] and the Fiat-Shamir transformation [FKMV12; FS87]. The following definitions are based on [Fis05; FKMV12; IV19] adapted to our notation. Hence, we rely on the explicitly programmable random oracle model [Wee09] where a simulator can explicitly program the random oracle. This means, a simulator can program the output of the random oracle on inputs adaptively and the simulator outputs the programming, e.g., as part of its state.

Definition 2.3.27 (Argument System in the ROM)
An argument system Arg in the ROM for relation R consists of algorithms Setup, Prove, and Verify with oracle access to random oracle RO.

- $pp \leftarrow \mathsf{Setup}(1^\lambda)$ is a ppt setup algorithm that on input security parameter λ returns public parameters pp.
- $\pi \leftarrow \mathsf{Prove}^{\mathsf{RO}}(pp, x, w, ctx)$ is a ppt proving algorithm that on input public parameters pp, instance-witness pair $(x, w) \in R$, and context ctx returns a proof π.
- $0/1 \leftarrow \mathsf{Verify}^{\mathsf{RO}}(pp, x, \pi, ctx)$ is a dpt verification algorithm that on input public parameters pp, an instance x, a proof π, and context ctx outputs either 0 or 1.

$\diamond$

We include a context ctx in the definition of argument systems in the ROM to explicitly state that the context of the proof should be considered during the proof generation and verification, e.g., a session identifier, verifier's identity, a timestamp or a random nonce. By this, we bound the proof to the context to avoid replay attacks, where a proof (potentially generated by a honest user) is just replayed by an adversary to a verifier. Since, we use an argument system in the ROM to build anonymous credentials (Chapter 5) where replay attacks are a real-world threat we already consider replay attacks in this building block.

We require that an argument system in the ROM is correct.

Definition 2.3.28 (Correctness)
An argument system Arg in the ROM for relation R is *correct*, if for all $\lambda \in \mathbb{N}$, all random oracles RO, all public parameters $pp \in \mathsf{Setup}(1^\lambda)$, all instance-witness pairs $(x, w) \in R$, all contexts $ctx \in \{0,1\}^*$, and all proofs $\pi \in \mathsf{Prove}^{\mathsf{RO}}(pp, x, w, ctx)$ it holds that

$$\Pr\Big[\mathsf{Verify}^{\mathsf{RO}}(pp, x, \pi, ctx) = 0\Big] = \mathsf{negl}(\lambda) \ .$$

◇

Next, we want to define the notion that a proof generated by an argument system in the ROM does not reveal any information to an adversary. We define this with the help of a *stateful* ppt simulator $\mathsf{Sim} := (\mathsf{SimRO}, \mathsf{SimProve})$ where ppt algorithm $\mathsf{SimRO}(v, st)$ programs the random oracle RO at position v and ppt algorithm $\mathsf{SimProve}(pp, x, ctx, st)$ simulates a proof for instance x and context ctx without a witness as input. Both algorithms use state st, that is synced between the algorithms. Hence, they share a common state that gets implicitly updated by each run of the algorithms of Sim.

Definition 2.3.29 (Zero-Knowledge)
Let Arg be an argument system in the ROM for relation R, let $\mathcal{A}$ be an adversary, and RO a random oracle. We define the advantage in the experiment $\mathrm{Exp}^{\mathsf{zk\text{-}rom}}_{\mathcal{A},\mathsf{Arg}}(\lambda)$ (Experiment 2.13) as

$$\mathrm{Adv}^{\mathsf{zk\text{-}rom}}_{\mathcal{A},\mathsf{Arg}}(\lambda) := \Big|2 \cdot \Pr\Big[\mathrm{Exp}^{\mathsf{zk\text{-}rom}}_{\mathcal{A},\mathsf{Arg}}(\lambda) = 1\Big] - 1\Big|$$

and call argument system Arg a non-interactive *zero-knowledge* argument system (in the ROM) if there exists a *stateful* ppt simulator $\mathsf{Sim} := (\mathsf{SimRO}, \mathsf{SimProve})$ such that for all ppt adversaries $\mathcal{A}$, it holds that $\mathrm{Adv}^{\mathsf{zk\text{-}rom}}_{\mathcal{A},\mathsf{Arg}}(\lambda) = \mathsf{negl}(\lambda)$. ◇

With zero-knowledge we have a security guarantee for the prover in place, but the verifier also wants to have the guarantee that no (adversarial) prover can convince it that a statement is true if it is indeed false. To this end, we define extractability in the random oracle model.

Definition 2.3.30 (Extractability - [Fis05])
Let Arg an argument system in the ROM for relation R and let $\mathcal{A}$ be an adversary, and RO a random oracle. We define the advantage in the experiment $\mathrm{Exp}^{\mathsf{extr\text{-}rom}}_{\mathcal{A},\mathsf{Arg}}(\lambda)$ (Experiment 2.14) as

$$\mathrm{Adv}^{\mathsf{extr\text{-}rom}}_{\mathcal{A},\mathsf{Arg}}(\lambda) := \Pr\Big[\mathrm{Exp}^{\mathsf{extr\text{-}rom}}_{\mathcal{A},\mathsf{Arg}}(\lambda) = 1\Big]$$

and call Arg *extractable* or *argument of knowledge* (in the ROM) if there exists a

$\underline{\mathrm{Exp}^{\mathsf{zk\text{-}rom}}_{\mathcal{A},\mathsf{Arg}}(\lambda)}$

1 : $pp \leftarrow \mathsf{Setup}(1^\lambda)$
2 : $b \leftarrow \{0,1\}$
3 : $b' \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{RO}_b},\mathcal{O}^{\mathsf{Prove}_b}_{pp}}(pp)$
4 : **return** 1 **if**
5 : $\quad b = b'$
6 : **else return** 0

$\underline{\mathcal{O}^{\mathsf{RO}_1}(v_i)}$

1 : **return** $\mathsf{RO}(v_i)$

$\underline{\mathcal{O}^{\mathsf{RO}_0}(v_i)}$

1 : $(y_i, st) \leftarrow \mathsf{SimRO}(v_i, st)$
2 : **return** y_i

$\underline{\mathcal{O}^{\mathsf{Prove}_1}_{pp}(x_i, w_i, ctx_i)}$

1 : **if** $(x_i, w_i) \notin R$ **then**
2 : $\quad$ **ignore**
3 : $\pi_i \leftarrow \mathsf{Prove}^{\mathcal{O}^{\mathsf{RO}_b}}(pp, x_i, w_i, ctx_i)$
4 : **return** π_i

$\underline{\mathcal{O}^{\mathsf{Prove}_0}_{pp}(x_i, w_i, ctx_i)}$

1 : **if** $(x_i, w_i) \notin R$ **then**
2 : $\quad$ **ignore**
3 : $(\pi_i, st) \leftarrow \mathsf{SimProve}(pp, x_i, ctx_i, st)$
4 : **return** π_i

Experiment 2.13: Zero-knowledge experiment for argument systems in the ROM.

ppt extractor Extr such that for all ppt adversaries $\mathcal{A}$, it holds that

$$\mathrm{Adv}^{\mathsf{extr\text{-}rom}}_{\mathcal{A},\mathsf{Arg}}(\lambda) = \mathsf{negl}(\lambda) \ .$$

◇

If the adversary even gets access to simulated proofs we still have to be able to extract. This notion is captured in the following simulation-extractability definition in the random oracle model.

Definition 2.3.31 (Simulation-Extractability)
Let Arg a non-interactive zero-knowledge argument system in the ROM for relation R, let $\mathcal{A}$ be an adversary, and let RO be a random oracle. We define the advantage

$\mathrm{Exp}^{\mathsf{extr\text{-}rom}}_{\mathcal{A},\mathsf{Arg}}(\lambda)$

$Q_H = \emptyset$
$pp \leftarrow \mathsf{Setup}(1^\lambda)$
$(x, \pi, ctx) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{RO}}}(pp)$
$w \leftarrow \mathsf{Extr}(pp, x, \pi, ctx, Q_H)$
return 1 **if**
  $\mathsf{Verify}^{\mathcal{O}^{\mathsf{RO}}}(pp, x, \pi, ctx) = 1 \wedge$
  $(x, w) \notin R$
else return 0

$\mathcal{O}^{\mathsf{RO}}(v_i)$

$y_i \leftarrow \mathsf{RO}(v_i)$
$Q_H = Q_H \cup \{(v_i, y_i)\}$
return y_i

Experiment 2.14: Extractability experiment for argument systems in the ROM.

in the experiment $\mathrm{Exp}^{\mathsf{sim\text{-}ext\text{-}rom}}_{\mathcal{A},\mathsf{Arg}}(\lambda)$ (Experiment 2.15) as

$$\mathrm{Adv}^{\mathsf{sim\text{-}ext\text{-}rom}}_{\mathcal{A},\mathsf{Arg}}(\lambda) \coloneqq \Pr\Big[\mathrm{Exp}^{\mathsf{sim\text{-}ext\text{-}rom}}_{\mathcal{A},\mathsf{Arg}}(\lambda) = 1\Big]$$

and call Arg *simulation-extractable* (in the ROM) if there exists a ppt extractor Extr such that for all ppt adversaries $\mathcal{A}$, it holds that $\mathrm{Adv}^{\mathsf{sim\text{-}ext\text{-}rom}}_{\mathcal{A},\mathsf{Arg}}(\lambda) = \mathsf{negl}(\lambda)$. ◇

CS notation. We use in Chapter 5 the standard notation, called CS notation, to denote argument system proofs which was introduced by Camenisch and Stadler [CS97]. This means a proof generated by an argument system Arg for a *statement* (corresponding to an instance) and context *ctx* with a list of *witnesses* is denoted as:

$$\mathsf{Arg}[(witnesses) : statement\ ](ctx)$$

For example,

$$\mathsf{Arg}[(u,a,c,r) : v_1 = g_1^u h_1^{c+10} \wedge v_2 = g_2^a h_2^r \wedge ((10 < c < 100) \vee a \in S)](ctx)$$

denotes a proof generated by algorithm Prove of an argument system Arg that proves knowledge of the witnesses u, a, c, r such that the predicates given in the statement are satisfied. In the above example we include a disjunction of a range proof $(10 < c < 100)$ and a set membership proof $a \in S$ to show the support for rich statement. There is a large body of work on efficient argument systems that realize the statements captured by the CS notation, e.g., [Bem+18; Cam+16b; CCs08; CDS94; CM99; Fis05; FKMV12; GOS12; Gro06; Gro16; GS08; GS12; MKWK19; Sch91]. This includes the definition of Sigma protocols [Dam10; FKMV12] as a generalization of Schnorr's protocol [Cam+16b; Sch91], composition of Sigma protocols in Boolean formulas [CDS94], set membership proofs and range proofs

$\mathrm{Exp}^{\mathsf{sim\text{-}ext\text{-}rom}}_{\mathcal{A},\mathsf{Arg}}(\lambda)$

$Q_H, Q_P = \emptyset$
$pp \leftarrow \mathsf{Setup}(1^\lambda)$
$(x, \pi, ctx) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{RO}}, \mathcal{O}^{\mathsf{SimProve}}_{pp}}(pp)$
$w \leftarrow \mathsf{Extr}(pp, st, x, \pi, ctx, Q_H, Q_P)$
return 1 **if**
  $\mathsf{Verify}^{\mathcal{O}^{\mathsf{RO}}}(pp, x, \pi, ctx) = 1 \wedge$
  $(x, \pi, ctx) \notin Q_P \wedge (x, w) \notin R$
else return 0

$\mathcal{O}^{\mathsf{RO}}(v_i)$

$(y_i, st) \leftarrow \mathsf{SimRO}(v_i, st)$
$Q_H = Q_H \cup \{(v_i, y_i)\}$
return y_i

$\mathcal{O}^{\mathsf{SimProve}}_{pp}(x_i, ctx_i)$

$(\pi_i, st) \leftarrow \mathsf{SimProve}(pp, x_i, ctx_i, st)$
$Q_P = Q_P \cup \{(x_i, \pi_i, ctx_i)\}$
return π_i

Experiment 2.15: Simulation-Extractability experiment for argument systems in the ROM.

[CCs08; MKWK19], and argument systems specialized on equations over bilinear groups [GS08]. Practically efficient argument systems based on Sigma protocols in the universal composability framework (UC) [Can01] were presented in [CKS11b], where the full version [CKS11a] shows the combination of the techniques mention above, i.e., set membership proofs, range proofs, equality proofs, Boolean composition, and more related techniques. Note, we do not use the CS notation if the statement to be proven demands a specialized argument system, e.g., for the ABS constructions from the literature presented in Section 3.5.

2.4 Universal Composability Framework

In the following we only want to introduce the concept of the universal composability framework (UC) by Canetti [Can01; Can20]. We present the formal definitions and details in Chapter 4.

The UC framework is interesting for cryptography, because the universal composability theorem of UC [Can01; Can20] allows us to build compositions of different schemes, individually shown to be secure in UC, without the need to redo all the proofs for the concrete composition. Additionally it gives us the guarantee that the individual schemes stay secure in any composition and any environment. In detail, considering a scheme (called protocols in UC) that is defined using other

subprotocols, e.g., a group signature that uses a signature scheme and proof system as building blocks. UC allows us to prove security of the protocol in a hybrid model, where any used UC secure subprotocol can be considered as ideal. Thereby, in the security proof we can assume that the signature scheme and proof system behave ideally, i.e., they are secure, and focus on the additional details of the group signature scheme. Thus it enables simpler security proofs for composed protocols (schemes) compared to the experiment-based definitions approach, provided the building blocks are already shown to be secure in UC.

In general the concept of UC security for a single protocol is to consider an ideal version of a scheme and give a real-world equivalent of it. The ideal version, formally called ideal protocol, includes an ideal functionality, that describes the ideal input, output behavior of the scheme and the information that an adversary gets. By this it defines correctness and security guarantees. For readers the ideal functionality can also provide a more intuitive understanding of what correctness and security ideally means for the scheme, compared to experiment-based definitions, where a reader typically has to parse the set of oracles available to the adversary to get an understanding of the definitions. Intuitively, modeling the ideal protocol of a cryptographic scheme is like modeling all involved parties via a trusted party, the ideal functionality that does everything for them, and precisely modeling which information is leaked to the adversary. Usually as a result, cryptography is no longer used in the ideal functionality. However, if we can replace the trusted party with real cryptography in a real protocol, executed by real parties with their own code, we have shown that the real protocol is as good as the ideal protocol. Intuitively this is what *UC security* captures; a formal definition is given in Section 4.2.

Overall, UC represents on the one side the ideal core of a scheme, which leads to results that look different from what cryptographers are accustomed to, e.g., UC-secure commitments [AKS19; CDR16; Lin11], and on the other side real protocols that capture the security defined by the ideal functionality. To get an idea of such an ideal functionality let us consider a sketch of ideal signatures, formally presented and proven to be equivalent to the standard EUF-CMA security of digital signature in [Can03]. Suppose a trusted party that you can ask to keep a signing record. The signing record stand for *that you authenticated a message of your choice*. The trusted party gives you a reference to this record as a response. Anyone else in possession of this reference can ask the trusted party if the referenced record is valid. In this simplified ideal signatures functionality there are no digital signatures in use. Just a trusted party that keeps record of who signed what. We present in Chapter 4 our ideal attribute-based signature functionality that follows the same idea as the ideal signatures.

Congruously, a formal definition of an ideal functionality in the universal composability framework [Can01; Can20] is technically challenging, since the formal details of UC include the consideration of network traffic, communications modes, state, and corruption of parties.

Part I

Attribute-Based Signatures

Enhanced Security of Attribute-Based Signatures

3

Summary. In this chapter we present and further extend the results of [BEJ18b] and its full version [BEJ18a]. The two main contributions of these works are that we enhance the security model of attribute-based signature (ABS) schemes in the form of a strengthened experiment-based definition and an ideal ABS functionality in the universal composability framework (UC). The ideal ABS functionality is presented in Chapter 4. In this chapter we focus on the experiment-based definition of ABS. Concretely, we strengthen the existing experiment-based security definitions for privacy in the form of simulation privacy (Sections 3.2.1 and 3.4.2). We then show that two major existing generic ABS constructions are indeed simulation private (Section 3.5). In addition to the results of [BEJ18a; BEJ18b] we strengthen the experiment-based soundness definition of ABS in the form of simulation-extractability in this thesis. Regarding this, we show a generic construction of ABS from signatures of knowledge. Our concrete instantiation based on succinct signatures of knowledge is, to the best of our knowledge, the first adaptively secure ABS scheme simultaneously achieving constant-size signatures (three group elements) and arithmetic circuits as policies.

In the following we give an introduction to ABS and present a detailed discussion of the related work in Section 3.1. From that we formulate our research questions and outline how we answer them in this chapter. The groundwork for this chapter are the experiment-based security definitions for ABS including simulation privacy, unforgeability, and simulation-extractability (Section 3.2). With respect to our security definitions we then show a secure ABS construction with constant-size signatures supporting expressive policies in Section 3.3. With the knowledge of how to build such an ABS scheme we discuss in Section 3.4 our experiment-based security definitions and show relations between them. In Section 3.5 we then show that two existing generic ABS constructions are simulation private and use this insight to outline how simulation privacy can be shown for other ABS constructions.

3.1 Introduction

Attribute-based signature (ABS) were introduced by Maji, Prabhakaran and Rosulek [MPR11]. Compared to digital signatures ABS extend the concept by considering multiple signers under one set of public parameters and an authority responsible for issuing secret keys to signers. Secret keys in ABS are issued on the *attributes* of signers, e.g., (`department`, `role`). A signature in ABS is then generated on a *message-policy* pair using a secret key with encoded attributes. Therefore, a signer does not authenticate a message in the sense of identification as it is the case with digital signatures. Instead, a valid attribute-based signature on a message-policy pair convinces a verifier that the message was signed by a signer with attributes that satisfy the policy. For example, consider a business environment where traditionally a specific employee has to sign a document, i.e., a sales confirmation. If digital signatures are used and the employee is on vacation (and only she has access to her secret key), no one else can sign the sales confirmation. Additionally, a digital signature valid under her public key identifies her and therefore leak internal processes, e.g., who is allowed to sign which type of documents. Using ABS, the signing policy for sales confirmations can be defined as: either the signer has to be employed at the sales department or is the chief executing officer. Hence, all employees of the sales department get a secret key where the `department` attribute is set to “sales” and the `role` attribute is set to “employee”. Further, the chief executing officer gets a secret key where the `department` attribute is set to “management” and the `role` attribute is set to “CEO” A document is then signed with a policy that checks if the `department` attribute is “sales” *and* the `role` attribute is “employee” *or* that the `role` attribute is “CEO” The advantage is that the ABS does not leak the identity of the signer

nor the attributes and secret key used to sign the document with respect to the policy. Considering authentication at service providers, the above example can be adapted to challenge-response protocol where a user has to sign a nonce (the challenge) and a policy get access to a specific service. Here, the policy can require the user has to to work at a specific department, e.g., research and development Application areas of ABS that are mentioned in the literature include leaking secrets [MPR11], attribute-based authentication [MPR11], an access control system [Li+10], attribute-based messaging [Bob+06; MPR11], trust-negotiation [MPR11], and a privacy-preserving certification mechanism [Kaa+17].

A secure ABS scheme guarantees unforgeability and privacy. Unforgeability captures that an adversary is not able to produce a valid signature on a message-policy pair without a secret key for a satisfying attribute set for the policy. Since there are multiple signers in an attribute-based signature scheme, unforgeability also captures that signers can not collude and combine secret keys to produce a signature if none of them has a single key that encodes satisfying attributes. Privacy of ABS guarantees that an adversary given a signature does neither learn the secret key nor the attributes used to generate a signature. Here, privacy has to be considered with respect to the policy, i.e., the adversary learns from a signature no more than what it can compute from the policy.

There are many ABS schemes and generic constructions presented in the literature [Che+13; DGM18; DOT19; EHM11; EK18; Gha15; GM19; Her14; Her16; MPR11; OT14; SAH16; SKAH18; Zha+19] with features such as revocation [GM22], traceability [EGK14; EHM11; Gha15; KCD09], multi-authorities [EGK14; Gha15; OT13], and hierarchical authorities [DGM18; GM19; GM22]. In multi-authority ABS schemes the responsibility for issuing secret keys on attributes is split across multiple authorities that can act independently from each other. In hierarchical ABS [DGM18; GM19; GM22] this is even further extended to featuring a root authority that delegates rights to issue secret keys to intermediate authorities which again can delegate some rights further and issue secret keys to users. During delegation the rights to issue secret keys can be restricted to a subset of attributes. In this thesis we focus on ABS with a single authority, since even in this case there are a variety of security definitions and ABS schemes in the literature with support for different policies and varying efficiency.

3.1.1 Related Work

Early works of ABS focussed mostly on the feasibility of schemes and started an interest to make the supported policy classes as expressive as possible while still ensuring security and efficiency. The concept of ABS started with a preliminary version [MPR08] of the later published work by Maji et al. [MPR11]. Maji et al. [MPR11] formally introduced ABS and the main contribution was a generic construction of ABS supporting threshold policies and monotone span programs (MSPs). Following the preliminary version [MPR08], there were works that presented ABS schemes with limited policies and privacy guarantees. The scheme by

Li and Kim [LK08] only supports conjunctions (n out of n, (n,n)-threshold) and therefore privacy of attributes is not considered. With normal privacy guarantees, Shahandashti and Safavi-Naini [SS09], Li and Au [Li+10], Chen et al. [Che+13], and El Bansarkhani and El Kaafarani [EE16] showed ABS schemes with (k,n)-threshold policies, where Khader et al. [KCD09] showed a scheme for threshold trees. From the groundwork of Maji et al. [MPR11] the expressiveness of supported policies paired with novel and interesting techniques to realize ABS were the main research focus. Over the years with many more schemes with the support for MSP as policies [DGM18; ECGD14; EG17; EGK14; EHM11; Gha15; GM19; Her14; Her16; HLLR12; Kaa+17; UKLC15] and featuring various approaches to the construction of ABS were presented.

Okamoto and Takashima were the first to widen the class of policies to non-monotone span programs [OT11; OT13; OT14] and Attrapadung et al. [AHY15] showed how to construct ABS from attribute-based encryption (ABE) techniques also supporting non-monotone span programs.

We summarize that the expressiveness of the policies and therefore the relation between attributes and policies is already sophisticated. The next major step was then the support of unbounded circuits (unbounded depth) and therefore a non-uniform computation model as policies. The two existing, and similar, constructions supporting this are by Sakai et al. [SAH16] and El Kaafarani and Katsumata [EK18] based on bilinear groups and lattices respectively.

ABS, rather key-policy ABS where the policy is encoded in the key, supporting an uniform computation model via Turing machines and nondeterministic finite automata as policies were presented by Datta et al. [DDM17] from indistinguishability obfuscation (IO) and more efficiently by Sakai et al. [SKAH18] with bilinear groups from standard assumptions. These constructions also lift previously existing bounds on the number of attributes.

An ABS construction that moves the policies to arithmetic computations was presented by Datta et al. [DOT19]. Their ABS supports arithmetic branching programs and therefore supports a wider class of policies than the most of the above schemes except the ones for Turing machines and circuits. In [DOT19] the authors show that the policies can be seen as polynomials over some finite field and mention that there are efficient transformations from a Boolean or arithmetic formula to arithmetic branching programs. Another in-between policy class supported by recent lattice-based ABS constructions [GM22; Zha+19] is realized via inner-product relations (conjunctive, disjunctive, threshold policies, and polynomial evaluation of attributes).

How to construct ABS. There are several approaches of constructing ABS in the literature. The most represented approach is to use a signature scheme and an argument system as building blocks. In ABS schemes constructed like this, the authority generates a signature on attributes to issue a secret key to a user. The user signs a message of its choice with a policy satisfied by the attributes encoded

in its secret key by generating a public verifiable proof via the argument system. The proof guarantees two parts. First, that the user has a secret key on satisfying attributes with respect to the policy. Second, the proof is bound to the signed message. Otherwise, an adversary could reuse the first part for arbitrary messages to construct forgeries. Note, we simplify the approach here and concrete constructions also rely on other building blocks to bound the message to the proof, e.g., commitment schemes, collision-resistant hash functions, and one-time signature schemes. Variants of this approach include the consideration of different argument systems, e.g., in the ROM and CRS model, different signature schemes, and replacing the argument system with a SoK. For example the ABS constructions presented in [AAS16; DGM18; EGK14; EK18; Her16; MPR11; SAH16; SKAH18] are based on the above described approach. Notable exceptions from that approach are ABS constructions from functional, identity-based, and attribute-based encryption schemes presented in [AHY15; DOT19; HLLR12; OT11] and the ABS construction based on indistinguishable obfuscation presented in [DDM17].

Extended features for ABS. Features to broaden the applicability of ABS such as the support for multiple secret key issuing authorities [MPR11] and traceability of users [EGK14; EHM11; Gha15; KCD09] were added over the years. The most advanced multi-authority ABS schemes are fully decentralized [EGK14; Gha15; OT13]. Another interesting feature is linkability [ECGD14; EG17; UKLC15] for establishing sessions between multiple authentications.

Security of ABS. For the purpose of this thesis we focus on signature-policy ABS, referred to as ABS, in the single authority setting without any extended features and summarize the state of the security model in the following. We have to note that the security model for ABS was not the focus of the overall ABS research. As we highlighted, the expressiveness of the policy class was the main driving factor. Maji et al. [MPR08; MPR10; MPR11] introduced the security model that was then adopted by subsequent works. The security model includes an experiment-based existential unforgeability (EUF) definition with a signing oracle, that is asked to find satisfying attributes for a given message-policy pair.

Okamoto and Takashima [OT14], were the first to observe that this definition is policy class specific and without restricting the policy class is problematic with respect to falsifiability [GW11; Nao03] (of the assumption) that a scheme under this EUF definition is secure. That is because the challenger modeling the experiment is not necessarily efficient depending on the policy class, e.g., for conjunctive normal form (CNF) formulas as policies, it is asked to solve the formulas by finding a satisfying input. Such a polynomial-time solver also means that we can decide boolean satisfiability problem (SAT) in polynomial-time. Hence, efficiently checking if an adversary breaks the scheme via the challenger is not possible in general. Furthermore, this can lead to non-efficient reductions (as part of a security proof) if one has to answer queries of an ABS adversary. Note, for ABS schemes that

just support monotone policies, e.g., threshold or MSP policies, the challenger is indeed efficient realizable. In this thesis, we show that the observation, that the original EUF definition depends on the policy class, also plays a major role in the definition of a stronger privacy notion. In the same paper as the observation [OT14] Okamoto and Takashima introduced a general EUF definition for ABS.

Maji et al. [MPR08; MPR10; MPR11] also introduced a privacy definition considering the distribution of signatures. In particular it requires that the distribution is independent of the secret key. This definition is not formulated via a security experiment and is widely adopted [AHY15; Che+13; DDM17; DOT19; OT13; OT14; SAH16; SKAH18]. An experiment-based version of the privacy definition was also introduced and used throughout the literature [EE16; EG17; EK18; Gha15; GM19; GM22; KCD09; Zha+19] where most of the schemes consider extended ABS, e.g., decentralized ABS.

Another approach to define privacy/anonymity is simulation and is widely used in privacy-preserving cryptography such as AC [BB18; BCKL08; CL01; CL04] and zero-knowledge argument systems [Fis05; FKMV12; FS87; GOS12; Gro06; GS08; GS12; IV19]. Simulatability for signatures was also formalized by Abe et al. [Abe+10; AHO10; AO09; AO12] in the context of blind signatures and structure-preserving signature (SPS) schemes and by Chase et al. [CKLM14] for malleable signatures. Intuitively, it requires the existence of a simulation signing algorithm that generates signatures with the help of a trapdoor instead of using a secret key as the normal signing algorithm.

Directly related to ABS is the work on policy-based signature (PBS) schemes by Bellare and Fuchsbauer [BF14]. In PBS the policy is associated to a signer's secret key and a signer can only sign a message if the message satisfies a policy and the signer knows a witness for the message-policy pair. Compared to ABS not only the attributes, here witness, but also the policy is hidden in a private PBS. PBS has interesting application areas where policies reveal secrets, e.g., organization structures and rules. Also, a verifier can be sure that a signer was permitted to sign the message, even if the verifier cannot inspect the policy. In [BF14] the authors also show how to build the generic ABS construction by Maji et al. [MPR11] from PBS, among other interesting implications such as group signatures, simulation-extractable NIZK, and public key encryption. A special case of PBS are functional signatures introduced in [BGI14]. In functional signatures signers can get secret keys for a function f and then sign any messages in the range of f. Mapped to PBS the witness is then the pre-image of f.

Regarding the equivalence of ABS to other primitives, there is the result by Tsabary [Tsa17] that states that ABS is equivalent to homomorphic signatures. The details of this equivalence are important to assess this result. Actually, Tsabary show the equivalence with a weak security model for ABS. In detail, it is only existential unforgeable with respect to a single key and the provided transformation to full existential unforgeability is only weak hiding. Here, weak hiding means that an adversary with a secret key can recognize signatures under its secret key. Such a definition is also called selfless anonymity, e.g., in group signatures. Following this

terminology, ABS are fully anonymous, i.e., an adversary cannot recognize its own signatures.

Efficiency of ABS. In addition to the support of a wider class of polices, it is important to study the effect of it on the efficiency of the ABS in the from of the size of the elements. Primarily the focus is on the signature size, since signatures are generated and transmitted often, whereas for keys this usually happens just once per signer.

With exceptions, that we list in the following, the signature size of almost all of the ABS schemes in the literature depend on the bound of the supported number of attributes. For example, the signature size of the ABS constructions in [MPR11; OT11; OT13; SAH16] is linear in the number of attributes and in case of [SKAH18] linear in the size of the policy.

Attribute-based signature schemes with constant-size signatures are presented in [EG17; GMZ12; HLLR12; WHHL15; ZXLZ12]. While featuring constant-size signatures, these ABS schemes only support threshold policies. Furthermore, the ABS schemes in [GMZ12; HLLR12; WHHL15; ZXLZ12] are only proven to be unforgeable under selective-policy attacks (non-adaptive unforgeability), which requires that the adversary submits a challenge policy at the beginning of the unforgeability experiment. An ABS scheme for threshold policies with constant-size signatures and claimed to be adaptively unforgeable was presented in [Che+13] using composite order groups instead of the common bilinear group setting. However, no formal security proofs for the ABS scheme was given. In [EG17] an ABS scheme supporting threshold policies and constant-size signatures is presented and shown to be adaptively unforgeable.

Concluding the related work, we can state that the research in the area of ABS has achieved the construction of ABS for very expressive policy classes (Turing machines, arithmetic branching programs, circuits) based on the security model introduced by Maji et al. [MPR08; MPR11]. Considering the signature size of the ABS constructions from the literature, we observe that constant-size signatures are only known for threshold policies.

3.1.2 Research Questions

As pointed out there are still open questions regarding the security of ABS in the experiment-based setting and in composed systems. Related to this is the question if we can strengthen the security notions of ABS. Furthermore, it is interesting to consider the efficiency (in terms of signature size) of schemes that support expressive policies. We identified the following questions.

Q1 Are there strong experiment-based security definitions that are not for specific policy classes and precisely capture both security aspects of ABS, namely privacy and unforgeability, and how are they related to existing security definitions?

Q2 Are there ABS constructions that are secure with respect to such experiment-based security definitions?

Q3 Is there a fully secure ABS construction supporting expressive policies with constant-size signatures?

3.1.3 Our Contribution

In this chapter we answer the above questions affirmatively in the following way.

Research question Q1. We answer the first question by combining the general ABS unforgeability definition of Okamoto and Takashima [OT14] and an adapted version of the PBS simulation-based privacy definition presented by Bellare and Fuchsbauer [BF14] into a ABS experiment-based security model (Section 3.2). As we noted in the related work, the idea of simulatability of signatures is not new and interestingly the early versions [MPR08; MPR10] of the Maji et al. construction [MPR11] presented a so called AltSign algorithm that is asked to output signatures without a secret key. In the concrete description of the AltSign algorithm the authors then require that it finds satisfying attributes for a given policy. As already mentioned, this is not efficiently feasible for all policy classes.

We concretize the simulatability of signatures in the form of simulation privacy independently from the policy class based on privacy definition presented in [BF14] for PBS. Therefore, we require that ppt simulation algorithms exists such that no adversary can distinguish them from the normal (non-simulation-based) algorithms of ABS. Concretely, we require that a ppt simulation algorithm for signing, called SimSign, with corresponding setup (SimSetup) and key generation algorithm (SimKeyGen) exists. The novelty of our experiment-based security model for ABS comes from the combination of the general (policy-independent) unforgeability definition [OT14] and the simulation privacy. In addition to that, we also introduce a simulation-extractability definition. Enabling security proofs of ABS schemes that use a building block that is simulation-extractable (e.g., SoK) to be more direct instead of incorporating a reduction to unforgeability.

We also show the relation of the security definition to one another in Section 3.4. There, we show that the simulation privacy definition is stronger than the standard privacy definition with respect to hard policy classes, where satisfying attributes for a policy are unique but not efficiently computable. Additionally, we show that the standard privacy definition for ABS does not guarantee the desired privacy. In detail, we present that for hard policy classes the standard privacy definition allows ABS schemes that append the attributes in the clear to generated signatures. Whereas our simulation privacy definition does not allow that. As a consequence of this result our simulation privacy definition should be considered. Regarding unforgeability, we show in Section 3.4.1 that simulation-extractability implies unforgeability for ABS.

Research question Q2. We show that the generic ABS constructions by Sakai et al. [SAH16] and by Maji et al. [MPR11] satisfy our simulation privacy definition in Section 3.5. Since originally for both schemes weaker privacy guarantees were shown, we think that simulation privacy is interesting on its own. Note, that for unforgeability there is nothing to show since we only changed the unforgeability definition such that the efficiency of the challenger does not depend on the policy class. Furthermore, the reductions in the respective unforgeability proofs given in [MPR11; SAH16] were already efficient. We also highlight lessons learned from that adaption of the existing ABS schemes and list additional schemes that we checked to be simulation private.

Research question Q3. We answer the question by showing a generic ABS construction from signatures of knowledge (SoKs) and digital signatures (Section 3.3), and by presenting a concrete instantiation with constant-size ABS signatures from a succinct SoK (Section 3.3.3). Our concrete instantiation supports arithmetic circuits as policies with signatures consisting of only three group elements. As a consequence of the generic construction (from simulation-extractable SoK), we introduce simulation-extractability to the experiment-based security model of ABS and show that it implies unforgeability (Section 3.4.1).

This chapter presents results published in [BEJ18b]. The extended proofs and descriptions presented in this thesis are based on the full version [BEJ18a] of the paper published at CANS'18. In detail, the simulation privacy definition (Section 3.2.1), that simulation privacy implies privacy (Section 3.4.2), and the result that two generic ABS construction from the literature are simulation private (Section 3.5) were published in [BEJ18b]. In this chapter we additionally present a generic ABS construction (Section 3.3), an ABS scheme with constant-size signatures (Section 3.3.3), show that simulation-extractibility implies unforgeability for ABS, and further analyze the relation of simulation privacy and privacy (Section 3.4).

3.2 Attribute-Based Signatures

To study and enhance the security definitions of attribute-based signature (ABS) we start with the formal definition of the syntax and then present the security definitions.

In an ABS scheme users get secret keys that are issued by an authority on their attributes and signatures are generated on message-policy pairs. We denote attributes $\mathbb{A}$ as a vector from an attribute universe $\mathcal{U}_{pp}$, where $\mathcal{U}_{pp}$ is implicitly defined by the public parameters pp of an ABS scheme. The attribute universe $\mathcal{U}_{pp}$ in turn implicitly defines a vector length l and a policy $\mathbb{P}$ is a map $\mathbb{P} : \mathcal{U}_{pp} \to \{0, 1\}$. By $\mathbb{A} \in \mathcal{U}_{pp}$ we denote that the attributes $\mathbb{A}$ are an element of $\mathcal{U}_{pp}$. Hence a vector of length l. Note, in the literature attributes are sometimes modeled as sets. Since our results do not depend on the concrete modeling of the attributes we use the vector modeling in this thesis which allows us to treat attributes as message vectors

for other schemes.

An ABS scheme supports a policy class $\{\mathbb{P}\}_{pp}$ that denotes the set of all supported policies that map from the attribute universe to $\{0,1\}$, i.e., policies $\mathbb{P}$ such that $\mathbb{P} : \mathcal{U}_{pp} \to \{0,1\}$. The supported policies depend on the concrete ABS scheme, e.g., boolean formulas, CNF formulas, boolean circuits or arithmetic circuits. We say that policy $\mathbb{P}$ is defined over the attributes of $\mathcal{U}_{pp}$ and denote it by $\mathbb{P} \in \mathcal{U}_{pp}$ if $\mathbb{P} \in \{\mathbb{P}\}_{pp}$. Further, we say that attributes $\mathbb{A}$ satisfy policy $\mathbb{P}$ if $\mathbb{P}(\mathbb{A}) = 1$.

Definition 3.2.1 (Attribute-based Signature Scheme)
An attribute-based signature scheme ABS consists of algorithms Setup, KeyGen, Sign, and Verify.

- $(pp, msk) \leftarrow \mathsf{Setup}(1^\lambda)$ is a ppt setup algorithm that on input security parameter 1^λ outputs public parameters pp and master secret key msk, where the public parameters pp implicitly define the message space $\mathcal{M}$ and attribute universe $\mathcal{U}_{pp}$.

- $sk_{\mathbb{A}} \leftarrow \mathsf{KeyGen}(pp, msk, \mathbb{A})$ is a ppt key generation algorithm that on input public parameters pp, a master secret key msk, and attributes $\mathbb{A} \in \mathcal{U}_{pp}$ outputs a secret key $sk_{\mathbb{A}}$.

- $\sigma \leftarrow \mathsf{Sign}(pp, sk_{\mathbb{A}}, m, \mathbb{P})$ is a ppt signing algorithm that on input public parameters pp, a message $m \in \mathcal{M}$, a policy $\mathbb{P}$ over the attributes of $\mathcal{U}_{pp}$, and a secret key $sk_{\mathbb{A}}$, outputs a signature σ or a failure symbol $\perp$.

- $0/1 \leftarrow \mathsf{Verify}(pp, m, \mathbb{P}, \sigma)$ is a ppt or dpt verification algorithm that on input public parameters pp, a message $m \in \mathcal{M}$, a policy $\mathbb{P}$, and a signature σ outputs $b \in \{0,1\}$.

$\diamond$

We require from an ABS scheme that is correct and consistent. Intuitively, correctness guarantees under honestly generated setup parameters that signatures on message-policy pairs computed with honestly generated secret keys are valid.

Definition 3.2.2 (Correctness)
An ABS scheme ABS with ppt algorithm Verify is *correct*, if for all $\lambda \in \mathbb{N}$, all $(pp, msk) \in \mathsf{Setup}(1^\lambda)$, all messages $m \in \mathcal{M}$, all attributes $\mathbb{A} \in \mathcal{U}_{pp}$, all $sk_{\mathbb{A}} \in \mathsf{KeyGen}(pp, msk, \mathbb{A})$, all policies $\mathbb{P} \in \mathcal{U}_{pp}$, such that $\mathbb{P}(\mathbb{A}) = 1$, and all signatures $\sigma \in \mathsf{Sign}(pp, sk_{\mathbb{A}}, m, \mathbb{P})$ it holds that

$$\Pr\left[\mathsf{Verify}(pp, m, \mathbb{P}, \sigma) = 0\right] \leq \mathsf{negl}(\lambda) \ .$$

$\diamond$

If Verify is a deterministic polynomial-time algorithm then we require that it outputs 1 with certainty.

Note, that our definition of ABS considers a probabilistic Verify algorithm. Therefore, consistency guarantees that a signature that was once declared by ppt algorithm Verify as valid, will be declared as invalid, by an independent run of Verify, only with negligible probability (and vice versa).

Definition 3.2.3 (Consistency)
An ABS scheme ABS with a probabilistic verification algorithm Verify is *consistent*, if for all $\lambda \in \mathbb{N}$, all $(pp, msk) \in \mathsf{Setup}(1^\lambda)$, all m, all $\mathbb{P}$, and all σ it holds that there exists $b \in \{0,1\}$ such that

$$\Pr[\mathsf{Verify}(pp, m, \mathbb{P}, \sigma) \neq b] \leq \mathsf{negl}(\lambda) \ .$$

$\diamond$

Note, we call a scheme perfectly correct and perfectly consistent if $\mathsf{negl}(\lambda)$ is 0 respectively.

Remark 3.2.1: We consider ABS in this work to be correct and consistent without mentioning it explicitly.

The following requirement allows a signer to verify secret keys. We use this for ABS in the universal composability framework (Chapter 4) where users can get secret keys generated by an adversary.

Remark 3.2.2: We further require that secret keys are efficiently verifiable. In detail we require that there is a ppt algorithm that for all $\lambda \in \mathbb{N}$, on input public parameters $pp \in \mathsf{Setup}(1^\lambda)$ and secret key $sk_{\mathbb{A}} \in \mathsf{KeyGen}(pp, msk, \mathbb{A})$ outputs 1 with certainty.

Note that this is a natural requirement, since a signer has to be able to verify if a secret key issued by an authority is valid with respect to the user's attributes. Almost all schemes that we checked fulfill this requirement since they are constructed such that secret keys are signatures on the attributes, e.g., [DGM18; EHM11; MPR11; OT13; SAH16].

3.2.1 Privacy

We present two privacy definitions. The first one captures that an adversary determining two attributes for two secret keys should not be able to tell which secret key was used to sign a given message-policy pair. The second definition is simulation-based and requires, that even the attributes used to generate a signature are hidden with respect to what is efficiently computable from the policy. Formally, both def-

initions are a specialization of the definitions for policy-based signatures (PBS) presented in [BF14] to ABS. We present the two formal privacy definitions in the following. Details of the differences between the privacy definitions are examined in Section 3.4.2. With regard to our simulation-based definition, we show in Section 3.5 simulation algorithms for two generic ABS constructions [MPR11; SAH16] and show that they satisfy our simulation-based privacy definition.

Standard Privacy

Privacy definitions that do not specify an experiment are common in ABS [Her16; MPR11; OT14; SAH16]. In more detail, they say that an ABS scheme is perfectly private, if for all honest setups, all honestly generated secret keys $sk_{\mathbb{A}_0}$ and $sk_{\mathbb{A}_1}$, and all message m, and all policies $\mathbb{P}$, such that $\mathbb{P}(\mathbb{A}_0) = \mathbb{P}(\mathbb{A}_1)$, the distributions of the signatures under $sk_{\mathbb{A}_0}$ and $sk_{\mathbb{A}_1}$ are identical [MPR11].

We capture the above in form of an equivalent security experiment, where the capabilities of the adversary are explicitly stated, e.g., oracle access, input and output behavior. Informally, our experiment-based privacy requires that it should be infeasible for an adversary to distinguish which secret key was used to generate a signature, even if it has access to the master secret key and the secret keys.

We refer to this privacy notion as standard privacy, if we compare it to other privacy notions. Our definition is adapted from [BF14].

Definition 3.2.4 (Privacy)
Let ABS be an attribute-based signature scheme and let $\mathcal{D}$ be a distinguisher. We define the advantage in the experiment $\mathrm{Exp}^{\mathsf{privacy},b}_{\mathcal{D},\mathsf{ABS}}(\lambda)$ (Experiment 3.1) as

$$\mathrm{Adv}^{\mathsf{privacy}}_{\mathcal{D},\mathsf{ABS}}(\lambda) \coloneqq \Big| \Pr[\mathrm{Exp}^{\mathsf{privacy},0}_{\mathcal{D},\mathsf{ABS}}(\lambda) = 1] - \Pr[\mathrm{Exp}^{\mathsf{privacy},1}_{\mathcal{D},\mathsf{ABS}}(\lambda) = 1] \Big|$$

and call ABS *perfectly private*, if for every (unbounded) distinguisher $\mathcal{D}$ it holds that

$$\mathrm{Adv}^{\mathsf{privacy}}_{\mathcal{D},\mathsf{ABS}}(\lambda) = 0$$

or *computationally private*, if for every ppt distinguishers $\mathcal{D}$ it holds that

$$\mathrm{Adv}^{\mathsf{privacy}}_{\mathcal{D},\mathsf{ABS}}(\lambda) = \mathsf{negl}(\lambda) \ .$$

◇

Note, that in the above privacy experiment the oracle outputs the secret keys together with the signature similar to full anonymity definitions for group signatures [BMW03].

Definition 3.2.4 states that the relation between the signature and the secret keys is hidden. In particular, an adversary cannot determine which secret key was used to issue a signature, for this the oracle outputs the generated secret keys. To

$\underline{\mathrm{Exp}^{\mathsf{privacy},b}_{\mathcal{D},\mathsf{ABS}}(\lambda)}$

1 : $(pp, msk) \leftarrow \mathsf{Setup}(1^\lambda)$
2 : $b' \leftarrow \mathcal{D}^{\mathcal{O}^b_{pp,msk}}(pp, msk)$
3 : **return** b'

$\underline{\mathcal{O}^b_{pp,msk}(m, \mathbb{P}, \mathbb{A}_0, \mathbb{A}_1)}$

1 : **if** $\mathbb{P}(\mathbb{A}_0) = 1, \mathbb{P}(\mathbb{A}_1) = 1,$ **then**
2 : $\quad sk_{\mathbb{A}_0} \leftarrow \mathsf{KeyGen}(pp, msk, \mathbb{A}_0),$
3 : $\quad sk_{\mathbb{A}_1} \leftarrow \mathsf{KeyGen}(pp, msk, \mathbb{A}_1),$
4 : $\quad \sigma \leftarrow \mathsf{Sign}(pp, sk_{\mathbb{A}_b}, m, \mathbb{P}),$
5 : $\quad$ **return** $(\sigma, sk_{\mathbb{A}_0}, sk_{\mathbb{A}_1})$
6 : **else return** $\perp$

Experiment 3.1: Privacy experiment for ABS.

further motivate the above privacy definition, let us momentarily consider a weaker definition. For this, we change the above definition such that it does not output the secret keys on oracle queries. Then let us assume a scheme, that generates secret keys that include unique identifiers. Furthermore, during signing the scheme appends this identifier to the signature. Notice, that in the described scheme all signatures under the same secret key are linkable via the unique identifier appended to each signature. The described scheme satisfies the weaker definition, since an adversary just see a signature output by the oracle and each secret key is just used once by the oracle. Hence, the included identifier does not help the adversary to link signatures. This weaker definition does not capture the real requirements, where we want signatures to be unlinkable. Therefore, the oracle in our privacy definition outputs both secret keys $sk_{\mathbb{A}_0}$ and $sk_{\mathbb{A}_1}$ in addition to the signature. By this, any scheme that appends identifiers to signatures is not considered private in our security model. In group signature terminology such a weaker definition as we described is called selfless-anonymity [BS04].

Simulation Privacy

Another way to describe privacy, is that given a valid signature σ on a policy $\mathbb{P}$, an adversary should not be able to learn which attributes are necessary to satisfy the policy $\mathbb{P}$ from the signature σ, except for what it can compute from the policy $\mathbb{P}$. Simulation privacy captures this privacy description. To argue why simulation privacy is desirable in practice consider the following example. Which is presented in [BF14] to motivate simulation privacy for policy-based signatures in a similar way. Consider an ABS scheme that is perfectly private according to Definition 3.2.4. Additionally, assume for every $\mathbb{P} \in \mathcal{U}_{pp}$ there is just one satisfying attribute $\mathbb{A}$. Therefore, the adversary (distinguisher) $\mathcal{D}$ in $\mathrm{Exp}^{\mathsf{privacy},b}_{\mathcal{D},\mathsf{ABS}}(\lambda)$ has to input $\mathbb{A}_0 = \mathbb{A}_1$ for $\mathbb{P}$ to its challenge oracle $\mathcal{O}^b_{pp,msk}$. Let us modify algorithm Sign such that it appends the attribute vector to each signature. As a result, the returned signatures are (of course) still indistinguishable as required in standard privacy Definition 3.2.4 (since $\mathbb{A}_0 = \mathbb{A}_1$), but the used attributes are not hidden.

This is not the desired privacy guarantee in a real-world application, where the attributes are secret or satisfying attributes of a policy are hard to compute, e.g., policies modeling NP-hard problems. To achieve the privacy level that is demanded in such applications we introduce simulation privacy for ABS. In Section 3.4.2 we show that simulation privacy is actually a stronger notion than standard privacy.

In the simulation-based definition of privacy we additionally require that the signatures are independent of the attributes. Therefore, simulation privacy is based on a simulation signing algorithm. The normal signing algorithm Sign in ABS gets a secret key for an attribute vector as an input, whereas the simulation signing algorithm $\mathsf{SimSign}$ does not. Intuitively, if signatures can be simulated without a given secret key for satisfying attributes (regarding the given policy), then signatures do not leak any information about the attributes used to generate it, except what can be computed from the policy. More formally, no adversary should be able to distinguish whether a signature was generated by the normal signing algorithm Sign or by the simulation signing algorithm $\mathsf{SimSign}$.

The following simulation-based definition is based on the definition presented in [BF14] for policy-based signatures.

Definition 3.2.5 (Simulation Privacy)
Let ABS be an attribute-based signature scheme and let $\mathcal{D}$ be a distinguisher. We define the advantage in the experiment $\mathrm{Exp}^{\mathsf{sim\text{-}privacy},b}_{\mathcal{D},\mathsf{ABS}}(\lambda)$ for $b \in \{0,1\}$ (Experiment 3.2) as

$$\mathrm{Adv}^{\mathsf{sim\text{-}privacy}}_{\mathcal{D},\mathsf{ABS}}(\lambda) \coloneqq \left|\Pr\left[\mathrm{Exp}^{\mathsf{sim\text{-}privacy},0}_{\mathcal{D},\mathsf{ABS}}(\lambda)=1\right] - \Pr\left[\mathrm{Exp}^{\mathsf{sim\text{-}privacy},1}_{\mathcal{D},\mathsf{ABS}}(\lambda)=1\right]\right| \ .$$

and call ABS *perfectly simulation private*, if there exists a tuple of ppt algorithms called $(\mathsf{SimSetup}, \mathsf{SimKeyGen}, \mathsf{SimSign})$ such that for every (unbounded) distinguisher $\mathcal{D}$ it holds that

$$\mathrm{Adv}^{\mathsf{sim\text{-}privacy}}_{\mathcal{D},\mathsf{ABS}}(\lambda) = 0$$

or *computational simulation private*, if there exists a tuple of ppt algorithms called $(\mathsf{SimSetup}, \mathsf{SimKeyGen}, \mathsf{SimSign})$ such that for every ppt distinguisher $\mathcal{D}$ it holds that

$$\mathrm{Adv}^{\mathsf{sim\text{-}privacy}}_{\mathcal{D},\mathsf{ABS}}(\lambda) = \mathsf{negl}(\lambda) \ .$$

Here $\mathsf{SimKeyGen}$ and $\mathsf{SimSign}$ can be stateful.

◇

We want to note that the oracle $\mathcal{O}^{\mathsf{Sign}_0}_{pp,msk,td_{sim}}$ in Definition 3.2.5 only outputs a simulated signature, if the attributes satisfy the policy ($\mathbb{P}(\mathbb{A}) = 1$), otherwise it outputs the failure symbol $\perp$. This is required, since $\mathsf{SimSign}$ unlike Sign can (potentially) simulate signatures even for false statements ($\mathbb{P}(\mathbb{A}) = 0$). Implicitly,

$\underline{\mathrm{Exp}^{\mathsf{sim\text{-}privacy},1}_{\mathcal{D},\mathsf{ABS}}(\lambda)}$

1 : $Q_{\mathsf{KeyGen}} \coloneqq \emptyset$
2 : $(pp, msk) \leftarrow \mathsf{Setup}(1^\lambda)$
3 : $b' \leftarrow \mathcal{D}^{\mathcal{O}^{\mathsf{KeyGen}_1}_{pp,msk}, \mathcal{O}^{\mathsf{Sign}_1}_{pp}}(pp, msk)$
4 : **return** b'

$\underline{\mathrm{Exp}^{\mathsf{sim\text{-}privacy},0}_{\mathcal{D},\mathsf{ABS}}(\lambda)}$

1 : $Q_{\mathsf{KeyGen}} \coloneqq \emptyset$
2 : $(pp, msk, td_{sim}) \leftarrow \mathsf{SimSetup}(1^\lambda)$
3 : $b' \leftarrow \mathcal{D}^{\mathcal{O}^{\mathsf{KeyGen}_0}_{pp,msk,td_{sim}}, \mathcal{O}^{\mathsf{Sign}_0}_{pp,msk,td_{sim}}}(pp, msk)$
4 : **return** b'

$\underline{\mathcal{O}^{\mathsf{KeyGen}_1}_{pp,msk}(\mathbb{A}_i) :}$

1 : $sk_{\mathbb{A}_i} \leftarrow \mathsf{KeyGen}(pp, msk, \mathbb{A}_i)$
2 : $Q_{\mathsf{KeyGen}} \coloneqq Q_{\mathsf{KeyGen}} \cup \{(i, sk_{\mathbb{A}_i}, \mathbb{A}_i)\}$
3 : **return** $sk_{\mathbb{A}_i}$

$\underline{\mathcal{O}^{\mathsf{KeyGen}_0}_{pp,msk,td_{sim}}(\mathbb{A}_i) :}$

1 : $sk_{\mathbb{A}_i} \leftarrow \mathsf{SimKeyGen}(pp, msk, td_{sim}, \mathbb{A}_i)$
2 : $Q_{\mathsf{KeyGen}} \coloneqq Q_{\mathsf{KeyGen}} \cup \{(i, sk_{\mathbb{A}_i}, \mathbb{A}_i)\}$
3 : **return** $sk_{\mathbb{A}_i}$

$\underline{\mathcal{O}^{\mathsf{Sign}_1}_{pp}(i, m_j, \mathbb{P}_j) :}$

1 : **if** $(i, sk_{\mathbb{A}_i}, \mathbb{A}_i) \notin Q_{\mathsf{KeyGen}}$ **then**
2 : **ignore**
3 : $\sigma \leftarrow \mathsf{Sign}(pp, sk_{\mathbb{A}_i}, m_j, \mathbb{P}_j)$
4 : **return** σ

$\underline{\mathcal{O}^{\mathsf{Sign}_0}_{pp,msk,td_{sim}}(i, m_j, \mathbb{P}_j) :}$

1 : **if** $(i, sk_{\mathbb{A}_i}, \mathbb{A}_i) \notin Q_{\mathsf{KeyGen}}$ **then**
2 : **ignore**
3 : **if** $\mathbb{P}_j(\mathbb{A}_i) = 0$ **then**
4 : **return** $\perp$
5 : **else**
6 : $\sigma \leftarrow \mathsf{SimSign}(pp, msk, td_{sim}, m_j, \mathbb{P}_j)$
7 : **return** σ

Experiment 3.2: Simulation Privacy experiment for ABS.

this demands that also $\mathsf{Sign}(pp, sk_{\mathbb{A}}, m, \mathbb{P})$ outputs the failure symbol $\perp$ if $\mathbb{P}(\mathbb{A}) = 0$. Intuitively, Sign should not output a (valid) signature in such a case anyway, hence it can just output $\perp$. In the above Definition 3.2.5, we introduce a simulation trapdoor td_{sim} to capture that SimKeyGen and SimSign use some additional information to simulate. For many of the ABS schemes in the literature the simulation trapdoor is the master secret key msk, e.g., [AHY15; ECGD14; EG17; EGK14; MPR11; SAH16; SKAH18; UKLC15]. In Section 3.5 we go into more details and show for two generic ABS constructions [MPR11; SAH16] that they are simulation private.

Defining simulation with a simulation trapdoor, instead of requiring that it is the master secret key, is more general and covers ABS schemes that simulate its signatures without msk, e.g., using a trapdoor for a zero-knowledge or signature of knowledge simulator as we present in Section 3.3.

3.2.2 Unforgeability

In contrast to digital signatures (Section 2.3.2) the unforgeability of ABS has to consider secret keys on attributes that were revealed to the adversary and the signatures that can be produced with them in respect to the policy that the adversary outputs as part of its forgery. Hence, the unforgeability notion captures that it should be infeasible for an adversary to output a valid signature on a message-policy pair where it never got a secret key that satisfies the policy nor a signature for that message-policy pair. We base the following unforgeability definition on the definition presented in [OT14].

Definition 3.2.6 (Unforgeability - [OT14])
Let ABS be an attribute-based signature scheme and $\mathcal{A}$ an adversary. We define the advantage in the experiment $\mathrm{Exp}^{\mathsf{euf}}_{\mathcal{A},\mathsf{ABS}}(\lambda)$ (Experiment 3.3) as

$$\mathrm{Adv}^{\mathsf{euf}}_{\mathcal{A},\mathsf{ABS}}(\lambda) \coloneqq \Pr\Big[\mathrm{Exp}^{\mathsf{euf}}_{\mathcal{A},\mathsf{ABS}}(\lambda) = 1\Big]$$

and call ABS existentially *unforgeable* regarding an adaptive chosen message-policy attack, if for all ppt adversaries $\mathcal{A}$ it holds that

$$\mathrm{Adv}^{\mathsf{euf}}_{\mathcal{A},\mathsf{ABS}}(\lambda) = \mathsf{negl}(\lambda) \ .$$

$\diamond$

Intuitively, an adversary $\mathcal{A}$ is successful in the above unforgeability experiment, if the following statements are true.

1. The signature σ^* output by $\mathcal{A}$ is valid for the message m^* and policy $\mathbb{P}^*$, i.e $\mathsf{Verify}(pp, m^*, \mathbb{P}^*, \sigma^*) = 1$,
2. the signing oracle never returned a signature for message-policy pair $(m^*, \mathbb{P}^*)$, i.e., $(m^*, \mathbb{P}^*) \notin Q_{\mathsf{Sign}}$,
3. and for all attributes $\mathbb{A}_i$, where the corresponding secret key $sk_{\mathbb{A}_i}$ was revealed by the oracle $\mathcal{O}^{\mathsf{Reveal}}$, it holds that the attributes $\mathbb{A}_i$ do not satisfy the policy $\mathbb{P}^*$, i.e., $\mathbb{P}^*(\mathbb{A}_i) = 0$.

Note, the above definition also guarantees collusion resistance in the following sense. The adversary $\mathcal{A}$ can get secret keys on attributes of its choice by first querying oracle $\mathcal{O}^{\mathsf{KeyGen}}_{pp,msk}$ and then oracle $\mathcal{O}^{\mathsf{Reveal}}$. Even then the adversary cannot output a valid signature for policy $\mathbb{P}^*$ if none of the revealed secret keys alone (representing a group of colluding signers) is sufficient to satisfy the policy $\mathbb{P}^*$.

Note, that the above definition is *weak* in a similar sense as our EUF-CMA definition for digital signatures (Definition 2.3.6). This means, that a successful adversary has to output a signature on a fresh message-policy pair and cannot reuse answers from the Sign oracle. This is usually viewed as a feature in privacy-preserving cryptography, since it allows that signatures are randomizable. Strongly

$\mathrm{Exp}^{\mathsf{euf}}_{\mathcal{A},\mathsf{ABS}}(\lambda)$

1 : $Q_{\mathsf{KeyGen}}, Q_{\mathsf{Sign}}, Q_{\mathsf{Reveal}} \coloneqq \emptyset$
2 : $(pp, msk) \leftarrow \mathsf{Setup}(1^\lambda)$
3 : $(m^*, \mathbb{P}^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{KeyGen}}_{pp,msk}, \mathcal{O}^{\mathsf{Reveal}}, \mathcal{O}^{\mathsf{Sign}}_{pp}}(pp)$
4 : **return** 1 **if**
5 : $\quad \mathsf{Verify}(pp, m^*, \mathbb{P}^*, \sigma^*) = 1 \wedge$
6 : $\quad (m^*, \mathbb{P}^*) \notin Q_{\mathsf{Sign}} \wedge$
7 : $\quad \forall \mathbb{A}_i \in Q_{\mathsf{Reveal}} : \mathbb{P}^*(\mathbb{A}_i) = 0$
8 : **else return** 0

$\mathcal{O}^{\mathsf{Sign}}_{pp}(i, m_j, \mathbb{P}_j)$:

1 : **if** $(i, sk_{\mathbb{A}_i}, \mathbb{A}_i) \notin Q_{\mathsf{KeyGen}}$ **then**
2 : $\quad$ **ignore**
3 : $\sigma \leftarrow \mathsf{Sign}(pp, sk_{\mathbb{A}_i}, m_j, \mathbb{P}_j)$
4 : $Q_{\mathsf{Sign}} \coloneqq Q_{\mathsf{Sign}} \cup \{(m_j, \mathbb{P}_j)\}$
5 : **return** σ

$\mathcal{O}^{\mathsf{Reveal}}(i)$:

1 : **if** $(i, sk_{\mathbb{A}_i}, \mathbb{A}_i) \notin Q_{\mathsf{KeyGen}}$ **then**
2 : $\quad$ **return** $\perp$
3 : **else**
4 : $\quad Q_{\mathsf{Reveal}} \coloneqq Q_{\mathsf{Reveal}} \cup \{\mathbb{A}_i\}$
5 : $\quad$ **return** $sk_{\mathbb{A}_i}$

$\mathcal{O}^{\mathsf{KeyGen}}_{pp,msk}(\mathbb{A}_i)$:

1 : $sk_{\mathbb{A}_i} \leftarrow \mathsf{KeyGen}(pp, msk, \mathbb{A}_i)$
2 : $Q_{\mathsf{KeyGen}} \coloneqq Q_{\mathsf{KeyGen}} \cup \{(i, sk_{\mathbb{A}_i}, \mathbb{A}_i)\}$

Experiment 3.3: Unforgeability experiment for ABS.

unforgeable signatures are not randomizable. For strong unforgeability in ABS we just need to change the second winning condition to $(m^*, \mathbb{P}^*, \sigma^*) \notin Q_{\mathsf{Sign}}$.

3.2.3 Simulation-Extractability

Next, we introduce simulation-extractability for ABS and in Section 3.4.1 we show that it implies unforgeability. The definition enables more direct constructions and proofs of ABS schemes that use a simulation-extractable building block. For example in Section 3.3 we show a generic ABS construction that uses a simulation-extractable SoK as a building block. In the following we introduce two variants of simulation-extractability. One that we call white-box, where the extractor depends in the adversary, and another that we call black-box, where the extractor is independent of the adversary. The naming is based on the naming that we use for

$\underline{\mathrm{Exp}^{\mathsf{wb\text{-}sim\text{-}ext}}_{\mathcal{A},\mathsf{Extr}_{\mathcal{A}},\mathsf{ABS}}(\lambda)}$

1 : $Q_{\mathsf{KeyGen}}, Q_{\mathsf{Sign}} \coloneqq \emptyset$
2 : $(pp, msk, td_{sim}) \leftarrow \mathsf{SimSetup}(1^\lambda)$
3 : $(m^*, \mathbb{P}^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{SimKeyGen}}_{pp,msk,td_{sim}}, \mathcal{O}^{\mathsf{SimSign}}_{pp,msk,td_{sim}}}(pp)$
4 : $(\mathbb{A}^*) \leftarrow \mathsf{Extr}_{\mathcal{A}}(trans_{\mathcal{A}})$
5 : **return** 1 **if**
6 : $\quad \mathsf{Verify}(pp, m^*, \mathbb{P}^*, \sigma^*) = 1 \wedge$
7 : $\quad (m^*, \mathbb{P}^*) \notin Q_{\mathsf{Sign}} \wedge$
8 : $\quad (\mathbb{A}^* \notin Q_{\mathsf{KeyGen}} \vee \mathbb{P}^*(\mathbb{A}^*) = 0)$
9 : **else return** 0

$\underline{\mathcal{O}^{\mathsf{SimKeyGen}}_{pp,msk,td_{sim}}(\mathbb{A}_i) :}$

1 : $sk_{\mathbb{A}_i} \leftarrow \mathsf{SimKeyGen}(pp, msk, td_{sim}, \mathbb{A}_i)$
2 : $Q_{\mathsf{KeyGen}} \coloneqq Q_{\mathsf{KeyGen}} \cup \{(i, sk_{\mathbb{A}_i}, \mathbb{A}_i)\}$
3 : **return** $sk_{\mathbb{A}_i}$

$\underline{\mathcal{O}^{\mathsf{SimSign}}_{pp,msk,td_{sim}}(m_j, \mathbb{P}_j) :}$

1 : $\sigma_j \leftarrow \mathsf{SimSign}(pp, msk, td_{sim}, m_j, \mathbb{P}_j)$
2 : $Q_{\mathsf{Sign}} \coloneqq Q_{\mathsf{Sign}} \cup \{(m_j, \mathbb{P}_j)\}$
3 : **return** σ_j

Experiment 3.4: White-box simulation-extractability experiment for ABS.

the security definitions of argument systems (Section 2.3.5).

Definition 3.2.7 (White-Box Simulation-Extractability)
Let ABS be a simulation private attribute-based signature scheme and $\mathcal{A}$ be an adversary $\mathcal{A}$. We define the advantage in the experiment $\mathrm{Exp}^{\mathsf{wb\text{-}sim\text{-}ext}}_{\mathcal{A},\mathsf{Extr}_{\mathcal{A}},\mathsf{ABS}}(\lambda)$ (Experiment 3.4) as

$$\mathrm{Adv}^{\mathsf{wb\text{-}sim\text{-}ext}}_{\mathcal{A},\mathsf{Extr}_{\mathcal{A}},\mathsf{ABS}}(\lambda) \coloneqq \Pr\left[\mathrm{Exp}^{\mathsf{wb\text{-}sim\text{-}ext}}_{\mathcal{A},\mathsf{Extr}_{\mathcal{A}},\mathsf{ABS}}(\lambda) = 1\right]$$

and call ABS white-box *simulation-extractable*, if for all ppt adversaries $\mathcal{A}$ there exists a ppt extractor $\mathsf{Extr}_{\mathcal{A}}$ such that $\mathrm{Adv}^{\mathsf{wb\text{-}sim\text{-}ext}}_{\mathsf{ABS},\mathcal{A},\mathsf{Extr}_{\mathcal{A}}}(\lambda) = \mathsf{negl}(\lambda)$.

$\diamond$

In the following we define the simulation-extractability with a black-box extractor. A black-box extractor is also called online extractor in the literature, since it does not rely on any rewinding techniques, rather it gets an extraction trapdoor for the CRS to extract.

Definition 3.2.8 (Black-Box Simulation-Extractability)
Let ABS be a simulation private attribute-based signature scheme and $\mathcal{A}$ be a

$\underline{\mathrm{Exp}^{\mathsf{bb\text{-}sim\text{-}ext}}_{\mathcal{A},\mathsf{Extr},\mathsf{ABS}}(\lambda)}$

$Q_{\mathsf{KeyGen}}, Q_{\mathsf{Sign}} \coloneqq \emptyset$
$(pp, msk, td_{sim}, td_{extr}) \leftarrow \mathsf{ExtrSetup}(1^\lambda)$
$(m^*, \mathbb{P}^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{SimKeyGen}}_{pp,msk,td_{sim}}, \mathcal{O}^{\mathsf{SimSign}}_{pp,msk,td_{sim}}}(pp)$
$(\mathbb{A}^*) \leftarrow \mathsf{Extr}(msk, td_{extr}, m^*, \mathbb{P}^*, \sigma^*, Q_{\mathsf{KeyGen}}, Q_{\mathsf{Sign}})$
return 1 **if**
  $\mathsf{Verify}(pp, m^*, \mathbb{P}^*, \sigma^*) = 1 \wedge$
  $(m^*, \mathbb{P}^*) \notin Q_{\mathsf{Sign}} \wedge$
  $(\mathbb{A}^* \notin Q_{\mathsf{KeyGen}} \vee \mathbb{P}^*(\mathbb{A}^*) = 0)$
else return 0

$\underline{\mathcal{O}^{\mathsf{SimKeyGen}}_{pp,msk,td_{sim}}(\mathbb{A}_i) :}$

$sk_{\mathbb{A}_i} \leftarrow \mathsf{SimKeyGen}(pp, msk, td_{sim}, \mathbb{A}_i)$
$Q_{\mathsf{KeyGen}} \coloneqq Q_{\mathsf{KeyGen}} \cup \{(i, sk_{\mathbb{A}_i}, \mathbb{A}_i)\}$
return $sk_{\mathbb{A}_i}$

$\underline{\mathcal{O}^{\mathsf{SimSign}}_{pp,msk,td_{sim}}(m_j, \mathbb{P}_j) :}$

$\sigma_j \leftarrow \mathsf{SimSign}(pp, msk, td_{sim}, m_j, \mathbb{P}_j)$
$Q_{\mathsf{Sign}} \coloneqq Q_{\mathsf{Sign}} \cup \{(m_j, \mathbb{P}_j)\}$
return σ_j

Experiment 3.5: Black-box simulation-extractability for ABS.

ppt adversary. We define the advantage in experiment $\mathrm{Exp}^{\mathsf{bb\text{-}sim\text{-}ext}}_{\mathcal{A},\mathsf{Extr},\mathsf{ABS}}(\lambda)$ (Experiment 3.5) as

$$\mathrm{Adv}^{\mathsf{bb\text{-}sim\text{-}ext}}_{\mathcal{A},\mathsf{Extr},\mathsf{ABS}}(\lambda) \coloneqq \Pr[\mathrm{Exp}^{\mathsf{bb\text{-}sim\text{-}ext}}_{\mathcal{A},\mathsf{Extr},\mathsf{ABS}}(\lambda) = 1]$$

and call ABS black-box *simulation-extractable*, if (1) there exists a ppt algorithm ExtrSetup such that for all ppt adversaries $\mathcal{A}$ it holds that

$$\Big|\Pr\Big[(pp, msk, td_{sim}) \leftarrow \mathsf{SimSetup}(1^\lambda) \colon \mathcal{A}(pp) = 1\Big] - \Pr\Big[(pp, msk, td_{sim}, td_{extr}) \leftarrow \mathsf{ExtrSetup}(1^\lambda) \colon \mathcal{A}(pp) = 1\Big]\Big|$$

is negligible and (2) there exists a ppt extractor Extr such that for all ppt adversaries $\mathcal{A}$ it holds $\mathrm{Adv}^{\mathsf{bb\text{-}sim\text{-}ext}}_{\mathsf{ABS},\mathcal{A},\mathsf{Extr}}(\lambda) = \mathsf{negl}(\lambda)$. ◇

3.3 Attribute-Based Signature Scheme from Signatures of Knowledge

In this section we present our generic ABS construction from signature of knowledge (SoK) and a digital signature scheme. We then show an instantiation that

supports arithmetic circuits with constant-size signatures consisting only of 3 group elements. We use here our enhanced experiment-security model, since we prove the construction secure in the sense of white-box simulation-extractability. This is a conscious decision, since we use a succinct SoK for the instantiation, where only white-box secure instantiations are known. Nonetheless, our ABS construction is not inherently bound to white-box security definitions. Therefore, we also discuss the security of the construction under black-box simulation-extractability.

The approach of a SoK-based construction is a natural candidate to construct ABS and other cryptographic constructions. For example, constructions that use SoK are presented in [BBS04; BCKL08; BEJ18c; BF14; BMW03; BSZ05; CDHK15; DP06; DS18] including but not limited to non-interactive anonymous credentials, reputation system, as well as static and dynamic group signatures. The first formal and universally composable security definition for SoKs was presented by Chase and Lysyanskaya in [CL06]. Initially, interactive zero-knowledge arguments of knowledge protocols were transformed to non-interactive variants by the so called Fiat-Shamir transformation [FS87]. The same technique yields a signature of knowledge. The efficiency of most of the constructions that rely on signatures of knowledge [BCKL08; BEJ18c; BF14; CDHK15; DS18] inherits the efficiency of the used SoK scheme. Recently, Groth and Maller [GM17] presented a succinct signature of knowledge based on SE-SNARK shown in the same work. The proofs generated by the SE-SNARK and therefore the resulting signatures of knowledge consist only of three group elements [GM17]. The SoK in [GM17] can be instantiated for relations given as an arithmetic circuit. With regard to ABS, an ABS scheme build from SoKs based on lattice assumptions was presented in [EE16]. However, the scheme only supports policies consisting of $(\wedge, \vee)$ combinations of attributes. Compared to this scheme we add in the following the formal definition and usage of an ABS relation and a generic construction based on SoKs and digital signatures. We also present a concrete instantiation with a wider class of policies, namely arithmetic circuits.

Before we present our generic ABS construction, we define what an attribute-based signature certifies as a formal relation, where the attributes are part of the witness and the policy is part of the instance. This formalization allows us to rely on a SoK for the ABS relation to formally define our generic ABS construction. Based on this we show our instantiation and analyze its efficiency.

Defining a relation for signatures with extended authentication semantics is also used in [BF14]. There, PBS are constructed by first defining an appropriate relation. The difference to ABS is that the policy is also part of the witness in the PBS relation. As described in Section 3.1.1 this means that the policy is hidden from verifiers in PBS where in ABS it is public. Regarding ABS, the ABS constructions presented in [DGM18; EK18; GM19] are also build from a relation. There, the relations are very specific to the building blocks that are used, i.e., how the policy is modeled.

Let us start with the definition of the relation for ABS.

Definition 3.3.1 (ABS Relation)
Let $\mathsf{Sig} = (\mathsf{Sig.Setup}, \mathsf{Sig.KeyGen}, \mathsf{Sig.Sign}, \mathsf{Sig.Verify})$ be a signature scheme for message space $\mathcal{U}$. Let $\mathcal{U}$ also be the attribute universe. We define relation $R_{\mathsf{Sig}}^{\mathrm{ABS}}$ as follows:

$$((x, w)) = ((pk, \mathbb{P}), (\mathbb{A}, \theta)) \in R_{\mathsf{Sig}}^{\mathrm{ABS}} \iff \mathsf{Sig.Verify}(pk, \theta, \mathbb{A}) = 1 \wedge \mathbb{P}(\mathbb{A}) = 1$$

where $(pk, \mathbb{P})$ is the instance and $(\mathbb{A}, \theta)$ is the witness. Let $R_{\mathsf{Sig}}^{\mathrm{ABS}}$ be implicitly defined by the message space $\mathcal{U}$. Further, let $\mathsf{RGen}_{\mathsf{Sig}}^{\mathrm{ABS}}$ be the relation generator for $R_{\mathsf{Sig}}^{\mathrm{ABS}}$ that on input 1^λ generates public parameters pp via $\mathsf{Sig.Setup}$. The public parameters also define the message space (and the attribute universe) $\mathcal{U}$. ◇

To present some intuition for the above definition and what it means to have a signature of knowledge for the ABS relation generator $\mathsf{RGen}_{\mathsf{Sig}}^{\mathrm{ABS}}$, consider $\mathsf{RGen}_{\mathsf{Sig}}^{\mathrm{ABS}}$ as the setup algorithm that creates the public parameters, e.g., a group description. Then, the role of the signature scheme Sig is to create ABS secret keys as signatures on attributes. Further, considering signature of knowledge scheme SoK for the ABS relation $R_{\mathsf{Sig}}^{\mathrm{ABS}}$, the scheme SoK signs message-policy pairs $(m, \mathbb{P})$ given satisfying attributes $\mathbb{A}$ (i.e., $\mathbb{P}(\mathbb{A}) = 1$) and a valid Sig signature on the attributes $\mathbb{A}$. Here, the validity of the signature on the attributes is determined by $\mathsf{Sig.Verify}$. Specifically, the verification equation, e.g., a collection of product pairing equations if Sig is based on bilinear groups.

3.3.1 Generic ABS Construction

In the following, we present a generic ABS construction from a signature of knowledge and digital signature scheme. The construction is similar to the second generic construction of policy-based signatures (PBS) in [BF14]. However, the PBS construction directly uses non-interactive zero-knowledge arguments of knowledge instead of a signature of knowledge for a specific PBS relation. This means their NIZK has to hide the policy and prove that the message and a provided witness satisfy the policy. In ABS we do not have to hide the policy and the policy does not involve the message to be signed.

Our approach for the generic ABS construction is that the ABS setup fixes public parameters for the signature scheme. A signer gets an ABS secret key consisting of a signature on its attributes $\mathbb{A}$. Then, to sign a message m with $\mathbb{P}$ the signer generates a signature of knowledge on m proving that its secret key satisfies $\mathbb{P}$ and that the secret key is a valid signature on attributes $\mathbb{A}$.

Construction 3.3.1 (Generic ABS Construction)
Let $\mathsf{Sig} = (\mathsf{Sig.KeyGen}, \mathsf{Sig.Sign}, \mathsf{Sig.Verify})$ be signature scheme and let $\mathsf{SoK} = (\mathsf{SoK.Setup}, \mathsf{SoK.SimSetup}, \mathsf{SoK.Sign}, \mathsf{SoK.SimSign}, \mathsf{SoK.Verify})$ be a signature of knowledge for ABS relation generator $\mathsf{RGen}_{\mathsf{Sig}}^{\mathrm{ABS}}$. We define the generic attribute-based signature construction $\mathsf{ABS}^{\mathsf{gc}}$ in Figure 3.6.

We assume that the initial public parameters pp' are part of the public key pk and the public parameters pp_{SoK}.

Lemma 3.3.1. *If the signature scheme* Sig *is perfectly correct and the signature of knowledge scheme* SoK *is perfectly correct, then the ABS construction* $\mathsf{ABS}^{\mathsf{gc}}$ *(Construction 3.3.1) is also perfectly correct.*

Proof. From inspection it follows that the message space of $\mathsf{ABS}^{\mathsf{gc}}$ is the message space $\mathcal{M}_{\mathsf{SoK}}$ of the signature of knowledge scheme SoK. Further, the attribute universe $\mathcal{U}_{pp}$ is the message space $\mathcal{M}_{\mathsf{Sig}}$ of the signature scheme Sig. Let $\lambda \in \mathbb{N}$, $(pp, msk) \leftarrow \mathsf{ABS}^{\mathsf{gc}}.\mathsf{Setup}(1^\lambda)$, message $m \in \mathcal{M}_{\mathsf{SoK}}$, attributes $\mathbb{A} \in \mathcal{M}_{\mathsf{Sig}}$, $sk_{\mathbb{A}} \leftarrow \mathsf{ABS}^{\mathsf{gc}}.\mathsf{KeyGen}(pp, msk, \mathbb{A})$, policy $\mathbb{P} \in \mathcal{U}_{pp}$, such that $\mathbb{P}(\mathbb{A}) = 1$, and let signature $\sigma \leftarrow \mathsf{ABS}^{\mathsf{gc}}.\mathsf{Sign}(pp, sk_{\mathbb{A}}, m, \mathbb{P})$. Then by definition of the algorithms, we have that the public parameters pp consists of the public parameters of SoK and the public key of generated by Sig.KeyGen and the master secret key is the secret key of the signature scheme Sig. In detail, $(pp, msk) = ((pp_{\mathsf{SoK}}, pk), sk)$ where $(pk, sk) \leftarrow \mathsf{Sig}.\mathsf{KeyGen}(pp')$ and $(pp_{\mathsf{SoK}}, td_{sim}) \leftarrow \mathsf{SoK}.\mathsf{SimSetup}(pp')$ for $pp' \leftarrow \mathsf{RGen}_{\mathsf{Sig}}^{\mathrm{ABS}}(1^\lambda)$. Further, we have that a secret key of $\mathsf{ABS}^{\mathsf{gc}}$ consists of a Sig signature and a message from its message space $\mathcal{M}_{\mathsf{Sig}}$, i.e., for $sk_{\mathbb{A}} \leftarrow \mathsf{ABS}^{\mathsf{gc}}.\mathsf{KeyGen}(pp, msk, \mathbb{A})$ we have that $sk_{\mathbb{A}} = (\mathbb{A} \in \mathcal{M}_{\mathsf{Sig}}, \theta \leftarrow \mathsf{Sig}.\mathsf{Sign}(pk, sk, \mathbb{A}))$. By definition of the signing algorithm of $\mathsf{ABS}^{\mathsf{gc}}$ we know that a signature $\sigma \leftarrow \mathsf{ABS}^{\mathsf{gc}}.\mathsf{Sign}(pp, sk_{\mathbb{A}}, m, \mathbb{P})$ is just a signature generated by the signature of knowledge scheme SoK, i.e., $\sigma \leftarrow \mathsf{SoK}.\mathsf{Sign}(pp_{\mathsf{SoK}}, (pk, \mathbb{P}), (\mathbb{A}, \theta), m)$. Putting it together, we have that $\mathsf{ABS}^{\mathsf{gc}}.\mathsf{Verify}(pp, m, \mathbb{P}, \sigma)$ outputs 1, if and only if

$$\mathsf{SoK}.\mathsf{Verify}(pp_{\mathsf{SoK}}, (pk, \mathbb{P}), m, \sigma)$$

outputs 1. From the perfect correctness of the signature scheme Sig it follows that $\mathsf{Sig}.\mathsf{Verify}(pk, \theta \leftarrow \mathsf{Sig}.\mathsf{Sign}(pk, sk, \mathbb{A}), \mathbb{A}) = 1$. We also know that the attributes $\mathbb{A}$ satisfy the policy $\mathbb{P}$, i.e., $\mathbb{P}(\mathbb{A}) = 1$. Then, by the definition of the ABS relation $R_{\mathsf{Sig}}^{\mathrm{ABS}}$ (Definition 3.3.1) it follows that $((pk, \mathbb{P}), (\mathbb{A}, \theta)) \in R_{\mathsf{Sig}}^{\mathrm{ABS}}$. Finally, by the perfect correctness of SoK we know that, if $((pk, \mathbb{P}), (\mathbb{A}, \theta)) \in R_{\mathsf{Sig}}^{\mathrm{ABS}}$ holds we have that $\mathsf{SoK}.\mathsf{Verify}(pp_{\mathsf{SoK}}, (pk, \mathbb{P}), m, \sigma)$ outputs 1. □

Perfect correctness of Construction 3.3.1, follows directly from the perfect correctness of the signature scheme Sig and the signature of knowledge SoK. Since SoK.Verify is deterministic and hence also $\mathsf{ABS}^{\mathsf{gc}}.\mathsf{Verify}$ is deterministic, we have that $\mathsf{ABS}^{\mathsf{gc}}$ is perfectly consistent (Definition 3.2.3).

Privacy. We start the formal proofs by showing the perfect simulation privacy for $\mathsf{ABS}^{\mathsf{gc}}$ Construction 3.3.1. Subsequently, we focus on the unforgeability of Construction 3.3.1. In the privacy proof we build an adversary against the perfect simulatability of the signature of knowledge scheme SoK from a distinguisher against the simulation privacy of $\mathsf{ABS}^{\mathsf{gc}}$.

$\mathsf{ABS}^{\mathsf{gc}}.\mathsf{Setup}(1^\lambda)$

$pp' \leftarrow \mathsf{RGen}_{\mathsf{Sig}}^{\mathrm{ABS}}(1^\lambda)$
$(pk, sk) \leftarrow \mathsf{Sig.KeyGen}(pp')$
$(pp_{\mathsf{SoK}}, td_{sim}) \leftarrow \mathsf{SoK.SimSetup}(pp')$
return $(pp \coloneqq (pp_{\mathsf{SoK}}, pk), msk \coloneqq sk)$

$\mathsf{ABS}^{\mathsf{gc}}.\mathsf{KeyGen}(pp, msk, \mathbb{A})$

parse $pp = (pp_{\mathsf{SoK}}, pk), msk = sk$
if $\mathbb{A} \notin \mathcal{U}_{pp}$ **then**
  return $\bot$
else
  $\theta \leftarrow \mathsf{Sig.Sign}(pk, sk, \mathbb{A})$
  return $sk_{\mathbb{A}} = (\mathbb{A}, \theta)$

$\mathsf{ABS}^{\mathsf{gc}}.\mathsf{Sign}(pp, sk_{\mathbb{A}}, m, \mathbb{P})$

parse $pp = (pp_{\mathsf{SoK}}, pk), sk_{\mathbb{A}} = (\mathbb{A}, \theta)$
if $\mathbb{P} \notin \mathcal{U}_{pp} \vee \mathbb{P}(\mathbb{A}) = 0$ **then**
  return $\bot$
else
  set $w \coloneqq (\mathbb{A}, \theta)$
  set $x \coloneqq (pk, \mathbb{P})$
  $\sigma \leftarrow \mathsf{SoK.Sign}(pp_{\mathsf{SoK}}, x, w, m)$
  return σ

$\mathsf{ABS}^{\mathsf{gc}}.\mathsf{Verify}(pp, m, \mathbb{P}, \sigma)$

parse $pp = (pp_{\mathsf{SoK}}, pk)$
set $x \coloneqq (pk, \mathbb{P})$
set $b \coloneqq \mathsf{SoK.Verify}(pp_{\mathsf{SoK}}, x, m, \sigma)$
return b

$\mathsf{ABS}^{\mathsf{gc}}.\mathsf{SimSetup}(1^\lambda)$

same as $\mathsf{ABS}^{\mathsf{gc}}.\mathsf{Setup}(1^\lambda)$ but with
additional output of the trapdoor td_{sim}
return $(pp \coloneqq (pp_{\mathsf{SoK}}, pk), msk \coloneqq sk, td_{sim})$

$\mathsf{ABS}^{\mathsf{gc}}.\mathsf{SimKeyGen}(pp, msk, td_{sim}, \mathbb{A})$

return $\mathsf{ABS}^{\mathsf{gc}}.\mathsf{KeyGen}(pp, msk, \mathbb{A})$

$\mathsf{ABS}^{\mathsf{gc}}.\mathsf{SimSign}(pp, msk, td_{sim}, m, \mathbb{P})$

parse $pp = (pp_{\mathsf{SoK}}, pk), msk = sk$
set $x \coloneqq (pk, \mathbb{P})$
$\sigma \leftarrow \mathsf{SoK.SimSign}(pp_{\mathsf{SoK}}, td_{sim}, x, m)$
return σ

Figure 3.6: Generic ABS construction $\mathsf{ABS}^{\mathsf{gc}}$.

Theorem 3.3.1. *If* SoK *is perfectly simulatable (Definition 2.3.13), then* $\mathsf{ABS}^{\mathsf{gc}}$ *(Construction 3.3.1) is perfectly simulation private (Definition 3.2.5) with respect to* $\mathsf{RGen}_{\mathsf{Sig}}^{\mathrm{ABS}}$.

Proof. Let $\mathcal{D}$ be a distinguisher for simulation privacy of $\mathsf{ABS}^{\mathsf{gc}}$ with advantage $\mathrm{Adv}^{\mathsf{sim\text{-}privacy}}_{\mathcal{D},\mathsf{ABS}^{\mathsf{gc}}}(\lambda) = \Pr[\mathrm{Exp}^{\mathsf{sim\text{-}privacy},1}_{\mathcal{D},\mathsf{ABS}}(\lambda) = 1] - \Pr[\mathrm{Exp}^{\mathsf{sim\text{-}privacy},0}_{\mathcal{D},\mathsf{ABS}}(\lambda) = 1]$. We use $\mathcal{D}$ to define an adversary $\mathcal{A}$ against the simulatability of SoK ($\mathrm{Exp}^{\mathsf{sim}}_{\mathcal{A},\mathsf{SoK}}(\lambda)$).

We start with the definition of $\mathcal{A}$ and then we analyze its advantage.

1. $\mathrm{Exp}^{\mathsf{sim}}_{\mathcal{A},\mathsf{SoK}}(\lambda)$ provides $\mathcal{A}$ the public parameters pp_b and access to a signing oracle $\mathcal{O}^{\mathsf{Sign}_b}_{pp_b,td_{sim}}(\cdot,\cdot,\cdot)$

2. $\mathcal{A}$ generates a signature key pair $(pk, sk) \leftarrow \mathsf{Sig.KeyGen}(pp_b)$, sets master secret key $msk \coloneqq sk$ and public parameters $pp \coloneqq (pp_b, pk)$.

3. $\mathcal{A}$ runs $\mathcal{D}$ on input (pp, msk) and answers $\mathcal{D}$'s oracle queries as follows.

   $\mathcal{O}^{\mathsf{KeyGen}_b}(\mathbb{A}_i)$: On i-th query, given attribute vector $\mathbb{A}_i$, $\mathcal{A}$ generates secret key $(\mathbb{A}_i, \theta_i) \leftarrow \mathsf{ABS}^{\mathsf{gc}}.\mathsf{KeyGen}(pp, msk, \mathbb{A}_i)$, adds $(i, sk_{\mathbb{A}_i}, \mathbb{A}_i)$ to Q_{KeyGen}, and outputs $sk_{\mathbb{A}_i} \coloneqq (\mathbb{A}_i, \theta_i)$.

   $\mathcal{O}^{\mathsf{Sign}_b}_{pp,msk}(i, m_j, \mathbb{P}_j)$: On j-th query, given $(i, m_j, \mathbb{P}_j)$ for an already recorded i, if $\mathbb{P}_j(\mathbb{A}_i) = 0$ return $\perp$, else $\mathcal{A}$ first sets $w_j \coloneqq (\mathbb{A}_i, \theta_i)$ and $x_j \coloneqq (pk, \mathbb{P}_j)$. Then $\mathcal{A}$ outputs $\sigma_j \leftarrow \mathcal{O}^{\mathsf{Sign}_b}_{pp_b,td_{sim}}(x_j, w_j, m_j)$.

4. Eventually $\mathcal{D}$ outputs bit b'. $\mathcal{A}$ outputs also b'.

First, let us analyze how the oracle queries of $\mathcal{D}$ are answered by $\mathcal{A}$. If the bit b chosen in the simulatability experiment $\mathrm{Exp}^{\mathsf{sim}}_{\mathcal{A},\mathsf{SoK}}(\lambda)$ of SoK, is equal to 1 the public parameters pp_1 are generated by the real setup $\mathsf{SoK.Setup}$ in the experiment. This means that the signing oracle $\mathcal{O}^{\mathsf{Sign}_1}$ outputs real signatures generated by the signing algorithm $\mathsf{SoK.Sign}$. If the bit b is 0, the public parameters pp_0 are generated by the simulation setup $\mathsf{SoK.SimSetup}$. Thus, the signing oracle $\mathcal{O}^{\mathsf{Sign}_0}$ outputs simulated signatures generated by the simulation signing algorithm $\mathsf{SoK.SimSign}$. Since the simulation trapdoor td_{sim} is not used in either cases of b and except for td_{sim} output/input algorithms $\mathsf{ABS}^{\mathsf{gc}}.\mathsf{SimSetup}$ and $\mathsf{ABS}^{\mathsf{gc}}.\mathsf{Setup}$, and $\mathsf{ABS}^{\mathsf{gc}}.\mathsf{SimKeyGen}$ and $\mathsf{ABS}^{\mathsf{gc}}.\mathsf{KeyGen}$ are the same, $\mathcal{A}$ can generate the setup parameters in the same way for both cases. Therefore, $\mathcal{A}$ knows msk and can just execute $\mathsf{ABS}^{\mathsf{gc}}.\mathsf{KeyGen}$ to answer the key generation oracle queries of $\mathcal{D}$ in both cases of b. This means in case $b = 1$, adversary $\mathcal{A}$ simulates the experiment $\mathrm{Exp}^{\mathsf{sim\text{-}privacy},1}_{\mathcal{D},\mathsf{ABS}}(\lambda)$ for distinguisher $\mathcal{D}$, with real setup and real signatures, perfectly. In case $b = 0$, adversary $\mathcal{A}$ simulates the experiment $\mathrm{Exp}^{\mathsf{sim\text{-}privacy},0}_{\mathcal{D},\mathsf{ABS}}(\lambda)$, with simulated setup and simulated signatures, perfectly.

Next, we analyze the success probability of $\mathcal{A}$, where b is the internal bit of

$\mathrm{Exp}^{\mathsf{sim}}_{\mathcal{A},\mathsf{SoK}}(\lambda)$.

$$\begin{aligned}
\Pr\Big[\mathrm{Exp}^{\mathsf{sim}}_{\mathcal{A},\mathsf{SoK}}(\lambda)=1\Big] &= \Pr[b'=b\wedge b=1]+\Pr[b'=b\wedge b=0]\\
&= \frac{1}{2}\cdot\left(\Pr[b'=1\mid b=1]+\Pr[b'=0\mid b=0]\right)\\
&= \frac{1}{2}\cdot\left(\Pr[b'=1\mid b=1]+(1-\Pr[b'=1\mid b=0])\right)\\
&= \frac{1}{2}+\frac{1}{2}\cdot\left(\Pr[b'=1\mid b=1]-\Pr[b'=1\mid b=0]\right)\\
&= \frac{1}{2}+\frac{1}{2}\cdot\Big(\Pr\Big[\mathrm{Exp}^{\mathsf{sim\text{-}privacy},1}_{\mathcal{D},\mathsf{ABS}}(\lambda)=1\Big]\\
&\qquad\qquad -\Pr\Big[\mathrm{Exp}^{\mathsf{sim\text{-}privacy},0}_{\mathcal{D},\mathsf{ABS}}(\lambda)=1\Big]\Big)\\
&= \frac{1}{2}+\frac{1}{2}\cdot\mathrm{Adv}^{\mathsf{sim\text{-}privacy}}_{\mathcal{D},\mathsf{ABS}^{\mathsf{gc}}}(\lambda)
\end{aligned}$$

Consequently, $\mathrm{Adv}^{\mathsf{simul}}_{\mathsf{SoK},\mathcal{A}}(\lambda)=2\cdot\Pr\Big[\mathrm{Exp}^{\mathsf{sim}}_{\mathcal{A},\mathsf{SoK}}(\lambda)\Big]-1=\mathrm{Adv}^{\mathsf{sim\text{-}privacy}}_{\mathcal{D},\mathsf{ABS}^{\mathsf{gc}}}(\lambda)$ □

Unforgeability. In the following we focus on the unforgeability of $\mathsf{ABS}^{\mathsf{gc}}$ (Construction 3.3.1). We show that the unforgeability of $\mathsf{ABS}^{\mathsf{gc}}$ follows from the EUF-CMA security of the digital signature scheme Sig and from the simulation-extractability of the signature of knowledge scheme SoK. The latter gives us the choice of white-box simulation-extractability (for each adversary a specific extractor) or the black-box variant (one extractor for all adversaries). Our instantiation, that we present in Section 3.3.3, is based on a succinct SoK. Since secure succinct SoKs (and succinct non-interactive argument of knowledge (SNARK)) are only known with respect to the white-box variant [BCCT12; GW11], we opt to use white-box simulation-extractability for ABS as well. To look ahead, the advantage of this approach is that our instantiation is a succinct ABS, i.e., signatures consists only of 3 group elements.

Before starting with the actual unforgeability proof of $\mathsf{ABS}^{\mathsf{gc}}$ we have to introduce some preliminaries. Currently the only known secure succinct constructions of SoK, that we later use, is based on a simulation-extractable SNARK [GM17]. Even proving unforgeability of constructions that combine digital signatures with SNARKs, that are (only) white-box knowledge sound (extraction without simulation), turns out to be a delicate task, as pointed out by Fiore and Nitulescu [FN16a; FN16b]. To understand the problem behind this, let us give an informal theorem and outline the proof structure. In the proof outline we highlight the problem and describe the solutions that Fiore and Nitulescu [FN16a; FN16b] gave in related context of SNARK-based constructions.

Informal theorem: If Sig *is EUF-CMA (Definition 2.3.6), and* SoK *is white-box strong simulation-extractable (Definition 2.3.14), then* $\mathsf{ABS}^{\mathsf{gc}}$ *is white-box simulation-extractable (Definition 3.2.7).*

In the proof, the first major step is to prove the existence of a white-box extractor

for $\mathsf{ABS}^{\mathsf{gc}}$ given the white-box extractor of the signature of knowledge scheme SoK.

The next major step, that needs more explanation, is then to show a forger for Sig that uses a white-box sim-ext adversary $\mathcal{A}'$ and extractor. The existence of this extractor is challenging, since the adversary $\mathcal{A}'$ in this situation has oracle access to a signing oracle from the unforgeability experiment of the digital signature scheme. This signing oracle holds the secret key of the digital signature scheme Sig. Therefore, adversary $\mathcal{A}'$ is formally not a sim-ext adversary against the SoK. Note, the sim-ext experiment (Definition 2.3.14) only provides specific oracle access to an oracle that outputs simulated SoK signatures. Hence, without further assumptions we cannot assume that an extractor exists for the white-box sim-ext adversary $\mathcal{A}'$.

As Fiore and Nitulescu [FN16b] pointed out, the described problem is common, if one uses white-box extractors, e.g., [BBFR15; BF11; BGI14]. Fiore and Nitulescu [FN16b] presented a range of solutions for the problem in the context of SNARKs. As mentioned above SNARKs are knowledge sound and not necessarily simulation-extractable.

First they formalize SNARK in the presence of oracles (O-SNARK), where the adversary in the knowledge soundness experiment has access to oracles. For a detailed discussion of this matter we refer to the full version [FN16a] of the paper by Fiore and Nitulescu. We do not have to introduce O-SNARKs formally in this thesis, since fortunately in [FN16a; FN16b] the authors show that if one uses a hash-then-sign EUF-CMA secure signature scheme, then classical SNARKs are O-SNARKs in the random oracle model. Note, that the hash-then-sign technique relies on a hash function modeled in the random oracle model (ROM). This comes with some shortcomings according to Fiore and Nitulescu [FN16a; FN16b]. In detail, the verification algorithm of the hash-then-sign signature scheme uses a random oracle. This means the SNARK used to prove validity of the signatures, has to be able to prove pre-images of the random oracle. However, in real-world applications we instantiate the random oracle with a hash function such as SHA-256 or SHA-3. Hence, in real-world applications we have to consider a SNARK that proves pre-images of a hash function instead of the random oracle. As done in [FN16a; FN16b] we have to assume that this is still secure. For further discussions we refer to [FN16a; FN16b].

Since the O-SNARK results of Fiore and Nitulescu [FN16a; FN16b] only consider knowledge soundness (SNARKs) and not simulation-extractability (SE-SNARKs) we have to assume that the results also expand to simulation-extractability. In our case we assume that SoK is white-box strong simulation-extractable with respect to the signing oracle specified by Sig. Note, the O-SNARK results should also hold for SE-SNARKs since they are as SNARKs based on knowledge assumptions. However, to be clear, no concrete results for SE-SNARKs are known to us. As a consequence, we additionally present a theorem with black-box simulation-extractability and therefore fewer assumptions after the following Theorem 3.3.2 with white-box simulation-extractability and the corresponding proof.

Theorem 3.3.2. *If the signature scheme* Sig *is EUF-CMA (Definition 2.3.6) and*

uses the hash-then-sign technique, and the SoK scheme SoK *is white-box strong simulation-extractable (Definition 2.3.14) with respect to* Sig*, then* $\mathsf{ABS}^{\mathsf{gc}}$ *(Construction 3.3.1) is white-box simulation-extractable (Definition 3.2.7) with respect to* $\mathsf{RGen}^{\mathrm{ABS}}_{\mathsf{Sig}}$ *and in the random oracle model.*

The following proof is in structure and techniques based on the proof of white-box simulation-extractable SoK from white-box simulation-extractable non-interactive zero-knowledge arguments (NIZK) and collision-resistant hash functions presented in [GM17, Proposition 3.1]. Note, in [GM17] the authors do not have to consider O-SNARKs, since their SoK construction does not use building blocks with additional problematic oracles. We adapt the proof from [GM17] to our setting. In our setting, we start with a white-box simulation-extractable SoK, instead of a NIZK argument, and an unforgeable signature scheme, instead of a collision-resistant hash function. The similarity is that we take something (SoK) that is white-box simulation-extractable to build something different (ABS) that is also white-box simulation-extractable.

Let us start with the formal proof of Theorem 3.3.2.

Proof. As a general outline of the proof, we first have to give simulation algorithms for $\mathsf{ABS}^{\mathsf{gc}}$ before we deal with the extraction part. We can skip this, since we already showed that $\mathsf{ABS}^{\mathsf{gc}}$ is simulation private and gave simulation algorithms for $\mathsf{ABS}^{\mathsf{gc}}$ in the above proof of Theorem 3.3.1. Therefore, we assume that $\mathsf{ABS}^{\mathsf{gc}}$ is perfectly simulation private and focus on the extraction part in this proof.

We bound the advantage $\mathrm{Adv}^{\mathsf{wb\text{-}sim\text{-}ext}}_{\mathcal{A},\cdot,\mathsf{ABS}^{\mathsf{gc}}}(\lambda)$ of an adversary $\mathcal{A}$ against $\mathsf{ABS}^{\mathsf{gc}}$ in the white-box simulation-extractability experiment. We proceed in three steps, starting by showing the existence of an ABS extractor. For this, on a high level, we construct an adversary $\mathcal{B}$ against the white-box strong simulation-extractability ($\mathsf{wb\text{-}sok\text{-}ext}$) of SoK using an adversary $\mathcal{A}$ against the white-box simulation-extractability of $\mathsf{ABS}^{\mathsf{gc}}$ as a subroutine. In the remainder of this proof we omit "white-box" and "strong" to shorten notation. Then, the $\mathsf{wb\text{-}sok\text{-}ext}$ supplies that for adversary $\mathcal{B}$ there exists an extractor $\mathsf{SoK.Extr}_{\mathcal{B}}$, such that $\mathrm{Adv}^{\mathsf{wb\text{-}sok\text{-}ext}}_{\mathcal{B},\mathsf{Extr}_{\mathcal{B}},\mathsf{SoK}}(\lambda) = \mathsf{negl}$. From $\mathsf{SoK.Extr}_{\mathcal{B}}$ we then build the $\mathsf{ABS}^{\mathsf{gc}}$ extractor $\mathsf{ABS}^{\mathsf{gc}}.\mathsf{Extr}_{\mathcal{A}}$. Based on this, we then bound the advantage of adversary $\mathcal{A}$ against the simulation-extractability of $\mathsf{ABS}^{\mathsf{gc}}$ via two reductions. One reduction to the $\mathsf{wb\text{-}sok\text{-}ext}$ of the SoK and one to the unforgeability of the signature scheme Sig. More formally, we show that for all adversaries $\mathcal{A}$ against the simulation-extractability of $\mathsf{ABS}^{\mathsf{gc}}$, there exists an adversary $\mathcal{B}$ against the simulation-extractability of SoK, such that for any extractor $\mathsf{SoK.Extr}_{\mathcal{B}}$, there exists an extractor $\mathsf{ABS}^{\mathsf{gc}}.\mathsf{Extr}_{\mathcal{A}}$, and there exists a forger $\mathcal{F}$ against the unforgeability of Sig such that

$$\mathrm{Adv}^{\mathsf{wb\text{-}sim\text{-}ext}}_{\mathcal{A},\mathsf{Extr}_{\mathcal{A}},\mathsf{ABS}^{\mathsf{gc}}}(\lambda) \leq \mathrm{Adv}^{\mathsf{wb\text{-}sok\text{-}ext}}_{\mathcal{B},\mathsf{Extr}_{\mathcal{A}},\mathsf{SoK}}(\lambda) + \mathrm{Adv}^{\mathsf{euf\text{-}cma}}_{F,\mathsf{Sig}}(\lambda).$$

Let us start with the construction of the $\mathsf{wb\text{-}sok\text{-}ext}$ adversary $\mathcal{B}$ in the experiment $\mathrm{Exp}^{\mathsf{wb\text{-}sok\text{-}ext}}_{\mathcal{B},\mathsf{Extr}_{\mathcal{B}},\mathsf{SoK}}(\lambda)$ (Definition 2.3.14).

1. $\mathcal{B}$ receives input pp_{SoK} and is given oracle access to a simulation signing oracle $\mathcal{O}^{\mathrm{S}}_{pp_{\mathsf{SoK}},td_{sim}}$. Note, that pp_{SoK} includes the public parameters pp'. Let r denote the randomness of $\mathcal{B}$.

2. $\mathcal{B}$ generates a Sig key pair $(pk, sk) \leftarrow \mathsf{Sig.KeyGen}(pp)$,
   - sets $msk := sk$ and
   - sets $pp := (pp_{\mathsf{SoK}}, pk)$.

3. $\mathcal{B}$ runs $\mathcal{A}(pp; r)$ and answers the oracle queries by $\mathcal{A}$ as follows.

   $\mathcal{O}^{\mathsf{SimKeyGen}}_{pp,msk,td_{sim}}(\mathbb{A}_i)$: $\mathcal{B}$ generates $(\mathbb{A}_i, \theta_i) \leftarrow \mathsf{ABS^{gc}.KeyGen}(pp, msk, \mathbb{A}_i)$, adds $(i, sk_{\mathbb{A}_i}, \mathbb{A}_i)$ to Q_{KeyGen} and outputs secret key $sk_{\mathbb{A}_i} := (\mathbb{A}_i, \theta_i)$.

   $\mathcal{O}^{\mathsf{SimSign}}_{pp,msk,td_{sim}}(m_j, \mathbb{P}_j)$: $\mathcal{B}$ sets $x_j := (pk, \mathbb{P}_j)$, queries its oracle for a signature $\sigma_j \leftarrow \mathcal{O}^{\mathrm{S}}_{pp_{\mathsf{SoK}},td_{sim}}(x_j, m_j)$, adds $(m_j, \mathbb{P}_j)$ to Q_{Sign}, and outputs σ_j.

4. Eventually $\mathcal{A}$ outputs $(m^*, \mathbb{P}^*, \sigma^*)$, then $\mathcal{B}$ outputs $(x^* := (pk, \mathbb{P}^*), m^*, \sigma^*)$.

Extractor for $\mathsf{ABS^{gc}}$. Next, let us move to the construction of the ABS extractor $\mathsf{ABS^{gc}.Extr}_{\mathcal{A}}$ from $\mathsf{SoK.Extr}_{\mathcal{B}}$ for adversary $\mathcal{B}$ as defined above. We observe that a SoK extractor is sufficient, since the witness w that it extracts for $R^{\mathrm{ABS}}_{\mathsf{Sig}}$ is an attribute $\mathbb{A}^*$ and Sig signature θ^* on $\mathbb{A}^*$. Together they form the $\mathsf{ABS^{gc}}$ secret key $sk_{\mathbb{A}^*} = (\mathbb{A}^*, \theta^*)$. Formally, to use $\mathsf{SoK.Extr}_{\mathcal{B}}$ we need a transcript $trans_{\mathcal{B}}$ of $\mathcal{B}$ as an input. The definition of white-box simulation-extractability for ABS (Definition 3.2.7) supplies us with the transcript $trans_{\mathcal{A}}$ of $\mathcal{A}$ in the experiment.

Observe, that given transcripts $trans_{\mathcal{A}}$, we can efficiently compute $trans_{\mathcal{B}}$. Transcripts $trans_{\mathcal{A}}$ and $trans_{\mathcal{B}}$ only differ in the key generation (step 2) for Sig and the key generation oracle queries that $\mathcal{B}$ answers on its own. Recall that we start adversary $\mathcal{A}$ with adversary $\mathcal{B}$'s randomness, hence given $trans_{\mathcal{A}}$ (inputs, outputs, and randomness of $\mathcal{A}$), we can efficiently compute the keys via $\mathsf{Sig.KeyGen}$ and $\mathsf{ABS^{gc}.KeyGen}$ respectively. Hence, let T be a polynomial-time algorithm that does what we described above, computing $trans_{\mathcal{B}}$ given $trans_{\mathcal{A}}$ as input. Formally, our $\mathsf{ABS^{gc}}$ extractor $\mathsf{ABS^{gc}.Extr}_{\mathcal{A}}$ is then defined as follows.

$\mathsf{ABS^{gc}.Extr}_{\mathcal{A}}(trans_{\mathcal{A}})$

$trans_{\mathcal{B}} := \mathsf{T}(trans_{\mathcal{A}})$

return $\mathsf{SoK.Extr}_{\mathcal{B}}(trans_{\mathcal{B}})$

With the extractor for $\mathsf{ABS^{gc}}$ in place, we proceed and look at the winning condition of $\mathcal{A}$ in $\mathrm{Exp}^{\mathsf{wb\text{-}sim\text{-}ext}}_{\mathcal{A},\mathsf{Extr}_{\mathcal{A}},\mathsf{ABS}}(\lambda)$ and its advantage. We then relate it to the winning condition of $\mathcal{B}$ in $\mathrm{Exp}^{\mathsf{wb\text{-}sok\text{-}ext}}_{\mathcal{B},\mathsf{Extr}_{\mathcal{B}},\mathsf{SoK}}(\lambda)$ and later to F in $\mathrm{Exp}^{\mathsf{euf\text{-}cma}}_{F,\mathsf{Sig}}(\lambda)$. After $\mathcal{A}$'s final output and using extractor $\mathsf{ABS^{gc}.Extr}_{\mathcal{A}}$ to extract $\mathbb{A}^*$, adversary $\mathcal{A}$ wins, if all the following conditions hold.

- $\mathsf{ABS^{gc}.Verify}(pp, m^*, \mathbb{P}^*, \sigma^*) = 1 \wedge$

- $(m^*, \mathbb{P}^*) \notin Q_{\mathsf{Sign}} \wedge$
- $(\mathbb{A}^* \notin Q_{\mathsf{KeyGen}} \vee \mathbb{P}^*(\mathbb{A}^*) = 0)$

We distinct two cases based on the disjunction in the last item. In the first case, we bound the advantage of $\mathcal{A}$ against wb-sim-ext of $\mathsf{ABS}^{\mathsf{gc}}$ with the simulation-extractability of SoK and in the second case with the unforgeability of Sig. In the following, we refer to adversary $\mathcal{A}$ that wins in case 1 as $\mathcal{A}_1$ and $\mathcal{A}_2$ in case 2. In the first case let the winning condition of $\mathcal{A}_1$ be:

($\mathcal{A}_1$.a) $\mathsf{ABS}^{\mathsf{gc}}.\mathsf{Verify}(pp, m^*, \mathbb{P}^*, \sigma^*) = 1 \wedge$

($\mathcal{A}_1$.b) $(m^*, \mathbb{P}^*) \notin Q_{\mathsf{Sign}} \wedge$

($\mathcal{A}_1$.c) $(\mathbb{A}^* \notin Q_{\mathsf{KeyGen}} \vee \mathbb{P}^*(\mathbb{A}^*) = 0)$

In the second case let the winning condition of $\mathcal{A}_2$ be:

($\mathcal{A}_2$.a) $\mathsf{ABS}^{\mathsf{gc}}.\mathsf{Verify}(pp, m^*, \mathbb{P}^*, \sigma^*) = 1 \wedge$

($\mathcal{A}_2$.b) $(m^*, \mathbb{P}^*) \notin Q_{\mathsf{Sign}} \wedge$

($\mathcal{A}_2$.c) $(\mathbb{A}^* \notin Q_{\mathsf{KeyGen}} \wedge \mathbb{P}^*(\mathbb{A}^*) = 1)$

Case 1. We show that whenever sim-ext ABS adversary $\mathcal{A}_1$ wins, then sim-ext SoK adversary $\mathcal{B}$ also wins. Therefore, let us also consider $\mathcal{B}$'s winning condition in $\mathrm{Exp}^{\mathsf{wb\text{-}sok\text{-}ext}}_{\mathcal{B},\mathsf{Extr}_{\mathcal{B}},\mathsf{SoK}}(\lambda)$ and relate it in the following to $\mathcal{A}_1$'s winning condition.

($\mathcal{B}$.a) $\mathsf{SoK}.\mathsf{Verify}(pp_{\mathsf{SoK}}, x^* \coloneqq (pk, \mathbb{P}^*), m^*, \sigma^*) = 1 \wedge$

($\mathcal{B}$.b) $(x^*, m^*, \sigma^*) \notin Q \wedge$

($\mathcal{B}$.c) $(x^*, w^* \coloneqq (\mathbb{A}^*, \theta^*)) \notin R^{\mathrm{ABS}}_{\mathsf{Sig}}$

By construction it holds that ($\mathcal{A}$.a) implies ($\mathcal{B}$.a). In detail, $\mathsf{ABS}^{\mathsf{gc}}.\mathsf{Verify}$ just runs $\mathsf{SoK}.\mathsf{Verify}$, hence whenever $\mathsf{ABS}^{\mathsf{gc}}.\mathsf{Verify}(pp, m^*, \mathbb{P}^*, \sigma^*) = 1$ holds, it also holds that $\mathsf{SoK}.\mathsf{Verify}(pp_{\mathsf{SoK}}, x^*, m^*, \sigma^*) = 1$. Regarding ($\mathcal{A}$.b), if $(m^*, \mathbb{P}^*,) \notin Q_{\mathsf{Sign}}$ holds, then $\mathcal{B}$ did not query his signing oracle and therefore it also holds that $(x^*, m^*, \sigma^*) \notin Q$ ($\mathcal{B}$.b) for any σ^*. Since ($\mathcal{A}_1$.c) is a disjunction, if $\mathbb{P}^*(\mathbb{A}^*) = 0$ holds, then the instance $x^* = (pk, \mathbb{P}^*)$ and witness $w^* = (\mathbb{A}^*, \theta^*)$ is not an element of the relation $R^{\mathrm{ABS}}_{\mathsf{Sig}}$ ($\mathcal{B}$.c). Note, for an instance-witness pair $(x', w') \in R^{\mathrm{ABS}}_{\mathsf{Sig}}$, it has to hold that $\mathbb{P}^*(\mathbb{A}^*) = 1$. By the above we conclude, if $\mathcal{A}_1$ wins $\mathrm{Exp}^{\mathsf{wb\text{-}sim\text{-}ext}}_{\mathcal{A}_1,\mathsf{Extr}_{\mathcal{A}},\mathsf{ABS}}(\lambda)$, then adversary $\mathcal{B}$ wins $\mathrm{Exp}^{\mathsf{wb\text{-}sok\text{-}ext}}_{\mathcal{B},\mathsf{Extr}_{\mathcal{B}},\mathsf{SoK}}(\lambda)$.

Case 2. In the following we build a forger F against the signature scheme Sig in experiment $\mathrm{Exp}^{\mathsf{euf\text{-}cma}}_{F,\mathsf{Sig}}(\lambda)$ using adversary $\mathcal{A}_2$. In case 2 we know that the $\mathsf{ABS}^{\mathsf{gc}}$ signature is valid, a signature for $(m^*, \mathbb{P}^*)$ was never queried, and most importantly a secret key for the satisfying attribute $\mathbb{A}^*$ was also never queries. Since secret keys are Sig signatures on attributes and ABS signatures are just signatures of knowledges generated by SoK, we extract the SoK to obtain the corresponding Sig signature θ^* on $\mathbb{A}^*$. Finally, we can then output $(\mathbb{A}^*, \theta^*)$ as the forgery for Sig. More formally, assuming $\mathcal{A}_2$ in $\mathrm{Exp}^{\mathsf{wb\text{-}sim\text{-}ext}}_{\mathcal{A}_2,\mathsf{Extr}_{\mathcal{A}},\mathsf{ABS}}(\lambda)$, and that the SoK is $\mathsf{wb\text{-}sok\text{-}ext}$, we build a forger F in $\mathrm{Exp}^{\mathsf{euf\text{-}cma}}_{F,\mathsf{Sig}}(\lambda)$, which is defined as follows.

1. F receives input pk and is given oracle access to a signing oracle $\mathcal{O}^{\mathsf{Sign}}_{pk,sk}$. Note, that pk includes the public parameters pp'.
2. F gets $(pp_{\mathsf{SoK}}, td_{sim}) \leftarrow \mathsf{SoK.SimSetup}(pp')$, and sets $pp := (pp_{\mathsf{SoK}}, pk)$, $msk := sk$, and $td_{sim} := td_{sim}$. (– $\mathsf{ABS}^{\mathsf{gc}}$ setup finished –)
3. F runs $\mathcal{A}_2(pp)$ and answers oracle queries by $\mathcal{A}_2$ as follows.

   $\mathcal{O}^{\mathsf{SimKeyGen}}_{pp,msk,td_{sim}}(\mathbb{A}_i)$: if $\mathbb{A} \notin \mathcal{U}$, F outputs $\perp$. Else F queries his own oracle for a signature $\theta_i \leftarrow \mathcal{O}^{\mathsf{Sign}}_{pk,sk}(\mathbb{A}_i)$. Sets $sk_{\mathbb{A}_i} := (\mathbb{A}_i, \theta_i)$ adds $(i, sk_{\mathbb{A}_i}, \mathbb{A}_i)$ to Q_{KeyGen} and outputs secret key $sk_{\mathbb{A}_i}$.

   $\mathcal{O}^{\mathsf{SimSign}}_{pp,msk,td_{sim}}(m_j, \mathbb{P}_j)$: F sets $x_j := (pk, \mathbb{P}_j)$, adds $(m_j, \mathbb{P}_j)$ to Q_{Sign} and outputs $\sigma_j \leftarrow \mathsf{SoK.SimSign}(pp_{\mathsf{SoK}}, td_{sim}, x_j, m_j)$.
4. On output $(m^*, \mathbb{P}^*, \sigma^*)$ by $\mathcal{A}_2$, set $(x^* := (pk, \mathbb{P}^*), m^*, \sigma^*)$ and use the extractor $\mathsf{ABS}^{\mathsf{gc}}.\mathsf{Extr}_{\mathcal{A}_2}$ to get the witness $w = (\mathbb{A}^*, \theta^*)$ for $R^{\mathrm{ABS}}_{\mathsf{Sig}}$.
5. If $\mathcal{A}_2$ winning condition ($\mathcal{A}_2$.c) $\mathbb{A}^* \notin Q_{\mathsf{KeyGen}} \wedge \mathbb{P}^*(\mathbb{A}^*) = 1$ is not fulfilled abort, else proceed.
6. Output $(\mathbb{A}^*, \theta^*)$ as a forgery.

First, observe that F is a ppt and it simulates the experiment for $\mathcal{A}_2$ perfectly by executing the $\mathsf{ABS}^{\mathsf{gc}}$ algorithms and for the key generation queries F uses its own signing oracle. Also observe, F uses $\mathcal{A}_2$, where $\mathcal{A}_2$ is just $\mathcal{A}$ conditioned on the case 2. An extractor for F exists by the result of [FN16a; FN16b] (see O-SNARK discussion above). Formally, this involves the definition of an adversary $\mathcal{B}$' that works similar to $\mathcal{B}$ above, but using a successful forger instead of $\mathcal{A}$, and having access to pre-sampled signing oracle answers. Intuitively, $\mathcal{B}$' casts a forger into the structure of a $\mathsf{wb\text{-}sok\text{-}ext}$ adversary in $\mathrm{Exp}^{\mathsf{wb\text{-}sok\text{-}ext}}_{\mathcal{B}',\mathsf{Extr}_{\mathcal{B}'},\mathsf{SoK}}(\lambda)$ (Definition 2.3.14). The details of the transformation from extraction with oracles and a hash-then-sign signature scheme to extraction without oracles are not insightful here and we refer to [FN16a, Section 4.3]. Next, let us look at the winning condition of F and relate it to ($\mathcal{A}_2$.c).

(F.a) $\mathsf{Sig.Verify}(pk, \mathbb{A}^*, \theta^*) = 1 \wedge$

$(F.\text{b})$ $(\mathbb{A}^*) \notin Q$

From the first part of the conjunction in $(\mathcal{A}_2.\text{c})$, $\mathbb{A}^* \notin Q_{\mathsf{KeyGen}} \wedge \mathbb{P}^*(\mathbb{A}^*) = 1$, we know that a Sig signature on $\mathbb{A}^*$ was never queried. Therefore, $(F.\text{b})$ holds. Further, from $(\mathcal{A}_2.\text{a})$, $\mathsf{ABS}^{\mathsf{gc}}.\mathsf{Verify}(pp, m^*, \mathbb{P}^*, \sigma^*) = 1$, we know that the $\mathsf{ABS}^{\mathsf{gc}}$ secret key is valid, since the verification runs $\mathsf{SoK.Verify}$. In detail, if $\mathsf{SoK.Verify}$ outputs 1 we know that σ^* is a valid SoK for the relation $R_{\mathsf{Sig}}^{\mathrm{ABS}}$ (Definition 3.3.1), which says that the secret key θ^* on the attributes $\mathbb{A}^*$ used to generate σ^* has to be valid, i.e., $\mathsf{Sig.Verify}(pk, \mathbb{A}^*, \theta^*) = 1$. Hence, $(F.\text{a})$ holds.

In conclusion, we have shown that for all adversaries $\mathcal{A}$ against the simulation-extractability of $\mathsf{ABS}^{\mathsf{gc}}$, with advantage $\mathrm{Adv}^{\mathsf{wb\text{-}sim\text{-}ext}}_{\mathcal{A},\mathsf{Extr}_{\mathcal{A}},\mathsf{ABS}^{\mathsf{gc}}}(\lambda)$, there exists an adversary $\mathcal{B}$ against the white-box strong simulation-extractability of SoK, with advantage $\mathrm{Adv}^{\mathsf{wb\text{-}sok\text{-}ext}}_{\mathcal{A},\mathsf{Extr}_{\mathcal{A}},\mathsf{SoK}}(\lambda)$, such that for any extractor $\mathsf{SoK.Extr}_{\mathcal{B}}$, there exists an extractor $\mathsf{ABS}^{\mathsf{gc}}.\mathsf{Extr}_{\mathcal{A}}$, and there exists a forger $\mathcal{F}$ against the unforgeability of Sig, with advantage $\mathrm{Adv}^{\mathsf{euf\text{-}cma}}_{F,\mathsf{Sig}}(\lambda)$, such that

$$\mathrm{Adv}^{\mathsf{wb\text{-}sim\text{-}ext}}_{\mathcal{A},\mathsf{Extr}_{\mathcal{A}},\mathsf{ABS}^{\mathsf{gc}}}(\lambda) \leq \mathrm{Adv}^{\mathsf{wb\text{-}sok\text{-}ext}}_{\mathcal{A},\mathsf{Extr}_{\mathcal{A}},\mathsf{SoK}}(\lambda) + \mathrm{Adv}^{\mathsf{euf\text{-}cma}}_{F,\mathsf{Sig}}(\lambda) \ .$$

□

3.3.2 Discussion of the Generic ABS Construction

Our construction is similar to the generic policy-based signature (PBS) scheme by Bellare and Fuchsbauer [BF14] in that they are both based on a relation. The difference in the PBS is that the message is part of the instance and the policy is part of the witness. Also, where we use a signature of knowledge as a black-box, Bellare and Fuchsbauer rely on simulation-extractable NIZK compatible with their PBS relation. As a result (to hide the policy and bind the message) they have to instantiate Groth-Sahai proofs [GS08; GS12] with custom proof equations. The resulting PBS [BF14] is not succinct and is black-box simulation-extractable. Also note, that the proofs are more straightforward in [BF14], because of the black-box variant of simulation-extractability. To be clear, if one changes their PBS construction to use a succinct simulation-extractable SNARK (SE-SNARK), then the resulting PBS scheme is also succinct. Additionally our construction is similar to the succinct functional signature construction in [BGI14], where we have to note that their unforgeability claim does not hold without assuming the existence of O-SNARKs as shown in [FN16a; FN16b].

Overall, the idea of employing SNARKs in combination with a signature scheme to get succinctness is not novel, but we have formally proven our construction secure by considering and stating all necessary assumptions.

For the proof of Theorem 3.3.2 we have to assume that the extraction in presence of oracles, introduced in [FN16b], also works for simulation-extractability (SE-SNARK) and not only for knowledge soundness (SNARK). Another way of instantiating our generic $\mathsf{ABS}^{\mathsf{gc}}$ is to consider black-box extraction. Then, we cannot

achieve succinctness of ABS, however we do not have to deal with the white-box extraction in presence of an oracle.

Generalization to black-box simulation-extractability. The Theorem 3.3.2 is specifically stated for the white-box variant of simulation-extractability, where the extractor depends on the adversary. Hence, the extractor has access to the code and the transcript of the adversary. As already mentioned, the reason for the white-box variant in Theorem 3.3.2 is that we use a succinct SoK based on a SE-SNARK to instantiate the generic construction in the next section (cf. Section 3.3.3). According by Gentry and Wichs [GW11] constructing a *black-box* SE-SNARK is an open question and all known constructions rely on special non-falsifiable assumptions, so called knowledge assumptions, with white-box extractors. The current state of research is that such assumptions seem inherent for SE-SNARKs [BCCT12; GW11].

Additionally, we note that white-box definitions are more involved to work with in security proofs (reductions) compared to the black-box variant. We have already seen this by considering O-SNARKs (and extraction in the presence of oracles) in the proof of Theorem 3.3.2. Considering the proof of the Theorem 3.3.2, we observe that the white-box notion is only used in the extra steps taken to show the existence of the extractor for $\mathsf{ABS}^{\mathsf{gc}}$, e.g., we showed that the transcript can be efficiently computed. Theorem 3.3.2 and its proof generalizes to a black-box variant of simulation-extractability (for both SoK and $\mathsf{ABS}^{\mathsf{gc}}$), if we handle the steps differently. Let us first state the theorem as a black-box variant before we describe the steps in detail.

Theorem 3.3.3. *If* Sig *is EUF-CMA (Definition 2.3.6) and* SoK *is black-box strong simulation-extractable (Definition 2.3.16), then* $\mathsf{ABS}^{\mathsf{gc}}$ *(Construction 3.3.1) is black-box simulation-extractable (Definition 3.2.8) with respect to* $\mathsf{RGen}_{\mathsf{Sig}}^{\mathrm{ABS}}$.

Here we can use any EUF-CMA signature scheme and we do not need to require that it is based on the hash-then-sign paradigm. This was only necessary for the O-SNARK solution. We can also drop that the simulation-extractability holds with respect to the specific signature scheme compared to Theorem 3.3.2, i.e., the signing oracle specified by the signature scheme. We do not provide a formal proof for Theorem 3.3.3, since it is similar to the proof of Theorem 3.3.2. However we give an outline next.

The case distinction and corresponding structure of reductions, presented in the proof of Theorem 3.3.2, stay the same. Only the steps involving the white-box extractor have to be addressed. Overall, the proof in the black-box setting is straightforward, since given a black-box simulation-extractable SoK scheme SoK, we know that there exists *one* extractor for all adversaries and we can just use this extractor as our $\mathsf{ABS}^{\mathsf{gc}}$ extractor. Hence, there is no need for extra steps to show that a specific adversary dependent extractor exists and how to get a transcript. In the remainder of the proof, we just use the SoK extractor to extract the final

output of the adversaries, i.e., we extract from a signature of knowledge to get the witness necessary in the reductions for case 1 to bb-sok-ext and case 2 to euf-cma.

Note, if we use a SoK with a black-box extractor (also called online extractor) in the ROM, for example one based on the Fischlin transformation [BFW15; Fis05] (which is based on the Fiat-Shamir transformation [FS87] on Sigma protocols), or the CRS based transformation from [Gro06], we directly get simulatability via the ROM. However, the security of the resulting attribute-based signature (Theorem 3.3.3) is then also in the ROM. Note that supported policies depend on the concrete relations that the signatures of knowledge support. For example, satisfiability of boolean formulas over attributes if we use a SoK based on Sigma protocols [CDS94; FKMV12].

Another option, in the standard model, is the black-box simulation-extractable signature of knowledge for boolean circuits presented in [BGPR20]. Using this signature of knowledge to instantiate Construction 3.3.1 gives us an ABS scheme supporting boolean circuits as policies and with signatures of size $\mathcal{O}(l + d)$ (d is the depth of the circuit and l the size of the input).

Consequently, there are instantiations of our generic attribute-based signature scheme (Construction 3.3.1) using black-box and white-box simulation-extractable signatures of knowledge. Since, ABS schemes using similar black-box constructions exist [AAS16; EE16; EK18; Gha15], even though they do not explicitly use simulation-extractability, we present in the following a concrete instantiation of Construction 3.3.1 using a *succinct* white-box simulation-extractable signature of knowledge. We deem the resulting succinct attribute-based signature scheme interesting, even under the consideration of the assumptions necessary to prove Theorem 3.3.2.

3.3.3 ABS Instantiation from SNARKY Signatures

In this section we show an instantiation of our generic ABS construction $\mathsf{ABS}^{\mathsf{gc}}$ that is succinct, i.e, constant-size signatures (3 group elements) and therefore independent of the number of attributes, size of the policy and message. To prove the security of $\mathsf{ABS}^{\mathsf{gc}}$ (Theorem 3.3.2) we relied on a hash-then-sign and EUF-CMA secure signature scheme Sig, and a white-box strong simulation-extractable SoK scheme SoK with respect to Sig (and $\mathsf{RGen}_{\mathsf{Sig}}^{\mathrm{ABS}}$). Intuitively, we can use any hash-then-sign and EUF-CMA secure signature scheme Sig, as long as the signature of knowledge can prove the validity of the Sig signatures. More interestingly, is to instantiate the signature of knowledge for $R_{\mathsf{Sig}}^{\mathrm{ABS}} \leftarrow \mathsf{RGen}_{\mathsf{Sig}}^{\mathrm{ABS}}$ to get an efficient attribute-based signature scheme.

SNARKY Signatures from succinct SE-SNARK. In [GM17] Groth and Maller show a pairing-based succinct simulation-extractable non-interactive zero-knowledge argument of knowledge (SE-NIZK) with a CRS, called simulation-extractable SNARK (SE-SNARK). See Section 2.3.5 for a formal definition of NIZK and SNARK. Importantly for us, the authors show a transformation from

SE-SNARKs to a *succinct signature of knowledge* scheme, which we refer to as GM-SoK. They call this SNARKY signatures. In [BKSV21] it is shown that this transformation not only works with strong simulation-extractable SE-SNARKs as presented in [GM17], but also with weak simulation-extractable SE-SNARKs. Interesting for our goal to construct an ABS scheme with constant-size signatures is that GM-SoK features constant-size signatures consisting of only 3 group elements. This is proven to be optimal for pairing-based succinct SoK in [GM17]. In general, the transformation to a SoK can be instantiated with any simulation-extractable non-interactive zero-knowledge argument of knowledge (SE-NIZK), as long as it supports a specific relation that we present in the following. However, if the SE-NIZK is also succinct (a SE-SNARK) the SoK inherits the succinctness property. Furthermore, the SE-SNARK and GM-SoK presented by Groth and Maller [GM17] can prove satisfiability of arithmetic circuits via representing them as square arithmetic programs.

We use the SoK scheme GM-SoK to instantiate our generic ABS construction $\mathsf{ABS}^{\mathsf{gc}}$. This means, that the features of GM-SoK directly transfers to our resulting ABS scheme. Hence, it features constant-size signatures and expressive policies in the form of arithmetic circuits. Note, for a sound instantiation of our generic ABS construction we have to assume that the SE-SNARK of [GM17] is simulation-extractable with respect to the signing oracle specified by the signature scheme Sig that we use, see the discussion in Section 3.3.1. Regarding security, in [GM17] the authors prove their SE-SNARK secure under the extended power knowledge of exponent (XPKE) and computational polynomial (Poly) assumption for type 3 pairings. Both assumptions hold in the generic group model (cf. [GM17]). To prove the security of the SoK scheme GM-SoK the authors additionally assume a target collision-resistant hash function (Definition 2.3.3). Since we instantiate our generic ABS construction $\mathsf{ABS}^{\mathsf{gc}}$ with the succinct signature of knowledge GM-SoK, we describe GM-SoK for completeness in the following. Subsequently, we discuss the details of our instantiation of $\mathsf{ABS}^{\mathsf{gc}}$.

Let us first give an intuition of the GM-SoK scheme before presenting the formal definition. Given a SE-SNARK Arg we can generate proofs for instance-witness pairs of a relation R' (satisfiability of arithmetic circuits), but for a signature of knowledge we need to bind the message m to be signed to the proofs. In GM-SoK this is done via a target collision-resistant hash function Hash (Definition 2.3.3), where the hash value $\mathsf{H}(hk, m)$ and the hash key hk is part of an extended instance of R'. Hence, the message is bound to an instance-witness pair of R' via a simple extension of the relation, that guarantees that the hash key and value is of the right length. This means, that GM-SoK proves that the signer knows a hash key and value together with a valid instance-witness pair of R'. Since the SE-SNARK Arg [GM17] proves satisfiability of arithmetic circuits, the extended relation, in the following called R, is just a constant extension of the arithmetic circuit of the inner relation R'. For details on representing relations as arithmetic circuit we refer to [GM17].

Definition 3.3.2 (GM-SoK [GM17])
Given an inner relation generator RGen'. We define the following relation *relation* for $R' \leftarrow \mathsf{RGen}'(1^\lambda)$ and l_K, l_h polynomial in λ:

$$R = \left\{((hk, h, x), w) : hk \in \{0,1\}^{l_K(\lambda)} \wedge h \in \{0,1\}^{l_h(\lambda)} \wedge (x, w) \in R'\right\}.$$

Let RGen denote a relation generator that first runs $R' \leftarrow \mathsf{RGen}'(1^\lambda)$ and returns R defined as above. Let $\mathsf{Hash} = (\mathsf{Hash.KeyGen}, \mathsf{Hash.H})$ be a target collision-resistant hash function with key length l_K and hash length l_h and let $\mathsf{Arg} = (\mathsf{Arg.Setup}, \mathsf{Arg.Prove}, \mathsf{Arg.Verify}, \mathsf{Arg.SimProve})$ be a SE-SNARK for RGen. The following describes $\mathsf{GM\text{-}SoK}(\mathsf{RGen}')$ for the inner relation generator RGen'.

$\mathsf{SoK.Setup}(R)$
$(crs, td_{sim}) \leftarrow \mathsf{Arg.Setup}(R)$
return $pp = crs$

$\mathsf{SoK.Sign}(pp, x, w, m)$
$hk \leftarrow \mathsf{Hash.KeyGen}(pp)$
$h = \mathsf{Hash.H}(hk, m)$
$\pi \leftarrow \mathsf{Arg.Prove}(crs, (K, h, x), w)$
return $\sigma = (K, \pi)$

$\mathsf{SoK.Verify}(pp, x, m, \sigma)$
parse $\sigma = (K, \pi)$
$h = \mathsf{Hash.H}(hk, m)$
return $\mathsf{Arg.Verify}(crs, (K, h, x), \pi)$

$\mathsf{SoK.SimSetup}(R')$
$(crs, td_{sim}) \leftarrow \mathsf{Arg.Setup}(R)$
return $pp = (crs, td_{sim})$

$\mathsf{SoK.SimSign}(pp, td_{sim}, x, m)$
$hk \leftarrow \mathsf{Hash.KeyGen}(pp)$
$h = \mathsf{Hash.H}(hk, m)$
$\pi \leftarrow \mathsf{Arg.SimProve}(crs, td_{sim}, (K, h, x))$
return $\sigma = (K, \pi)$

◇

The GM-SoK and the following theorem are originally presented in [GM17].

Theorem 3.3.4. *If* Hash *is a target collision-resistant hash function and* Arg *is an secure SE-SNARK argument (perfect completeness, perfect zero-knowledge, simulation-extractable) for* RGen*, then the SoK scheme* GM-SoK *is a secure (perfect correctness, perfect simulatability, white-box simulation-extractability) signature of knowledge for* RGen' *with respect to the message space* $\mathcal{M} = \{0,1\}^*$.

The perfect correctness and perfect simulatability of GM-SoK follow from the perfect completeness of Arg respectively that Arg is perfectly zero-knowledge. Since Arg is white-box simulation-extractable the white-box simulation-extractability of GM-SoK follows. The full proof is given by Groth and Maller in [GM17], where the authors also show the reverse direction of the above theorem.

Succinct ABS from GM-SoK. Next, let us turn to the instantiation of our generic ABS construction $\mathsf{ABS}^{\mathsf{gc}}$ with GM-SoK. Since the signatures of GM-SoK are succinct (only 3 group elements), it is our signature of knowledge of choice to instantiate $\mathsf{ABS}^{\mathsf{gc}}$ (Construction 3.3.1). By using GM-SoK we also fix the policy class to be arithmetic circuits. To formalize the instantiation of Construction 3.3.1, let Sig be a hash-then-sign and EUF-CMA secure signature scheme. Then, $\mathsf{GM\text{-}SoK}(\mathsf{RGen}_{\mathsf{Sig}}^{\mathrm{ABS}})$ instantiated with our ABS relation generator $\mathsf{RGen}_{\mathsf{Sig}}^{\mathrm{ABS}}$ as the inner relation generator yields a succinct signature of knowledge for $\mathsf{ABS}^{\mathsf{gc}}$. Hence, $\mathsf{ABS}^{\mathsf{gc}}$ (Construction 3.3.1) instantiated with Sig and $\mathsf{GM\text{-}SoK}(\mathsf{RGen}_{\mathsf{Sig}}^{\mathrm{ABS}})$ is an ABS scheme with constant-size signatures (3 group elements) and we refer to it as $\mathsf{ABS}^{\mathsf{succi}}$.

Note, a resulting signature of $\mathsf{ABS}^{\mathsf{succi}}$ is essentially a succinct SE-SNARK proof, where we arithmetized the relation (including the verification of the Sig signature). It proves that the ABS signer of a message-policy pair knows a valid Sig signature on attributes that satisfy the policy.

Bandwidth concerned applications benefit the most from constant-size ABS. For example, $\mathsf{ABS}^{\mathsf{succi}}$ can improve existing authentication mechanisms, e.g., challenge-response authentication, by providing a rich attribute-based authentication without straining the bandwidth of involved parties.

3.3.4 Evaluation of our Succinct ABS

Important for practical application of ABS are the concrete sizes of the signatures and keys and the supported policies. Therefore, we give a comparison to other ABS schemes and report on a prototype implementation of $\mathsf{ABS}^{\mathsf{succi}}$. The latter provides an outlook on practical performance and key sizes.

Comparison of ABS schemes. We present a comparison of existing ABS schemes with $\mathsf{ABS}^{\mathsf{succi}}$ in Table 3.1. The table is not exhaustive, but lists attribute-based signature schemes that are presented in a similar model compared to ours.

Prototype implementation of our Succinct ABS. In the following we analyze the signature and key sizes of the scheme $\mathsf{ABS}^{\mathsf{succi}}$ on the basis of a prototype implementation. This serves an outlook for real-world applications of the scheme $\mathsf{ABS}^{\mathsf{succi}}$. Since our scheme $\mathsf{ABS}^{\mathsf{succi}}$ is instantiated with GM-SoK which is based on a SE-SNARK over arithmetic circuits [GM17], it is not straightforward to count the size of the elements (public parameters, keys) by studying the construction of the scheme. To get a (non optimized) estimate we evaluated $\mathsf{ABS}^{\mathsf{succi}}$ in the `libsnark` [SCI20] library as a prototype. Hence, it is a proof of concept. Note, the SE-SNARK by Groth and Maller [GM17] used in GM-SoK was already implemented in `libsnark`. For the prototype we used the pairing-based signature scheme by Pointcheval and Sanders [PS16] as Sig, i.e., for the issuing of the $\mathsf{ABS}^{\mathsf{succi}}$ secret keys. To setup the system we used the type 3 pairing bilinear group `mnt4` provided by `libsnark`. In a type 3 pairing, other than type 2, there is no efficiently

Table 3.1: Overview of signature sizes of attribute-based signature schemes. We list the signature size, model, assumptions, and supported policies for each scheme.

Scheme	Signature size	Model	Assumption	Policies
(1) [MPR11]	$36s + 2t + 24ks$	std	q-SDH, SXDH	MSP
(2) [MPR11]	$28s + 2t + 12k + 8$	std	SXDH	MSP
(3) [MPR11]	$s + t + 2$	GGM	coll-res hash	MSP
[HLLR12]*	3	std	n-DHE	Threshold
[OT11]	$7s + 11$	std	DLIN, coll-res hash	Non-monotone SP
[OT13]	$13s$	ROM	DLIN	Non-monotone SP
[Che+13]**	3	std	SDP	Threshold
[SAH16]	$12l + 20N + 26$	std	SXDH	Non-monotone circuit
[EG17]	$27\|\mathbb{G}_1\| + 28\|\mathbb{G}_2\|$	std	SXDH	Threshold
$\mathsf{ABS}^{\mathsf{succi}}$	3	GGM	XPKE, Poly	Arithmetic circuit

* [HLLR12] is only selective-policy unforgeable. All others are adaptive unforgeable.

** [Che+13] composite order groups, no detailed security proofs provided.

$s \times t$ is the size of the (monotone) span program (MSP or non-monotone SP) where s corresponds to the number of attributes, λ security parameter, l input size of the circuit and number of attributes, and N number of gates in the circuit. GGM: generic group model. q-SDH: q-Strong Diffie-Hellman assumption [BBS04], SXDH: Symmetric External Diffie-Hellman assumption [GS12], n-DHE: n-Diffie-Hellman Exponent assumption [HLLR12], DLIN: Linear Diffie-Hellman assumption (Decision Linear assumption) [BBS04], SPD: Subgroup decision problem for 3 primes [LW10], XPKE: extended power knowledge of exponent assumption [GM17], Poly: computational polynomial assumption [GM17].

computable isomorphism from group $\mathbb{G}_2$ to $\mathbb{G}_1$. As mentioned, this is a prototype and to optimize the key sizes and timings we suggest to utilize other EUF-CMA signature schemes that are not pairing-based and other SNARK libraries, since active development of `libsnark` stopped.

In Table 3.2 we present the resulting sizes and the timings needed to generate the elements. For the evaluation we used a MacBook Pro (version late 2016, Intel Core i5-6287U, 3.1 GHz, 16 GB 2133 MHz LPDDR3). Hence, Table 3.2 shows that even on older hardware the prototype gets verification times lower than a second. The public parameters can be split for the signer and verifier, which yields a more practical scheme, cf. [GM17] for details. Note, that the signer's public parameters increases by 3 elements in $\mathbb{G}_1$ and 1 element in $\mathbb{G}_2$ (plus a constant) for each gate in the policy. The verifier's public parameters only increase by 1 element in $\mathbb{G}_1$. Further, our prototype does not use hash-then-sign to simplify its implementation. For this an additional constant number of elements, depending on the hash function, gets added to the public parameters.

3.4 On the Experiment-Based Security of ABS

In this section we analyze the relation of the presented experiment-based security notions of ABS. We show that simulation privacy is a stronger security notion than privacy and that simulation-extractability implies unforgeability. In detail,

Table 3.2: Overview of the libsnark prototype of our ABS scheme $\mathsf{ABS}^{\mathsf{succi}}$ using GM-SoK. We used a simple policy consisting of just one attribute. With $(\cdot, \cdot)$ we denote the number of elements in $\mathbb{G}_1$ and $\mathbb{G}_2$.

	Size	Time	Elements
Signer public params	12 MB	$\downarrow$	$(205291, 49703)$
Verifier public params	0.35 MB	$20s$	$(9357, 3)$
Signature	145 B	$15s$	$(2, 1)$
Verification		$0.65s$	
Verification (pre-computation included)		$1.1s$	

we show for ABS that simulation privacy (Definition 3.2.5) implies privacy (Definition 3.2.4) and that privacy does not imply simulation privacy. We show the latter with respect to policy classes, where satisfying attributes given a policy are not efficiently computable. Further, we show that a simulation-extractable ABS scheme is unforgeable, with respect to our definitions. This bridges the gab from ABS schemes constructed from simulation-extractable building blocks, like our generic ABS construction in Section 3.3, to the unforgeability notion of ABS schemes. Another ABS scheme from the literature that is build using SoK [EE16] does not use a simulation-extractability notion and directly shows that it is unforgeable. In terms of unforgeability, we our experiment-based unforgeability definition (Definition 3.2.6) does not implicitly assume a specific policy class as it is the case if we assume a policy class where satisfying attributes for a given policy are efficiently computable. It is therefore more general than the notion introduced by Maji et al. [MPR08; MPR10; MPR11]. Maji et al. define unforgeability with a signing oracle that computes satisfying attributes given a policy, generates a secret key on these attributes, and then outputs a signature under this secret key. Hence, the authors either implicitly assumed that satisfying attributes given just a policy are efficiently computable, which corresponds to their monotone policies, or their challenger is not efficient for general policies. Interestingly, Maji et al. [MPR10; MPR11] present an efficient algorithm to simulate signatures in the first step of their unforgeability proof. We observed that this step is common in ABS unforgeability proofs in the literature and present in [AHY15; ECGD14; EG17; EGK14; Li+10; SKAH18; UKLC15].

Overall, the security model of ABS was not the focus of the research community unlike security models of other privacy-preserving schemes, e.g., group signatures [BHSB19; BMW03; DS18; Gro07] and anonymous credentials [BB18; CDHK15; CKLM14; CL01]. In this section we show that the experiment-security model of ABS benefits from a simulation-based privacy, general unforgeability and simulation-extractability definition. We believe that our results simplifies security proofs of ABS. Namely, (1) simulation-extractable primitives can directly be used to build simulation-extractable ABS without taking the extra steps to show unforgeability separately, and (2) unforgeability proofs of simulation private ABS schemes

do not have to show that signatures are simulatable. Let us start with analyzing the relation of the unforgeability and simulation-extractability definitions.

3.4.1 On the Unforgeability of ABS

In the following we show that a simulation-extractable ABS scheme is also unforgeable. The proof is done via a sequence of games [Sho04]. Note, our definitions of simulation-extractability Definitions 3.2.7 and 3.2.8 require that the ABS scheme is simulation private. However, in the following we state simulation privacy as an explicit assumption to make it more clear.

Theorem 3.4.1. *If an ABS scheme is perfect simulation private (Definition 3.2.5) and white-box simulation-extractable (Definition 3.2.7), then it is unforgeable (Definition 3.2.6).*

Proof. The proof is done via a sequence of games (experiments), where we (1) start with the unforgeability experiment, (2) introduce simulation algorithms in the next experiment and (3) in the last experiment we change the winning condition to the one from the extractability experiment. Let us start with the definition of the corresponding experiments for an ABS scheme called ABS. We define the experiments $\mathrm{Exp}^{\mathsf{UF}}(\lambda)$, $\mathrm{Exp}^{\mathsf{UF+Sim}}(\lambda)$, and $\mathrm{Exp}^{\mathsf{UF+Sim+Ext}}(\lambda)$ in Experiment 3.7, where changes from one experiment to the next are highlighted. For $\mathrm{Exp}^{\mathsf{UF+Sim+Ext}}(\lambda)$ the extractor $\mathsf{Extr}_{\mathcal{A}}$ is a white-box extractor for an adversary $\mathcal{A}$, i.e., it has access to the transcript of $\mathcal{A}$. Note, $\mathrm{Exp}^{\mathsf{UF}}(\lambda)$ is our unforgeability experiment (Definition 3.2.6) repeated here for completeness. We refer to the experiments also as UF, $\mathsf{UF}+\mathsf{Sim}$, and $\mathsf{UF}+\mathsf{Sim}+\mathsf{Ext}$.

Suppose an adversary $\mathcal{A}$ against unforgeability (UF). We show in the following that, if adversary $\mathcal{A}$ recognizes a change in the experiments, i.e., adversary $\mathcal{A}$ distinguishes two consecutive experiments, we can construct adversaries against the assumptions. This means, if adversary $\mathcal{A}$ can distinguish experiment UF and $\mathsf{UF}+\mathsf{Sim}$, we build an adversary $\mathcal{B}$ against simulation privacy. Next, we show that from $\mathcal{A}$ in experiment $\mathsf{UF}+\mathsf{Sim}$ we construct an adversary $\mathcal{C}$ in experiment $\mathsf{UF}+\mathsf{Sim}+\mathsf{Ext}$. Finally, we show that from adversary $\mathcal{C}$ we can construct an adversary $\mathcal{D}$ against the white-box simulation-extractability (Definition 3.2.7).

Oracles in all experiments. The reveal oracle is present in all experiments.

$\mathcal{O}^{\mathsf{Reveal}}(i)$:

if $(i, sk_{\mathbb{A}_i}, \mathbb{A}_i) \notin Q_{\mathsf{KeyGen}}$ **then**
  return $\perp$
else
  $Q_{\mathsf{Reveal}} \coloneqq Q_{\mathsf{Reveal}} \cup \{\mathbb{A}_i\}$
  return $sk_{\mathbb{A}_i}$

$\underline{\mathrm{Exp}^{\mathsf{UF}}_{\mathcal{A},\mathsf{ABS}}(\lambda)}$

$Q_{\mathsf{KeyGen}}, Q_{\mathsf{Sign}}, Q_{\mathsf{Reveal}} := \emptyset$

$(pp, msk) \leftarrow \mathsf{Setup}(1^\lambda)$

$(m^*, \mathbb{P}^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{KeyGen}}_{pp,msk}, \mathcal{O}^{\mathsf{Reveal}}, \mathcal{O}^{\mathsf{Sign}}_{pp}}(pp)$

return 1 **if**

$\mathsf{Verify}(pp, m^*, \mathbb{P}^*, \sigma^*) = 1 \wedge$

$(m^*, \mathbb{P}^*) \notin Q_{\mathsf{Sign}} \wedge$

$\forall \mathbb{A}_i \in Q_{\mathsf{Reveal}} : \mathbb{P}^*(\mathbb{A}_i) = 0$

else return 0

$\underline{\mathrm{Exp}^{\mathsf{UF+Sim}}_{\mathcal{A},\mathsf{ABS}}(\lambda)}$

$Q_{\mathsf{KeyGen}}, Q_{\mathsf{Sign}}, Q_{\mathsf{Reveal}} := \emptyset$

$(pp, msk, td_{sim}) \leftarrow \mathsf{SimSetup}(1^\lambda)$

$(m^*, \mathbb{P}^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{KeyGen}}_{pp,msk}, \mathcal{O}^{\mathsf{Reveal}}, \mathcal{O}^{\mathsf{Sign}}_{pp,msk,td_{sim}}}(pp)$

return 1 **if**

$\mathsf{Verify}(pp, m^*, \mathbb{P}^*, \sigma^*) = 1 \wedge$

$(m^*, \mathbb{P}^*) \notin Q_{\mathsf{Sign}} \wedge$

$\forall \mathbb{A}_i \in Q_{\mathsf{Reveal}} : \mathbb{P}^*(\mathbb{A}_i) = 0$

else return 0

$\underline{\mathrm{Exp}^{\mathsf{UF+Sim+Ext}}_{\mathcal{A},\mathsf{Extr}_{\mathcal{A}},\mathsf{ABS}}(\lambda)}$

$Q_{\mathsf{KeyGen}}, Q_{\mathsf{Sign}}, Q_{\mathsf{Reveal}} := \emptyset$

$(pp, msk, td_{sim}) \leftarrow \mathsf{SimSetup}(1^\lambda)$

$(m^*, \mathbb{P}^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{KeyGen}}_{pp,msk}, \mathcal{O}^{\mathsf{Reveal}}, \mathcal{O}^{\mathsf{Sign}}_{pp,msk,td_{sim}}}(pp)$

$(\mathbb{A}^*) \leftarrow \mathsf{Extr}_{\mathcal{A}}(trans_{\mathcal{A}})$

return 1 **if**

$\mathsf{Verify}(pp, m^*, \mathbb{P}^*, \sigma^*) = 1 \wedge$

$(m^*, \mathbb{P}^*) \notin Q_{\mathsf{Sign}} \wedge$

$(\mathbb{A}^* \notin Q_{\mathsf{Reveal}} \vee \mathbb{P}^*(\mathbb{A}^*) = 0)$

else return 0

Experiment 3.7: Experiments for the proof of Theorem 3.4.1.

Oracles in $\mathrm{Exp}^{\mathsf{UF}}$. The unforgeability experiment has additionally a key generation and signing oracle using algorithms KeyGen and Sign respectively.

$\underline{\mathcal{O}^{\mathsf{KeyGen}}_{pp,msk}(\mathbb{A}_i) :}$

$sk_{\mathbb{A}_i} \leftarrow \mathsf{KeyGen}(pp, msk, \mathbb{A}_i)$

$Q_{\mathsf{KeyGen}} := Q_{\mathsf{KeyGen}} \cup \{(i, sk_{\mathbb{A}_i}, \mathbb{A}_i)\}$

$\underline{\mathcal{O}^{\mathsf{Sign}}_{pp}(i, m_j, \mathbb{P}_j) :}$

if $(i, sk_{\mathbb{A}_i}, \mathbb{A}_i) \notin Q_{\mathsf{KeyGen}}$ **then**

ignore

$\sigma \leftarrow \mathsf{Sign}(pp, sk_{\mathbb{A}_i}, m_j, \mathbb{P}_j)$

$Q_{\mathsf{Sign}} := Q_{\mathsf{Sign}} \cup \{(m_j, \mathbb{P}_j)\}$

return σ

Oracles in $\mathrm{Exp}^{\mathsf{UF+Sim}}$ and $\mathrm{Exp}^{\mathsf{UF+Sim+Ext}}$. The last two experiments share the key generation and signing oracle. The change here is that they use algorithms SimKeyGen and SimSign respectively.

$\mathcal{O}^{\mathsf{KeyGen}}_{pp,msk}(\mathbb{A}_i)$:

$sk_{\mathbb{A}_i} \leftarrow \mathsf{SimKeyGen}(pp, msk, td_{sim}, \mathbb{A}_i)$

$Q_{\mathsf{KeyGen}} \coloneqq Q_{\mathsf{KeyGen}} \cup \{(i, sk_{\mathbb{A}_i}, \mathbb{A}_i)\}$

$\mathcal{O}^{\mathsf{Sign}}_{pp,msk,td_{sim}}(i, m_j, \mathbb{P}_j)$:

if $(i, sk_{\mathbb{A}_i}, \mathbb{A}_i) \notin Q_{\mathsf{KeyGen}}$ **then**
  ignore
if $\mathbb{P}_j(\mathbb{A}_i) = 0$ **then**
  return $\perp$
else
  $\sigma \leftarrow \mathsf{SimSign}(pp, msk, td_{sim}, m_j, \mathbb{P}_j)$
  $Q_{\mathsf{Sign}} \coloneqq Q_{\mathsf{Sign}} \cup \{(m_j, \mathbb{P}_j)\}$
return σ

Note, that the two simulation oracles $\mathcal{O}^{\mathsf{KeyGen}}$ and $\mathcal{O}^{\mathsf{Sign}}$ in the experiments $\mathrm{Exp}^{\mathsf{UF+Sim}}$ and $\mathrm{Exp}^{\mathsf{UF+Sim+Ext}}$ differ compared to the corresponding oracles in the simulation-extractability experiment $\mathrm{Exp}^{\mathsf{wb\text{-}sim\text{-}ext}}$ (Definition 3.2.7). Oracle $\mathcal{O}^{\mathsf{KeyGen}}$ uses $\mathsf{SimKeyGen}$, but it does not output the generated secret key, where in simulation-extractability it does. Further, $\mathcal{O}^{\mathsf{Sign}}$ only outputs a simulated signature, if the attributes satisfy the policy ($\mathbb{P}(\mathbb{A}) = 1$). This is necessary, since $\mathsf{SimSign}$, unlike Sign, simulates signatures even for $\mathbb{P}(\mathbb{A}) = 0$. Hence, without this check ($\mathbb{P}(\mathbb{A}) = 1$) the experiment UF and $\mathsf{UF} + \mathsf{Sim}$ are easy to distinguish. An adversary only has to query a signature with ($\mathbb{P}(\mathbb{A}) = 0$) and it would get different answers in the experiments. Note, that simulation privacy (Definition 3.2.5) requires from a scheme that Sign outputs $\perp$ for inputs where $\mathbb{P}(\mathbb{A}) = 0$ holds.

1. UF to $\mathsf{UF} + \mathsf{Sim}$. Suppose adversary $\mathcal{A}$ distinguishes the two experiments $\mathrm{Exp}^{\mathsf{UF}}$ and $\mathrm{Exp}^{\mathsf{UF+Sim}}$ with advantage

$$\Big|\Pr\Big[\mathrm{Exp}^{\mathsf{UF}}_{\mathcal{A},\mathsf{ABS}}(\lambda) = 1\Big] - \Pr\Big[\mathrm{Exp}^{\mathsf{UF+Sim}}_{\mathcal{A},\mathsf{ABS}}(\lambda) = 1\Big]\Big|.$$

Using adversary $\mathcal{A}$ we build an adversary $\mathcal{B}$ against the simulation privacy of ABS. Adversary $\mathcal{B}$ has access to two oracles $\mathcal{O}^{\mathsf{KeyGen}_b}$ and $\mathcal{O}^{\mathsf{Sign}}$. Depending on the setting ($b \in \{0, 1\}$) the oracles use the real ($b = 1$) or simulation algorithms ($b = 0$) in $\mathrm{Exp}^{\mathsf{sim\text{-}privacy},b}_{\mathcal{B},\mathsf{ABS}}(\lambda)$ (Definition 3.2.5). The adversary $\mathcal{A}$ expects a sign, key generation, and reveal oracle. $\mathcal{A}$'s signature queries are directly mapped by $\mathcal{B}$ to its signing oracle. $\mathcal{B}$ in its simulation privacy experiment has no reveal oracle, since its key generation oracle directly outputs the generated secret keys. Therefore, $\mathcal{B}$ stores them internally on key generation queries by $\mathcal{A}$ and outputs the key on a corresponding reveal query by $\mathcal{A}$.

Next, we construct $\mathcal{B}$ in $\mathrm{Exp}^{\mathsf{sim\text{-}privacy},b}_{\mathcal{B},\mathsf{ABS}}(\lambda)$ using $\mathcal{A}$.

1. $\mathcal{B}$ receives input (pp, msk), runs $\mathcal{A}(pp)$
2. $\mathcal{B}$ answers oracle queries by $\mathcal{A}$ as follows:
   $\mathcal{O}^{\mathsf{KeyGen}}_{pp,msk}(\mathbb{A}_i)$: calls own oracle $sk_{\mathbb{A}_i} \leftarrow \mathcal{O}^{\mathsf{KeyGen}_b}(\mathbb{A}_i)$ and adds $(i, sk_{\mathbb{A}_i})$ to Q_{KG} for the i-th query.

$\mathcal{O}^{\mathsf{Reveal}}(i)$: returns $sk_{\mathbb{A}_i}$, if $(i, sk_{\mathbb{A}_i}) \in Q_{\mathsf{KG}}$ and adds $\mathbb{A}_i$ to Q_{Rev}, otherwise returns $\perp$.

$\mathcal{O}^{\mathsf{Sign}}(i, m_j, \mathbb{P}_j)$: returns $\sigma \leftarrow \mathcal{O}^{\mathsf{Sign}_b}(i, m_j, \mathbb{P}_j)$ and adds $(m_j, \mathbb{P}_j)$ to Q_{S}.

3. On $\mathcal{A}$'s output $(m^*, \mathbb{P}^*, \sigma^*)$, $\mathcal{B}$ outputs 1, if $\mathcal{A}$'s winning conditions hold:
   - $\mathsf{Verify}(pp, m^*, \mathbb{P}^*, \sigma^*) = 1 \wedge$
   - $(m^*, \mathbb{P}^*) \notin Q_{\mathsf{S}} \wedge$
   - $\forall \mathbb{A}_i \in Q_{\mathsf{Rev}} : \mathbb{P}^*(\mathbb{A}_i) = 0$

   and 0 otherwise.

Before analyzing $\mathcal{B}$'s advantage, we note that $\mathcal{B}$'s running time is polynomial in the security parameter, since it only runs ppt adversary $\mathcal{A}$ and has to store polynomial many queries of $\mathcal{A}$.

The main part of the analysis of adversary $\mathcal{B}$ is to relate the settings of $\mathcal{B}$ in $\mathrm{Exp}^{\mathsf{sim\text{-}privacy},b}_{\mathcal{B},\mathsf{ABS}}(\lambda)$ to the experiments $\mathrm{Exp}^{\mathsf{UF}}_{\mathcal{A},\mathsf{ABS}}(\lambda)$ and $\mathrm{Exp}^{\mathsf{UF+Sim}}_{\mathcal{A},\mathsf{ABS}}(\lambda)$ of adversary $\mathcal{A}$. Adversary $\mathcal{B}$ has access to the simulation setting with the simulation algorithms $\mathsf{SimSetup}$, $\mathsf{SimKeyGen}$, and $\mathsf{SimSign}$, if $b = 1$ holds and to the real setting $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign})$, if $b = 0$ holds. In case $b = 0$, the oracle queries by $\mathcal{A}$ are answered by $\mathcal{B}$'s oracles, that use the real algorithms KeyGen and Sign as in experiment $\mathrm{Exp}^{\mathsf{UF}}$. In case $b = 1$, $\mathcal{A}$'s oracle queries are answered by $\mathcal{B}$'s oracles, that use the simulation algorithms $\mathsf{SimKeyGen}$ and $\mathsf{SimSign}$ as in experiment $\mathrm{Exp}^{\mathsf{UF+Sim}}$. In both cases, $\mathcal{B}$ simulates the experiments for $\mathcal{A}$ perfectly. Hence, we conclude that

$$\begin{aligned}\mathrm{Adv}^{\mathsf{sim\text{-}privacy}}_{\mathcal{B},\mathsf{ABS}}(\lambda) &= \left|\Pr\left[\mathrm{Exp}^{\mathsf{sim\text{-}privacy},1}_{\mathcal{B},\mathsf{ABS}}(\lambda) = 1\right] - \Pr\left[\mathrm{Exp}^{\mathsf{sim\text{-}privacy},0}_{\mathcal{B},\mathsf{ABS}}(\lambda) = 1\right]\right| \\ &= \left|\Pr\left[\mathrm{Exp}^{\mathsf{UF}}_{\mathcal{A},\mathsf{ABS}}(\lambda) = 1\right] - \Pr[\mathrm{Exp}^{\mathsf{UF+Sim}}_{\mathcal{A},\mathsf{ABS}}(\lambda) = 1]\right| \ .\end{aligned}$$

2. $\mathsf{UF} + \mathsf{Sim}$ and $\mathsf{UF} + \mathsf{Sim} + \mathsf{Ext}$. In the second step, we construct from adversary $\mathcal{A}$ in $\mathrm{Exp}^{\mathsf{UF+Sim}}_{\mathcal{A},\mathsf{ABS}}(\lambda)$ an adversary $\mathcal{C}$ against $\mathrm{Exp}^{\mathsf{UF+Sim+Ext}}_{\mathcal{C},\mathsf{Extr}_{\mathcal{C}}\mathsf{ABS}}(\lambda)$. In other words we show $\mathsf{UF} + \mathsf{Sim} + \mathsf{Ext} \Rightarrow \mathsf{UF} + \mathsf{Sim}$. We conclude this in the third and final step, by showing that we construct an adversary against simulation-extractability (Definition 3.2.7) from adversary $\mathcal{C}$.

Back to step 2, observe that all the changes in the game $\mathsf{UF} + \mathsf{Sim} + \mathsf{Ext}$ occur after the final output of the adversary. Therefore, we will show that all winning outputs in $\mathsf{UF} + \mathsf{Sim}$ are also winning in $\mathsf{UF} + \mathsf{Sim} + \mathsf{Ext}$. Let us start with the description of adversary $\mathcal{C}$. The oracle queries of $\mathcal{A}$ in $\mathrm{Exp}^{\mathsf{UF+Sim}}$ are answered by $\mathcal{C}$'s oracles, since the experiments $\mathrm{Exp}^{\mathsf{UF+Sim}}$ and $\mathrm{Exp}^{\mathsf{UF+Sim+Ext}}$ share the same oracles. Next, we formally construct $\mathcal{C}$ in $\mathrm{Exp}^{\mathsf{UF+Sim+Ext}}_{\mathcal{C},\mathsf{Extr}_{\mathcal{C}},\mathsf{ABS}}(\lambda)$ from $\mathcal{A}$.

1. $\mathcal{C}$ receives input pp and runs $\mathcal{A}(pp)$.
2. $\mathcal{C}$ answers oracle queries by $\mathcal{A}$ as follows:

$\mathcal{O}^{\mathsf{KeyGen}}_{pp,msk}(\mathbb{A}_i)$: returns $sk_{\mathbb{A}_i} \leftarrow \mathcal{O}^{\mathsf{KeyGen}}_{pp,msk}(\mathbb{A}_i)$
$\mathcal{O}^{\mathsf{Reveal}}(i)$: returns $sk_{\mathbb{A}_i} \leftarrow \mathcal{O}^{\mathsf{Reveal}}(i)$.
$\mathcal{O}^{\mathsf{Sign}}(i, m_j, \mathbb{P}_j)$: returns $\sigma \leftarrow \mathcal{O}^{\mathsf{Sign}}(i, m_j, \mathbb{P}_j)$.

3. On $\mathcal{A}$'s output $(m^*, \mathbb{P}^*, \sigma^*)$, $\mathcal{C}$ outputs $(m^*, \mathbb{P}^*, \sigma^*)$.

$\mathcal{C}$ is efficient since it only runs the ppt adversary $\mathcal{A}$. It also simulates $\mathcal{A}$'s experiment $\mathrm{Exp}^{\mathsf{UF+Sim}}_{\mathcal{A},\mathsf{ABS}}(\lambda)$ perfectly, since $\mathcal{C}$ has access to the same oracles and just relays on $\mathcal{A}$'s queries. To analyze $\mathcal{C}$'s winning probability (experiment outputs 1) in experiment $\mathrm{Exp}^{\mathsf{UF+Sim+Ext}}_{\mathcal{C},\mathsf{Extr}_\mathcal{C},\mathsf{ABS}}$ we look at the winning condition of adversary $\mathcal{A}$ in $\mathrm{Exp}^{\mathsf{UF+Sim}}_{\mathcal{A},\mathsf{ABS}}$ and show what the condition means for adversary $\mathcal{C}$. Let us assume $\mathcal{A}$ wins $\mathrm{Exp}^{\mathsf{UF+Sim}}_{\mathcal{A},\mathsf{ABS}}$. Since the first two parts of the winning condition of both experiments are the same (see Experiment 3.7), we only look at the third part. For $\mathcal{A}$ in $\mathrm{Exp}^{\mathsf{UF+Sim}}_{\mathcal{A},\mathsf{ABS}}$ this is:

$(\mathcal{A}.3)$ $\forall \mathbb{A}_i \in Q_{\mathsf{Reveal}} : \mathbb{P}^*(\mathbb{A}_i) = 0$

and for $\mathcal{C}$ in $\mathrm{Exp}^{\mathsf{UF+Sim+Ext}}_{\mathcal{C},\mathsf{Extr}_\mathcal{C},\mathsf{ABS}}(\lambda)$ this is:

$(\mathcal{C}.3)$ $(\mathbb{A}^* \notin Q_{\mathsf{Reveal}} \vee \mathbb{P}^*(\mathbb{A}^*) = 0)$

We assume that extractor $\mathsf{Extr}_\mathcal{C}$ for $\mathcal{C}$ exists for the following analysis. We show how to construct it in the next proof step. Let $(\mathbb{A}^*) \leftarrow \mathsf{Extr}_\mathcal{C}(msk, m^*, \mathbb{P}^*, \sigma^*)$. From $(\mathcal{A}.3)$ we know that all revealed attributes do not satisfy the policy. So, if $\mathbb{A}^* \in Q_{\mathsf{Reveal}}$ it holds that $\mathbb{P}^*(\mathbb{A}^*) = 0$ and the second part of the disjunction in $(\mathcal{C}.3)$ is fulfilled. Thus $\mathcal{C}$ wins. Otherwise, $\mathbb{A}^* \notin Q_{\mathsf{Reveal}}$ holds, then $\mathcal{C}$ also wins, because of the first part of the disjunction in $(\mathcal{C}.3)$.

There is also a special case in which $\mathcal{A}$ loses, but $\mathcal{C}$ wins. This is expected, since the extraction allows for a more specific third winning condition. In detail, assume the first two parts of the winning condition hold and let us consider the following set

$$\mathsf{Attr} = \{\mathbb{A} \in Q_{\mathsf{Reveal}} \mid \mathbb{P}^*(\mathbb{A}) = 1 \wedge \mathbb{A} \neq \mathbb{A}^*\} \quad .$$

Attr contains all revealed attribute vectors that satisfy the output policy and are not the extracted attribute $\mathbb{A}^*$. If Attr is not the empty set, adversary $\mathcal{A}$ loses, because of $(\mathcal{A}.3)$. This means there is one revealed attribute vector that satisfies the output policy $\mathbb{P}^*$ and therefore the best that we can do in $\mathrm{Exp}^{\mathsf{UF+Sim}}_{\mathcal{A},\mathsf{ABS}}$, without extraction, is to declare this as a loss. Without extraction, we cannot check, if such a revealed attribute vector was really used to generate the claimed forgery. For $\mathcal{C}$ with extraction in $\mathrm{Exp}^{\mathsf{UF+Sim+Ext}}_{\mathcal{C},\mathsf{Extr}_\mathcal{C},\mathsf{ABS}}$ there are two cases where $\mathcal{C}$ then wins:

- $\mathbb{A}^* \in Q_{\mathsf{Reveal}} \wedge \mathbb{P}^*(\mathbb{A}^*) = 0$. Here, $\mathcal{C}$ wins because of the second part of the conjunction in $(\mathcal{C}.3)$.
- $\mathbb{A}^* \notin Q_{\mathsf{Reveal}} \wedge (\mathbb{P}^*(\mathbb{A}^*) = 0 \vee \mathbb{P}^*(\mathbb{A}^*) = 1)$. Here, $\mathcal{C}$ wins because of the first part of the conjunction in $(\mathcal{C}.3)$.

Overall, we see that ($\mathcal{C}$.3) can still hold even if there is a revealed attribute that satisfies the output policy $\mathbb{P}^*$.

Consequently, it holds that

$$\Pr\left[\mathrm{Exp}^{\mathsf{UF+Sim}}_{\mathcal{A},\mathsf{ABS}}(\lambda)=1\right] \leq \Pr\left[\mathrm{Exp}^{\mathsf{UF+Sim+Ext}}_{\mathcal{C},\mathsf{Extr}_{\mathcal{C}},\mathsf{ABS}}(\lambda)=1\right]$$

where $\leq$ comes from the fact that $\mathcal{C}$ wins in the above described cases where $\mathcal{A}$ loses.

3. $\mathsf{UF}+\mathsf{Sim}+\mathsf{Ext}$ and sim-ext. In the last step we finish the proof by showing that from adversary $\mathcal{C}$ in $\mathrm{Exp}^{\mathsf{UF+Sim+Ext}}_{\mathcal{C},\mathsf{Extr}_{\mathcal{C}},\mathsf{ABS}}$ we can construct an adversary $\mathcal{D}$ against the simulation-extractability of ABS in the experiment $\mathrm{Exp}^{\mathsf{wb\text{-}sim\text{-}ext}}_{\mathcal{D},\mathsf{Extr}_{\mathcal{D}},\mathsf{ABS}}(\lambda)$ (Definition 3.2.7). Looking at the definitions of the two experiments $\mathrm{Exp}^{\mathsf{UF+Sim+Ext}}$ and $\mathrm{Exp}^{\mathsf{wb\text{-}sim\text{-}ext}}$, explicitly their oracles, we see that in experiment $\mathrm{Exp}^{\mathsf{UF+Sim+Ext}}$ there are separate key generation and reveal oracles. Here, the key generation oracle only generates keys and the reveal oracle returns them to the adversary. In $\mathrm{Exp}^{\mathsf{wb\text{-}sim\text{-}ext}}$ we have a key generation oracle that generates and *returns* the generated key. This makes a separate reveal oracle obsolete.

We can of course simulate this behavior in adversary $\mathcal{D}$. On i-th key generation query by adversary $\mathcal{C}$ for attribute $\mathbb{A}_i$, $\mathcal{D}$ stores the query $(i, \mathbb{A}_i)$. Then on a reveal query for i, $\mathcal{D}$ queries its own oracle $sk_{\mathbb{A}_i} \leftarrow \mathcal{O}^{\mathsf{SimKeyGen}}(\mathbb{A}_i)$ and outputs secret key $sk_{\mathbb{A}_i}$ if a stored query $(i, \mathbb{A}_i)$ exists, otherwise it returns $\perp$. For subsequent reveal queries, $\mathcal{D}$ stores already revealed keys. For the signing oracle, we also have to consider that in $\mathrm{Exp}^{\mathsf{UF+Sim+Ext}}$ the signing oracle only simulates signatures for attributes that satisfy the policy, where in $\mathrm{Exp}^{\mathsf{wb\text{-}sim\text{-}ext}}$ the simulation signing oracle does it regardless. In fact, in $\mathrm{Exp}^{\mathsf{wb\text{-}sim\text{-}ext}}$ the attributes are not an input to the simulation signing oracle. Next, we construct $\mathcal{D}$ in $\mathrm{Exp}^{\mathsf{wb\text{-}sim\text{-}ext}}_{\mathcal{D},\mathsf{Extr}_{\mathcal{D}},\mathsf{ABS}}(\lambda)$ from $\mathcal{C}$.

1. $\mathcal{D}$ receives input pp and runs $\mathcal{C}(pp)$.

2. $\mathcal{D}$ answers oracle queries by $\mathcal{C}$ as follows:
   $\mathcal{O}^{\mathsf{KeyGen}}_{pp,msk}(\mathbb{A}_i)$: adds $(i, \perp, \mathbb{A}_i)$ to Q_{KG} for the i-th query.
   $\mathcal{O}^{\mathsf{Reveal}}(i)$:
   - if $(i, \perp, \mathbb{A}_i) \in Q_{\mathsf{KG}}$ returns $sk_{\mathbb{A}_i} \leftarrow \mathcal{O}^{\mathsf{SimKeyGen}}_{pp,msk,td_{sim}}(\mathbb{A}_i)$ and changes entry to $(i, sk_{\mathbb{A}_i}, \mathbb{A}_i)$,
   - else if $(i, sk_{\mathbb{A}_i}, \mathbb{A}_i) \in Q_{\mathsf{KG}}$ returns $sk_{\mathbb{A}_i}$, else returns $\perp$.

   $\mathcal{O}^{\mathsf{Sign}}(i, m_j, \mathbb{P}_j)$:
   - if $(i, sk_{\mathbb{A}_i} \neq \perp, \mathbb{A}_i) \in Q_{\mathsf{KG}}$ and $\mathbb{P}_j(\mathbb{A}_i) = 1$, returns signature $\sigma \leftarrow \mathcal{O}^{\mathsf{SimSign}}_{pp,msk,td_{sim}}(m_j, \mathbb{P}_j)$,
   - Else, if $(i, \perp, \mathbb{A}_i) \in Q_{\mathsf{KG}}$ and $\mathbb{P}_j(\mathbb{A}_i) = 1$, runs $sk_{\mathbb{A}_i} \leftarrow \mathcal{O}^{\mathsf{SimKeyGen}}_{pp,msk,td_{sim}}(\mathbb{A}_i)$, changes entry to $(i, sk_{\mathbb{A}_i}, \mathbb{A}_i)$, and returns $\sigma \leftarrow \mathcal{O}^{\mathsf{SimSign}}_{pp,msk,td_{sim}}(m_j, \mathbb{P}_j)$.

3. On $\mathcal{C}$'s output $(m^*, \mathbb{P}^*, \sigma^*)$, $\mathcal{C}$ outputs $(m^*, \mathbb{P}^*, \sigma^*)$.

Regarding the efficiency of $\mathcal{D}$, it only runs ppt adversary $\mathcal{C}$ and does efficient operations on a polynomially sized set. $\mathcal{D}$ simulates the experiment for $\mathcal{C}$ perfectly, since, as we described above, the oracles behave exactly like in $\mathcal{C}$'s experiment $\mathrm{Exp}^{\mathsf{UF+Sim+Ext}}$. Let us next look at the winning condition of both adversaries $\mathcal{C}$ and $\mathcal{B}$ in their respective experiments $\mathrm{Exp}^{\mathsf{wb\text{-}sim\text{-}ext}}$ and $\mathrm{Exp}^{\mathsf{UF+Sim+Ext}}$. Note, the first two parts of the winning condition are the same in both experiments. Hence, we focus on the third part. For $\mathcal{C}$ in $\mathrm{Exp}^{\mathsf{UF+Sim+Ext}}$ this is:

$(\mathcal{C}.3)$ $(\mathbb{A}^* \notin Q_{\mathsf{Reveal}} \vee \mathbb{P}^*(\mathbb{A}^*) = 0)$

and for $\mathcal{D}$ in $\mathrm{Exp}^{\mathsf{wb\text{-}sim\text{-}ext}}$ this is:

$(\mathcal{D}.3)$ $(\mathbb{A}^* \notin Q_{\mathsf{KeyGen}} \vee \mathbb{P}^*(\mathbb{A}^*) = 0)$

The change in $(\mathcal{D}.3)$ compared to $(\mathcal{C}.3)$ is only due to the above described difference in the oracles, but represents the same behavior, i.e., the $\mathcal{C}$'s reveal oracle and $\mathcal{D}$'s key generation oracle output secret keys. To finish the proof let us construct an extractor $\mathsf{Extr}_{\mathcal{C}}$ for adversary $\mathcal{C}$. Since we assume that ABS is simulation-extractable, an extractor $\mathsf{Extr}_{\mathcal{B}}$ for adversary $\mathcal{D}$ in $\mathrm{Exp}^{\mathsf{wb\text{-}sim\text{-}ext}}$ gives us an extractor $\mathsf{Extr}_{\mathcal{C}}$ for adversary $\mathcal{C}$. Due to the fact that the experiments, besides the slight difference in the oracles, differ only after the final output of the adversaries. In detail, given a transcript of $\mathcal{C}$ (input, output, randomness) we can compute a transcript of $\mathcal{D}$ efficiently, i.e., we just have to interpret each new revel query by $\mathcal{C}$ as a key generation query for $\mathcal{D}$ and ignore repeated revel queries. Therefore, let T be a polynomial-time algorithm that computes $trans_{\mathcal{D}}$ given $trans_{\mathcal{C}}$ as described above. The extractor $\mathsf{Extr}_{\mathcal{C}}$ is then defined as follows.

$$\begin{array}{l}\underline{\mathsf{Extr}_{\mathcal{C}}(trans_{\mathcal{C}})}\\ trans_{\mathcal{D}} := \mathsf{T}(trans_{\mathcal{C}})\\ \textbf{return } \mathsf{Extr}_{\mathcal{D}}(trans_{\mathcal{D}})\end{array}$$

Overall, we conclude that

$$\mathrm{Adv}^{\mathsf{wb\text{-}sim\text{-}ext}}_{\mathcal{D},\mathsf{Extr}_{\mathcal{D}},\mathsf{ABS}}(\lambda) = \Pr\left[\mathrm{Exp}^{\mathsf{wb\text{-}sim\text{-}ext}}_{\mathcal{D},\mathsf{Extr}_{\mathcal{D}},\mathsf{ABS}}(\lambda) = 1\right] = \Pr\left[\mathrm{Exp}^{\mathsf{UF+Sim+Ext}}_{\mathcal{C},\mathsf{Extr}_{\mathcal{C}},\mathsf{ABS}}(\lambda) = 1\right].$$

This finishes the last step and collecting all of the above steps we get the following.

$$\begin{aligned}\Pr\left[\mathrm{Exp}^{\mathsf{UF}}_{\mathcal{A},\mathsf{ABS}}(\lambda) = 1\right] =\ & \Pr\left[\mathrm{Exp}^{\mathsf{UF}}_{\mathcal{A},\mathsf{ABS}}(\lambda) = 1\right] - \Pr\left[\mathrm{Exp}^{\mathsf{UF+Sim}}_{\mathcal{A},\mathsf{ABS}}(\lambda) = 1\right]\\ & + \Pr\left[\mathrm{Exp}^{\mathsf{UF+Sim}}_{\mathcal{A},\mathsf{ABS}}(\lambda) = 1\right]\\ \leq\ & \left|\Pr\left[\mathrm{Exp}^{\mathsf{UF}}_{\mathcal{A},\mathsf{ABS}}(\lambda) = 1\right] - \Pr\left[\mathrm{Exp}^{\mathsf{UF+Sim}}_{\mathcal{A},\mathsf{ABS}}(\lambda) = 1\right]\right|\\ & + \Pr\left[\mathrm{Exp}^{\mathsf{UF+Sim}}_{\mathcal{A},\mathsf{ABS}}(\lambda) = 1\right]\\ =& \mathrm{Adv}^{\mathsf{sim\text{-}privacy}}_{B,\mathsf{ABS}}(\lambda) + \Pr\left[\mathrm{Exp}^{\mathsf{UF+Sim}}_{\mathcal{A},\mathsf{ABS}}(\lambda) = 1\right]\end{aligned}$$

$$\leq \mathrm{Adv}^{\mathsf{sim\text{-}privacy}}_{B,\mathsf{ABS}}(\lambda) + \Pr\left[\mathrm{Exp}^{\mathsf{UF+Sim+Ext}}_{\mathcal{C},\mathsf{Extr}_{\mathcal{C}}\mathsf{ABS}}(\lambda) = 1\right]$$
$$= \mathrm{Adv}^{\mathsf{sim\text{-}privacy}}_{B,\mathsf{ABS}}(\lambda) + \mathrm{Adv}^{\mathsf{wb\text{-}sim\text{-}ext}}_{\mathcal{D},\mathsf{Extr}_{\mathcal{D}},\mathsf{ABS}}(\lambda)$$

□

We can adapt Theorem 3.4.1 and the above proof to black-box simulation-extractability of ABS (Definition 3.2.8), since we only relied on the white-box variant of the extractor to argue that an extractor exists for a specific adversary. In case of black-box simulation-extractability, we have one extractor for all adversaries that we use throughout the proof. Everything else stays the same. Note, this only simplifies the proof. Hence, the following theorem follows.

Theorem 3.4.2. *If an ABS scheme is perfect simulation private (Definition 3.2.5) and black-box simulation-extractable (Definition 3.2.8), then it is unforgeable (Definition 3.2.6).*

Considering the reverse direction of Theorems 3.4.1 and 3.4.3 that an unforgeable ABS scheme is simulation private and simulation-extractable we do not have a formal negative result. However, we have also not found a way to argue the existence of efficient simulators or an efficient extractor from unforgeability.

Next, we focus on the relation of the privacy definitions for ABS.

3.4.2 On the Privacy of ABS

In the following we analyze the relationship between our privacy and simulation privacy definitions. We first show that simulation privacy implies privacy. Then we show that the reverse direction is not true for all policy classes.

Theorem 3.4.3. *If an ABS scheme* ABS *is perfectly simulation private (Definition 3.2.5), then it is also perfectly private (Definition 3.2.4).*

Proof. Suppose that ABS is not perfectly private. Then, there exists a distinguisher $\mathcal{A}$ such that the advantage

$$\mathrm{Adv}^{\mathsf{privacy}}_{\mathcal{A},\mathsf{ABS}}(\lambda) = \Pr[\mathrm{Exp}^{\mathsf{privacy},1}_{\mathcal{A},\mathsf{ABS}}(\lambda) = 1] - \Pr[\mathrm{Exp}^{\mathsf{privacy},0}_{\mathcal{A},\mathsf{ABS}}(\lambda) = 1] > 0 \ .$$

We construct a distinguisher $\mathcal{D}$ for perfect simulation privacy using $\mathcal{A}$ as a black-box in the following. Distinguisher $\mathcal{D}$ in $\mathrm{Exp}^{\mathsf{sim\text{-}privacy},b}_{\mathcal{D},\mathsf{ABS}}(\lambda)$ works as follows.

1. $\mathcal{D}$ receives (pp, msk) and runs $\mathcal{A}(pp, msk)$.
2. $\mathcal{D}$ flips a coin $d \leftarrow \{0,1\}$.
3. $\mathcal{D}$ on the k-th oracle query $(m, \mathbb{P}, \mathbb{A}_0, \mathbb{A}_1)$ from $\mathcal{A}$ $(k > 0)$:
   a) It checks whether $\mathbb{P}(\mathbb{A}_0) = 1$ and $\mathbb{P}(\mathbb{A}_1) = 1$ hold, if not ignore query
   b) Get $sk_{\mathbb{A}_0} \leftarrow \mathcal{O}^{\mathsf{KeyGen}_b}(\mathbb{A}_0)$ and $sk_{\mathbb{A}_1} \leftarrow \mathcal{O}^{\mathsf{KeyGen}_b}(\mathbb{A}_1)$

c) Get $\sigma_d \leftarrow \mathcal{O}^{\mathsf{Sign}_b}(2k-1+d, m, \mathbb{P})$

d) Return $(\sigma_d, sk_{\mathbb{A}_0}, sk_{\mathbb{A}_1})$

4. Eventually $\mathcal{A}$ outputs d'.

5. $\mathcal{D}$ sets $b' := 1$ if $d = d'$, otherwise $\mathcal{D}$ sets $b' := 0$.

6. $\mathcal{D}$ outputs b'.

Let us analyze the advantage of distinguisher $\mathcal{D}$. First, we analyze the case in which distinguisher $\mathcal{D}$ is in the simulation privacy experiment $\mathrm{Exp}^{\mathsf{sim\text{-}privacy},1}_{\mathcal{D},\mathsf{ABS}}(\lambda)$ with $b = 1$. In this case, the experiment including the oracles use the normal algorithms $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign})$. Hence, for $d \in \{0,1\}$ distinguisher $\mathcal{D}$ simulates the experiment $\mathrm{Exp}^{\mathsf{privacy},d}_{\mathcal{A},\mathsf{ABS}}(\lambda)$ for adversary $\mathcal{A}$ perfectly. Therefore, we get the following.

$$\begin{aligned}
\Pr[\mathrm{Exp}^{\mathsf{sim\text{-}privacy},1}_{\mathcal{D},\mathsf{ABS}}(\lambda) = 1] &= \Pr[d' = 1 \mid d = 1] \cdot \Pr[d = 1] \\
&\quad + \Pr[d' = 0 \mid d = 0] \cdot \Pr[d = 0] \\
&= \frac{1}{2}(\Pr[d' = 1 \mid d = 1] - \Pr[d' = 1 \mid d = 0]) + \frac{1}{2} \\
&= \frac{1}{2}(\Pr[\mathrm{Exp}^{\mathsf{privacy},1}_{\mathcal{A},\mathsf{ABS}}(\lambda) = 1] \\
&\quad - \Pr[\mathrm{Exp}^{\mathsf{privacy},0}_{\mathcal{A},\mathsf{ABS}}(\lambda) = 1]) + \frac{1}{2} \\
&= \frac{1}{2}\mathrm{Adv}^{\mathsf{privacy}}_{\mathcal{A},\mathsf{ABS}}(\lambda) + \frac{1}{2}
\end{aligned}$$

Second, in the case $b = 0$ distinguisher $\mathcal{D}$ is in the simulation privacy experiment $\mathrm{Exp}^{\mathsf{sim\text{-}privacy},0}_{\mathcal{D},\mathsf{ABS}}(\lambda)$. Here, the signatures are generated independently from the bit d that $\mathcal{D}$ chooses. In detail, the signing oracle $\mathcal{O}^{\mathsf{Sign}_0}_{pp,msk,td_{sim}}$ generates signatures using the simulation signing algorithm $\mathsf{SimSign}$. Hence, signatures are generated independent of the secret key for the attributes $\mathbb{A}^*_d$. Consequently, the view of $\mathcal{A}$ is independent of d and $\Pr[\mathrm{Exp}^{\mathsf{sim\text{-}privacy},0}_{\mathcal{D},\mathsf{ABS}}(\lambda) = 1] = \Pr[d = d'] = \frac{1}{2}$. Overall, it holds that

$$\mathrm{Adv}^{\mathsf{sim\text{-}privacy}}_{\mathcal{D},\mathsf{ABS}}(\lambda) = \frac{1}{2} \cdot \mathrm{Adv}^{\mathsf{privacy}}_{\mathcal{A},\mathsf{ABS}}(\lambda).$$

□

Corollary 3.4.1. *If an ABS scheme* ABS *is computationally simulation private (Definition 3.2.5), then it is also computationally private (Definition 3.2.4).*

Corollary 3.4.1 follows from Theorem 3.4.3 since, if we only consider ppt adversaries as in the computational definitions, the reduction in the proof of Theorem 3.4.3 is efficient.

For policy-based signature (PBS) Bellare and Fuchsbauer [BF14] show without restricting the polices the reverse direction of Corollary 3.4.1 does not hold by

presenting a counter-example. In the following, we present a counter-example for ABS that shows that the reverse direction of Corollary 3.4.1 and Theorem 3.4.3 does not hold in general for ABS, i.e., standard privacy does not imply simulations privacy without restricting the policy class.

Counter-example. To show that simulation-based privacy is a stronger notion than privacy without restricting the policy class, we fix a specific policy class based on discrete logarithms (dlog) and present an ABS scheme for it as a counter-example. The ABS scheme is an adaptation of our generic ABS construction $\mathsf{ABS}^{\mathsf{gc}}$ (Section 3.3). Then, we show that the presented ABS scheme is private, but not simulation private. After that we generalize this to hard policy classes, i.e., where computing a satisfying attribute vector for a policy is infeasible for an ppt adversary. We formulate this in the following in a definition that, similar to our simulation privacy definition, requires that the setup of the setup algorithm Setup and of a simulation setup algorithm $\mathsf{SimSetup}$ is indistinguishable. Further, it captures that it should be infeasible for an ppt adversary to compute a satisfying attribute vector even if it gets access to the master secret key and simulation trapdoor.

Definition 3.4.1 (Hard Policy Class)
Let $\{\mathbb{P}\}_{pp}$ be a policy class of an ABS scheme ABS with public parameters pp. We call $\{\mathbb{P}\}_{pp}$ a *hard policy class*, if (1) there exists a ppt algorithm $\mathsf{SimSetup}$ such that for all ppt adversaries $\mathcal{A}'$ it holds that

$$\Big|\Pr\Big[(pp, msk) \leftarrow \mathsf{ABS.Setup}(1^\lambda)\colon \mathcal{A}'(pp, msk) = 1\Big] - \Pr\Big[(pp, msk, td_{sim}) \leftarrow \mathsf{ABS.SimSetup}(1^\lambda)\colon \mathcal{A}'(pp, msk) = 1\Big]\Big|$$

and (2) for all ppt adversaries $\mathcal{A}$ and all $\mathbb{P} \in \{\mathbb{P}\}_{pp}$ it holds that

$$\Pr[(pp, msk, td_{sim}) \leftarrow \mathsf{ABS.SimSetup}(1^\lambda), \mathbb{A} \leftarrow \mathcal{A}(\mathbb{P}, pp, msk, td_{sim}) : \mathbb{P}(\mathbb{A}) = 1] = \mathsf{negl}(\lambda).$$

◇

Let us start with an informal counter-example to outline the general idea. We consider our example ABS scheme that we used before, where the Sign algorithm on input attributes and a message-policy pair generates a signature and then appends to the signature the attributes that where used to generate it. Suppose a hard policy class. Then we have that the example scheme with this hard policy class is not perfectly simulation private. In detail, assume there is a ppt algorithm $\mathsf{SimSign}$ which output is indistinguishable from the output of algorithm Sign, i.e., $\mathsf{SimSign}$ computes a satisfying attribute vector for a policy and appends it to the signature given just the master secret key, the simulation trapdoor, and a message-policy pair. This algorithm $\mathsf{SimSign}$ is then an adversary for the hard policy class that

computes a satisfying attribute vector with non-negligible probability. Hence, by assumption that we have an ABS scheme for a hard policy class the described algorithm $\mathsf{SimSign}$ cannot exist.

For the concrete counter-example we use a policy class based on the discrete logarithm (dlog) of an element. In detail, a policy $\mathbb{P}$ in the dlog policy class specifies a group element Y of a prime order group $(\mathbb{G}_1, p, g_1)$ and the unique satisfying attribute for it is then $y \in \mathbb{Z}_p$, i.e., $g_1^y = Y \in \mathbb{G}_1$. Computing a policy with a satisfying attribute is easy. Pick its dlog $y \leftarrow \mathbb{Z}_p$, set $\mathbb{P} := Y = g_1^y \in \mathbb{G}_1$. However, under the dlog assumption (Definition 2.2.6) given only $\mathbb{P}$ (and the group description) it is hard to compute $y \in \mathbb{Z}_p$, i.e., there is no ppt algorithm that computes y. The main arguments that we need for the following counter-example are that the satisfying attribute is a unique element in $\mathbb{Z}_p$ and that under the dlog assumption computing the element is hard.

We base our counter-example scheme, called $\mathsf{ABS}^{\mathsf{ce}}$, on our generic scheme $\mathsf{ABS}^{\mathsf{gc}}$. For this we adapt the ABS relation $R_{\mathsf{Sig}}^{\mathrm{ABS}}$ (Definition 3.3.1) and then present $\mathsf{ABS}^{\mathsf{ce}}$ in the following. In detail, we adapt the policy part of the $R_{\mathsf{Sig}}^{\mathrm{ABS}}$ to the dlog policy class and call it $R_{\mathsf{Sig}}^{\text{ABS-dlog}}$. Let $\mathbb{P}(\mathbb{A}) = 1 \Leftrightarrow Y = g_1^y \in \mathbb{G}_1$ where policy $\mathbb{P} := Y \in \mathbb{G}_1$ and attributes $\mathbb{A} := y \in \mathbb{Z}_p$. Hence, we work with a signature of knowledge for the statement $\mathsf{Sig.Verify}(pk, \theta, \mathbb{A}) = 1 \wedge Y = g_1^y \in \mathbb{G}_1$, meaning that a signer has a valid key θ on attribute $\mathbb{A} = y \in \mathbb{Z}_p$ and the attribute is the dlog of the policy $\mathbb{P} = Y \in \mathbb{G}_1$. Next, we present our counter-example scheme $\mathsf{ABS}^{\mathsf{ce}}$ based on $\mathsf{ABS}^{\mathsf{gc}}$. For $\mathsf{ABS}^{\mathsf{ce}}$ we only alter the signing algorithm $\mathsf{ABS}^{\mathsf{gc}}.\mathsf{Sign}$, such that it additionally outputs the attribute vector next to the signature and ignore, for now, the $\mathsf{ABS}^{\mathsf{gc}}.\mathsf{SimSign}$ algorithm. We call this altered algorithm Sign'.

Example 3.4.1: Let SoK be a signature of knowledge with respect to a relation generator for $R_{\mathsf{Sig}}^{\text{ABS-dlog}}$ and let Sig be a signature scheme with message space $\mathbb{Z}_p$. We define an ABS scheme $\mathsf{ABS}^{\mathsf{ce}}$ based on $\mathsf{ABS}^{\mathsf{gc}}$ (Construction 3.3.1) with signing algorithm $\mathsf{ABS}^{\mathsf{ce}}.\mathsf{Sign}'$ instead of $\mathsf{ABS}^{\mathsf{gc}}.\mathsf{Sign}$.

$\mathsf{ABS}^{\mathsf{ce}}.\mathsf{Sign}'(pp, sk_{\mathbb{A}}, m, \mathbb{P})$

1 : **parse** $pp = (pp_{\mathsf{SoK}}, pk), sk_{\mathbb{A}} = (\mathbb{A}, \theta)$
2 : $\sigma \leftarrow \mathsf{ABS}^{\mathsf{gc}}.\mathsf{Sign}(pp, sk_{\mathbb{A}}, m, \mathbb{P})$
3 : **if** $\sigma = \perp$ **then**
4 : **return** $\perp$
5 : **else**
6 : **return** $\sigma' = (\sigma, \mathbb{A})$

$\mathsf{ABS}^{\mathsf{ce}}$ is an ABS scheme for the dlog policy class, since attributes are elements from $\mathbb{Z}_p$ and SoK generates signatures of knowledge with respect to $R_{\mathsf{Sig}}^{\text{ABS-dlog}}$.

In the following, we start with showing that $\mathsf{ABS}^{\mathsf{ce}}$ is still private (Definition 3.2.4). However, subsequently we show that under the dlog assumption $\mathsf{ABS}^{\mathsf{ce}}$ is not simulation private (Definition 3.2.5).

Theorem 3.4.4. *If* $\mathsf{ABS}^{\mathsf{gc}}$ *(Construction 3.3.1) is perfectly simulation private (Definition 3.2.5), then* $\mathsf{ABS}^{\mathsf{ce}}$ *(Example 3.4.1) is perfectly private (Definition 3.2.4).*

Let us outline the proof of the theorem. Note, we have already shown that $\mathsf{ABS}^{\mathsf{gc}}$ is perfectly simulation private, see Theorem 3.3.1. Therefore, in the following we have to show that no adversary that adaptively queries signatures of his choice by providing policy $\mathbb{P}_{Y_i}$, two attribute vectors $(\mathbb{A}_0, \mathbb{A}_1)$, and a message m can distinguish whether the signatures in the privacy experiment are generated using $\mathbb{A}_0$ or $\mathbb{A}_1$. Such queries are answered by a signature under one of the attributes and two freshly generated secret keys for $\mathbb{A}_0$ and $\mathbb{A}_1$. See the experiment in Definition 3.2.4 for reference.

From the perfect simulation privacy of $\mathsf{ABS}^{\mathsf{gc}}$ we directly get that the actual signature σ of Sign''s output is simulatable without an attribute vector as an input. Hence, σ is independent of the attribute choices ($\mathbb{A}_0$ or $\mathbb{A}_1$) of the experiment in each signature query. It remains to show that Sign' outputting the attribute vector next to σ does not help the adversary to guess which attributes were used to generate the signatures, i.e., it does not change its advantage. For this we have to look at the oracle given to the adversary in the privacy experiment. There, signatures are only generated under the condition that $\mathbb{P}(\mathbb{A}_0) = 1$, $\mathbb{P}(\mathbb{A}_1) = 1$, and $\mathbb{P} \in \mathcal{U}_{pp}$ hold, where the attributes are given by the adversary. Translated to dlog policies this means, that the policy $\mathbb{P}$ defines a group element Y, such that $\mathbb{A}_0$ and $\mathbb{A}_1$ are the discrete logarithm of this group element Y with respect to g_1, given in the public parameters. Since the discrete logarithm is unique (in $\mathbb{Z}_p$) the adversary, for valid queries, is forced to input two attributes that are equal, $\mathbb{A}_0 = \mathbb{A}_1 = \log_{g_1}(Y)$. Hence, additionally outputting the used attribute does not give the adversary any information, that he did not have before the query, regardless of the adversary's setting ($b = 0$, the oracle outputs signatures under $\mathbb{A}_0$, or $b = 1$, the oracle outputs signatures under $\mathbb{A}_1$).

Note, this is not specific to dlog policies, since we only use the fact that for any policy the satisfying attributes are unique. In general, this means if the policy class includes only policies with unique satisfying attributes, then appending attributes to signatures does not clash with the privacy notion.

The formal proof is skipped here, since it is the same as the one for Theorem 3.4.3 (simulation privacy implies standard privacy) with an additional argument, that Sign' additionally outputting the attribute vector does not affect the adversary's advantage. As detailed above, the argument is that the satisfying attributes for a policy are unique. Therefore the two attribute vectors given by the adversary in valid queries have to be equal, i.e., the unique attributes which are already known to the adversary before the query.

If we consider policy classes that only include policies with at least two distinct satisfying attribute vectors, the above argument does not work. In this case, an

adversary can query the signing oracle with two distinct attribute vectors $\mathbb{A}_0$ and $\mathbb{A}_1$ and a policy that both attribute vectors satisfy. Then, any signing algorithm that appends the used attribute vector to signatures leaks the challenge bit of the privacy experiment to the adversary. Hence, on output $(\sigma, \mathbb{A}_b)$ an adversary can just output b and wins the privacy experiment.

Next, we present that without restricting the policy class to policies where computing satisfying attributes is efficient, not every private ABS scheme is also simulation private.

Theorem 3.4.5. *Under the discrete logarithm assumption (Definition 2.2.6) with respect to the group generated by the relation generator for* $R_{\mathsf{Sig}}^{\text{ABS-dlog}}$, $\mathsf{ABS}^{\mathsf{ce}}$ *(Example 3.4.1) with the dlog policy class is not computational simulation private (Definition 3.2.5).*

Let us assume that $\mathsf{ABS}^{\mathsf{ce}}$ is (computational) simulation private, then there exist ppt algorithms $\mathsf{SimSetup}$, $\mathsf{SimKeyGen}$, and $\mathsf{SimSign}$ such that no ppt adversary can distinguish whether it is interacting with the normal or the simulation algorithms (cf. Definition 3.2.5). Hence, $\mathsf{SimSign}$ is a ppt that on input $(pp, msk, td_{sim}, m, \mathbb{P})$ outputs simulated signatures $\sigma' = (\sigma, \mathbb{A})$ that are indistinguishable from signatures produced by $\mathsf{Sign}'(pp, sk_{\mathbb{A}}, m, \mathbb{P})$ for $\mathbb{P}(\mathbb{A}) = 1$. Outputting the attribute $\mathbb{A}$, that was used to create the signature, is easy for Sign', since it gets the secret key for $\mathbb{A}$ as input, but $\mathsf{SimSign}$ does not. This means, that $\mathsf{SimSign}$ outputs the discrete logarithm $\mathbb{A}$ of a given group element $\mathbb{P}$ in polynomial-time. In the following we outline the proof details. Given a dlog instance $(\mathbb{G}, p, g, u, h)$ with the challenge to compute $\log_u(h)$ as described in the dlog experiment (Definition 2.2.6), we generate the rest of the parameters of the $\mathsf{ABS}^{\mathsf{ce}}$ accordingly. Then, we run $\mathsf{SimSign}$ on policy h (and an arbitrary message) and output the attribute that $\mathsf{SimSign}$ returns in its signature as the dlog of h. Note, $\mathsf{SimSign}$ only fails with negligible probability, since we assumed that $\mathsf{ABS}^{\mathsf{ce}}$ is computational simulation private.

Theorem 3.4.5 can also be generalized to hard policy classes. We used dlog as an illustrative example, since the dlog assumption is simple and essential in many cryptographic schemes. Note, if we only consider policy classes that are not hard, i.e., with policies where satisfying attributes are efficiently computable, called easy policy classes in the remainder, a simulation signing algorithm can just compute satisfying attributes in polynomial-time, e.g., ABS schemes that support monotone policies. Then, the simulation signing algorithm can just generate a secret key on the satisfying attributes (using msk) and output a signature under the secret key by running the normal signing algorithm. We formalize this in the following theorem.

Theorem 3.4.6. *If an ABS scheme* ABS *is computational private (Definition 3.2.4) and defined for an easy policy class, then* ABS *is also computational simulation private (Definition 3.2.5).*

Proof. Let $\mathsf{ABS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$. We have to define ppt simulation algorithms $\mathsf{SimSetup}$, $\mathsf{SimKeyGen}$, and $\mathsf{SimSign}$ and show that no adversary can

distinguish them from the normal algorithms, except with negligible probability. We define SimSetup such that it sets $td_{sim} \coloneqq \perp$, runs Setup and outputs whatever Setup outputs. Also, we define SimKeyGen such that it runs KeyGen and outputs KeyGen's output. Further, we define SimSign as follows.

$\mathsf{SimSign}(pp, msk, td_{sim}, m, \mathbb{P})$

compute $\mathbb{A}$ such that $\mathbb{P}(\mathbb{A}) = 1$
$sk_{\mathbb{A}} \leftarrow \mathsf{KeyGen}(pp, msk, \mathbb{A})$
$\sigma \leftarrow \mathsf{Sign}(pp, sk_{\mathbb{A}}, m, \mathbb{P})$
return σ

The first step of the simulation signing algorithm SimSign is efficiently computable by the assumption that the policy class is easy. The remaining steps are also efficient since SimSign runs ppt algorithms KeyGen and Sign. SimSign only uses the master secret key msk, hence we set td_{sim} to $\perp$ in SimSetup. Further, SimSign generates signatures exactly as Sign does with a secret key $sk_{\mathbb{A}}$ as input. In conclusion, since the simulation algorithms use the normal algorithms to produce their output, i.e., the output distributions are identical, no adversary can distinguish an interaction with the normal algorithms from an interaction with the simulation algorithms. □

Let us summarize the relation of privacy and simulation privacy in the following. We showed that simulation privacy is a stronger notion than privacy for policy classes that include only policies with unique satisfying attributes where these satisfying attributes are not efficiently computable (hard policy classes). This might sound familiar. In fact it resembles the results with respect to witness indistinguishable (WI), witness hiding (WH), and zero-knowledge (ZK) argument systems [FS90]. As a reminder, in an argument system a prover wants to convince a verifier that a common input x is a valid instance of a relation R, i.e., there exists a witness w for x such that $(x, w) \in R$. ABS privacy can be compared to WI. WI can be trivially fulfilled by a prover that outputs the witness to the verifier, if the witness is unique. The simulatability of simulation privacy is similar to the one required by ZK. Intuitively, ZK requires a simulator that outputs proofs (without a witness as input) that are indistinguishable from real proofs (generated with a witness as input). To conclude, let us first repeat the results known for argument systems. As presented in [FS90] the relations between witness indistinguishable, witness hiding, and zero-knowledge protocol are as follows. Every zero-knowledge argument system is also witness indistinguishable, but not every witness indistinguishable argument system is witness hiding. The relation between zero-knowledge and witness hiding (zero-knowledge implies witness hiding) only holds under additional assumptions, i.e., the existence of a claw free function and a proper generator for instance and witness, cf. [FS90, Theorem 4.1]. Regarding ABS, our results above seems logical, since most ABS schemes are build from (non-interactive) witness indistinguishable, zero-knowledge argument systems or signatures of knowledge schemes based on NIZK argument systems.

This result and resemblance to argument systems are also a motivation to answer the question "how can ABS privacy be modeled ideally". In Chapter 4 we discuss this question and answer it with a universally composable ideal functionality for attribute-based signatures. Before this, we focus on existing ABS schemes from the literature and show that they are simulation private.

3.5 On the Security of Existing Schemes

We have seen that our new notion of simulation privacy for ABS is strictly stronger than the (standard) privacy notion, if we consider policy classes that include only policies with unique satisfying attributes and where these satisfying attributes are not efficiently computable. The goal of this section is to show that there are existing schemes that satisfy our simulation privacy definition. This result is not only interesting on its own, but also with respect to our universal composability result for ABS in Chapter 4. There, our UC result Theorem 4.5.1 shows that if and only if an ABS scheme is correct, consistent, unforgeable, and *simulation private* then the ABS scheme achieves UC security. Therefore, we show in this section that the generic ABS constructions of Maji et al. [MPR11] and Sakai et al. [SAH16] satisfy our perfect simulation privacy definition. Combined with our UC result (Theorem 4.5.1), this shows that the two generic ABS constructions achieve UC security. Since the two ABS constructions cover two generic constructions, we use them in this work to show how our result actually improves the state of the art security for ABS. For example, the construction in [SAH16] covers many common approaches in the literature to define ABS. In detail, it uses NIWI argument systems, collision-resistant hash functions, and SPS as building blocks. Both constructions were originally shown to satisfy correctness, consistency, unforgeability, and the (weaker) perfect privacy. As we have shown by our counter-example (Theorem 3.4.5), not every private ABS is simulation private. Therefore, we show that the two generic ABS constructions are perfectly simulation private by defining three simulation algorithms ($\mathsf{SimSetup}$, $\mathsf{SimKeyGen}$, $\mathsf{SimSign}$) for each construction. Then, we prove that they guarantee simulation privacy according to Definition 3.2.5. To define the simulation signing algorithms $\mathsf{SimSign}$ of the generic ABS constructions we only apply minor modifications to the original signing algorithms. In particular these modifications do not imply changes to the setup and key generation algorithms.

An interesting insight is that both constructions implicitly define the simulation signing algorithm in the first step of their unforgeability proof. In detail, the first proof step is to replace real signatures with simulated signatures. This typically prepares further proof steps that are then more scheme specific. Our literature research and analysis of other ABS schemes (and generic constructions) show that this first step is common and present in [AHY15; ECGD14; EG17; EGK14; Li+10; SKAH18; UKLC15] among others. In other cases this step is not present. However, if the schemes are build from simulatable argument systems, then signatures can be simulated via the zero-knowledge simulator, i.e., [DGM18; EE16; EK18; Gha15;

GM19; GM22]. Simulation privacy is not formalized but implicitly used in the works by Okamoto et al. [DOT19; OT13; OT14]. There, simulated signatures are used to show standard privacy directly. Hence, we expect that the techniques and insight used to define simulation algorithms in the following for the constructions by Sakai et al. [SAH16] and Maji et al. [MPR11] can be used for the other schemes.

Concretely, to formally define a simulation signing algorithm SimSign for the constructions, we exploit a similarity in the signing algorithms of both schemes. The signing algorithms fulfill two basic properties. First, the signature on a message-policy pair proves that the signer knows a valid secret key on attributes satisfying the policy. Second, it binds the policy to the signed message. In detail, in the constructions from [MPR11; SAH16] a signature on a message-policy pair $(m, \mathbb{P})$ of the signing algorithm Sign, proves that the signer knows a secret key for attributes satisfying the given policy $\mathbb{P}$ *or* that he knows a special signature on $(m, \mathbb{P})$ that one can only generate with access to the master secret key. We exploit the second part of the *or* to define the simulation signing algorithms for both constructions [MPR11; SAH16]. This so called (parallel) OR-proof technique originated from [CDS94; JSI96] and in general it allows a user to show that an original statement is true or that the user knows a value where actually the user does not know it. Only in the security reduction we can then argue that we know the value, because we alternate the setup accordingly, e.g., we generate a trapdoor. For example, Groth [Gro06] uses the technique to add simulatability to an extractable proof system, Bernhard et al. [BFG13] use it to construct a SoK in the standard model and Derler and Slamanig [DS19] use it together with key-homomorphic signatures to construct a SE-NIZK from a NIWI system. Furthermore, the technique is used to construct tightly-secure schemes [Bad+15; GJ18; HJ12]. There are also sequential OR-proofs [AOS02; FHJ20] and they are used to build signature schemes [AOS02; FHJ20] and tightly-secure signature schemes [DGJL21].

3.5.1 Generic ABS Construction by Sakai et al.

Next, we present simulation algorithms for the generic construction by Sakai, Attrapadung and Hanaoka [SAH16] and show that the construction is simulation private (Definition 3.2.5). Then, we discuss concrete instantiations of the generic construction and their efficiency originally presented by Bemmann [Bem17]. Here, the concrete instantiations also highlight that the generic construction by Sakai et al. covers many approaches to define an ABS scheme.

Recall that simulation privacy demands the existence of three ppt simulation algorithms, namely SimSetup, SimKeyGen, and SimSign. We have to define such simulation algorithms and show that no adversary can distinguish the output of the simulation algorithms from the normal algorithms Setup, KeyGen, and Sign of an ABS scheme as given in Definition 3.2.1. The simulation signing algorithm SimSign that we present is implicitly given in the unforgeability proof by Sakai et al. in [SAH16, Theorem 1, Game 2]. Note that we focus on the construction given by Sakai et al. in [SAH16] which is also one of the two constructions presented in

the journal version [SAH18]. Considering the journal version, the first construction shows the idea of the construction more clearly whereas the second construction is more efficient, but also more complicated, since it is an optimization of the first construction.

Building Blocks

Let us start with the description of the building blocks of the Sakai et al. construction. Originally the construction by Sakai et al. uses a non-interactive witness indistinguishable argument of knowledge (NIWI) (Definition 2.3.22) for circuits [GOS12; GS12], a collision-resistant hash function (Definition 2.3.2), a structure-preserving signature (SPS) [KPW15], an extractable commitment scheme [GS08; GS12] and supports circuits C as policies.

NIWI. Sakai et al. [SAH16] introduce a combination of the techniques from Groth-Ostrovsky-Sahai proofs [GOS12] and Groth-Sahai proofs [GS12] to build an efficient NIWI for circuit satisfiability. Their NIWI supports arbitrary circuits of unbounded size and depth. Is is the reason why the generic ABS construction of Sakai et al. [SAH16] support arbitrary circuits as policies.

Circuits. Let us briefly describe how circuits are represented in [SAH16]. A circuit C has L input wires and overall N NAND gates, including the last output gate. Wires are references by indices $1, \ldots, L, L+1, \ldots, L+N$, where the first L indices are input wires, the internal wires are indices $L+1, \ldots, L+N-1$, and the output wire is $L+N$. To define which NAND gate involves which two wires, then called incoming wires of the gate, there are topology functions $I_1, I_2 : \{L+1, \ldots, L+N\} \to \{1, \ldots, L+N-1\}$, where it is required that $I_1(i) < i$ and $I_2(i) < i$. The topology functions take as input a non-input wire corresponding to the outgoing wire of a NAND gate and output the first and second incoming wires of this gate. Hence given C, we know the wires, their role, and how they are connected.

Putting the above together, Sakai et al. [SAH16] employ the Groth-Ostrovsky-Sahai proof system [GOS12] approach to prove that for each gate the NAND relation $\neg(u \wedge v) = w$ is satisfied, where u, v are incoming wires and w is the outgoing wire. The combination with Groth-Sahai proofs [GS12] allows them to generate commitments to each wire and prove the arithmetized NAND relation $1 - u \cdot v = w$ via a product pairing equation, which is the specialty of Groth-Sahai proofs. The corresponding relation for circuit satisfiability is then $(x = C, w = (x_1, \ldots, x_L)) \in R \Leftrightarrow C((x_1, \ldots, x_L)) = 1$.

Commitments. The Sakai et al. generic ABS construction also relies on extractable commitment scheme that are a part of the NIWI system that they use. Security of extractable commitment schemes includes binding and hiding. As in commitment schemes, binding means that it is infeasible for an adversary to open a

commitment in two different ways. Furthermore, hiding means that it is infeasible for an adversary to distinguish two commitments on two distinct messages. Additionally in extractable commitment schemes, there is an ppt extraction algorithm that extracts the committed message from a commitment. Sakai et al. [SAH16] use the Groth-Sahai extractable commitment scheme that is part of the Groth-Sahai proof system and that is presented in [GS08; GS12]. It allows two setups. The first setup provides perfect hiding and computational binding commitments. The second setup provides computational hiding and perfect binding commitments. For a detailed formal definition we refer to [Fuc10].

SPS. The last building block is the so called structure-preserving signature (SPS) [Abe+16]. These are signature schemes with the additional requirement that messages, signatures, and public keys are all group elements. Sakai et al. [SAH16] use the SPS scheme by Kiltz et al. [KPW15]. The SPS scheme is used to issue secret keys on attributes in the Sakai et al. generic ABS construction.

Construction

With the building blocks in place, we move on with the description of how the construction works. As usual, secret keys for attributes $\mathbb{A}$ are signatures, here a SPS on the attributes. A message-policy pair $(m, \mathbb{P} := C)$ consists of a message $m \in \{0,1\}^*$, and a policy $\mathbb{P}$ represented as an arbitrary circuit. The challenge in creating a signature on such a message-policy pair is that several statement has to be proven efficiently. That is, one has to prove the satisfiability of the circuit with respect to the attributes of a secret key, the validity of a signature on the attributes, and one has to bind the message to the proof. The last part is necessary, since the NIWI proof is otherwise independent of the message. Sakai et al. use a special NIWI for circuits and an OR-proof, involving the hash of the message-policy pair, to achieve this. As mentioned above, the OR-proof is a common technique to bind messages to proofs and, as we will see, to get simulatability. For further details on the NIWI proofs we refer to [SAH16]. In this work we only focus on how the OR-proof technique is used.

In the following, let $\mathsf{Hash} = (\mathsf{KeyGen}, \mathsf{H})$ be a collision-resistant hash function and $h \leftarrow \mathsf{H}(hk, (m, C))$ be the hash value of the message-policy pair (m, C). The satisfiability proof of the circuit C is done via the NIWI. To bind the message m to the NIWI proof, the circuit is at first extended in such a way that it is additionally satisfied by the hash value h. This is done via an OR-gate in the circuit. Intuitively, for each signature generation we extend the set of satisfying inputs of circuit C by an additional input corresponding to the hash value of the message-policy pair to be signed. Furthermore, for the ABS construction the NIWI is used to prove the validity of a secret key, a SPS signature, for attributes that satisfy the circuit. Note, secret keys for the extended satisfying input are never generated for users. This is exactly where the simulation signing algorithm $\mathsf{SimSign}$ starts to formulate. Basically, $\mathsf{SimSign}$ is the original signing algorithm Sign executed with a special

secret key generated on the hash value of the message-policy pair (m, C) (the extended satisfying input). This is done to satisfy the circuit via the extended OR part. Note, this special secret key, an SPS signature, can only be generated with the master secret key (the SPS secret key). Hence, we set the simulation trapdoor td_{sim}, that allows $\mathsf{SimSign}$ to generate simulated signatures, to the master secret key msk.

Next, we present the construction by Sakai et al. [SAH16] for completeness and adapt the notation to ours. Subsequently, we present the simulation algorithms. The construction by Sakai et al. represents attributes as bit vectors $\mathbb{A} = (x_1, \ldots, x_l) \in \{0,1\}^l$ of length $l \in \mathbb{N}$. This can be interpreted in different ways. One interpretation is that they consider binary attributes. That is, if $x_i = 1$ holds then an attribute $a_i \in \mathcal{U}$ is set and if $x_i = 0$ holds then it is not set. The attribute universe is therefore $\mathcal{U} = \{a_1, \ldots, a_l\}$. Another interpretation is that Sakai et al. support a single attribute representing a numerical value encoded in l bits or a combination of both, where some bits correspond to numerical values and others indicate binary attributes. The latter is a more expressive interpretation, where the policy can also include some threshold and range checks.

Construction 3.5.1 (Generic Construction by Sakai et al. - [SAH16])
Let GrGen be a bilinear group generator, let $\mathsf{Hash} = (\mathsf{Hash.KeyGen}, \mathsf{Hash.H})$ be a collision-resistant hash function with output length l_h, let $\mathsf{NIWI} = (\mathsf{NIWI.Setup}, \mathsf{NIWI.Prove}, \mathsf{NIWI.Verify})$ be a NIWI for circuit satisfiability relation, let $\mathsf{SPS} = (\mathsf{SPS.Setup}, \mathsf{SPS.KeyGen}, \mathsf{SPS.Sign}, \mathsf{SPS.Verify})$ be a structure-preserving signature scheme, let $\mathsf{Com} = (\mathsf{Com.Commit}, \mathsf{Com.Extr})$ be an extractable commitment scheme, and let $l \in \mathbb{N}$, $l \geq l_h$ be the size of the attribute vectors and let $l + 1$ be the input length of the circuits. The attribute-based signature scheme $\mathsf{ABS}^{\mathrm{SAH}}$ for message space $\{0,1\}^*$ is defined in Figure 3.8.

Note, a secret key $sk_{\mathbb{A}}$ consists of attributes $\mathbb{A}$ and SPS signature θ on $\mathbb{A}$. Hence, a user can efficiently verify a secret key $sk_{\mathbb{A}}$ by running $\mathsf{SPS.Verify}(pk, \mathbb{A}, \theta)$. The Sign algorithm is the most complex part of the construction, since it combines the two techniques of the Groth-Ostrovsky-Sahai [GOS12] and Groth-Sahai proof systems [GS12]. Hence, it outputs a proof that the extended circuit $\hat{C}$ evaluates to 1 on input the attributes *or* the hash value of the message-policy pair. Note, that the key generation algorithm KeyGen of $\mathsf{ABS}^{\mathrm{SAH}}$ only generates secret keys where the first attribute is fixed to be 0. To achieve this, it prepends the 0 to the attributes given, before generating the signature using SPS. Also note, that by definition of the extended circuit $\hat{C}$ the first input X_1 has to be 0 on left side of the *or* and on the right side the input X_1 has to be 1. Hence, honestly generated secret keys (output of KeyGen) never satisfy the right side of the disjunction. Even if one asks for a secret key on a hash value of a message-policy pair the key generation by KeyGen adds a zero attribute in front of the hash value. For completeness, we include the following theorem in this thesis and refer for the proof to [SAH16].

$\mathsf{ABS}^{\mathrm{SAH}}.\mathsf{Setup}(1^\lambda)$

$\mathbb{GD} \coloneqq (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, e, g, \tilde{g}) \leftarrow \mathsf{GrGen}(1^\lambda)$
$crs \leftarrow \mathsf{NIWI.Setup}(\mathbb{GD})$
$(pk, sk) \leftarrow \mathsf{SPS.KeyGen}(\mathbb{GD}, 1^{l+1})$
$hk \leftarrow \mathsf{Hash.KeyGen}(1^\lambda)$
$pp \coloneqq (l, crs, pk, hk), msk \coloneqq sk$
return (pp, msk)

$\mathsf{ABS}^{\mathrm{SAH}}.\mathsf{KeyGen}(pp, msk, \mathbb{A})$

parse $pp = (l, crs, pk, hk), msk = sk$
parse $\mathbb{A} = (x_1, \ldots, x_l)$
$A \coloneqq (g^0, g^{x_1}, \ldots, g^{x_l})$
$\theta \leftarrow \mathsf{SPS.Sign}(pk, sk, A)$
return $sk_{\mathbb{A}} \coloneqq (\mathbb{A}, \theta)$

$\mathsf{ABS}^{\mathrm{SAH}}.\mathsf{Sign}(pp, sk_{\mathbb{A}}, M, \mathbb{P})$

parse $pp = (l, crs, pk, hk), sk_{\mathbb{A}} = (\mathbb{A}, \theta), \mathbb{A} = (x_1, \ldots, x_l), \mathbb{P}$ as circuit C
$h \leftarrow \mathsf{Hash.H}(hk, (M, C))$
// extend circuit C to $\hat{C}$, where hash value h is hard-wired
$\hat{C}(X_1, \ldots, X_{l+1}) = 1 \Leftrightarrow (X_1 = 0 \wedge C(X_2, \ldots, X_{l+1}) = 1) \vee (X_1 = 1 \wedge X_2 || \cdots || X_{l_h+1} = h)$
// use leading 0 and attributes $\mathbb{A} = (x_1, \ldots, x_l)$ to set the input wires
$X_1 \coloneqq 0, X_2 \coloneqq x_1, \ldots, X_{l+1} \coloneqq x_l$
// compute assignment for each non-input wire in $\hat{C}$ with NAND input arithmetization
for $i = (l+1)+1, \ldots, (l+1)+(N-1)$ **do** $X_i \coloneqq 1 - X_{I_1(i)} \cdot X_{I_2(i)}$
// compute group elements for all wires except output wire
for $i = 1, \ldots, (l+1)+(N-1)$ **do** $W_i \coloneqq g^{X_i}, \tilde{W}_i \coloneqq \tilde{g}^{X_i}$
// compute commitment to the secret key element θ
$com_\theta \leftarrow \mathsf{Com.Commit}(pp, \theta)$
// compute commitments to all wire group elements
for $i = 1, \ldots, (l+1)+(N-1)$ **do**
  $com_{W_i} \leftarrow \mathsf{Com.Commit}(pp, W_i), com_{\tilde{W}_i} \leftarrow \mathsf{Com.Commit}(pp, \tilde{W}_i)$
// prove that the secret key is valid for the attributes that are also the input of the circuit
use NIWI.Prove to generate proof π_{Sign} for equation $\mathsf{SPS.Verify}(pk, (W_1, \ldots, W_{l+1}), \theta) = 1$
// prove that the input wire commitments are to the same value
for $i = 1, \ldots, l+1$ **do**
  use NIWI.Prove to generate proof π_i for equation $e(g, \tilde{W}_i) = e(W_i, \tilde{g})$
// prove internal wire commitments are to the same value and NAND relation holds
for $i = (l+1)+1, \ldots, (l+1)+(N-1)$ **do**
  use NIWI.Prove to generate proof π_i for equations
    $e(g, \tilde{W}_i) = e(W_i, \tilde{g}), \qquad e(W_{I_1(i)}, \tilde{W}_{I_2(i)}) \cdot e(W_i, \tilde{g}) = e(g, \tilde{g})$
// prove that the NAND relation holds for the last gate
use NIWI.Prove to generate proof $\pi_{(l+1)+N}$ for output wire equation
    $e(W_{I_1((l+1)+N)}, \tilde{W}_{I_2((l+1)+N)}) = 1$
return $\sigma \coloneqq (com_\theta, com_{W_1}, \ldots, com_{W_{(l+1)+(N-1)}}, com_{\tilde{W}_1}, \ldots, com_{\tilde{W}_{(l+1)+(N-1)}}, \pi_{\mathsf{Sign}},$
  $\pi_1, \ldots, \pi_{(l+1)+N})$

Figure 3.8: Generic Construction by Sakai et al. [SAH16].

$\mathsf{ABS}^{\mathrm{SAH}}.\mathsf{Verify}(pp, M, \mathbb{P}, \sigma)$

parse $pp = (l, crs, pk, hk)$
parse $\mathbb{P}$ as circuit C
parse $\sigma = (com_\theta, com_{W_1}, \ldots, com_{W_{(l+1)+(N-1)}}, com_{\tilde{W}_1}, \ldots, com_{\tilde{W}_{(l+1)+(N-1)}},$
$\pi_{\mathsf{Sign}}, \pi_1, \ldots, \pi_{(l+1)+N})$
extend C as above to $\hat{C}$
use $\mathsf{NIWI.Verify}$ and the commitments in σ to verify every proof π_j in σ
if all proofs are valid set $b \coloneqq 1$, otherwise set $b \coloneqq 0$
return b

Figure 3.8: Generic Construction by Sakai et al. [SAH16] (continued).

Theorem 3.5.1. *If* NIWI *is a perfect witness-indistinguishable proof system,* Hash *is a collision-resistant hash function,* SPS *is an existential unforgeable structure-preserving signature scheme, and* Com *is an perfectly hiding and perfectly extractable commitment scheme, then the attribute-based signature scheme* $\mathsf{ABS}^{\mathrm{SAH}}$ *is perfectly private (Definition 3.2.4) and unforgeable (Definition 3.2.6).*

Intuitively, perfect privacy follows directly from the witness indistinguishability of the proof system and the perfect hiding property of the commitment scheme. For the unforgeability we move to the perfect binding setup of the commitment scheme. This gives us perfect extractability for the commitments and therefore access to the witness used in the proofs. Additionally, signatures are only generated with witnesses for the right side of the OR in the extended circuit. Then, an adversary either finds a collision in the hash function or generates a forgery for the SPS scheme.

Next, we present that $\mathsf{ABS}^{\mathrm{SAH}}$ is simulation private. For this we have to define simulation algorithms SimSetup, SimKeyGen, and SimSign. Recall that the simulation signing algorithm SimSign was already given in the unforgeability proof of the Sakai et al. construction in [SAH16]. In the following, we formalize it according to our simulation privacy notion.

Theorem 3.5.2. *If* NIWI *is perfectly witness-indistinguishable and* Com *is perfectly hiding, then* $\mathsf{ABS}^{\mathrm{SAH}}$ *is perfectly simulation private (Definition 3.2.5).*

Proof. Major changes to Setup and KeyGen are not necessary, since the SimSign algorithm uses the master secret key *msk* to simulate signatures, which is already output by Setup. Therefore, SimSetup behaves exactly as Setup, but SimSetup additionally outputs $td_{sim} \coloneqq msk$, and KeyGen behave exactly as SimKeyGen, but with additional input td_{sim} that SimKeyGen ignores. Major changes compared to Sign are necessary to define the simulation algorithm SimSign that uses the master secret key *msk* to generate ABS signatures without any attributes as an input. To achieve this, it uses the right side of the *or* in the extended circuit $\hat{C}$ as we

described above. We present simulation algorithm SimSign in the following where we highlight the changes.

$\mathsf{ABS}^{\mathrm{SAH}}.\mathsf{SimSign}(pp, msk, td_{sim}, m, \mathbb{P})$

parse $pp = (l, crs, pk, hk), msk = sk, td_{sim} = sk$
parse $\mathbb{P}$ as circuit C
$h \leftarrow \mathsf{Hash.H}(hk, (M, C))$
// parse h as bits and pad with 0 to the length $l + 1$
parse $h = (h_1, \cdots, h_{l_h})$
$h \coloneqq (h_1, \cdots, h_{l_h}, 0, \ldots, 0) \in \{0,1\}^{l+1}$
$A_h \coloneqq (g^1, g^{h_1}, \ldots, g^{h_{l_h}}, g^0, \ldots, g^0) \in \mathbb{G}_1^{l+1}$
$\theta \leftarrow \mathsf{SPS.Sign}(pk, sk, A_h)$
$sk_h \coloneqq (h, \theta)$
// next extend circuit C to $\hat{C}$, where hash value h is hard-wired
$\hat{C}(X_1, \ldots, X_{l+1}) = 1 \Leftrightarrow$
$\quad (X_1 = 0 \wedge C(X_2, \ldots, X_{l+1}) = 1) \vee (X_1 = 1 \wedge X_2 || \cdots || X_{l_h+1} = h)$
// use leading 1 and padded hash value h to set the input wires
$X_1 \coloneqq 1, X_2 \coloneqq h_1, \ldots, X_{l_h+1} \coloneqq h_{l_h}, X_{l_h+2} \coloneqq 0, \ldots, X_{l+1} \coloneqq 0$
// the remaining steps are as defined in Sign
// ...
return σ

Note, that all three simulation algorithms are ppt algorithms, since they are based on existing ppt algorithms of the construction and only add efficiently computable steps. In the definition of SimSign above we omit at the end the unchanged steps of Sign. Here, the only internal difference is that the NIWI proofs are generated with respect to the special secret key (the SPS signature on the group representation of the padded hash value) and with respect to the input wires (the padded hash value with a leading zero). Therefore, the NIWI proves that the extended circuit $\hat{C}$ is satisfied as before in Sign. However, internally it uses a different witness, namely the special secret key for the right side of the disjunction.

SimSetup and SimKeyGen are unchanged from Setup and KeyGen, except for the additional output of SimSetup and input of the simulation trapdoor td_{sim} for SimKeyGen. Hence, their distribution of the other outputs is unchanged. Thus, it remains to show, that SimSign generates the same distribution as Sign.

Sign uses the secret key of the signer as the commit messages and the witness for the proofs. Whereas SimSign generates a special secret key sk_h on the hash value of the message-policy pair via the master secret key msk on-demand. Then, SimSign uses sk_h as for the commitments and as a witness for the proofs. Notice, that both algorithms expand the given circuit C such that it is additionally satisfied by the hash value used to compute the special secret key. The final output of both algorithms, SimSign and Sign, consists of commitments and proofs where just

Table 3.3: Overview of signature sizes of instantiations of the Sakai et al. [SAH16] generic construction (Construction 3.5.1) originally presented in [Bem17]
l input size of the circuit and length of attribute vector, and N number of gates in the circuit.

Used SPS	$\mathbb{G}_1$	$\mathbb{G}_2$	Assumption of SPS
$\mathsf{SPS}_{\mathrm{KPW}}$ [KPW15]	$6l + 10N + 20$	$6l + 10N + 6$	SXDH
$\mathsf{SPS}_{\mathrm{JR}}$ [JR17]	$6l + 10N + 18$	$6l + 10N + 6$	SXDH
$\mathsf{SPS}_{\mathrm{AGHO}}$ [AGHO11]	$6l + 10N + 12$	$6l + 10N + 6$	GGM

different messages and witnesses were used. From the perfect hiding property of the commitment scheme and the perfect witness indistinguishability of the proof system it follows that the distributions of $\mathsf{SimSign}$ and Sign are independent of the witness (messages) used and that they are identical.

□

Instantiations and efficiency. Next, we want to give some examples of how to instantiate the Sakai et al. generic construction $\mathsf{ABS}^{\mathrm{SAH}}$ and their efficiency originally presented by Bemmann [Bem17]. Since the proof system is specific for the construction, as described above, and implicitly defines the commitment scheme, the only have to instantiate the SPS scheme in the following. Originally Sakai et al. [SAH16] present an instantiation with the SPS scheme by Kiltz, Pan and Wee [KPW15], denoted by $\mathsf{SPS}_{\mathrm{KPW}}$, and list the signature size for this instantiation. In [Bem17] the generic construction is also instantiated by using the SPS schemes by Jutla and Roy [JR17], denoted by $\mathsf{SPS}_{\mathrm{JR}}$, and by Abe, Groth, Haralambiev and Ohkubo [AGHO11], denoted by $\mathsf{SPS}_{\mathrm{AGHO}}$. Details of the SPS schemes are out of scope of this thesis. However, Table 3.3 from [Bem17] shows the resulting signature sizes of the corresponding instantiations of the Sakai et al. generic constructions. Note that the used SPS scheme only influences the size of the commitment com_θ to the SPS signature θ (included in ABS secret keys) and the size of the NIWI proof π_{Sign} of the validity of the SPS signature. The other parts of the ABS signature are unchanged, since they depend only on the NIWI and how the circuit is presented. Since, the SPS scheme $\mathsf{SPS}_{\mathrm{AGHO}}$ of Abe et al. is proven to be optimal (signature size of 3 elements) [AGHO11] the corresponding instantiation of $\mathsf{ABS}^{\mathrm{SAH}}$ is also optimal regarding the used SPS scheme.

3.5.2 Generic ABS Construction by Maji et al.

Maji, Prabhakaran and Rosulek present in [MPR11] a generic construction of ABS supporting monotone boolean functions as policies. A full version of the work is available [MPR10], with full formal proofs and extended explanations. There is also a preliminary version of the work available [MPR08], which has only historical significance. In the following, we show similar to Section 3.5.1 that the construction of Maji et al., originally shown to be private and unforgeable, is simulation private.

Hence, we have to define ppt simulation algorithms SimSetup, SimKeyGen, and SimSign and then show that they are indistinguishable from the algorithms Setup, KeyGen, and Sign in the simulation privacy experiment Definition 3.2.5.

Building Blocks

Let us start with a description of the construction and its building blocks. The generic construction by Maji et al. is based on a NIWI system and a so called credential bundle. A credential bundle is a tagged signature $(t, \sigma) \leftarrow \mathsf{Sig.Sign}(sk, t||m_i)$ on a uniformly random tag t and message m_i realized by a EUF-CMA secure signature scheme Sig. Intuitively, the tag is a nonce that binds signatures under the same tag together. This is used to generate ABS secret keys on attribute vectors $\mathbb{A} = (a_1, \ldots, a_l) \in \mathcal{U}$ consisting of l individual signatures σ_i on the attributes a_i $(i = 1, \ldots, l)$ with the same tag t. Hence, in the following we use a signature scheme in the Maji et al. construction to sign attributes in the key generation. Maji et al. [MPR11] show three instantiations of their generic ABS construction. For the first two instantiations they use the Groth-Sahai proof system [GS08; GS12] as the NIWI system and combine it with the Boneh-Boyen signature scheme [BB04; BB08] and with the Waters signature scheme [Wat05] respectively. The third instantiation is directly proven to be secure in the generic group model [MPR10].

Construction

As we have discussed before, the challenge in constructing ABS from proof systems is to bind the message to be signed to the proof and therefore the policy. Hence, the generic Maji et al. construction [MPR11] supports a universe of attributes that contains a pseudo-attribute for every message-policy pair $(m, \mathbb{P})$. The construction defines a ABS signing algorithm Sign, that on input $(m, \mathbb{P})$ and a secret key $sk_{\mathbb{A}}$ extends the policy to policy $\hat{\mathbb{P}} := \mathbb{P} \vee$ "pseudo-attribute $(m, \mathbb{P})$ is in $\mathbb{A}$", i.e., a disjunction of the original policy and the statement that the signer has a secret key for the pseudo-attribute for $(m, \mathbb{P})$. Important here is that secret keys on pseudo-attributes are not generated by the key generation algorithm. Second, it uses the signers secret key $sk_{\mathbb{A}}$ (a signature on attributes $\mathbb{A}$) to generate the ABS signature. Here, the ABS signature is a NIWI proof. In detail, the NIWI proof shows that the signer knows a valid signature on attributes that satisfy the extended policy $\hat{\mathbb{P}}$. Here, the the left side of the disjunction in policy $\hat{\mathbb{P}}$ is satisfied.

Similar to the Sakai et al. construction (Section 3.5.1), the set of satisfying attributes with respect to the policy is extended to include a special element (the pseudo-attribute). Signers do not have a valid signature on this special element. However, the simulation signing algorithm utilize this. Concretely, the simulation signing algorithm SimSign uses the master secret key msk to issue a secret key $\hat{sk}$ for the pseudo-attribute $(m, \mathbb{P})$ and uses the extended policy $\hat{\mathbb{P}}$ to generate an ABS signature in the same way as Sign generates it. By using the pseudo-attribute $(m, \mathbb{P})$ the right side of the disjunction in the extend policy $\hat{\mathbb{P}}$ is satisfied

and is proven by using $\hat{sk}$ in the NIWI proof. The indistinguishability of the simulated and normal signatures then follows from the witness indistinguishability of the NIWI system. Hence, the approach of the SimSign algorithm for this generic construction by Maji et al. is similar to the one that we used above for the Sakai et al. construction. Also here simulated signatures and an alternative signing algorithm are at least mentioned by Maji et al. in all available versions of the work [MPR08; MPR10; MPR11]. Maji et al. define an alternative signing algorithm primarily to use it in the unforgeability definition. Note, that they define the alternative signing algorithm in such a way that it limits the formal usability. Namely, given just the master secret key and a policy it should find satisfying attributes for the policy, generate a secret key for these attributes and then produce a signature with this secret key by using the normal signing algorithm. This alternative signing algorithm is not a (probabilistic) polynomial-time algorithm if the pseudo-attributes technique is not used and more importantly for hard policy classes, where one cannot compute satisfying attributes in polynomial-time from a given policy. Besides their alternative signing algorithm, there is another signing algorithm presented in the first step of the unforgeability proof of the generic construction. The simulation signing algorithm that we present is based on the latter.

Next, we describe the Maji et al. construction [MPR11] with adapted notation. Then, we present the simulation algorithms in detail. Their generic construction describes five algorithms TSetup, ASetup, KeyGen, Sign, and Verify. The key generation algorithm KeyGen is originally called AttrGen. They define two setup algorithms: TSetup for the so called signature trustee and ASetup for the attribute-issuing authority. In detail, this separates the setup of the NIWI proofs from the setup for the secret key generation on attributes. The separation is a preparation for multi-authority attribute-based signatures that are also covered by Maji et al. [MPR10; MPR11]. There attribute-issuing authorities have to register at the signature trustee. Since we focus on the single attribute-issuing authority case we combine the setups. Hence, we define algorithm Setup, such that it combines TSetup and ASetup in one algorithm to match our syntax. Accordingly, we define the generic ABS construction by Maji et al. as $\mathsf{ABS}^{\mathrm{MPR}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$. Let $\mathcal{U}$ be the universe of attributes and let $\hat{\mathcal{U}}$ denote the space of pseudo-attributes, where $\mathcal{U} \cap \hat{\mathcal{U}} = \emptyset$. One pseudo-attribute $a_{m,\mathbb{P}} \in \hat{\mathcal{U}}$ is added for each message-policy pair $(m, \mathbb{P})$. A policy is defined as a monotone boolean function $\mathbb{P}\colon 2^{\mathcal{U}} \to \{0,1\}$, where $2^{\mathcal{U}}$ denotes the powerset of $\mathcal{U}$.

Construction 3.5.2 (Generic Construction by Maji et al. - [MPR11])
Let GrGen be a bilinear group generator, let $\mathsf{CB} = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme with message space $\mathcal{U} \cup \hat{\mathcal{U}}$, and let $\mathsf{NIWI} = (\mathsf{NIWI.Setup}, \mathsf{NIWI.Prove}, \mathsf{NIWI.Verify})$ be a non-interactive witness indistinguishable proof system for monotone boolean functions. The generic ABS construction $\mathsf{ABS}^{\mathrm{MPR}}$ for message space $\{0,1\}^*$ is defined in Figure 3.9.

Note, a secret key $sk_{\mathbb{A}}$ consists of attributes $\mathbb{A}$ and signature θ on $\mathbb{A}$. Therefore, to efficiently verify a secret key $sk_{\mathbb{A}}$ a user runs $\mathsf{CB.Verify}(apk, \mathbb{A}, \theta)$. Let us explain the above construction in more detail. The setup is straightforward. Maji et al. [MPR11] define it such that it generates two key-pairs such that the authors can later extend the construction to multi-authorities and to separate the validity of the pseudo-attributes and the attributes (issued by the authority). The authority also checks in the key generation KeyGen that the given attributes do not include any pseudo-attributes. Hence, secret keys that are honestly generated by an authority never include any pseudo-attributes. Therefore, in Sign we set the i-th public key pk_i to the public key used to generate attribute a_i. In detail, we set public key pk_i to be tpk for the pseudo-attribute and to apk for attributes. Consequently, if the pseudo-attribute is part of the NIWI proof the statement includes the signature trustee's public key tpk. To understand the statement that the NIWI proves, we have to mention that in the construction of Maji et al. a signer only needs valid signatures for a subset of attributes that are sufficient to satisfy the policy. This is what the equation for the NIWI proof in algorithm Sign expresses. Hence, the statement that the NIWI proves is that the signer has sufficient attributes in his secret key to satisfy the extended policy. For this to work, all elements were set appropriately beforehand. Here, we have to mention that setting the signatures $\hat{\sigma}_i$ to 0 is not a problem since they are not used in the NIWI statement. The concrete instantiation of a NIWI that can efficiently proof such statements is based on the Groth-Sahai proof system [GS08; GS12] and is given by Maji et al. [MPR11]. For further details on the concrete statements for the Groth-Sahai proof system we refer to [MPR10]. The following theorem is included for completeness and we refer for the full proof to [MPR10; MPR11].

Theorem 3.5.3. *If* NIWI *is a perfectly witness indistinguishable argument of knowledge,* CB *is an EUF-CMA signature scheme, then the attribute-based signature scheme* $\mathsf{ABS}^{\mathrm{MPR}}$ *is perfectly private (Definition 3.2.4) and unforgeable (Definition 3.2.6).*

The perfect privacy of the construction $\mathsf{ABS}^{\mathrm{MPR}}$ follows from the perfect witness indistinguishability of the NIWI proof system. Unforgeability of $\mathsf{ABS}^{\mathrm{MPR}}$ is shown by two reductions, that use the extractor of the NIWI. Given an adversary against the unforgeability of $\mathsf{ABS}^{\mathrm{MPR}}$ and extracting a valid proof, i.e., an ABS signature, output by the adversary as its forgery, gives us a witness that is either a valid CB signature on a pseudo-attribute under the trustee's public key tpk or a valid CB signature under the authority's public key apk. The extracted CB signature can then be used to claim a forgery in a corresponding reduction to one of the CB key generations for signature trustee or authority. For the full proof see [MPR10]. Next, we show simulation privacy for $\mathsf{ABS}^{\mathrm{MPR}}$.

Theorem 3.5.4. *If* NIWI *is perfectly witness indistinguishable, then* $\mathsf{ABS}^{\mathrm{MPR}}$ *is perfectly simulation private (Definition 3.2.5).*

$\mathsf{ABS}^{\mathrm{MPR}}.\mathsf{Setup}(1^\lambda)$

$\mathbb{GD} \coloneqq (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, e, g, \tilde{g}) \leftarrow \mathsf{GrGen}(1^\lambda)$
$crs \leftarrow \mathsf{NIWI.Setup}(\mathbb{GD})$
// generate signature trustee key pair
$(tpk, tsk) \leftarrow \mathsf{CB.KeyGen}(\mathbb{GD})$
// generate attribute-issuing authority key pair
$(apk, ask) \leftarrow \mathsf{CB.KeyGen}(\mathbb{GD})$
$pp \coloneqq (crs, tpk, apk), msk \coloneqq (tsk, ask)$
return (pp, msk)

$\mathsf{ABS}^{\mathrm{MPR}}.\mathsf{KeyGen}(pp, msk, \mathbb{A})$

parse $pp = (crs, tpk, apk), msk = (tsk, ask)$
parse $\mathbb{A} = (a_1, \ldots, a_l)$
if $\mathbb{A}$ contains pseudo-attributes, **abort**
else
  $\theta \coloneqq (t, \sigma_1, \ldots, \sigma_l) \leftarrow \mathsf{CB.Sign}(apk, ask, \mathbb{A})$
  return $sk_{\mathbb{A}} \coloneqq (\mathbb{A}, \theta)$

$\mathsf{ABS}^{\mathrm{MPR}}.\mathsf{Sign}(pp, sk_{\mathbb{A}}, m, \mathbb{P})$

parse $pp = (crs, tpk, apk), sk_{\mathbb{A}} = (\mathbb{A}, \theta), \mathbb{A} = (a_1, \ldots, a_l), \theta = (t, \sigma_{a_1}, \ldots, \sigma_{a_l})$
if $\mathbb{P} \notin \mathcal{U} \vee \mathbb{P}(\mathbb{A}) = 0$ **return** $\perp$ **else**
// extend policy with pseudo-attribute $a_{m,\mathbb{P}} \in \hat{\mathcal{U}}$
$\hat{\mathbb{P}} \coloneqq \mathbb{P} \vee a_{m,\mathbb{P}}$
Let $\{a_1, \ldots, a_k\}$ be the attributes in $\hat{\mathbb{P}}$
for $i = 1, \ldots, k$ **do**
  if $\sigma_{a_i} \in sk_{\mathbb{A}}$ **then**
    $\hat{\sigma_i} \coloneqq \sigma_{a_i}$
  else $\hat{\sigma_i} \coloneqq 0$
  if $a_i = a_{m,\mathbb{P}}$ **then**
    $pk_i \coloneqq tpk$
  else
    $pk_i \coloneqq apk$
use $\mathsf{NIWI.Prove}$ to generate proof π for the statement
  $\exists (t, \hat{\sigma_1}, \ldots, \hat{\sigma_k}) \colon \hat{\mathbb{P}}\big(\{a_i \mid \mathsf{CB.Verify}(pk_i, a_i, (t, \hat{\sigma_i})) = 1\}\big) = 1$
return $\sigma \coloneqq \pi$

$\mathsf{ABS}^{\mathrm{MPR}}.\mathsf{Verify}(pp, M, \mathbb{P}, \sigma)$

parse $pp = (crs, tpk, apk)$
parse $\sigma = \pi$
// extend policy with pseudo-attribute $a_{m,\mathbb{P}} \in \hat{\mathcal{U}}$
$\hat{\mathbb{P}} \coloneqq \mathbb{P} \vee a_{m,\mathbb{P}}$
use $\mathsf{NIWI.Verify}$ to verify the proof π
if it is valid set $b \coloneqq 1$, otherwise set $b \coloneqq 0$
return b

Figure 3.9: Generic Construction by Maji et al. [MPR11].

Proof. To show simulation privacy we start with the definition of ppt algorithms SimSetup, SimKeyGen, and SimSign. We can use Setup as SimSetup and KeyGen as SimKeyGen. We only have to add the additional output of the simulation trapdoor td_{sim} to SimSetup and give td_{sim} to SimKeyGen as an additional output that it ignores. The simulation trapdoor is set to the signature trustee's secret key tsk. The following simulation signing algorithm SimSign is mentioned as an idea for the reduction in the unforgeability proof in the conference version [MPR11, Theorem 1] and is defined in the full version [MPR10, Appendix C.1]. We highlight the changes compared to the signing algorithm Sign in the following definition of the simulation signing algorithm SimSign.

$\mathsf{ABS}^{\mathrm{MPR}}.\mathsf{SimSign}(pp, msk, td_{sim}, m, \mathbb{P})$

parse $pp = (crs, tpk, apk)$, $msk = (tsk, ask)$, $td_{sim} = tsk$

// generate secret key for pseudo-attribute $a_{m,\mathbb{P}} \in \hat{\mathcal{U}}$

$\hat{sk} \leftarrow \mathsf{CB.Sign}(tpk, tsk, a_{m,\mathbb{P}})$

// the remaining steps are the same as in Sign

$\sigma \leftarrow \mathsf{ABS}^{\mathrm{MPR}}.\mathsf{Sign}(pp, \hat{sk}, m, \mathbb{P})$

return σ

The SimSign algorithm uses the signature trustee secret key tsk to generate a signature on the pseudo-attribute $a_{m,\mathbb{P}}$ for the given message-policy pair $m, \mathbb{P}$. Thus, the right side of the extend policy $\hat{\mathbb{P}} := \mathbb{P} \vee a_{m,\mathbb{P}}$ is satisfied and the NIWI proof can be generated as in Sign. However, internally it uses only the pseudo-attribute $a_{m,\mathbb{P}}$ and the signature on it as the witness. Finally, note that all three simulation algorithms are ppt algorithms, since they extend existing ppt algorithms only with efficiently computable steps.

Regarding simulation privacy, the algorithms SimSetup and SimKeyGen are unchanged compared to Setup and KeyGen with respect to what the adversary gets as input. Hence, we focus on the signing algorithms in the following. The signatures generated by Sign and SimSign are just NIWI proofs that use different witnesses for the same extended policy $\hat{\mathbb{P}} := \mathbb{P} \vee a_{m,\mathbb{P}}$. Therefore, the extend policy has the same set of satisfying attributes in both algorithms. As we previously described, internally the NIWI proofs of Sign are over attributes issued by the authority. Therefore, the proofs are with respect to the authority public key apk. The witness that Sign uses is the ABS secret key for these attributes including the issued attributes under apk. This is a witness for the left side of the disjunction in the extended policy $\hat{\mathbb{P}}$. Internally SimSign uses as the witness the pseudo-attribute $a_{m,\mathbb{P}}$ and the signature under the trustee public key tpk on it. This is a witness for the right side of the disjunction in the extend policy $\hat{\mathbb{P}}$. Note, that technically SimSign can also sign policies $\mathbb{P}$ where there are no satisfying attributes, because it always extends the policy to be satisfiable by the pseudo-attribute $a_{m,\mathbb{P}}$. We handle this in the simulation privacy experiment (Definition 3.2.5) by requiring that the simulation oracle only answers with a signature, if the given attributes satisfy the

policy $\mathbb{P}$. Otherwise, it outputs the failure symbol $\perp$. The same behavior is defined in the second step of $\mathsf{ABS}^{\mathrm{MPR}}.\mathsf{Sign}$. Hence, trivially distinguishing $\mathsf{SimSign}$ and Sign is precluded. It follows from the perfect witness indistinguishability of NIWI that the output distributions of Sign and $\mathsf{SimSign}$ algorithms are independent of the witness an identical. □

In conclusion, we have shown that the generic ABS constructions by Maji et al. [MPR11] and Sakai et al. [SAH16] are our perfectly simulation private. Looking ahead, our UC result (Theorem 4.5.1) that we present in Chapter 4 shows that the two generic ABS constructions achieve UC security.

Universal Composable Attribute-based Signatures

4

Summary. In this chapter we present the attribute-based signatures in the universal composability framework (UC) results of [BEJ18b] and its full version [BEJ18a]. The main contribution of these works that we present in this thesis is that we enhance the security model of ABS in the form of an ideal ABS functionality in UC (Section 4.3).

Furthermore, we show that an ABS scheme is simulation private and existential unforgeable regarding our experiment-based definitions (Chapter 3) if and only if it is UC secure. This shows that the two existing generic ABS construction that are simulation private, as presented in Section 3.5, are also UC secure (Section 4.5).

We present the basic concepts of Canetti's universal composability framework (UC) [Can01] as an introduction to the formalisms of the framework before we define UC preliminaries including the overall framework, structure, model of computation, and UC security in (Section 4.1). Based on this we present our universally composable ideal functionality for attribute-based signatures in Section 4.3. Subsequently, we introduce a protocol that transforms ABS schemes to the UC model (Section 4.4). Finally, we show the relation of the ideal ABS functionality and the experiment-based security definition of Chapter 3. Hence, in Section 4.5 we show our main result that the protocol realizes our ideal ABS functionality, i.e., the protocol is UC secure, if and only if the used ABS scheme is correct, consistent, unforgeable and computationally simulation private.

In this thesis we use the most recent version of UC presented in [Can20], since this version fixes some flaws present in previous versions including [Can01]. The years from the original publication of the UC framework [Can01] to the current version [Can20] showed that security in UC is difficult to formalize, see discussion in [Can20]. This is the reason why we base our ideal functionality for attribute-based signatures in this thesis on the most recent version of UC [Can20]. Earlier versions included some bugs and oversights. For a list of major changes see [Can20]. Note, most of the changes are mentioned inline and not listed separately.

Compared to [BEJ18b] we add the current formal security definitions of UC, properly consider balanced and identity-bounded environments, structured protocols and use the current probabilistic polynomial-time definition for UC from [Can20]. Intuitively, our UC result shown in [BEJ18b] does not change, however formally we present it here based on a sound UC definition.

4.1 Introduction

Universal composability of cryptographic schemes is a key concept of analyzing the security of schemes in arbitrary compositions. It facilitates a modular approach to security analyses. Many frameworks and models of universal composability were introduced in the literature [Can01; CCL15; CDPW07; CKKR19; HS15; Küs06; Mau11; PW01]. The universal composability framework (UC) introduced by Canetti [Can01] is the most used framework in the literature. The basic approach in UC is to define an ideal protocol that captures the ideal behavior of a scheme without relying on concrete cryptographic schemes. The main part of the ideal protocol is a so called ideal functionality that acts as a trusted party in the name of all parties in the protocol. Then one gives a real protocol using concrete cryptographic schemes (e.g., a digital signature scheme) replacing the ideal functionality. Intuitively, to show security one proves that in any execution the real protocol behaves like the ideal protocol. As already mentioned in the conceptual description of UC in Section 2.4 the appeal of UC comes from the universal composability theorem [Can01] that allows us to compose multiple schemes without compromising the security of the individual schemes. To this end, cryptographers

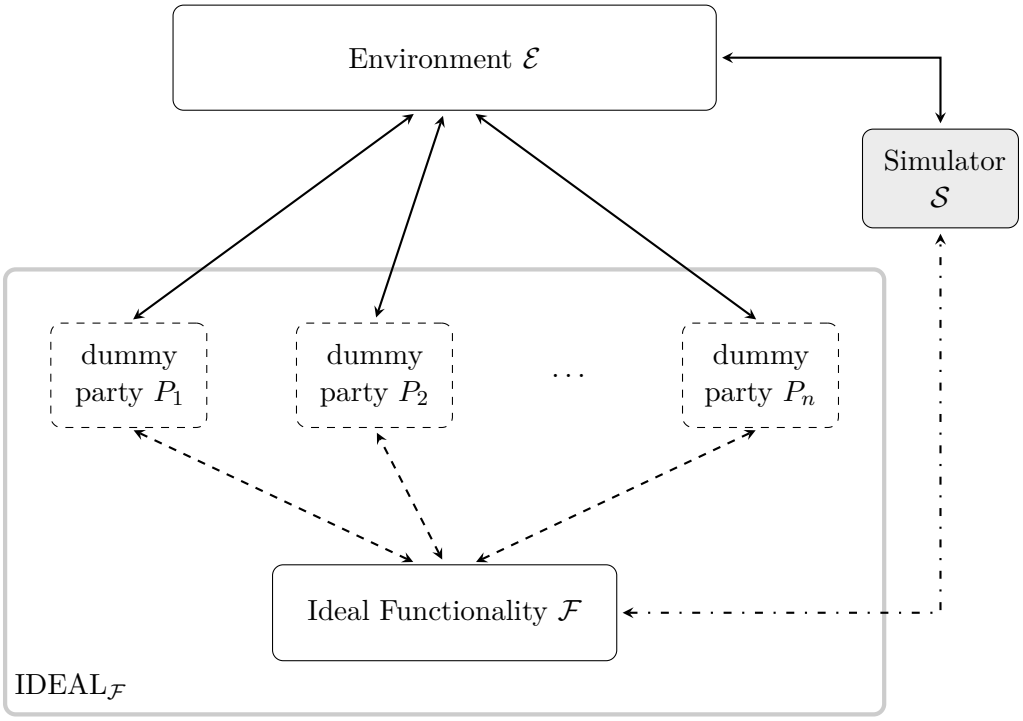

Figure 4.1: Ideal setting with ideal protocol IDEAL$_{\mathcal{F}}$. The dashed lines represent that the dummy parties just forward any messages (input) received by the environment $\mathcal{E}$ to the ideal functionality $\mathcal{F}$ and vice versa. The connection between the ideal functionality $\mathcal{F}$ and simulator $\mathcal{S}$ represents backdoor communication that is only visible to $\mathcal{F}$ and $\mathcal{S}$. The connection between the environment $\mathcal{E}$ and the simulator $\mathcal{S}$ represents communication that the simulator has to be able to simulate for $\mathcal{E}$, i.e., communication that happens in the real setting.

show that individual schemes are secure in UC. Then, they can rely on the universal composability theorem to build larger systems out of them. Intuitively, the universal composability theorem asserts that regardless of the environment of the larger system the individual schemes remain secure. This universality with regard to any execution environments is the strength of UC.

Let us describe Canetti's UC [Can20] on a high level. In UC a scheme, or more general a task, is described by an ideal functionality $\mathcal{F}$ as a part of an ideal protocol IDEAL$_{\mathcal{F}}$ (Figure 4.1) and is then compared to a (real) protocol ρ (Figure 4.2). In the ideal protocol IDEAL$_{\mathcal{F}}$, the ideal functionality $\mathcal{F}$ models a trusted party that handles all tasks in the name of all other parties. Hence, parties in the ideal protocol are just dummies and forward any messages. Therefore, the communication of parties is also handled by the ideal functionality $\mathcal{F}$. The real protocol ρ involves parties that handle their tasks on their own. Hence, parties run their own code and can directly communicate with each other. Figure 4.1 shows the ideal setting with an environment $\mathcal{E}$, a simulator $\mathcal{S}$, and an ideal protocol IDEAL$_{\mathcal{F}}$

consisting of dummy parties P_1 to P_n and an ideal functionality $\mathcal{F}$. Figure 4.2 shows the real setting with an environment $\mathcal{E}$, an adversary $\mathcal{A}$, and a protocol ρ consisting of parties P_1 to P_n. The solid lines in Figure 4.1 for the ideal setting represent the same communication that is present in the real setting (Figure 4.2). The environment $\mathcal{E}$, present in both the real and the ideal setting, is considered as the execution environment of the task (scheme) captured by the protocols. As such it controls the inputs and activations (computations) of parties. The capabilities of the environment capture that in reality, protocols do not exist in isolation. There are other tasks, like higher-level protocols (and schemes), that interact with each other. The environment $\mathcal{E}$ also communicates with the adversary $\mathcal{A}$ in the real setting (Figure 4.2) and with the simulator (ideal adversary) in the ideal setting (Figure 4.1).

Regarding security in UC, an environment $\mathcal{E}$ acts as an distinguisher that has to decide whether it is in the ideal or real setting. The role of the simulator in the ideal setting is to translate the behavior of the real protocol and an adversary $\mathcal{A}$ to the ideal protocol, such that an environment $\mathcal{E}$ cannot distinguish the ideal and real settings. Intuitively, a protocol ρ is called UC secure, if for every adversary $\mathcal{A}$ against the protocol ρ, there exists a simulator $\mathcal{S}$ considered as the ideal adversary for ideal functionality $\mathcal{F}$, such that no environment $\mathcal{E}$ can distinguish, if it talks to the ideal protocol and simulator $\mathcal{S}$ or to the protocol ρ and adversary $\mathcal{A}$. To formally define security we have to introduce some UC preliminaries in the following. We then present the formal UC security definition in Section 4.2.5. The universal composability framework by Canetti [Can20] captures ideal protocols and (real) protocols by the same definitional framework for protocols. Indeed, ideal protocols and protocols are instantiation of the same formalism.

Before going into the formal details of Canetti's UC we discuss other schemes in UC that are related to digital signatures and state the research questions that arise from considering attribute-based signature in UC.

4.1.1 Related Work

Universal composability has been considered as the framework for many important constructions in the literature [ACHM05; CDHK15; CDL16; CLNS17; FHH14; FLM11; KM16; Lin11]. Among others there are UC secure multi-commitments by Lindell [Lin11], string commitments by Fischlin et al. [FLM11], group signatures by Ateniese et al. [ACHM05], anonymous credentials by Camenisch et al. [CDHK15], non-interactive public-key encryption [CLNS17], and non-interactive key exchange by Freire et al. [FHH14]. In this chapter we present the first universal composable ABS. The ABS schemes and construction in the literature as presented in Section 3.1.1 are all defined using an experiment-based model. These ABS schemes (constructions) and their security definitions look at ABS as an isolated primitive, where a composition with other schemes demand additional specialized security proofs. However, in real applications, ABS is combined and composed with other cryptographic primitives to achieve more comprehensive security goals. For exam-

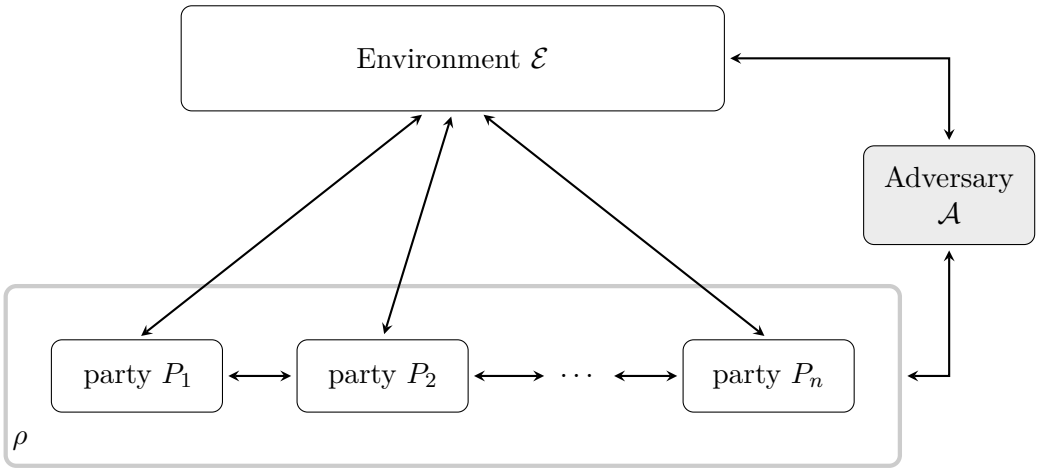

Figure 4.2: Real setting with protocol ρ. The connections between the environment $\mathcal{E}$ and parties P_1 to P_n represent input from $\mathcal{E}$ to the parties and output from the parties to $\mathcal{E}$. All parties can communicate with each other represented by the simplified connections between them. The connections between the adversary $\mathcal{A}$ and the protocol ρ represents that the adversary (potentially) controls the communication network and that it is able to interact with the parties, e.g., it sends messages to them or corrupts parties. Adversary $\mathcal{A}$ also exchanges messages with the environment $\mathcal{E}$ represented by the connection between them.

ple, application areas of ABS mentioned in the literature include attribute-based messaging [Bob+06; MPR11], leaking secrets [MPR11], attribute-based authentication [MPR11], access control systems [Li+10], trust-negotiation [MPR11], and a privacy-preserving certification mechanism [Kaa+17]. For example, ABS can be deployed as an authentication mechanism by service providers, i.e., in a challenge-response protocol. In such a protocol the user is asked to sign a policy and a nonce given by the service provider. Usually such authentication mechanisms are deployed in large scale applications. Canetti [Can01] introduced the UC to describe and prove security with such applications in mind. In the field of digital signatures in UC the research started with the result by Canetti [Can03] that a presented ideal digital signature functionality is equivalent to the standard experiment-based security model (EUF-CMA, Definition 2.3.6) of digital signatures [GMR88]. Another equivalence result was given by Abe and Ohkubo [AO12] for blind signatures. They define UC secure blind signatures and an equivalent experiment-based security notion. More complex signature related constructions were presented in UC by Ateniese et al. [ACHM05] with the construction of a group signature scheme and Camenisch et al. [CDHK15] presented an anonymous credential system.

4.1.2 Research Questions

Based on the related work, research questions and answers of Chapter 3 we identify the following questions regarding ABS in UC.

Q1 Are ABS universal composable? In detail, how to define ABS as an ideal functionality in UC? Including the question of how to model privacy ideally?

Q2 Are the experiment-based security definitions and the UC definition equivalent?

4.1.3 Contribution

In this chapter we answer the research questions affirmatively in the following way.

Research question Q1. To answer the UC question, we model the first universally composable ideal attribute-based signature functionality in Section 4.3. Our ideal ABS functionality is rooted in the ideal digital signatures functionality by Canetti [Can03] that allows signature creation on a message with a identity-based secret key. The novelty (and difficulty) comes from extending it to support multiple signing parties and signature creation on message-policy pairs under secret keys with attributes. Then, we show that unforgeable and simulation private ABS schemes are UC secure. This lifts existing schemes that we listed in Section 3.5 to be simulation private to UC security. This directly leads to the answer of the next question.

Research question Q2. Our main result of this chapter shows the equivalence of the experiment-based and UC security of ABS under standard requirements on the environment (Section 4.5). In detail, we first show that an ABS scheme that is secure regarding our experiment-based definitions (simulation privacy and unforgeability) implies UC security for the ABS scheme with minimal restrictions on the environment. Essential for this result is the simulation aspect of the privacy definition. We also show the reverse direction, i.e., a UC secure ABS scheme is secure with respect to our experiment-based security definitions. The proof techniques are based on the work of Abe and Ohkubo [AO12] for UC secure blind signature schemes. The main result highlights the applicability of our ABS security models even under such strong requirements such as in UC. Note, that for the main result we consider the experiment-based definitions (Chapter 3) of simulation privacy (Definition 3.2.5) and unforgeability (Definition 3.2.6), but not simulation-extractability (Definitions 3.2.7, 3.2.8). The reason for this is that (1) simulation-extractability is a strong requirement, (2) it is only known that schemes (argument systems) with black-box extraction (i.e., without rewinding, access to the code and randomness of the environment) are UC secure [Bag19; BPR20; CLOS02; GOS12; Kos+15]. In more detail, in UC one cannot assume to be able to rewind arbitrary environments or to have access to the code and randomness of the environment as it is required for white-box extactability. Regarding (1), non of the ABS schemes from the literature is proven to be simulation-extractable. Our generic construction in Section 3.3 is the first. Hence, basing our main result on simulation-extractable ABS is in conflict with our goal to show that existing schemes are UC secure.

We can also use the same approach as we used for simulation-privacy and show that existing schemes fulfill simulation-extractable. However, close inspection of the building blocks of the existing ABS schemes reveals that they do not fulfill simulation-extractability. Furthermore, if we also consider (2) the approach is even more limiting, since then we have to limit the main result to ABS schemes that are black-box extractable, i.e., straight-line extractable [Fis05]. Consequently, we use the less limiting experiment-based security definitions (simulation privacy and unforgeability) such that our main result is applicable to as many ABS schemes from the literature as possible.

4.2 Preliminaries of UC

The goal of the following detailed introduction is to present all necessary preliminaries to define security in UC. To do this formally, we introduce important preliminaries for protocols and their execution in UC. As mentioned above, the UC framework has a challenging goal: to enable universal composition of protocols while guaranteeing the security of the individual protocols. Therefore, the UC framework consists of many definitions that build on each other. We do not include all the definitions and details given by Canetti [Can20], since this is out of scope. We rather give intuitive descriptions and repeat formal definitions, if they are essential for the understanding and definition of security in UC. For the full formal treatment of UC we refer to [Can20]. In the following we describe the model of computation, how protocols are executed and emulation of protocols to finally define security. Let us start with the model of computation in the UC framework to understand what happens in the background, if we write that a party does something in a protocol or a party gets activated in an execution.

4.2.1 Model of Computation

A protocol is modeled as an interactive Turing machine (ITM) that extends the notion of Turing machines by interaction with other interactive Turing machines via dedicated communication tapes and a read only identity tape, see [Can20, Definition 4] for details. The identity tape of an ITM μ contains the program description of μ and an (unique) identity string id. Formally the two strings form the extended identity of μ. An interactive Turing machine instance (ITI) is denoted as $M = (\mu, id)$, where μ in this context refers to the program description. An activation (computation) of an ITI $M = (\mu, id)$ is a computation of program μ [Can20]. A party in UC is an ITI. Model details such as the extended identity and communication tapes are used implicitly in this work. For example we refer to parties via a unique name, by using that parties know their program (and identity) and are able to communicate with each other. Concretely, if we write that one party sends a message to another party then this means that an ITI writes the message on a communication tape of another ITI.

We skip defining the detailed semantics of executing ITMs in the UC by Canetti, since it is out of scope. We refer for the full formal treatment to [Can20] and only recap important formal definitions and give intuitions for the semantics where possible. The following descriptions are based on the definitions and descriptions given in [Can20]. Execution of ITMs is modeled via a system of ITIs $(\mathcal{E}, C)$ consisting of an initial ITM $\mathcal{E}$ and a control function C. The control function C determines programs of ITIs and if an information flow between ITIs is allowed or not. Concretely, the initial ITM is the environment and the control function, among others, encodes the adversary and the protocol program.

An execution is thereby a sequence of activations of ITIs guided by the control function. The questions here is, how can it be a system of ITMs, if it just starts with an initial ITM. The technique here is that an ITI of the initial ITM (and any other to be precise) can invoke new ITIs. The mechanism that allows this is the same that is used for the communication of ITIs. Intuitively, if a message is sent to an ITI $M' = (\mu', id')$ that so far did not exist in the system, then a new ITI M' with code μ' and identity id' is created. If the ITI M' does exist, it just receives the message (via a write operation on a tape) and is activated next. The execution of the system ends with the initial ITI halting and the final output written on a tape. Later we will use a random variable describing the final output of an execution on some input to define security in UC.

Now that the formal model of computation is set, let us relate it to the informal description of UC given before. A protocol in UC is defined as a ITM. If a protocol has roles (for different parties) then the program of the protocol ITM consists of programs for each of the roles. For example, a party that is responsible for the setup role has its own program different from the program for a party that has the role of a signer. To bind ITIs, invoked by the initial protocol ITM, together in one session of a protocol, UC uses that an identity of an ITI consists of two strings: the session identifier (sid) and the actual party identifier. Hence, ITI with the same session identifier form a session of a protocol. Hence, we can now concretely state, that, if we write party P_i, we actually refer to an ITI with a party identifier P_i and a session identifier sid. Collecting the above, we now can say that parties in an execution of a UC protocol are ITIs and activations of parties are executed in a system of ITMs. Still open is some form of resource bound for the system to be useful in cryptography.

4.2.2 Bounded Computation

As described in [Can20], the definition of bounded computation in UC is a combination of two notions. It combines the notion of a function in the input length to capture how many computational steps are required, with the notion of what can be computed given a specific bound. Intuitively, runtime is handled in UC by a budget of runtime tokens. There is an initial budget of tokens and tokens are included in every message exchanged. This is called the import of a message. Therefore, receiving a message gives some extra runtime by the token included

in the import. Hence, probabilistic polynomial-time in UC is with respect to the tokens received and tokens sent, such that the steps taken by every ITI in the system (protocol) is bounded by a polynomial in the difference of tokens received and sent. More specifically, to define ppt for ITMs, UC extends the notion of a T-bounded Turing machine that halts after at most $T(k)$ steps on an input with length k. The extension is to consider T-bounded ITMs that take at most $T(n)$ steps at any point of the execution of a system of ITMs. Here, n is defined to be $n \coloneqq n_{\text{Input}} - n_{\text{Output}}$, i.e., the difference of tokens received and sent. A ITM μ is said to be ppt, if there exists a polynomial q such that μ is q-bounded in the above sense. For more formal details confer [Can20, Section 3.2, Definition 6]. To bridge this ppt definition for ITMs to the experiment-based definition framework, UC considers ITMs that are parameterized with the security parameter λ. Intuitively, this means that the runtime token bucket is at least of size polynomial in the security parameter.

We skip here further discussions of the ITM model and parameterized security. For a detailed discussion including the mechanism for the unique identities, deletion and halting of ITIs, and a mechanism for the session identifier, we refer to [Can20, Section 3]. For our purpose it is sufficient to know that parameterizing the ITMs (protocols) with a security parameter captures the usual "for all *ppt* algorithms" model that cryptographers are more accustomed to in the framework of experiment-based definitions.

4.2.3 Protocol Execution

We have seen the formal background mainly to define probabilistic polynomial-time in UC. Furthermore, we have seen the ITM computational model of environment, adversary, protocol and parties. In UC we execute a system of ITMs $(\mathcal{E}, C_{\text{EXEC}}^{\rho,\mathcal{A}})$ for protocol ρ, environment $\mathcal{E}$, and adversary $\mathcal{A}$. Note, we only consider environments that are parameterized with the security parameter. As before, the initial ITM of the system is the environment $\mathcal{E}$ and the control function $C_{\text{EXEC}}^{\rho,\mathcal{A}}$ determines the program of all party ITIs to be ρ. Hence, every party in the system takes part in the same protocol. Further, control function $C_{\text{EXEC}}^{\rho,\mathcal{A}}$ regulates the communication of the adversary and the parties, e.g., it allows or disallows communication between parties. Intuitively this means, there is a control function that regulates communication, there is an environment that is the initiator of any activation and parties know their program code to run in the protocol. With this in place, we present a core concept on the road to defining security in UC.

4.2.4 UC-Emulation

An important concept, to formally define security, is emulation. To that end, we first define UC-emulation some preliminaries. Intuitively, if one protocol ρ UC-emulates another protocol ϕ it means that there is a simulator that translates an execution of protocol ρ to the other protocol ϕ. Anticipatory, for security we

consider that protocol ϕ is ideal and protocol ρ is real. However, before talking about security we define some preliminaries and UC-emulation first.

UC considers balanced environments and optionally identity-bounded environments. Identity-bounded environments are restricted to communicate with a limited set of parties. A detailed description of the latter is omitted, since this restriction is not needed in this work, i.e., formally the identity-bound is $\{0,1\}^*$ and we leave it out of the formal definitions. Intuitively, this means that the environment can use any identity in a protocol. In turn, balanced environment is a core principle, since it circumvents unnatural situations that have no counter-part in a realistic scenario, but formally lead to problems, see [Cam+16a; KTR20] for details. Following [Can20] we only consider balanced environments ensuring that the adversary has enough import (i.e., runtime tokens) at any point of the execution to read all the inputs which are given by the environment to all other parties. This is to circumvent unnatural situations, where the adversary is not able to process messages sent by the parties or to send messages to other parties, just because his computational resources (i.e., runtime tokens) are exhausted. In formal definitions we mention that the environment is balanced, but for readability we omit it in textual description.

Next, in preparation of the emulation definition, let us define two binary random variables over the internal randomness of all machines involved. For ppt environment $\mathcal{E}$, ppt protocols ρ, ϕ, ppt adversaries $\mathcal{A}$, $\mathcal{S}$, and q polynomial in λ:

- Let $\mathcal{E}[\rho, \mathcal{A}](\lambda, z)$ be a binary random variable describing the output of environment $\mathcal{E}$ of an execution with protocol ρ, adversary $\mathcal{A}$ and security parameter $\lambda \in \mathbb{N}$ on input $z \in \{0,1\}^{q(\lambda)}$ and
- let $\mathcal{E}[\phi, \mathcal{S}](\lambda, z)$ be a binary random variable describing the output of environment $\mathcal{E}$ of an execution with protocol ϕ, adversary $\mathcal{S}$, and security parameter$\lambda \in \mathbb{N}$ on input $z \in \{0,1\}^{q(\lambda)}$.

Now, we are ready to formally define UC-emulation.

Definition 4.2.1 (UC-Emulation - [Can20, Definition 9])
Let ρ and ϕ be ppt protocols. We say that ρ UC-emulates ϕ, if for any ppt adversary $\mathcal{A}$ there exists a ppt simulator $\mathcal{S}$ such that for any balanced ppt environment $\mathcal{E}$ there exists a negligible function negl such that for all inputs $z \in \{0,1\}^{q(\lambda)}$ it holds

$$|\Pr[\mathcal{E}[\phi, \mathcal{S}](\lambda, z) = 1] - \Pr[\mathcal{E}[\rho, \mathcal{A}](\lambda, z) = 1]| \leq \mathsf{negl}(\lambda).$$

$\diamond$

In Definition 4.2.1 the ppt environment $\mathcal{E}$ acts as an interactive distinguisher that outputs either 0 or 1. Also, there are two adversarial machines, the environment and the adversary $\mathcal{A}$. The simulator $\mathcal{S}$ has a special role that we describe later. Considering the environment $\mathcal{E}$ and adversary $\mathcal{A}$ separately renders the intuition

and formal proofs more difficult, since the simulator depends on the adversary $\mathcal{A}$. With the definition above we have to present for every adversary $\mathcal{A}$ a customized simulator $\mathcal{S}$. Fortunately, Canetti shows in [Can20] two additional and equivalent definitions for UC-emulation: one with a dummy adversary and another with a black-box simulator. Here, the dummy adversary is a simplified adversary that just passes on any messages that the environment exchanges with any party. Canetti [Can01] shows that the dummy adversary is the strongest adversary in UC, since $\mathcal{E}$ takes over the responsibilities of $\mathcal{A}$ and has therefore control over the communication with the protocol. For more details see [Can20, Section 4.3.1].

Let us next describe the special role of the simulator. The simulator $\mathcal{S}$ is also called the ideal adversary in the literature. The special role of the simulator $\mathcal{S}$ is to translate everything that can happen in the real protocol with adversary $\mathcal{A}$ to the ideal protocol. To this end, there is a special backdoor communication mechanism that allows the simulator and ideal functionality $\mathcal{F}$ to directly communicate with each other. The backdoor messages are used in ideal functionalities to ask the simulator for help and to model information that leaks to the adversary in a real protocol. In Section 4.2.7 describe how these backdoor messages are handled by so called responsive environments.

In this work, we formally use the UC-emulation with black-box simulator. We have to note that a black-box simulator $\mathcal{S}^{\mathcal{A}}$ in UC just has the adversary $\mathcal{A}$ as a subroutine without access to the program, internal state or random tape of adversary $\mathcal{A}$. Therefore, the protocol execution is as defined above. As a consequence, black-box simulation in UC is without rewinding access to the adversary $\mathcal{A}$. The reason for this is that the adversary $\mathcal{A}$ still exchanges messages with the environment and rewinding then also means rewinding the environment. Otherwise, the environment can easily recognize a rewind of adversary $\mathcal{A}$ and detect an execution with simulator $\mathcal{S}^{\mathcal{A}}$. In conclusion, rewinding the environment is an unrealistic model. Let us next define UC-emulation with a black-box simulator.

Definition 4.2.2 (UC-Emulation with Black-Box Simulator - [Can20, Def. 12]) Let ρ and ϕ be ppt protocols. We say that ρ UC-emulates ϕ, if there exists a ppt (black-box) simulator $\mathcal{S}^{(\cdot)}$ such that for any ppt adversary $\mathcal{A}$ and any balanced ppt environment $\mathcal{E}$ there exists a negligible function negl such that for all inputs $z \in \{0,1\}^{q(\lambda)}$ it holds:

$$\left|\Pr\left[\mathcal{E}[\phi, \mathcal{S}^{\mathcal{A}}](\lambda, z) = 1\right] - \Pr\left[\mathcal{E}[\rho, \mathcal{A}](\lambda, z) = 1\right]\right| \leq \mathsf{negl}(\lambda) \ .$$

$\diamond$

The following equivalence between the UC-emulation definitions is formally proven in [Can20].

Theorem 4.2.1 (UC-Emulation Equivalence - [Can20, Claim 13])**.** *Let ρ and ϕ be ppt protocols, then ρ UC-emulates ϕ as in Definition 4.2.1 if and only if ρ*

UC-emulates ϕ with black-box simulation as in Definition 4.2.2.

Using UC-emulation (black-box simulator) we define UC security in the following.

4.2.5 UC Security

Roughly, security in UC is defined via a real protocol that UC-emulates an ideal protocol, this process is called realization of an ideal functionality. Let us first describe preliminaries regarding the structure of protocols. A common theme in UC is that there are many model details that are important solely for the security analysis and some of them have no real-world counterpart. To give protocol designers a level of abstraction of these model details the UC framework defines structured protocols. A structured protocol is a shell around a protocol, called the body. The shell code contains most of the code necessary for the model details and the body protocol itself can be defined by protocol designers more directly. The shell is therefore responsible for backdoor messages exchanged between the simulator and the ideal functionality, communication in general and corruption of parties, among others. Furthermore, UC as presented in [Can20] considers *subroutine respecting* protocols to circumvent runtime and communication problems that were present in previous versions. Basically, a subroutine respecting protocol restricts the communication to protocol parties and their subroutines in one session. Thus eliminating unwanted inputs from other protocols (and sessions) that might be executed in the environment. This makes it easier to define protocols, since otherwise protocol designers have to specify how to react on any incoming messages. This can be compared to experiment-based security definitions. Where in one run of the experiment we usually only consider the security of the current session defined by the setup of the experiment. The formal details of both mechanisms go to far and for our purpose we assume structured protocols and use the sandbox shell given in [Can20] to handle the model details and to get subroutine respecting protocols. This means we only define the body protocol and implicitly apply the sandbox shell.

We formally define security via realizing ideal functionalities in the following. Let us repeat what an ideal protocol is. An ideal protocol $\text{IDEAL}_{\mathcal{F}}$ consists of multiple dummy parties and an ideal functionality $\mathcal{F}$. A dummy party just forwards all input to the ideal functionality, all outgoing messages to the specified recipient, and outputs to the environment. Dummy parties exists to syntactically match the real protocol, where parties execute their own program code. The ideal functionality is a machine modeled as a trusted party that describes the ideal process including state information, and input-output behavior. The simulator $\mathcal{S}$ and the ideal functionality $\mathcal{F}$ communicate via a special backdoor tape (modeled in the shell). The formal ITM details of the ideal functionality are omitted here and we refer to [Can20, Section 5.3] for the details.

Definition 4.2.3 (Realizing Ideal Functionalities - [Can20, Definition 20])
Let $\mathcal{F}$ be an ideal functionality, ρ be a protocol and let $\text{IDEAL}_{\mathcal{F}}$ be the ideal protocol for $\mathcal{F}$. We say that ρ UC-realizes $\mathcal{F}$, if ρ is subroutine respecting, and ρ UC-emulates $\text{IDEAL}_{\mathcal{F}}$. ◇

Intuitively, the above definition captures correctness and security. Correctness as defined by the ideal functionality transfers to the realizing protocol (and vice versa), because the output of the environment has to be indistinguishable in both, ideal and real, protocol executions. Regarding security, the indistinguishability of the (ideal and real) protocol executions gives us also that any attack against the (real) protocol ρ by an adversary is translated to an attack against the ideal protocol $\text{IDEAL}_{\mathcal{F}}$ by a simulator. Similarly, if any information is leaked to an adversary in ρ it is translated to information provided by the simulator in the ideal protocol $\text{IDEAL}_{\mathcal{F}}$. The above definition captures security of (single) protocols referred to as realization of an ideal functionality. This is not the universal composability that UC promises, however it is an essential part.

4.2.6 Universal Composition

The core of the universal composability framework is that the universal composability theorem presented in [Can20] states that, if a ppt protocol ρ UC-emulates ppt protocol ϕ, then for any ppt protocol ζ, that uses ϕ as a subroutine, the composed protocol $\zeta^{\phi \to \rho}$ UC-emulates protocol ζ. Here $\phi \to \rho$ means that any call to subprotocol ϕ in protocol ζ is replaced by a call to ρ. The called hybrid-model in the UC framework make use of this composed protocol principle. In detail, a security analysis in the hybrid-model can be simplified by working with protocol ζ that calls an ideal protocol internally and focus the analysis on what ζ adds to the composed protocol. The universal composability theorem takes care of the rest, i.e., securely replacing the ideal protocol with a real protocol. This is a small example and can be extended to multiple subprotocols.

We only state the formal universal composability theorem by Canetti [Can20] for completeness in the following. However, we skip detailed description, further formal preliminary definitions (e.g., compliant protocols, subroutine exposing), and the proof of the universal composability theorem since they are out of scope and better presented in context in [Can20, Section 6].

Theorem 4.2.2 (Universal Composition - [Can20, Theorem 22])**.** *Let ρ, ϕ, and ζ be ppt protocols and let ξ be a ppt predicate on identities, such that ζ is (ρ, ϕ, ξ)-compliant, protocol ϕ and ρ are subroutine respecting and subroutine exposing, and ρ ξ-UC-emulates ϕ. Then protocol $\zeta^{\phi \to \rho}$ UC-emulates protocol ζ.*

Roughly, (ρ, ϕ, ξ)-compliant means that there is a shell code that enforces that ρ and ϕ behave the same, i.e., write operations of instances of ρ and ϕ are either for ρ or ϕ and never mixed. Also instances do not leak which protocol they are executing and all extended identities of all communicating instances of ρ and ϕ satisfy the

predicate ξ. Hence, ξ can be used to limit the set of instances (and identities) that are allowed to communicate. For the formal definition of compliant protocols see [Can20, Section 6.1]. A protocol is subroutine exposing, if the protocol on a query of an identity *id* by the (real or ideal) adversary answers if *id* is part of the session of the protocol. This requirement is again provided by UC with a shell code, for details we refer to [Can20, Definition 21].

4.2.7 UC Framework Utilities

In the following, we present some conventions of how to write ideal functionalities and utilities to deal with standard tasks, such as corruption and authenticated channels for the communication of the parties. To this end, Canetti [Can20, Section 7] presents mechanisms and modeling details that we can use out of the box. Let us introduce some notation before we explain the mechanisms that we use in this work: For environment $\mathcal{E}$, ideal functionality $\mathcal{F}$, simulator $\mathcal{S}^{\cdot)}$, protocol ρ, and adversary $\mathcal{A}$, we call an execution $\mathcal{E}[\mathcal{F}, \mathcal{S}^{(\cdot)}]$ the ideal setting (Figure 4.1) and with $\mathcal{E}[\rho, \mathcal{A}]$ the real setting (Figure 4.2). Here, we omit the security parameter λ and input z for $\mathcal{E}$ to simplify notation.

Corruption, erasure and secure channels. In UC parties can be corrupted by the adversary $\mathcal{A}$, which means that the adversary $\mathcal{A}$ takes control of the party. Hence, in the real setting a corrupted party cannot be assumed to act honestly (i.e., follow the protocol). In the ideal setting parties are just dummy parties. Hence, to show that a protocol UC-realizes an ideal functionality the simulator needs to handle corruptions by the adversary $\mathcal{A}$. This means the simulator has to provide $\mathcal{A}$ with the same information that it gets by a corruption in the real setting. [Can20] presents variants of corruption model and which information the adversary gets after a corruption. Our corruption model is based on adaptive corruptions. Adaptive corruptions allow the adversary $\mathcal{A}$ to trigger a corruption of a party at any time. Adversary $\mathcal{A}$ is in full control of a corrupted party. Further, we consider the erasure model [Can20], denoted as *erasure*, where honest parties erase ephemeral randomness immediately after usage. This means that on corruption of a party the adversary $\mathcal{A}$ only gets the result of previous computations, but not the randomness used. This models the real world, where usually used randomness is not stored. For the setting of ABS, we restrict the corruption, such that the party responsible for the setup of the ABS scheme can only be corrupted after an honest setup was completed. In a corruption of the setup party we only output the public parameters and master secret key, and not the simulation trapdoor. This is not a strong restriction, in the sense that it matches the ABS security definitions. There, we have that the adversary, in case of privacy (Definition 3.2.4, Definition 3.2.5), indeed gets the master secret key, but obviously not the simulation trapdoor which is only available in one of the experiment settings. Intuitively, it would be trivial to distinguish the settings by only getting the simulation trapdoor in the simulation setting. This restriction is denoted as *adaptive* and is explained in more detail in

Section 4.5.

For communication between the parties we assume authenticated secure channels, denoted as *secure channels*, such that the adversary is not able read secret keys and has to deliver them unmodified. For details of how to realize this assumption we refer to Canetti [Can20]. Overall, our results are shown in an *adaptive*, *erasure*, and *secure channels* model for an ideal functionality $\mathcal{F}$, denoted as $\mathcal{F}$[*adaptive, erasure, secure channels*].

Delayed output. Delayed output is a feature of the UC framework that allows the simulation of real-world transmission delay, e.g., network traffic, and allows the involvement of the adversary and simulator, e.g., adversarial withholding of messages. Hence, any problems with messages sent over the network is handled by this UC framework feature and we do not have to deal with all the details in the ideal functionality.

In detail, the ideal functionality instead of sending output x directly to the recipient party P, it informs the simulator (think of an ideal adversary) that it wants to output x to P over the backdoor tape, including a unique identifier. On an acknowledgement of the simulator the ideal functionality delivers the output x to P. Importantly, there are two modes: *public* and *private*. If it is a *public delayed output*, the output x is transmitted in the clear to the simulator. If it is a *private delayed output*, the output x is not included in the transmission to the simulator. This corresponds to transmitting secrets to a party (over a secure channel). Hence, the simulator, in its simulation, does not include the secret output x, as it is also not available in the real protocol, because of secure channels. The simulator acknowledges a delayed output by responding with the corresponding unique identifier via the backdoor tape.

We use this feature to inform the simulator about the intended output, where one of the involved parties is corrupted. In this situation, the simulator is then able to execute further steps that simulate the steps of a real-world protocol with a corrupted party. In general this allows it to simulate network delay, involvement of the adversary via corrupted parties, and computations that are necessary for the simulation.

Simulation and message exchange. In UC the ideal functionality and simulator often exchange some messages via a backdoor tape for modeling reasons. Typically protocol designers assume that the simulator answers immediately on such messages, where according to the UC framework the simulator can do anything it likes (activating other parties and even protocols) after receiving a message from the ideal functionality. This introduces state changes of parties and unexpected behavior of the ideal functionality that were not anticipated by the protocol designer. Therefore, the handling of the exchanged messages, and responses of the simulator makes the description of the ideal functionality unnecessarily more complex. Then again, the side effects of such message exchanges cannot be ignored. As Camenisch

et al. highlight in [Cam+16a] such messages can lead to runtime problems. To circumvent the problems the authors [Cam+16a] define responsive environments, where the adversary, simulator, and environment are bound to answer to so called restricting messages immediately. We model our ideal functionality with responsive environments. Therefore, we employ the generic restriction defined by Camenisch et al. [Cam+16a], where every message prefixed with "`Respond`" is restricting.

Writing ideal functionalities: process and challenges. Besides the already mentioned details of UC, there are some technical challenges that come with the definition of ideal functionalities. Already simple ideal functionalities, like the ideal signatures [Can03], have some details moved to the simulator that seem like they belong in the ideal functionality at first. This peculiarity of the UC framework comes even more pronounced in complex ideal functionalities. This also applies to our ideal functionality for attribute-based signatures.

The process of defining ideal functionalities can be described as a three step process. (1) The definition starts with just the core ideal process of a scheme. For example, we want to sign message-policies pairs with attributes. Hence, we define three activations: Setup, Key Generation, Signature, and Verify corresponding to the syntax of ABS (Definition 3.2.1). Then, we have to enable parties to activate the ideal functionality for a key generation on attributes and we have to store the result. To enable parties to sign a message-policy pair with attributes, via the Signature activation, we just have to look up, if there was a key generation result with these attributes and that they satisfy the policy. As a result of this, we mark the message-policy pair as validly signed. (2) The next step of the ideal functionality definition process is to deal with corrupted parties. This usually requires the involvement of the simulator, since if we departure from the ideal process the concrete steps typically depend on the real protocol. Hence, the ideal functionality asks the simulator to deal with the details of activations from corrupted parties and the ideal functionality is extended with steps that handle the results given by the simulator. Extending just one activation is often not enough. A key generation with a corrupted party affects the Signature activation and ultimately the Verify activation. (3) The last step of the definition process, is the consideration of the real protocol, since eventually we want to show a real protocol that UC-realizes the ideal functionality. This last step is, challenging since we have to consider all three components: ideal functionality, simulator, and real protocol, at once. For example a change in the input-output behavior of the real protocol either has to be simulatable by the simulator or leads to changes in the ideal functionality. In conclusion, this explains why ideal functionalities, simulators and partially the real protocols are so convoluted in UC and describes the process of defining an ideal functionality.

4.3 Ideal Attribute-Based Signatures Functionality

In this section, we formally define the ideal functionality for attribute-based signature, describe the security properties that it provides and compare them intuitively to the experiment-based security of ABS (Section 3.2). Subsequently, we present the definition of a real protocol for ABS that wraps an ABS scheme and transforms it to the UC protocol model.

The formal definition of the ideal functionality $\mathcal{F}_{\mathsf{ABS}}$ for ABS with access to a simulator $\mathcal{S}$ is presented in Figure 4.3. The ideal functionality for ABS models a scheme with an unique setup party P_{Setup} and arbitrary parties P_i that can query for secret keys, sign message-policy pairs and verify signatures via the corresponding activations. Note, all activations are instantiated by the environment $\mathcal{E}$. Hence, after a Setup activation through P_{Setup} was completed, any party P_i can ask for a secret key on attributes $\mathbb{A}$ using the Key Generation activation. A party P_i can generate a signature on a message-policy pair by a Signature activation and it can verify a signature through a Verify activation.

Let us first explain the notation used in the ideal functionality $\mathcal{F}_{\mathsf{ABS}}$, before explaining the definition of $\mathcal{F}_{\mathsf{ABS}}$ in more detail. We use "record $\langle$`keyword`, entry$_1$, ..., entry$_l\rangle$" to denote that the protocol (ideal functionality) stores the element (entry$_1$, ..., entry$_l$) via a list with the name `keyword` in state. The element (entry$_1$, ..., entry$_l$) is then a list element and called a record. We assume that new records are appended to the list (the latest record is the last in the list) and that records can be looked up and added in polynomial-time. To look up a record we use the notation $\langle$`keyword`, entry$_1$, ..., entry$_i$, $\cdot$, entry$_{i+2}$, ..., entry$_l$ $\rangle$ where $\cdot$ is a wildcard that can be mapped to any entry. During recording we denote by $\perp$ a placeholder for an entry that we might insert via an update later on.

4.3.1 Description of Ideal ABS Functionality $\mathcal{F}_{\mathsf{ABS}}$

Before we describe each activation of $\mathcal{F}_{\mathsf{ABS}}$ (Figure 4.3) in detail, we start with the explanation of general UC mechanisms for the definition of ideal functionalities that we use. After the detailed description of the activations, we focus on the security guarantees of $\mathcal{F}_{\mathsf{ABS}}$.

In the UC framework each ideal functionality instance has a unique session identifier *sid* to identify multiple instances of the same functionality and to know which instance is responsible for which message. Therefore messages include the *sid*. The session identifier *sid* can be determined in several ways (cf. [Can20], Section 3.1.3). For simplicity we let the first Setup activation determine the *sid*, which consists of the party's identifier and a unique session identifier *sid*$'$. Therefore, the *sid* also determines the unique party responsible for the setup and key generation and every party in the same session (same *sid*) knows the setup party. An instance of the functionality stores the *sid* in the first Setup activation and ignores all activations with a different *sid*$'$. In general, the inputs for the Setup, Key Generation, Signature and Verify activations are determined by the environment $\mathcal{E}$. Next, let us explain

Setup: On input $(\texttt{Setup}, sid)$ from a party P_{Setup}

1. If $sid = (P_{\mathsf{Setup}}, sid')$ for some sid' continue, else ignore.
2. If there is no record $\langle \texttt{SetupParams}, sid', P_{\mathsf{Setup}}, pp, msk, td_{sim} \rangle$, send $(\texttt{Respond}, (\texttt{Setup}, sid))$ to $\mathcal{S}$. Upon receiving $(\texttt{Setup}, sid, \mathcal{S}.\mathsf{Setup}, \mathcal{S}.\mathsf{KeyGen}, \mathcal{S}.\mathsf{Sign}, \mathcal{S}.\mathsf{Verify})$ from $\mathcal{S}$ do:
   a) Generate $(pp, msk, td_{sim}) \leftarrow \mathcal{S}.\mathsf{Setup}$
   b) Record $\langle \texttt{SetupParams}, sid', P_{\mathsf{Setup}}, pp, msk, td_{sim} \rangle$ and $\langle \texttt{Algorithms}, \mathcal{S}.\mathsf{Setup}, \mathcal{S}.\mathsf{KeyGen}, \mathcal{S}.\mathsf{Sign}, \mathcal{S}.\mathsf{Verify} \rangle$.
3. Else, use recorded $\langle \texttt{SetupParams}, sid', P_{\mathsf{Setup}}, pp, msk, td_{sim} \rangle$.
4. In both cases output $(\texttt{PublicParams}, sid, pp)$ to P_{Setup}.

Key Generation: On input $(\texttt{KeyGenRequest}, sid, kid, \mathbb{A})$ from party P_i

1. If $sid = (P_{\mathsf{Setup}}, sid')$ continue, else ignore.
2. Record the request as $\langle \texttt{KeyGenRequest}, sid, kid, \mathbb{A}, P_i, b' \coloneqq 0 \rangle$ with $b' \coloneqq 0$ to mark it unprocessed.
3. Send public delayed output $(\texttt{KeyGenRequest}, sid, kid, pp, \mathbb{A}, P_i)$ to P_{Setup}

Key Generation: On input $(\texttt{KeyGen}, sid, kid)$ from party P_{Setup}

1. If $sid = (P_{\mathsf{Setup}}, sid')$ continue, else ignore.
2. If there is an unprocessed record $\langle \texttt{KeyGenRequest}, sid, kid, \mathbb{A}, P_i, b' = 0 \rangle$ from party P_i, mark it processed by setting $b' = 1$ and continue. Else ignore.
3. If P_{Setup} is corrupt then record $\langle \texttt{KeyGen}, sid, kid, \bot, \bot, \mathbb{A}, \bot \rangle$ and send as a public delayed output $(\texttt{KeyGen}, sid, kid, \mathbb{A})$ to P_i.
4. Else, if $\langle sid', P_{\mathsf{Setup}}, pp, msk, td_{sim} \rangle$ is recorded proceed, else ignore.
   a) Generate secret key $sk_{\mathbb{A}} \leftarrow \mathcal{S}.\mathsf{KeyGen}(pp, msk, td_{sim}, \mathbb{A})$ and record $\langle \texttt{KeyGen}, sid, kid, \bot, pp, \mathbb{A}, sk_{\mathbb{A}} \rangle$. Send $(\texttt{Respond}, (\texttt{KeyGen}, sid, kid, \mathbb{A}, P_i, sk_{\mathbb{A}}))$ to $\mathcal{S}$.
   b) Upon receiving $(\texttt{KeyGen}, sid, kid, \mathbb{A}, P_i, sk_{\mathbb{A}}, 1)$ from $\mathcal{S}$ send $(\texttt{KeyGen}, sid, kid, \mathbb{A})$ as a public delayed output to P_i.
5. In both cases, when $(\texttt{KeyGen}, sid, kid, \mathbb{A})$ is delivered to P_i update the record by adding the party P_i to $\langle \texttt{KeyGen}, sid, kid, P_i, \cdot, \mathbb{A}, \cdot \rangle$.

Figure 4.3: Ideal attribute-based signatures functionality $\mathcal{F}_{\mathsf{ABS}}$.

Signature: On input $(\texttt{Signature}, sid, kid, pp', \mathbb{A}, m, \mathbb{P})$ from some party P_i

1. If $sid = (P_{\mathsf{Setup}}, sid')$, continue, else ignore.
2. Look up latest record $\langle \texttt{KeyGen}, sid, kid, P_i, pp', \mathbb{A}, \cdot \rangle$,
   a) if none exists ignore,
   b) else if $\mathbb{P}(\mathbb{A}) = 0$, record $\langle \texttt{Signature}, sid, kid, pp', \mathbb{A}, m, \mathbb{P}, \sigma := \bot, 0 \rangle$ and output $(\texttt{Signature}, sid, kid, \mathbb{A}, m, \mathbb{P}, \sigma := \bot)$ to P_i,
   c) else it holds that $\mathbb{P}(\mathbb{A}) = 1$ and proceed:
3. If $pp' = pp$ for recorded pp,
   a) generate $\sigma \leftarrow \mathcal{S}.\mathsf{Sign}(pp, msk, td_{sim}, m, \mathbb{P})$.
   b) if a record $\langle \texttt{Signature}, sid, kid, pp', \mathbb{A}, m, \mathbb{P}, \sigma, 0 \rangle$ exists, output error and halt.
   c) else, record valid signature $\langle \texttt{Signature}, sid, kid, pp', \mathbb{A}, m, \mathbb{P}, \sigma, f := 1 \rangle$
4. If $pp' \neq pp$, send $(\texttt{Respond}, (\texttt{Signature}, sid, kid, pp', \mathbb{A}, m, \mathbb{P}))$ to $\mathcal{S}$ and receive the answer $(\texttt{Signature}, sid, kid, pp', \mathbb{A}, m, \mathbb{P}, \sigma)$ from $\mathcal{S}$.
5. Output $(\texttt{Signature}, sid, kid, \mathbb{A}, m, \mathbb{P}, \sigma)$ to P_i.

Verify: On input $(\texttt{Verify}, sid, pp', m, \mathbb{P}, \sigma)$ from some party P_i

1. Compute $b \leftarrow \mathcal{S}.\mathsf{Verify}(pp', m, \mathbb{P}, \sigma)$
2. If $pp' = pp$ for recorded pp then
   a) If $(\texttt{Signature}, sid, kid, pp, \mathbb{A}, m, \mathbb{P}, \sigma, f)$ recorded for any f, then set output bit $f_{\text{out}} := f$
   b) Else, if for any σ' record $\langle \texttt{Signature}, sid, kid, pp, \mathbb{A}, m, \mathbb{P}, \sigma', 1 \rangle$ exists or P_{Setup} is corrupt or there exists at least one corrupted P_i with a record $\langle \texttt{KeyGen}, sid, kid, P_i, pp, \mathbb{A}', \cdot \rangle$ where $\mathbb{P}(\mathbb{A}') = 1$, then set output bit $f_{\text{out}} := b$ and record $\langle \texttt{Signature}, sid, kid := \bot, pp, \mathbb{A} := \bot, m, \mathbb{P}, \sigma, f_{\text{out}} \rangle$
   c) Else, set output bit $f_{\text{out}} := 0$ and record $\langle \texttt{Signature}, sid, kid := \bot, pp, \mathbb{A} :=, m, \mathbb{P}, \sigma, 0 \rangle$.
3. If $pp' \neq pp$ for recorded pp then
   a) If record $\langle \texttt{Signature}, sid, kid, pp', \mathbb{A}, m, \mathbb{P}, \sigma, f \rangle$ exists, then set output bit $f_{\text{out}} := f$
   b) Else set output bit $f_{\text{out}} := b$ and record $\langle \texttt{Signature}, sid, kid := \bot, pp', \mathbb{A} := \bot, m, \mathbb{P}, \sigma, f_{\text{out}} \rangle$
4. Output $(\texttt{Verified}, sid, m, \mathbb{P}, \sigma, f_{\text{out}})$ to P

Figure 4.3: Ideal attribute-based signatures functionality $\mathcal{F}_{\mathsf{ABS}}$ (continued).

the records that ideal functionality $\mathcal{F}_{\mathsf{ABS}}$ stores.

`SetupParams`: records the setup parameters generated during the first setup activation `Setup` as a record

$$\langle \texttt{SetupParams}, sid', P_{\mathsf{Setup}}, pp, msk, td_{sim} \rangle$$

with session identifier sid', setup party identity P_{Setup}, public parameters pp, master secret key msk, and simulation trapdoor td_{sim}. This enables the ideal functionality $\mathcal{F}_{\mathsf{ABS}}$ to look up recorded setup parameters, e.g., to determine if an activation is for the recorded public parameters (corresponds to one instance).

`Algorithms`: records the algorithms $\mathcal{S}.\mathsf{Setup}$, $\mathcal{S}.\mathsf{KeyGen}$, $\mathcal{S}.\mathsf{Sign}$, and $\mathcal{S}.\mathsf{Verify}$ output by the simulator during the first setup activation `Setup` as a record

$$\langle \texttt{Algorithms}, \mathcal{S}.\mathsf{Setup}, \mathcal{S}.\mathsf{KeyGen}, \mathcal{S}.\mathsf{Sign}, \mathcal{S}.\mathsf{Verify} \rangle.$$

This enables the ideal functionality $\mathcal{F}_{\mathsf{ABS}}$ to use the algorithms in its activations.

`KeyGenRequest`: records a key generation of a party P_i as a record

$$\langle \texttt{KeyGenRequest}, sid, kid, \mathbb{A}, P_i, b' \rangle$$

with session identifier sid, key identifier kid, attributes $\mathbb{A}$, party identity P_i, and bit $b' \in \{0,1\}$ ($0 \coloneqq$ unprocessed, $1 \coloneqq$ processed). This enables the ideal functionality $\mathcal{F}_{\mathsf{ABS}}$ to store key generation requests that are yet to be answered by a key generation by setting $b' \coloneqq 0$ and by setting $b' \coloneqq 1$ the ideal functionality $\mathcal{F}_{\mathsf{ABS}}$ marks the request as processed and therefore answered.

`KeyGen`: records a generated key for party P_i as a record

$$\langle \texttt{KeyGen}, sid, kid, P_i, pp, \mathbb{A}, sk_{\mathbb{A}} \rangle$$

with session identifier sid, key identifier kid, party identity P_i, public parameters pp, attributes $\mathbb{A}$, and secret key $sk_{\mathbb{A}}$ for attributes $\mathbb{A}$. This enables the ideal functionality $\mathcal{F}_{\mathsf{ABS}}$ to store key generations and to look up key generations of a party in a signature activation.

`Signature`: records signature and its validity as a record

$$\langle \texttt{Signature}, sid, kid, pp', \mathbb{A}, m, \mathbb{P}, \sigma, f \rangle$$

with session identifier sid, key identifier kid, public parameters pp', attribute $\mathbb{A}$, message m, policy $\mathbb{P}$, signature σ (can be $\perp$), and validity bit $f \in \{0,1\}$ ($0 \coloneqq$ invalid, $1 \coloneqq$ valid). This enables the ideal functionality $\mathcal{F}_{\mathsf{ABS}}$ to store

(valid and invalid) signatures and to answer verify activations, e.g., $\mathcal{F}_{\mathsf{ABS}}$ is able to answer verify activations consistently.

Next, we explain the activations of $\mathcal{F}_{\mathsf{ABS}}$ (Figure 4.3) to give an intuitive understanding of the ideal functionality. The explanation follows and references the steps of Figure 4.3. Hence, we recommend to read the explanations and Figure 4.3 alternately.

Setup. In the Setup activation the simulator $\mathcal{S}$ is responsible for providing ppt algorithms $\mathcal{S}.\mathsf{Setup}$, $\mathcal{S}.\mathsf{KeyGen}$, $\mathcal{S}.\mathsf{Sign}$, and $\mathcal{S}.\mathsf{Verify}$. These have to be stateless ppt algorithms such that the outputs of $\mathcal{F}_{\mathsf{ABS}}$ generated with these algorithms are independent of the internal state of the simulator and previous activations. Further, this modeling allows us to give a technically sound equivalence proof in Section 4.5. The algorithm $\mathcal{S}.\mathsf{Setup}$ is used to generate and fix the public parameters pp, the master secret key msk, and the simulation trapdoor td_{sim} of the functionality instance. The algorithm $\mathcal{S}.\mathsf{KeyGen}$ and $\mathcal{S}.\mathsf{Sign}$ always take as input the recorded (pp, msk, td_{sim}). We give an explicit input to highlight that we use the recorded elements.

Key Generation. The Key Generation activation models an exchange between a party P_i, that queries a key via `KeyGenRequest` on a given attribute vector $\mathbb{A}$, and the setup party P_{Setup} responsible for the actual key generation (`KeyGen`). In the `KeyGenRequest` activation from a party P_i the ideal functionality records the request $\langle \texttt{KeyGenRequest}, sid, kid, \mathbb{A}, P_i, b' \rangle$ and marks it as unprocessed by setting $b' := 0$. This bit is used in the corresponding key generation activation `KeyGen` of the setup party P_{Setup} to determine, if the key generation request was already answered or not. Note, the key identifier kid models that the requesting party (more specifically the environment) can use a unique key identifier kid for each request, e.g., to distinguish multiple secret keys for the same attributes. We do not enforce uniqueness of the key identifiers in the ideal functionality. Hence, in the a signature activation we use the latest recorded secret key. Before describing the signature activation in more detail let us explain the key generation of P_{Setup} in more detail.

The `KeyGen` activation for P_{Setup} models that it is triggered after receiving the output $(\texttt{KeyGenRequest}, sid, kid, pp, \mathbb{A}, P_i)$ from the `KeyGenRequest` activation of P_i; telling the setup party that P_i asks for a key on $\mathbb{A}$. This modeling also encompasses that P_{Setup} (and environment $\mathcal{E}$) in a higher level protocol can decide whether to answer a key generation request. P_{Setup} proceeds only if there is an unprocessed key generation request with the key identifier kid for the party P_i (Step 2 of `KeyGen` activation). It then changes the bit b' for the request to 1 to mark it as processed. In Step 3, if P_{Setup} is corrupt, the ideal functionality informs the party P_i and the simulator via a public delayed output and a `KeyGenRequest` is recorded by $\mathcal{F}_{\mathsf{ABS}}$ ($\perp$ for unknown). Skipping slightly ahead, the record with corrupted P_{Setup} enables $\mathcal{F}_{\mathsf{ABS}}$ in the `Signature` activation to ask the simulator $\mathcal{S}$

to sign a message under a key that corresponds to the record. $\mathcal{F}_{\mathsf{ABS}}$ delegates this to $\mathcal{S}$, since if P_{Setup} is corrupt $\mathcal{F}_{\mathsf{ABS}}$ can not record the generated secret keys. A corrupt P_{Setup} is controlled by environment $\mathcal{E}$ (and adversary $\mathcal{A}$) and therefore the secret keys are generated outside of the ideal functionality.

For honest P_{Setup} in Step 4, the ideal functionality $\mathcal{F}_{\mathsf{ABS}}$ first checks, if a setup was finished and then it uses the algorithm $\mathcal{S}.\mathsf{KeyGen}$ provided by $\mathcal{S}$ to generate the secret key for $\mathbb{A}$. Next, $\mathcal{S}$ is informed about the key generation. This is necessary since $\mathcal{S}$ has to simulate the transmission of the secret key to the party P_i. We explain the details in Section 4.5 and the definition of a simulator. In Step 5, if the final output to P_i is delivered, $\mathcal{F}_{\mathsf{ABS}}$ is assured that the simulator transmitted the secret key and $\mathcal{F}_{\mathsf{ABS}}$ can record a successful key generation for P_i.

Signature. The Signature activation is responsible for the generation of signatures. After a check if the inputs are valid, the Signature activation checks if the activated party has a secret key for the given attribute by looking for a matching KeyGen record. If none exists it ignores the activation. If there is a record but the attribute vector does not satisfy the policy ($\mathbb{P}(\mathbb{A}) = 0$) it outputs $\perp$ and records the signature as invalid ($f = 0$). In the other case where $\mathbb{P}(\mathbb{A}) = 1$ holds and if the activation is under registered pp, it utilizes $\mathcal{S}.\mathsf{Sign}$ to output a signature without using the secret key of the party and without the activated party's identity as an input to $\mathcal{S}.\mathsf{Sign}$. Otherwise it asks $\mathcal{S}$ for a signature under unregistered public parameters. Overall and if all checks pass, the signatures generated in $\mathcal{F}_{\mathsf{ABS}}$ are recorded as valid ($f = 1$).

Verify. In this activation we handle any public parameters. Hence, we cover the cases where pp' is invalid or belongs to another instance, even if the activated party is honest. Verify Step 2a handles the case where an exact record exists (every element matches). Then Verify uses the recorded bit f as the output. Verify Step 2b handles two cases. First, a presumably manipulated signature (e.g., randomized) that was not recorded in a Signature activation. Second, the existence of corrupted parties. In any case, to decide whether a signature is valid we use the algorithm $\mathcal{S}.\mathsf{Verify}$, provided by $\mathcal{S}$, and the results is recorded by $\mathcal{F}_{\mathsf{ABS}}$. By this we allow strongly and weakly unforgeable signatures depending on $\mathcal{S}.\mathsf{Verify}$ which, as we will see in Section 4.5, can depend on an actual verify algorithm of a scheme. The simplest verify case is, if there is no corrupted party. Then $\mathcal{F}_{\mathsf{ABS}}$ verifies all signatures where the corresponding message and policy was not signed in a Signature activation as invalid (Verify 2c, $f_{\text{out}} \coloneqq 0$).

Note, parties can be corrupted in our *adaptive* model, with the restriction that the setup party P_{Setup} can only be corrupted after a finished Setup activation. We explain this in more detail in Section 4.5.

4.3.2 Security of Ideal ABS Functionality $\mathcal{F}_{\mathsf{ABS}}$

In the following we explain the security guarantees modeled by the ideal ABS functionality $\mathcal{F}_{\mathsf{ABS}}$. This includes the description of the security guarantees and how the keys are managed.

Scope of Security. Let us describe the scope of the security of the ideal functionality $\mathcal{F}_{\mathsf{ABS}}$ by describing how the public parameters and simulator are used. The ideal ABS functionality $\mathcal{F}_{\mathsf{ABS}}$ is based on the ideal digital signatures functionality by Canetti [Can03]. In detail, the support of verification under any public key. This allows the functionality to be more modular and to be used in diverse applications. Motivated by this, $\mathcal{F}_{\mathsf{ABS}}$ supports Signature and Verify activations with unregistered public parameters pp'. Importantly and similar to the ideal digital signatures by Canetti [Can03], $\mathcal{F}_{\mathsf{ABS}}$ itself only guarantees security for honest parties under the registered (honestly generated) public parameters. Therefore, we have to check in Key Generation, Signature and Verify, if the given public parameters pp' are equal to the registered public parameters pp and act accordingly.

For unregistered public parameters, corrupted parties, and inputs that are not recorded not all guarantees are lost, rather we rely on the simulator to match the guarantees that the real setting has in such cases. This is important, since the environment (and adversary) can give arbitrary inputs in any party's activation, even if the party is honest (not corrupt). Hence, with unregistered public parameters, in the Signature activation (Step 4) we ask the simulator $\mathcal{S}$ and we rely on the algorithm supplied by $\mathcal{S}$ in the Verify activation (Step 1). Therefore, the guarantees are determined by $\mathcal{S}$. For example, if the supplied algorithms allow randomized signatures such that $\mathcal{S}.\mathsf{Verify}$ declares them as valid $\mathcal{F}_{\mathsf{ABS}}$ will also do this by using $\mathcal{S}.\mathsf{Verify}$. This also holds for signatures originally signed by $\mathcal{F}_{\mathsf{ABS}}$ under registered public parameters, since randomization of signatures happens outside of $\mathcal{F}_{\mathsf{ABS}}$. In such a case, the Verify activation ends up in Step 2b, because of the first part of the if-statement. Namely, it exists a record that marks the message-policy pair as signed and valid, but with a different signature. This corresponds to the unforgeability definition of ABS (Definition 3.2.6) that also allows rerandomization of signatures.

Important for the security guarantees is also, that we allow the corruption of any party with the restriction that P_{Setup} can only be corrupted after the Setup activation was executed once. The privacy guarantee of $\mathcal{F}_{\mathsf{ABS}}$ is closely related to the simulation privacy (Definition 3.2.5). The corresponding experiment in Definition 3.2.5, models that the setup of (pp, msk) is honestly executed by the experiment. After that the distinguisher gets as input (pp, msk). This is compatible to our model of corruption of the setup party denoted as *adaptive*, see Section 4.2.7.

Correctness and Consistency. Intuitively, the ideal functionality $\mathcal{F}_{\mathsf{ABS}}$ guarantees correctness and consistency in the form that honestly generated signatures are directly recorded as valid by setting the bit f to 1 (Signature, Step 3c). To

verify signatures, these records are consistently used, i.e., if there is a `Signature` record with $f \in \{0,1\}$ the ideal functionality $\mathcal{F}_{\mathsf{ABS}}$ uses this bit f as its output. In detail, if all checks hold in a Signature activation, it corresponds to an honest party (a honest signer) that signs a message-policy pair under registered public parameters. Under this conditions, a Signature activation like ($\texttt{Signature}, sid, kid, pp, \mathbb{A}, m, \mathbb{P}$) in $\mathcal{F}_{\mathsf{ABS}}$ always results in a record $\langle \texttt{Signature}, sid, kid, pp, \mathbb{A}, m, \mathbb{P}, \sigma, f \coloneqq 1\rangle$ (Signature, Step 3c), where $f \coloneqq 1$ marks the signature as valid, since it was honestly generated in $\mathcal{F}_{\mathsf{ABS}}$ under registered public parameters pp. This leads to a verification output with $f_{\text{out}} = 1$ in a corresponding Verify activation (Step 2a). Thus, correctness is guaranteed.

Consistency is captured by the Steps 2a and 3a of the Verify activation. There we just output what is recorded. To verify $(m, \mathbb{P}, \sigma)$, where $\mathcal{F}_{\mathsf{ABS}}$ already generated a different signature σ' for $(m, \mathbb{P})$, we rely on Step 2b and use the bit output by $\mathcal{S}.\mathsf{Verify}$. An example for this are randomized signatures as we mentioned before. Step 2b also handles the case of corrupted parties. Corrupted parties generate signatures without the involvement of $\mathcal{F}_{\mathsf{ABS}}$ and may share their secret keys. In these cases, we have to use the output b of $\mathcal{S}.\mathsf{Verify}$. Hence, the guarantees are provided by the simulator $\mathcal{S}$ and the algorithms that $\mathcal{S}$ outputs. For correctness and consistency the simulator $\mathcal{S}$ has to ensure that $\mathcal{S}.\mathsf{Verify}$ outputs the same bit as the real setting outputs, otherwise distinguishing the ideal setting from the real setting is easy.

Non-Colluding. That parties do not collude is handled in the Signature activation Step 2. There, we check if the activated party has a single secret key for attributes that satisfy the given policy. Therefore, attributes from different secret keys and other parties cannot be pooled.

Privacy. Privacy for honest users under registered public parameters pp is guaranteed by the algorithms returned by $\mathcal{S}$. That means the algorithms do not take any privacy relevant information as input. Concretely, The algorithm $\mathcal{S}.\mathsf{Sign}$ generates signatures with the public parameters, the master secret key, simulation trapdoor, message, and a policy as input. Hence, for an honest signing party and under the registered pp we guarantee that the signatures output by $\mathcal{F}_{\mathsf{ABS}}$ are independent of the party's identity, secret key, and attribute vector encoded in the secret key. Intuitively, this guarantees that an adversary cannot link signatures to a party, an attribute vector, or to a secret key. Since $\mathcal{F}_{\mathsf{ABS}}$ only guarantees privacy under the registered public parameters pp, we can ask $\mathcal{S}$ for a signature under unregistered public parameters pp' or if P_{Setup} was already corrupted during the corresponding key generation (Signature, Step 4).

In a Key Generation activation with honest P_{Setup} we guarantee that the secret key is independent of the party's identifier by using the $\mathcal{S}.\mathsf{KeyGen}$ algorithm where the identifier P_i is not an input. Even if P_{Setup} is corrupted after a successful key generation with party P_i, we require that a signature can be generated without

the knowledge of P_i's secret key by using algorithm $\mathcal{S}.\mathsf{Sign}$ with the registered (pp, msk) pair.

We show in Section 4.5 that we can instantiate the algorithms, returned by the simulator, with the simulation algorithms of a simulation private ABS scheme.

Unforgeability. Regarding unforgeability, as explained for correctness, assuming no corruptions, the ideal functionality $\mathcal{F}_{\mathsf{ABS}}$ only declares a signature valid and sets $f_{\text{out}} := 1$ (Verify, Step 2a), if it was signed by itself in a Signature activation (Step 3c). Unforgeability is then guaranteed in $\mathcal{F}_{\mathsf{ABS}}$ by the Verify Step 2c, where $\mathcal{F}_{\mathsf{ABS}}$ declares all other signatures invalid by setting the bit f_{out} to 0. In detail, if $\mathcal{F}_{\mathsf{ABS}}$ enters Verify Step 2c, we know that all possible signing parties are honest and that the corresponding *message-policy pair* was not signed by $\mathcal{F}_{\mathsf{ABS}}$ in a Signature activation, otherwise we are in one of the two previous Verify activation steps. Here, we only consider the message-policy pair and not also the signature on purpose, since as already mentioned, we also support rerandomized signatures. This captures the experiment-based unforgeability notion (Definition 3.2.6). There, a forger is only successful, if it outputs a forgery under honest setup for a policy that is not satisfied by any attribute vector $\mathbb{A}$ where the forger got a corresponding secret key $sk_{\mathbb{A}}$ by the reveal oracle. Translated to $\mathcal{F}_{\mathsf{ABS}}$, the reveal oracle corresponds to corruption of the party with secret key $sk_{\mathbb{A}}$. A successful forger means that its output includes a valid signature, where in $\mathcal{F}_{\mathsf{ABS}}$, as described above, we declare it as invalid in Verify Step 2c. We use this difference in behavior in the formal unforgeability proof of the UC realization with the real protocol (Lemma 4.5.4).

4.4 Attribute-Based Signatures Protocol

In this section we present our attribute-based signatures protocol ρ_{ABS} and we show that the protocol ρ_{ABS} UC-realizes the ideal functionality $\mathcal{F}_{\mathsf{ABS}}$ in the sense of Definition 4.2.3. Concretely, the protocol ρ_{ABS} serves as a transformation of an attribute-based signature scheme to UC. Therefore, our result shows that existing ABS schemes are UC secure. We give the formal definition of the protocol ρ_{ABS} in Figure 4.4. The protocol uses an ABS scheme ABS and the activations Setup, Key Generation, Signature and Verify use the algorithms of ABS. Since protocol ρ_{ABS} is not an ideal protocol each party runs an instance of the protocol in a session (same session identifier sid). The Setup activation of the protocol is run by a unique party with the identifier P_{Setup}. The Key Generation activation involves a requesting party P_i and the setup party P_{Setup} that generates the secret key for P_i. The Signature and Verify activations can be executed by any party P_i without the involvement of the setup party P_{Setup}. Note, secret keys are never output to $\mathcal{E}$ in the protocol ρ_{ABS}, since in a real-world application a honest party should never reveal its secret keys.

Let us describe the definition of the ABS protocol ρ_{ABS} (Figure 4.4) in more detail. The protocol ρ_{ABS} stores records `SetupParams`, `KeyGenRequest`, `KeyGen`

similar to the ideal functionality $\mathcal{F}_{\mathsf{ABS}}$ (see Section 4.3) with the difference that for `SetupParams` it does not store a simulation trapdoor, and in `KeyGen` records it does not store the party. Both values that ρ_{ABS} does not store are not used in the protocol. In the following we describe the activations defined in protocol ρ_{ABS}. Here we omit some parameters in the inputs and outputs (e.g., public parameters and key identifiers) to simply the description.

Setup. In the first Setup activation `Setup` the party P_{Setup} generates and records $(pp, msk) \leftarrow \mathsf{Setup}(1^\lambda)$ where pp are the public parameters and msk is the master secret key. If it is not the first setup activation, the party P_{Setup} uses the existing `SetupParams` record. In both cases party P_{Setup} then outputs the public parameters pp to the environment $\mathcal{E}$.

Key Generation. A party P_i in a `KeyGenRequest` activation gets from the environment the session identifier sid, key identifier kid and attributes $\mathbb{A}$ as input. Party P_i then sends a `KeyGenRequest` message to the setup party P_{Setup}. On an answer by the party P_{Setup} in the form of a message $(\texttt{KeyGen}, sid, kid, pp, \mathbb{A}, sk_{\mathbb{A}})$, it checks if the secret key $sk_{\mathbb{A}}$ is valid and if this is the case it records the secret key $sk_{\mathbb{A}}$ to be able to sign message-policy pairs with $sk_{\mathbb{A}}$. The role of the party P_{Setup} in the key generation is to answer the `KeyGen` messages by a party P_i. Therefore, it records an unprocessed request and notifies the environment $\mathcal{E}$ via an output that it is ready to process the `KeyGen` under key identifier kid. Then, on input $(\texttt{KeyGen}, sid, kid)$ from $\mathcal{E}$ the party P_{Setup} looks up the unprocessed request for key identifier kid, marks it processed and generates a secret key $sk_{\mathbb{A}} \leftarrow \mathsf{KeyGen}(pp, msk, \mathbb{A})$, if the `SetupParams` are already recorded. Otherwise, party P_{Setup} was activated by the environment $\mathcal{E}$ for a key generation before the first Setup activation. Finally, party P_{Setup} outputs the generated secret key $sk_{\mathbb{A}}$ to the requesting party P_i.

Signature. In a Signature activation a party P_i receives form the environment $\mathcal{E}$ a message $(\texttt{Signature}, sid, kid, pp', \mathbb{A}, m, \mathbb{P})$ where the key identifier kid, public parameters pp', and attributes $\mathbb{A}$ are used to identify the recorded secret key $sk_{\mathbb{A}}$ that party P_i should use to generate a signature on message-policy pair $(m, \mathbb{P})$. Party P_i then generates a signature using the recorded secret key $sk_{\mathbb{A}}$ and outputs the signature to the environment $\mathcal{E}$.

Verify. On a Verify activation by environment $\mathcal{E}$, a party P_i just runs the Verify algorithm and outputs the result to the environment $\mathcal{E}$. Hence, it does not rely on any records and uses the input that it gets from the environment $\mathcal{E}$. This means it verifies signatures also under any given public parameters.

Setup When party P_{Setup} receives input $(\mathtt{Setup}, sid)$ from $\mathcal{E}$

1. If $sid = (P_{\mathsf{Setup}}, sid')$ for some sid' continue, else ignore.
2. If a record $\langle \mathtt{SetupParams}, sid', P_{\mathsf{Setup}}, pp, msk \rangle$ does not exist, P_{Setup} runs $(pp, msk) \leftarrow \mathsf{Setup}(1^\lambda)$ and records $\langle \mathtt{SetupParams}, sid', P_{\mathsf{Setup}}, pp, msk \rangle$.
3. Else, it uses the record $\langle \mathtt{SetupParams}, sid', P_{\mathsf{Setup}}, pp, msk \rangle$.
4. P_{Setup} outputs $(\mathtt{PublicParams}, sid, pp)$ to $\mathcal{E}$.

Key Generation

- When party P_i receives $(\mathtt{KeyGenRequest}, sid, kid, \mathbb{A})$ from $\mathcal{E}$
  1. If $sid = (P_{\mathsf{Setup}}, sid')$ continue, else ignore.
  2. P_i sends $(\mathtt{KeyGenRequest}, sid, kid, \mathbb{A})$ to P_{Setup}.
- P_{Setup} on receiving $(\mathtt{KeyGenRequest}, sid, kid, \mathbb{A})$ from P_i, it records the request $\langle \mathtt{KeyGenRequest}, sid, kid, \mathbb{A}, P_i, b' := 0 \rangle$ as unprocessed, and outputs $(\mathtt{KeyGenRequest}, sid, kid, pp, \mathbb{A}, P_i)$ to $\mathcal{E}$.
- When party P_{Setup} receives $(\mathtt{KeyGen}, sid, kid)$ from $\mathcal{E}$
  1. If $sid = (P_{\mathsf{Setup}}, sid')$ continue, else ignore.
  2. Look up unprocessed request record $\langle \mathtt{KeyGenRequest}, sid, kid, \mathbb{A}, P_i, b' = 0 \rangle$ from P_i. If there is none, ignore. Else mark it processed by setting $b' := 1$ and continue.
  3. If $\langle \mathtt{SetupParams}, sid', P_{\mathsf{Setup}}, pp, msk \rangle$ is not recorded by P_{Setup}, then ignore.
  4. Else P_{Setup} computes $sk_{\mathbb{A}} \leftarrow \mathsf{KeyGen}(pp, msk, \mathbb{A})$ and sends $(\mathtt{KeyGen}, sid, kid, pp, \mathbb{A}, sk_{\mathbb{A}})$ to P_i.
- P_i on receiving $(\mathtt{KeyGen}, sid, kid, pp, \mathbb{A}, sk_{\mathbb{A}})$ as an answer from P_{Setup}. If $(sk_{\mathbb{A}}, \mathbb{A})$ is valid under received pp, P_i appends record $\langle \mathtt{KeyGen}, sid, kid, pp, \mathbb{A}, sk_{\mathbb{A}} \rangle$ and outputs $(\mathtt{KeyGen}, sid, kid, \mathbb{A})$ to $\mathcal{E}$.

Signature When party P_i receives $(\mathtt{Signature}, sid, kid, pp', \mathbb{A}, m, \mathbb{P})$ from $\mathcal{E}$

1. If $sid = (P_{\mathsf{Setup}}, sid')$, continue, else ignore.
2. P_i looks up last record $\langle \mathtt{KeyGen}, sid, kid, pp', \mathbb{A}, sk_{\mathbb{A}} \rangle$.
3. If there is no record, then ignore.
4. Else, P_i computes signature $\sigma \leftarrow \mathsf{Sign}(pp', sk_{\mathbb{A}}, m, \mathbb{P})$ and outputs $(\mathtt{Signature}, sid, kid, \mathbb{A}, m, \mathbb{P}, \sigma)$ to $\mathcal{E}$.

Verify When party P_i receives $(\mathtt{Verify}, sid, pp', m, \mathbb{P}, \sigma)$ from $\mathcal{E}$

- P_i runs $b \leftarrow \mathsf{Verify}(pp', m, \mathbb{P}, \sigma)$ and outputs $(\mathtt{Verified}, sid, m, \mathbb{P}, \sigma, b)$ to $\mathcal{E}$.

Figure 4.4: Attribute-based signature protocol ρ_{ABS}.

4.5 UC Security for ABS

In the following we show that the ABS protocol ρ_{ABS} UC-realizes the ideal ABS functionality $\mathcal{F}_{\mathsf{ABS}}$. To this end, we use an ABS scheme that is correct, consistent, unforgeable and computational simulation private in the protocol ρ_{ABS}. With the protocol ρ_{ABS} and ideal ABS functionality $\mathcal{F}_{\mathsf{ABS}}$ defined, we can describe how *adaptive* corruption works. On a high level, the adversary $\mathcal{A}$ (potentially triggered by the environment) can initiate a corruption of any party P by sending $(\texttt{corrupt}, P)$. In case of a corruption of P_{Setup}, adversary $\mathcal{A}$ gets the public parameters and the master secret key that was recorded for P_{Setup}. Then adversary $\mathcal{A}$ (and therefore environment $\mathcal{E}$) can issue arbitrary secret keys. This includes the issuing of secret keys to honest parties. For this case, $\mathcal{F}_{\mathsf{ABS}}$ also keeps records of successful key generations with an honest party P_i and corrupted P_{Setup}. Additionally, $\mathcal{F}_{\mathsf{ABS}}$ only generates signatures for P_i, if such a record exists.

Overall, the UC-realization is proven under the assumption that (1) the setup party can only be corrupted after the setup activation was executed once and (2) that the corruption output does not include the simulation trapdoor. The first assumption models the guarantees of our privacy definitions (Definitions 3.2.4 and 3.2.5) and mirrors existing experiment-based secure ABS schemes that are defined with an honest setup like [EHM11; MPR08; MPR11; OT14; SAH16]. The second assumption is in place to make notation easier. It can be avoided by hardcoding the simulation trapdoor in the algorithms that $\mathcal{S}$ outputs, by this the simulation trapdoor is not explicitly available to P_{Setup}. We instead use the common notation where the inputs, like the simulation trapdoor, are explicit inputs to the algorithms and not hardcoded to make readability easier.

Note that one can lift the restriction that the corruption of the setup party is allowed only after the setup has been executed once by defining ABS with a CRS. The CRS is setup trustworthy and includes honest setup parameters of an ABS scheme. This is similar to the definition of blind signatures by Abe and Ohkubo [AO12]. Since we do not want to restrict our result to ABS with a CRS we do not consider this option in this thesis. Further we want to note that it is not unusual to restrict the environment. Similar environment restrictions regarding the setup of schemes are present in other UC-realizations. To give an example, in the work of Buan et al. [Bsk06] the signer can only be corrupted after the key generation. For UC non-committing blind signatures [AO12] the environment is restricted to activate the key generation only once. Overall, our assumptions are a restriction on the UC environment and is denoted as *adaptive*.

With the details of the UC model clarified, let us state the main theorem of this chapter, where we denote with ABS the attribute-based signature schemes used in the protocol ρ_{ABS}.

Theorem 4.5.1. *Let* ABS *be an attribute-based signature scheme.* ABS *is correct, consistent, unforgeable and computationally simulation private if and only if protocol* ρ_{ABS} *UC-realizes ideal functionality* $\mathcal{F}_{\mathsf{ABS}}$[adaptive, erasure, secure channels].

The theorem is with respect to the UC-realization of Definition 4.2.3 and we use the UC-emulation with black-box simulator of Definition 4.2.2.

For the formal proof of the theorem we split it in two parts and prove them separately in Sections 4.5.1 and 4.5.2. To shorten notation we will omit the UC assumptions *adaptive*, *erasure*, and *secure channels* in textual descriptions and if the context is clear.

4.5.1 Experiment-Based Security implies UC Security

In this section we show that a correct, consistent, unforgeable, and computationally simulation private ABS scheme ABS is also UC secure. The UC security is shown with respect to our ideal ABS functionality $\mathcal{F}_{\mathsf{ABS}}$ (Figure 4.3) and the ABS protocol ρ_{ABS} (Figure 4.4) which uses the ABS scheme ABS.

Lemma 4.5.1. *If the attribute-based signature scheme* ABS *that protocol* ρ_{ABS} *uses is correct, consistent, unforgeable and computationally simulation private, then protocol* ρ_{ABS} *UC-realizes ideal functionality* $\mathcal{F}_{\mathsf{ABS}}$[adaptive, erasure, secure channels].

In the following, we define simulator $\mathcal{S}_0^{\mathcal{A}}$ that, with black-box access to any given adversary $\mathcal{A}$, interacts with ideal functionality $\mathcal{F}_{\mathsf{ABS}}$ such that for all environments $\mathcal{E}$ it holds that $[\mathcal{F}_{\mathsf{ABS}}, \mathcal{S}_0^{\mathcal{A}}]$ and $[\rho_{\mathsf{ABS}}, \mathcal{A}]$ are indistinguishable. Concretely, for an ABS scheme $\mathsf{ABS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{SimKeyGen}, \mathsf{SimSetup}, \mathsf{SimSign})$, any ppt adversary $\mathcal{A}$ and our ideal ABS functionality $\mathcal{F}_{\mathsf{ABS}}$ we define the simulator $\mathcal{S}_0^{\mathcal{A}}$ in Figure 4.5, where $\mathcal{S}_0^{\mathcal{A}}$ has black-box access to $\mathcal{A}$. Note, that in the `Setup` activation of $\mathcal{S}_0^{\mathcal{A}}$ it sends algorithms $\mathsf{SimSetup}(1^\lambda), \mathsf{SimKeyGen}, \mathsf{SimSign}, \mathsf{Verify}$ to $\mathcal{F}_{\mathsf{ABS}}$, where $\mathsf{SimSetup}(1^\lambda)$ means that the security parameter is hardcoded.

Before starting the proof of Lemma 4.5.1, note that $\mathcal{S}_0^{\mathcal{A}}$ is ppt with respect to the UC definition (Section 4.2) since it runs the ppt adversary $\mathcal{A}$ and its other steps consists of simple checks, look ups and writing records which are all efficiently computable.

Next, we prove Lemma 4.5.1 via a sequence of games. Starting in Game 0 with the ideal setting $[\mathcal{F}_{\mathsf{ABS}}, \mathcal{S}_0^{\mathcal{A}}]$ and ending in Game 5 with the real setting $[\rho_{\mathsf{ABS}}, \mathcal{A}]$. In the sequence of games we gradually modify $\mathcal{F}_{\mathsf{ABS}}$ and $\mathcal{S}_0^{\mathcal{A}}$ in each step. Thereby, every game expands the modifications introduced in the games before and we only present the modifications of the current game for better readability. With each modification we show that, if an environment $\mathcal{E}$ can distinguish Game $i-1$ and Game i, then there exists an adversary that breaks one of the security guarantees of ABS.

In Figure 4.6 we present an overview of the sequence of games and the related security guarantees; computational simulation privacy, unforgeability, correctness and consistency. Note, that the model [*adaptive, erasure, secure channels*] for the ideal functionalities stays in place in the sequence of games. For readability we omit it in the following lemmas.

Setup On input $(\texttt{Respond}, (\texttt{Setup}, sid))$ from $\mathcal{F}_{\mathsf{ABS}}$

- Set the algorithms as $\mathcal{S}.\mathsf{Setup} \coloneqq \mathsf{SimSetup}(1^\lambda), \mathcal{S}.\mathsf{KeyGen} \coloneqq \mathsf{SimKeyGen}$, $\mathcal{S}.\mathsf{Sign} \coloneqq \mathsf{SimSign}$, and $\mathcal{S}.\mathsf{Verify} \coloneqq \mathsf{Verify}$.
- Send $(\texttt{Setup}, sid, \mathcal{S}.\mathsf{Setup}, \mathcal{S}.\mathsf{KeyGen}, \mathcal{S}.\mathsf{Sign}, \mathcal{S}.\mathsf{Verify})$ to $\mathcal{F}_{\mathsf{ABS}}$.

Key Generation: If P_i is honest

- On seeing public delayed output $(\texttt{KeyGenRequest}, sid, kid, pp, \mathbb{A}, P_i)$ for P_{Setup} by $\mathcal{F}_{\mathsf{ABS}}$ send $(\texttt{KeyGenRequest}, sid, kid, \mathbb{A})$ from P_i to P_{Setup}.

Key Generation: If P_{Setup} is honest

- On receiving message $(\texttt{Respond}, (\texttt{KeyGen}, sid, kid, \mathbb{A}, P_i, sk_{\mathbb{A}}))$ from $\mathcal{F}_{\mathsf{ABS}}$ record $\langle \texttt{KeyGen}, sid, kid, pp, \mathbb{A}, P_i, sk_{\mathbb{A}} \rangle$ and send $(\texttt{KeyGen}, sid, kid, \mathbb{A}, P_i, sk_{\mathbb{A}}, 1)$ to $\mathcal{F}_{\mathsf{ABS}}$.

- -

1. On seeing the public delayed output $(\texttt{KeyGen}, sid, kid, \mathbb{A})$ for P_i by $\mathcal{F}_{\mathsf{ABS}}$ simulate the P_{Setup} key generation part:
2. Check for the existence of record $\langle \texttt{KeyGen}, sid, kid, pp, \mathbb{A}, P_i, sk_{\mathbb{A}} \rangle$. If it exists then send $(\texttt{KeyGen}, sid, kid, pp, \mathbb{A}, sk_{\mathbb{A}})$ from P_{Setup} to P_i.

Key Generation: If P_i is corrupt

- On receiving $(\texttt{KeyGenRequest}, sid, kid, \mathbb{A})$ sent to P_{Setup} from P_i (controlled by $\mathcal{A}$). Send $(\texttt{KeyGenRequest}, sid, kid, \mathbb{A})$ on behalf of P_i to $\mathcal{F}_{\mathsf{ABS}}$ and deliver public delayed output directly without notifying $\mathcal{A}$.

Key Generation: If P_{Setup} is corrupt

1. On $(\texttt{KeyGen}, sid, kid, pp', \mathbb{A}, sk_{\mathbb{A}})$ from P_{Setup} (controlled by $\mathcal{A}$). If $(sk, \mathbb{A})$ is valid under pp', send $(\texttt{KeyGen}, sid, kid)$ on behalf of P_{Setup} to $\mathcal{F}_{\mathsf{ABS}}$.
2. On seeing the corresponding public delayed output $(\texttt{KeyGen}, sid, kid, \mathbb{A})$ for P_i by $\mathcal{F}_{\mathsf{ABS}}$ record $\langle \texttt{KeyGen}, sid, kid, pp', \mathbb{A}, P_i, sk_{\mathbb{A}} \rangle$.

Signature On input $(\texttt{Respond}, (\texttt{Signature}, sid, kid, pp', \mathbb{A}, m, \mathbb{P}))$ from $\mathcal{F}_{\mathsf{ABS}}$

- Look up last record $\langle \texttt{KeyGen}, sid, kid, pp', \mathbb{A}, P_i, sk_{\mathbb{A}} \rangle$, if none exists ignore. Else, $\sigma \leftarrow \mathsf{Sign}(pp', sk_{\mathbb{A}}, m, \mathbb{P})$ and send $(\texttt{Signature}, sid, kid, pp', \mathbb{A}, m, \mathbb{P}, \sigma)$ to $\mathcal{F}_{\mathsf{ABS}}$.

Others

- On $(\texttt{corrupt}, P_i)$ from $\mathcal{A}$. Inform $\mathcal{F}_{\mathsf{ABS}}$ with $(\texttt{corrupt}, P_i)$, to get all input, output and exchanged messages including $(\texttt{KeyGen}, sid, kid, pp, \mathbb{A}_j, P_i)$ messages. Send it all to $\mathcal{A}$.
- Public delayed output send to honest parties is acknowledged after the specified steps are processed. For corrupted parties it is never acknowledged.

Figure 4.5: Simulator $\mathcal{S}_0^{\mathcal{A}}$

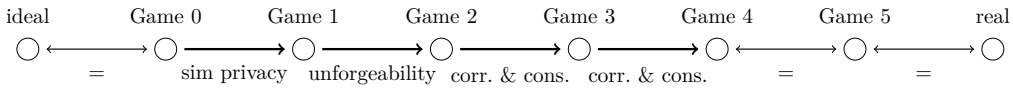

Figure 4.6: Sequence of Games

Game 0. Let $\mathcal{F}_0$ be the ideal functionality $\mathcal{F}_{\mathsf{ABS}}$ from Figure 4.3. Environment $\mathcal{E}$ interacts with $\mathcal{F}_0$ and the simulator $\mathcal{S}_0^{\mathcal{A}}$ from Figure 4.5. Therefore, the probability that $\mathcal{E}$ outputs 1 is the same as in the ideal setting. Hence, the following lemma follows directly.

Lemma 4.5.2. *For all ppt adversaries $\mathcal{A}$ and all ppt environments $\mathcal{E}$, it holds that*

$$\Pr\Big[\mathcal{E}[\mathcal{F}_{\mathsf{ABS}}, \mathcal{S}_0^{\mathcal{A}}] = 1\Big] = \Pr\Big[\mathcal{E}[\mathcal{F}_0, \mathcal{S}_0^{\mathcal{A}}] = 1\Big] \ .$$

Game 1. (Remove simulation algorithms) In this step we modify $\mathcal{F}_0$ to $\mathcal{F}_1$ and $\mathcal{S}_0^{\mathcal{A}}$ to $\mathcal{S}_1^{\mathcal{A}}$. The goal is to use the algorithms Setup, KeyGen, and Sign instead of $\mathsf{SimSetup}$, $\mathsf{SimKeyGen}$ and $\mathsf{SimSign}$ of ABS.

- In **Setup:** Simulator $\mathcal{S}_1^{\mathcal{A}}$ does not send algorithms $\mathsf{SimSetup}(1^\lambda)$, $\mathsf{SimKeyGen}$ and $\mathsf{SimSign}$ instead it sends $\mathsf{Setup}(1^\lambda), \mathsf{KeyGen}$ and Sign to $\mathcal{F}_1$. To match this change, $\mathcal{F}_1$ on running $\mathcal{S}.\mathsf{Setup} = \mathsf{Setup}$ in **Setup** Step 2a does not record the simulation trapdoor td_{sim}
- In the ideal functionality all usages of the algorithms, given by the simulator, are adapted by leaving out msk and td_{sim} in the inputs to match the inputs of KeyGen and Sign. Putting it together, $\mathcal{F}_1$ uses only the algorithms Setup, KeyGen, and Sign.
- In **Signature:** Steps 3a and 4 of $\mathcal{F}_0 = \mathcal{F}_{\mathsf{ABS}}$ (Figure 4.3) are modified. Step 4 is omitted completely. As a result, $\mathcal{F}_1$ never asks $\mathcal{S}_1^{\mathcal{A}}$ for a signature. In Step 3a the check $(pp' = pp)$ is omitted and the rest is changed such that $\mathcal{F}_1$ looks up last recorded entry $(\mathtt{KeyGen}, P_i, pp', \mathbb{A}, sk_{\mathbb{A}})$ with $\mathbb{P}(\mathbb{A}) = 1$ and generates the signature as $\mathsf{Sign}(pp', sk_{\mathbb{A}}, m, \mathbb{P})$ instead of using $\mathsf{SimSign}$ as in $\mathcal{F}_0$.

We change the setting of environment $\mathcal{E}$ from Game 0 to Game 1, thus from a game with simulated setup, simulated keys and simulated signatures to one with the signatures, keys and setup of the real protocol ρ_{ABS}. Hence, Game 0 and Game 1 correspond the two experiment settings in the simulation privacy experiment (Definition 3.2.5) and leads us to the following lemma.

Lemma 4.5.3. *If ABS scheme ABS is a computationally simulation private attribute-based signature scheme, then for all ppt adversaries $\mathcal{A}$ and all ppt environments $\mathcal{E}$ it holds that*

$$|\Pr[\mathcal{E}[\mathcal{F}_0, \mathcal{S}_0^{\mathcal{A}}] = 1] - \Pr[\mathcal{E}[\mathcal{F}_1, \mathcal{S}_1^{\mathcal{A}}] = 1]| \leq \mathsf{negl}_{0,1}(\lambda),$$

where $\mathsf{negl}_{0,1}(\cdot)$ *is negligible.*

Proof. We show that, if an environment $\mathcal{E}$ distinguishes $[\mathcal{F}_0, \mathcal{S}_0^{\mathcal{A}}]$ and $[\mathcal{F}_1, \mathcal{S}_1^{\mathcal{A}}]$ with non-negligible probability, then it distinguishes the output of the simulation algorithms from the output of the real algorithms. Let us look at the view of environment $\mathcal{E}$ in both settings. In $[\mathcal{F}_0, \mathcal{S}_0^{\mathcal{A}}]$ the instances resulting from the Setup activations are simulated, involving SimSetup, SimKeyGen and SimSign. In $[\mathcal{F}_1, \mathcal{S}_1^{\mathcal{A}}]$ the instances run the algorithms Setup, KeyGen and Sign. Recall, all algorithms are defined by the ABS scheme ABS. Given an environment $\mathcal{E}$ and adversary $\mathcal{A}$ that distinguishes $[\mathcal{F}_0, \mathcal{S}_0^{\mathcal{A}}]$ from $[\mathcal{F}_1, \mathcal{S}_1^{\mathcal{A}}]$, we define algorithm $\mathcal{D}$ to attack computational simulation privacy (Definition 3.2.5).

Definition of algorithm $\mathcal{D}$. In the following description of algorithm $\mathcal{D}$, we define how $\mathcal{D}$ emulates for environment $\mathcal{E}$ the behavior of $[\mathcal{F}_b, \mathcal{S}_b^{\mathcal{A}}]$ by interacting with its challenger from the simulation privacy experiment $\mathrm{Exp}_{\mathcal{D},\mathsf{ABS}}^{\mathsf{sim\text{-}privacy},b}(\lambda)$ ($b \in \{0,1\}$). To make the definition of $\mathcal{D}$ as readable as possible, we omit session identifier *sid* and parameter checks. To shorten the description, we omit the concrete format of the input and output messages, and recording of entries. All of this works as in the ideal functionality and simulator and is therefore omitted. We also refer to Figure 4.5 to see how corruption works in $\mathcal{D}$. Overall this means, to emulate $[\mathcal{F}_b, \mathcal{S}_b^{\mathcal{A}}]$, algorithm $\mathcal{D}$ follows the input and output behavior determined by the ideal functionality, i.e., the message formats for inputs and outputs.

Initialization: $\mathcal{D}$ on input (pp, msk) from $\mathrm{Exp}_{\mathcal{D},\mathsf{ABS}}^{\mathsf{sim\text{-}privacy},b}(\lambda)$, first runs environment $\mathcal{E}$ and adversary$\mathcal{A}$. If $\mathcal{E}$ and $\mathcal{A}$ want to exchange messages, i.e., activations and results of a corruptions, then $\mathcal{D}$ lets them communicate. Overall, whenever $\mathcal{F}_{\mathsf{ABS}}$ uses the algorithms $\mathcal{S}.\mathsf{KeyGen}$ or $\mathcal{S}.\mathsf{Sign}$ provided by a simulator, $\mathcal{D}$ will relate it to the oracles provided by $\mathrm{Exp}_{\mathcal{D},\mathsf{ABS}}^{\mathsf{sim\text{-}privacy},b}(\lambda)$, i.e., oracles $\mathcal{O}^{\mathsf{KeyGen}_b}$ and $\mathcal{O}^{\mathsf{Sign}_b}$.

Setup: On $\mathcal{E}$ sending $(\mathtt{Setup}, sid)$ to P_{Setup}, $\mathcal{D}$ outputs $(\mathtt{PublicParams}, sid, pp)$ where pp is given by the experiment

Key Generation: On $\mathcal{E}$ sending $(\mathtt{KeyGenRequest}, sid, kid, \mathbb{A}_l)$ to a party P.

- $\mathcal{D}$ behaves like $[\mathcal{F}_0, \mathcal{S}_0^{\mathcal{A}}]$ (note: not changed in $[\mathcal{F}_1, \mathcal{S}_1^{\mathcal{A}}]$)

Key Generation: On $\mathcal{E}$ sending $(\mathtt{KeyGen}, sid, kid)$ to party P_{Setup}.

- If P and P_{Setup} are honest, $\mathcal{D}$ behaves like $[\mathcal{F}_0, \mathcal{S}_0^{\mathcal{A}}]$, but for key generations it queries oracle $\mathcal{O}^{\mathsf{KeyGen}_b}(\mathbb{A}_l)$.
- If P is corrupt and P_{Setup} is honest, $\mathcal{D}$ first queries $\mathcal{O}^{\mathsf{KeyGen}_b}(\mathbb{A}_l)$, then queries $\mathcal{O}^{\mathsf{Reveal}}(l)$ to get $sk_{\mathbb{A}_l}$, and outputs $(\mathtt{KeyGen}, sid, kid, pp, \mathbb{A}_l, sk_{\mathbb{A}_l})$ to $\mathcal{A}$ for P.
- If P_i is honest and P_{Setup} is corrupt, $\mathcal{D}$ follows the steps of $\mathcal{S}_0^{\mathcal{A}}$ in Figure 4.5 (not changed in $\mathcal{S}_1^{\mathcal{A}}$) to simulate the communication with the

corrupted party. This includes the recording of $(\texttt{KeyGen}, sid, kid, pp', \mathbb{A}_l, P_i, sk_{\mathbb{A}_l})$.

Signature: If $\mathcal{E}$ sends $(\texttt{Signature}, sid, kid, pp', \mathbb{A}_i, m_j, \mathbb{P}_j)$ for a corrupted party P_l, $\mathcal{D}$ lets $\mathcal{A}$ handle it as the simulator $\mathcal{S}_b^{\mathcal{A}}$ does. If P_l is honest and

- $pp' = pp$: $\mathcal{D}$ answers with a signature from $\mathcal{O}^{\mathsf{Sign}_b}(i, m_j, \mathbb{P}_j)$
- $pp' \neq pp$: If $(P_l, pp', \mathbb{A}_i, sk_{\mathbb{A}_i})$ is recorded, $\mathcal{D}$ answers with a signature by running $\mathsf{Sign}(pp', sk_{\mathbb{A}_i}, m_j, \mathbb{P}_j)$.

Verification: If $\mathcal{E}$ sends $(\texttt{Verify}, sid, pp, m, \mathbb{P}, \sigma)$ to a corrupted P_l, $\mathcal{D}$ lets $\mathcal{A}$ handle it. If P_l is honest, then $\mathcal{D}$ executes the verification steps of $\mathcal{F}_0$ (not changed in $\mathcal{F}_1$).

Output: If $\mathcal{E}$ outputs a bit $\tilde{b}$, $\mathcal{D}$ outputs it as well.

Analysis. Since in UC an environment $\mathcal{E}$ is parameterized with the security parameter λ, environment $\mathcal{E}$, adversary $\mathcal{A}$, ideal functionality $\mathcal{F}_b$ and simulator $\mathcal{S}_b^{\mathcal{A}}$ are ppt algorithms. Hence, algorithm $\mathcal{D}$ is also a ppt algorithm. Let us analyze algorithm $\mathcal{D}$ and its emulation for environment $\mathcal{E}$ in the following. First of all, the view of $\mathcal{E}$ regarding the verification was not altered, since the Verify activation was not changed from $\mathcal{F}_0$ to $\mathcal{F}_1$. In the case of $pp' \neq pp$ in a Signature activations, $[\mathcal{F}_0, \mathcal{S}_0^{\mathcal{A}}]$ and $[\mathcal{F}_1, \mathcal{S}_1^{\mathcal{A}}]$ use Sign to generate a signature under pp'. The same holds for $\mathcal{D}$. All the other steps of $\mathcal{D}$ also perfectly emulate the behavior of $[\mathcal{F}_0, \mathcal{S}_0^{\mathcal{A}}]$ and $[\mathcal{F}_1, \mathcal{S}_1^{\mathcal{A}}]$.

If $\mathcal{D}$ is taking part in $\mathrm{Exp}^{\mathsf{sim\text{-}privacy},0}_{\mathcal{D},\mathsf{ABS}}(\lambda)$, then the public parameters given by the game are output by $\mathsf{SimSetup}$, keys returned by $\mathcal{O}^{\mathsf{KeyGen}_0}_{pp,msk,td_{sim}}$ are generated by $\mathsf{SimKeyGen}$, and signatures returned by the $\mathcal{O}^{\mathsf{Sign}_0}_{pp,msk,td_{sim}}$ are computed using $\mathsf{SimSign}$. Therefore, the view of $\mathcal{E}$ in this case is distributed as in $[\mathcal{F}_0, \mathcal{S}_0^{\mathcal{A}}]$. If $\mathcal{D}$ is taking part in $\mathrm{Exp}^{\mathsf{sim\text{-}privacy},1}_{\mathcal{D},\mathsf{ABS}}(\lambda)$, then the given public parameters are the output of Setup, keys returned by $\mathcal{O}^{\mathsf{KeyGen}_1}_{pp,msk}$ are generated by KeyGen, and the signatures returned by $\mathcal{O}^{\mathsf{Sign}_1}_{pp}$ are computed using the Sign algorithm. Hence, the view of environment $\mathcal{E}$ is distributed as in $[\mathcal{F}_1, \mathcal{S}_1^{\mathcal{A}}]$. At the end the final output of $\mathcal{D}$ is the output of environment $\mathcal{E}$. Overall, for a computationally simulation private ABS scheme ABS we can conclude that,

$$\begin{aligned}\left|\Pr\left[\mathcal{E}[\mathcal{F}_0, \mathcal{S}_0^{\mathcal{A}}] = 1\right] - \Pr\left[\mathcal{E}[\mathcal{F}_1, \mathcal{S}_1^{\mathcal{A}}] = 1\right]\right| &\leq \left|\Pr\left[\mathrm{Exp}^{\mathsf{sim\text{-}privacy},0}_{\mathcal{D},\mathsf{ABS}}(\lambda) = 1\right] - \right. \\ &\qquad \left. \Pr\left[\mathrm{Exp}^{\mathsf{sim\text{-}privacy},1}_{\mathcal{D},\mathsf{ABS}}(\lambda) = 1\right]\right| \\ &= \mathrm{Adv}^{\mathsf{sim\text{-}privacy}}_{\mathcal{D},\mathsf{ABS}}(\lambda),\end{aligned}$$

where the advantage $\mathrm{Adv}^{\mathsf{sim\text{-}privacy}}_{\mathcal{D},\mathsf{ABS}}(\lambda)$ is negligible in λ. □

Game 2. (modifying unforgeability) Recall that for honest parties under registered *pp* our ideal ABS functionality guarantees unforgability with certainty, see security of $\mathcal{F}_{\mathsf{ABS}}$ in Section 4.3.2. In this step we modify the setting $[\mathcal{F}_1, \mathcal{S}_1^{\mathcal{A}}]$ and denote the result as $[\mathcal{F}_2, \mathcal{S}_2^{\mathcal{A}}]$, where we only modify the functionality $\mathcal{F}_1$ and the simulator is not changed, hence $\mathcal{S}_1^{\mathcal{A}} = \mathcal{S}_2^{\mathcal{A}}$. To define $\mathcal{F}_2$, we introduce one modification to the Verify Step 2c (see Figure 4.3, the step was not changed up to now), where $f_{\text{out}} := 0$ is set. This step is changed to $f_{\text{out}} := b$, where b is the output of the verification done with $\mathcal{S}.\mathsf{Verify}$.

Lemma 4.5.4. *If ABS scheme* ABS *is unforgeable (Definition 3.2.6), then for all ppt adversaries* $\mathcal{A}$ *and all ppt environments* $\mathcal{E}$ *it holds that*

$$|\Pr[\mathcal{E}[\mathcal{F}_1, \mathcal{S}_1^{\mathcal{A}}] = 1] - \Pr[\mathcal{E}[\mathcal{F}_2, \mathcal{S}_2^{\mathcal{A}}] = 1]| \leq \mathsf{negl}_{1,2}(\lambda),$$

where $\mathsf{negl}_{1,2}(\cdot)$ *is negligible.*

Proof. Notice that the modification of Game 2 only introduces a difference to Game 1 in Verify, Step 2c only if the verify bit $b = 1$ holds, i.e., we set $f := b = 1$ where we set $f := 0$ in Game 1. We show that this event only happens, if the unforgeability of the scheme was broken. Let us denote the event that $b = 1$ holds in Step 2c of the Verify activation with Forge.

First, let us make clear what it means if Verify Step 2c in the ideal functionality (Figure 4.3) is reached and Forge happens for input $(\texttt{Verify}, sid, pp, m, \mathbb{P}, \sigma)$. If the verification of $\mathcal{F}_1$ and $\mathcal{F}_2$ reaches Step 2c, we know that the conditions for the previous steps of the Verify activation did not hold. In particular, we know that the signature σ on $(m, \mathbb{P})$ was not generated by the ideal functionalities and it was not verified in a previous Verify activation (Step 2a). Further, we know that $(m, \mathbb{P})$ was never signed (Step 2b) by the ideal functionalities. Importantly, from Step 2b it follows that the setup party P_{Setup} is honest. Hence, as in $\text{Exp}_{\cdot,\mathsf{ABS}}^{\mathsf{euf}}(\lambda)$ the master secret key is kept secret. Furthermore, from Step 2b it follows that there is no corrupted signer P_i with a record $\langle \texttt{KeyGen}, sid, kid, P_i, pp, \mathbb{A}', \cdot \rangle$ where $\mathbb{P}(\mathbb{A}') = 1$. Therefore, no corrupted party could have legitimately generated the signature σ. We conclude that, if Forge happens, then the signature is one of a party without a satisfying secret key for $\mathbb{P}$ and the signature was not generated by the ideal functionalities, i.e., it is a forgery. More formally we define in the following a forger F that utilizes the forging event Forge.

Definition of forger F. Let us define a forger F against the unforgeability of ABS in $\text{Exp}_{F,\mathsf{ABS}}^{\mathsf{euf}}(\lambda)$ (Definition 3.2.6) using an adversary $\mathcal{A}$ and environment $\mathcal{E}$. Similar to the distinguisher from Lemma 4.5.3, forger F behaves like the ideal functionality and simulator $[\mathcal{F}_1, \mathcal{S}_1^{\mathcal{A}}]$. Hence, we just describe the situations where F behaves differently to utilize the forging event Forge. Overall, F follows the input and output message format as defined by $\mathcal{F}_{\mathsf{ABS}}$ in Figure 4.3 (same as in $\mathcal{F}_1$ and $\mathcal{F}_2$). Also, F uses the oracle access provided by the unforgeability experiment $\text{Exp}_{F,\mathsf{ABS}}^{\mathsf{euf}}(\lambda)$ to answer $\mathcal{E}$'s Sign and Key Generation activations, instead of using

the algorithms provided by the simulator. To shorten the description we omit in the following the output messages, *sid* and parameter checks, and details of the corruption. They are the same as in Figure 4.3 and Figure 4.5.

Initialization: On input pp from $\text{Exp}^{\mathsf{euf}}_{F,\mathsf{ABS}}(\lambda)$, forger F runs environment $\mathcal{E}$ and adversary $\mathcal{A}$. Messages between $\mathcal{A}$ and $\mathcal{E}$ are forwarded by F. F emulates $\mathcal{F}_{\mathsf{ABS}}$ and $\mathcal{S}$ as described in the following.

Setup: If $\mathcal{E}$ sends $(\texttt{Setup}, sid)$ to P_{Setup}, F returns $(\texttt{PublicParams}, sid, pp)$.

Key Generation $\mathbb{A}_i$: F while simulating the key generation part of P_{Setup}. For honest party P_i, F queries the oracle $\mathcal{O}^{\mathsf{KeyGen}_1}_{pp,msk}(\mathbb{A}_i)$ and returns $(\texttt{KeyGen}, sid, kid, \mathbb{A}_i)$. If P_i is corrupt, F queries $\mathcal{O}^{\mathsf{KeyGen}_1}_{pp,msk}(\mathbb{A}_i)$ gets $sk_{\mathbb{A}_i}$ by querying $\mathcal{O}^{\mathsf{Reveal}}(i)$ and outputs $(\texttt{KeyGen}, sid, kid, pp, \mathbb{A}_i, sk_{\mathbb{A}_i})$ to $\mathcal{A}$ for P_i.

Signature: If $\mathcal{E}$ sends $(\texttt{Signature}, sid, kid, pp, \mathbb{A}_i, m_j, \mathbb{P}_j)$ to a corrupted party P_l, F lets $\mathcal{A}$ handle it. If P_l is honest and a key for ($\mathbb{A}_i$,P_l) with $\mathbb{P}_j(\mathbb{A}_i) = 1$ was generated, F computes the signature with a query to $\mathcal{O}^{\mathsf{Sign}}_{pp}(i, m_j, \mathbb{P}_j)$, otherwise F ignores the activation.

Verification/Output: If $\mathcal{E}$ sends $(\texttt{Verify}, sid, pp', m, \mathbb{P}, \sigma)$ to a corrupted party P_l, F lets $\mathcal{A}$ handle it. If P_l is honest, F executes the verification steps of $\mathcal{F}_{\mathsf{ABS}}$. If Forge happens during the verification checks, F uses $(m, \mathbb{P}, \sigma)$ as its final output. Otherwise, F outputs $(\perp, \perp, \perp)$.

Analysis. Since an environment $\mathcal{E}$ in UC is parameterized with the security parameter λ, environment $\mathcal{E}$, adversary $\mathcal{A}$, the ideal functionality and simulator are ppt algorithms. Hence, forger F is also a ppt algorithm.

Forger F answers the activations send by $\mathcal{E}$ with the outputs of its oracles. The oracles use the same algorithms (i.e., $\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}$) as the simulator $\mathcal{S}_2^{\mathcal{A}}$ of the current Game 2. From the argumentation above it follows that, if $\mathcal{E}$ causes Forge, then forger F outputs a valid forgery and wins experiment $\text{Exp}^{\mathsf{euf}}_{F,\mathsf{ABS}}(\lambda)$. Hence, it holds that

$$\Big|\Pr\Big[\mathcal{E}[\mathcal{F}_1, \mathcal{S}_1^{\mathcal{A}}] = 1\Big] - \Pr\Big[\mathcal{E}[\mathcal{F}_2, \mathcal{S}_2^{\mathcal{A}}] = 1\Big]\Big| \leq \Pr[\mathsf{Forge}] = \Pr\Big[\text{Exp}^{\mathsf{euf}}_{F,\mathsf{ABS}}(\lambda) = 1\Big].$$

□

Game 3: (Verification with algorithm Verify) To define $[\mathcal{F}_3, \mathcal{S}_3^{\mathcal{A}}]$ we modify $[\mathcal{F}_2, \mathcal{S}_2^{\mathcal{A}}]$ such that the interaction between them works as follows.

$\mathcal{F}_3$ on input $(\texttt{Verify}, sid, pp', m, \mathbb{P}, \sigma)$ from some party P:

1. Send $(\mathsf{Verify}, sid, pp', m, \mathbb{P}, \sigma)$ to $\mathcal{S}_3^{\mathcal{A}}$ and $\mathcal{S}_3^{\mathcal{A}}$ runs $b \leftarrow \mathsf{Verify}(pp', m, \mathbb{P}, \sigma)$ and sends $(\texttt{Verified}, sid, pp', m, \mathbb{P}, \sigma, b)$ to $\mathcal{F}_3$.

2. Upon $(\texttt{Verified}, sid, pp', m, \mathbb{P}, \sigma, b)$ from $\mathcal{S}_3^{\mathcal{A}}$ record $\langle \texttt{Signature}, sid, kid\ pp', \mathbb{A} \coloneqq \bot, m, \mathbb{P}, \sigma, f_{\text{out}} \coloneqq b \rangle$ and output $(\texttt{Verified}, sid, m, \mathbb{P}, \sigma, f_{\text{out}})$ to P.

This replaces all verification steps that were present in $\mathcal{F}_2$. Everything else stays as in $[\mathcal{F}_2, \mathcal{S}_2^{\mathcal{A}}]$.

Lemma 4.5.5. *If ABS scheme* ABS *is correct (Definition 3.2.2) and consistent (Definition 3.2.3), then for all ppt adversaries $\mathcal{A}$ and all ppt environments $\mathcal{E}$ it holds that*

$$|\Pr[\mathcal{E}[\mathcal{F}_2, \mathcal{S}_2^{\mathcal{A}}] = 1] - \Pr[\mathcal{E}[\mathcal{F}_3, \mathcal{S}_3^{\mathcal{A}}] = 1]| \leq \mathsf{negl}_{2,3}(\lambda),$$

where $\mathsf{negl}_{2,3}(\cdot)$ *is negligible.*

Proof. In $\mathcal{F}_3$ the output of a Verification activation is determined by the output of the algorithm Verify of the ABS scheme ABS. In the following, we show that for every Verification activation input to $\mathcal{F}_3$ the output bit f_{out} is equal to the bit that is used in $\mathcal{F}_2$, assuming ABS is correct and consistent. Recall that $\mathcal{F}_2$ guarantees correctness and consistency for honest users (under hostly generated keys) with certainty. Hence, if consistency or correctness fails an environment can easily recognize this and distinguish Game 3 from Game 2. First, observer that the Steps 2b, 2c, and 3b in $\mathcal{F}_2$ (see Figure 4.3) already set $f_{\text{out}} = b$, where b is the output of Verify. The interesting Steps are 2a and 3a in $\mathcal{F}_2$. In both steps, the bit f_{out} is set to a previously recorded bit f, but due to the changes in Game 3 we now use the bit b of Verify in $\mathcal{F}_3$ instead. The bit f could previously been recorded in two cases. First during the corresponding Signature activation (Step 3c in Figure 4.3) and second in a previous Verification activation. In the first case, for honest signers, parameters and keys $\mathcal{F}_2$ directly records signatures that it generates itself as valid. This is exactly what correctness of ABS requires. Hence, it follows from the correctness that $f \neq b$ only occurs with negligible probability. For corrupted signers (or not recorded (dishonest) parameters) the same follows from the consistency of ABS. In the second case, we can conclude that f was previously recorded in Verify activation Steps 2b, 2c (modified in Game 2) or 3b. Thus, by definition of the steps the bit b was already used in $\mathcal{F}_2$. Overall, the environment $\mathcal{E}$ can distinguish $[\mathcal{F}_2, \mathcal{S}_2^{\mathcal{A}}]$ and $[\mathcal{F}_3, \mathcal{S}_3^{\mathcal{A}}]$ only if the correctness or consistency fails. □

Game 4. (Remove halting condition in Signature activation) We use $[\mathcal{F}_3, \mathcal{S}_3^{\mathcal{A}}]$ as a starting point and modify them to define $[\mathcal{F}_4, \mathcal{S}_4^{\mathcal{A}}]$. We remove from $\mathcal{F}_3$ the halting condition in Step 3b of the Signature activation (was still unchanged from $\mathcal{F}_0$ in Figure 4.3) and directly record $\langle \texttt{Signature}, sid, kid, pp, \mathbb{A}, m, \mathbb{P}, \sigma, f \coloneqq 1 \rangle$ and output $(\texttt{Signature}, sid, kid, \mathbb{A}, m, \mathbb{P}, \sigma)$ to P_i in Step 3c. The simulator is unchanged, hence $\mathcal{S}_4^{\mathcal{A}} = \mathcal{S}_3^{\mathcal{A}}$.

Lemma 4.5.6. *If ABS scheme* ABS *is correct (Definition 3.2.2) and consistent (Definition 3.2.3), then for all ppt adversaries $\mathcal{A}$ and all environments $\mathcal{E}$ it holds that*

$$|\Pr[\mathcal{E}[\mathcal{F}_3, \mathcal{S}_3^{\mathcal{A}}] = 1] - \Pr[\mathcal{E}[\mathcal{F}_4, \mathcal{S}_4^{\mathcal{A}}] = 1]| \leq \mathsf{negl}_{3,4}(\lambda),$$

where $\mathsf{negl}_{3,4}(\cdot)$ *is negligible.*

Proof. In the Signature Step 3b (see Figure 4.3) the condition that a record $\langle \texttt{Signature}, sid, kid\ pp, \mathbb{A}, m, \mathbb{P}, \sigma, 0\rangle$ exists can only hold, if previously a Verification activation ($\texttt{Verify}$, $sid, pp, m, \mathbb{P}, \sigma$) recorded $f_{\text{out}} = 0$. However, this means that the output of the algorithm $\mathsf{Verify}(pp, m, \mathbb{P}, \sigma)$ was $b = 0$. Since correctness holds, we conclude that signatures for honest signers generated by the algorithm Sign are verified by algorithm Verify with $b = 0$ only with negligible probability. From the consistency we conclude the following. If the same signature σ was previously generated by a corrupted signer and verified by a Verification activation (in other words by Verify) the result $f_{\text{out}} = 0$ is recorded only with negligible probability. □

Game 5: (Remove records for signatures) Game 5 is a modification of Game 4 where the simulator is unchanged. Hence $\mathcal{S}_5^{\mathcal{A}} = \mathcal{S}_4^{\mathcal{A}}$. Only $\mathcal{F}_4$ is modified to $\mathcal{F}_5$, such that it does not create records for signatures like $\langle \texttt{Signature}, sid, kid\ pp', \mathbb{A}', m', \mathbb{P}', \sigma', f'\rangle$ in Signature and Verification activations.

Lemma 4.5.7. *For all ppt adversaries $\mathcal{A}$ and all ppt environments $\mathcal{E}$ it holds that*

$$\Pr[\mathcal{E}[\mathcal{F}_4, \mathcal{S}_4^{\mathcal{A}}] = 1] = \Pr[\mathcal{E}[\mathcal{F}_5, \mathcal{S}_5^{\mathcal{A}}] = 1] \ .$$

Proof. Through the modifications in Game 3 and Game 4 the signature records $\texttt{Signature}$ were never read by $\mathcal{F}_4$. Hence we just removed unused records in Game 5 and hence the view of an environment *env* does not change. Looking at all the changes introduced here and in all the previous steps that are now present in $\mathcal{F}5$ we observe that $\mathcal{F}_5$ only performs parameter and *sid* checks, asks its simulator $\mathcal{S}_5^{\mathcal{A}}$ and outputs what $\mathcal{S}_5^{\mathcal{A}}$ outputs. Overall, we moved all of the steps that generate ABS elements (public parameters, secret keys, signatures, verification bits) to the simulator $\mathcal{S}_5^{\mathcal{A}}$. The simulator $\mathcal{S}_5^{\mathcal{A}}$ uses internally the algorithms of the ABS scheme ABS, i.e., the same algorithms as used in the protocol ρ_{ABS}. □

As noted above, $\mathcal{F}_5$ in comparison to $\mathcal{F}_0$ only does parameter checks and the output of $\mathcal{F}_5$ is determined by the simulator $\mathcal{S}_5^{\mathcal{A}}$. Additionally, the simulator $\mathcal{S}_0^{\mathcal{A}}$ was modified such that $\mathcal{S}_5^{\mathcal{A}}$ behaves like protocol ρ_{ABS} (Figure 4.4).

Lemma 4.5.8. *For all ppt adversaries $\mathcal{A}$ and all ppt environments $\mathcal{E}$ it holds that*

$$\Pr[\mathcal{E}[\mathcal{F}_5, \mathcal{S}_5^{\mathcal{A}}] = 1] = \Pr[\mathcal{E}[\rho_{\mathsf{ABS}}, \mathcal{A}] = 1] \ .$$

Proof. For every activation the output of $\mathcal{F}_5$ is determined by the output of simulator $\mathcal{S}_5^{\mathcal{A}}$ and the algorithms $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ used in the simulator, where $\mathcal{S}_5^{\mathcal{A}}$ simulates the communication of the honest parties with $\mathcal{A}$. In detail, the same algorithms with the same inputs are used as in protocol ρ_{ABS}. Further, as in protocol ρ_{ABS} the simulator $\mathcal{S}_5^{\mathcal{A}}$ let $\mathcal{A}$ determine the behavior and output of corrupted parties. Consequently, the view of environment $\mathcal{E}$ in $[\mathcal{F}_5, \mathcal{S}_5^{\mathcal{A}}]$ is equal to the view of $\mathcal{E}$ in $[\rho_{\mathsf{ABS}}, \mathcal{A}]$. □

Proof of Lemma 4.5.1. From Lemmas 4.5.2, 4.5.3, 4.5.4, 4.5.5, 4.5.6, 4.5.7, and 4.5.8 it follows that

$$\begin{aligned} |\Pr[\mathcal{E}[\mathcal{F}_{\mathsf{ABS}}, \mathcal{S}_0^{\mathcal{A}}] = 1] - \Pr[\mathcal{E}[\rho_{\mathsf{ABS}}, \mathcal{A}](\lambda) = 1]| \\ \leq \mathsf{negl}_{0,1}(\lambda) + \mathsf{negl}_{1,2}(\lambda) + \mathsf{negl}_{2,3}(\lambda) + \mathsf{negl}_{3,4}(\lambda) \\ \leq \nu(\lambda) \end{aligned}$$

where $\nu(\cdot)$ is negligible. Hence, the protocol ρ_{ABS} UC-realizes the ideal ABS functionality $\mathcal{F}_{\mathsf{ABS}}[\textit{adaptive, erasure, secure channels}]$.

4.5.2 UC Security implies Experiment-Based Security

We show the second part of Theorem 4.5.1 in the following.

Lemma 4.5.9. *If protocol ρ_{ABS} UC-realizes the ideal functionality $\mathcal{F}_{\mathsf{ABS}}$*[adaptive, erasure, secure channels]*, then the ABS scheme* ABS *used in protocol ρ_{ABS} is correct, consistent, computationally simulation private, and unforgeable.*

Lemma 4.5.9 follows from Lemmas 4.5.10, 4.5.11, 4.5.12 and 4.5.13 stated in the following. Regarding the lemmas, we use the contraposition of Lemma 4.5.9 to make the description easier to follow. Concretely, ABS is **not** (correct ∧ consistent ∧ unforgeable ∧ computationally simulation private) implies that protocol ρ_{ABS} does not UC-realize ideal functionality $\mathcal{F}_{\mathsf{ABS}}$. Here, protocol "$\rho_{\mathsf{ABS}}$ does not UC-realize $\mathcal{F}_{\mathsf{ABS}}$" denotes that there is an adversary $\mathcal{A}$ such that for all simulators $\mathcal{S}$, there exists an environment $\mathcal{E}$ such that $[\mathcal{F}_{\mathsf{ABS}}, \mathcal{S}]$ and $[\rho_{\mathsf{ABS}}, \mathcal{A}]$ are not computationally indistinguishable with respect to the security parameter. Intuitively, this approach allows us in the following lemmas to focus on the individual assumptions and to exclude cases that we have already dealt with. In the following, we start with correctness (Lemma 4.5.10), then show consistency (Lemma 4.5.11), and then we focus on simulation privacy assuming correct and consistency of the ABS scheme ABS (Lemma 4.5.12). Finally, we focus on the unforgeability of ABS and assume correctness, consistency, and simulation privacy (Lemma 4.5.13).

Lemma 4.5.10. *Assume* ABS *is not correct (Definition 3.2.2), then protocol ρ_{ABS} does not UC-realize the ideal functionality $\mathcal{F}_{\mathsf{ABS}}$*[adaptive, erasure, secure channels].

Proof. If ABS is not correct then there exists a message $m \in \mathcal{M}$, attributes $\mathbb{A} \in \mathcal{U}_{pp}$, and policy $\mathbb{P} \in \mathcal{U}_{pp}$, such that $\mathbb{P}(\mathbb{A}) = 1$, and a signature $\sigma \in \mathsf{Sign}(pp, sk_{\mathbb{A}}, m, \mathbb{P})$

such that

$$\Pr\left[\mathsf{Verify}(pp, m, \mathbb{P}, \sigma) = 0\right] = \kappa(\lambda)$$

is non-negligible. Let environment $\mathcal{E}$ be such that it uses these elements to distinguishes $[\mathcal{E}[\rho_{\mathsf{ABS}}, \mathcal{A}](\lambda) = 1]$ and $[\mathcal{F}_{\mathsf{ABS}}, \mathcal{S}]$. The environment $\mathcal{E}$ works as follows. Environment $\mathcal{E}$ first sets $sid = (P_{\mathsf{Setup}}, 0)$ and runs the setup through P_{Setup} with input $(\mathtt{Setup}, sid)$ and obtains as a result the public parameter pp. Secondly, $\mathcal{E}$ activates some party P with $(\mathtt{KeyGenRequest}, sid, kid, \mathbb{A})$ and then sends the corresponding activation $(\mathtt{KeyGen}, sid, kid)$ to P_{Setup}. After that, it activates the same party P with $(\mathtt{Signature}, sid, kid, pp, \mathbb{A}, m, \mathbb{P})$ and obtains the signature σ. At last, environment $\mathcal{E}$ activates a party P_V with $(\mathtt{Verify}, sid, pp, m, \mathbb{P}, \sigma)$ to verify the signature and outputs the result. Observe that environment $\mathcal{E}$ always outputs 1 in the ideal setting $[\mathcal{F}_{\mathsf{ABS}}, \mathcal{S}]$, since the ideal setting guarantees for honest parties perfect correctness, but environment $\mathcal{E}$ outputs 0 with non-negligible probability in the real setting $[\rho_{\mathsf{ABS}}, \mathcal{A}]$. Hence, $|\Pr[\mathcal{E}[\rho_{\mathsf{ABS}}, \mathcal{A}](\lambda) = 1] - \Pr[\mathcal{E}[\mathcal{F}_{\mathsf{ABS}}, \mathcal{S}](\lambda) = 1]| = \kappa(\lambda)$. □

Lemma 4.5.11. *Assume ABS scheme* ABS *is not consistent (Definition 3.2.3), then protocol* ρ_{ABS} *does not UC-realize the ideal functionality* $\mathcal{F}_{\mathsf{ABS}}$[adaptive, erasure, secure channels].

Proof. If ABS is not consistent then there exists a message m, policy $\mathbb{P}$, and signature σ, and for all $b \in \{0, 1\}$ it holds that

$$\Pr[\mathsf{Verify}(pp, m, \mathbb{P}, \sigma) \neq b] = \kappa(\lambda)$$

is non-negligible. Intuitively, this means algorithm Verify outputs on different bits for the same input. Overall, the argument here is similar to the argument in the proof of Lemma 4.5.10. Let environment $\mathcal{E}$ be such that it uses the above elements. Environment $\mathcal{E}$ then works as described in the proof of Lemma 4.5.10 except that it activates P_V polynomial-many times with $(\mathtt{Verify}, sid, pp, m, \mathbb{P}, \sigma)$ to trigger that it produces two different verify outputs, i.e., b and $1 - b$. Let us denote this event with bad. Environment $\mathcal{E}$ outputs 1 if event bad does occur and 0 otherwise. In the ideal setting $[\mathcal{F}_{\mathsf{ABS}}, \mathcal{S}]$ the event bad never occurs, because the ideal verification guarantees consistency with certainty. Hence, the environment $\mathcal{E}$ always outputs 0. In the real setting $[\rho_{\mathsf{ABS}}, \mathcal{A}]$, it follows from our assumption that ABS is not consistent that event bad occurs with non-negligible probability. Consequently, $|\Pr[\mathcal{E}[\rho_{\mathsf{ABS}}, \mathcal{A}](\lambda) = 1] - \Pr[\mathcal{E}[\mathcal{F}_{\mathsf{ABS}}, \mathcal{S}](\lambda) = 1]| = \Pr[\mathsf{bad}] = \kappa(\lambda)$ □

Simulation Privacy. Intuitively, to show simulation privacy we define an environment that distinguishes the output of the simulation algorithms and the non-simulation algorithms. The structure and technique how to define the simulation algorithms in the proof of the following lemma is based on the simulation blindness proof in [AO12].

Lemma 4.5.12. *Let* ABS *be correct and consistent. If* ABS *is not computationally simulation private (Definition 3.2.5), then protocol* ρ_{ABS} *does not UC-realize the ideal functionality* $\mathcal{F}_{\mathsf{ABS}}$[adaptive, erasure, secure channels].

Let us first outline the proof. Since we assume that ABS is not computationally simulation private, it holds that for all tuples of ppt algorithms (SimSetup, SimKeyGen, SimSign) according to Definition 3.2.5, that there exists an algorithm $\mathcal{D}$ that distinguishes the experiment $\mathrm{Exp}^{\mathsf{sim\text{-}privacy},0}_{\mathcal{D},\mathsf{ABS}}(\lambda)$ from $\mathrm{Exp}^{\mathsf{sim\text{-}privacy},1}_{\mathcal{D},\mathsf{ABS}}(\lambda)$ with non-negligible advantage.

For a fixed adversary $\mathcal{A}$ and for every simulator $S^{\mathcal{A}}$ we first show how to define such a tuple of algorithms (SimSetup, SimKeyGen, SimSign). This is necessary since the definition of simulation privacy requires the existence of these algorithms. Then, from the assumption that ABS is not computationally simulation private we know that there exists a distinguisher $\mathcal{D}_{\mathcal{S}^{\mathcal{A}}}$ for the three algorithms (SimSetup, SimKeyGen, SimSign) that we define. Finally, we use this distinguisher $\mathcal{D}_{\mathcal{S}^{\mathcal{A}}}$ to construct an environment $\mathcal{E}$ that distinguishes the two settings $[\mathcal{F}_{\mathsf{ABS}}, \mathcal{S}^{\mathcal{A}}]$ and $[\rho_{\mathsf{ABS}}, \mathcal{A}]$.

Proof. Note, since we show that the protocol ρ_{ABS} does not UC-realize the ideal functionality $\mathcal{F}_{\mathsf{ABS}}$ we are in full control of the environment and adversary and can define them in the following. Hence, we define and use an adversary $\mathcal{A}$ that corrupts the setup party P_{Setup} after the Setup activation is completed. That means, if adversary $\mathcal{A}$ sees the output (`PublicParams`, *sid*, *pp*) from P_{Setup} it corrupts P_{Setup} and outputs (*pp*, *msk*) to the environment. After that it lets the environment handle P_{Setup} and therefore forwards every message for P_{Setup} to the environment. For every other message it behaves as the dummy adversary. Note, intuitively this definition of adversary $\mathcal{A}$ forces a simulator $\mathcal{S}^{\mathcal{A}}$ to act similar (up to computational difference) in the described corruption case. Otherwise, an environment can directly distinguish, if it talks to adversary $\mathcal{A}$ or a simulator $\mathcal{S}^{\mathcal{A}}$.

To define simulation algorithms we use that by definition of $\mathcal{F}_{\mathsf{ABS}}$ the simulator $\mathcal{S}^{\mathcal{A}}$ has to send, among other things, stateless ppt algorithms ($\mathcal{S}$.Setup, $\mathcal{S}$.KeyGen, $\mathcal{S}$.Sign) to $\mathcal{F}_{\mathsf{ABS}}$. Hence, the technique that we apply to get simulation algorithms is to put the appropriate parts of the ideal functionality and simulator in the three algorithms. This means, SimSetup runs the setup like $\mathcal{F}_{\mathsf{ABS}}$ does with $\mathcal{S}$ adjusted to the corruption on P_{Setup}, which allows us later in the proof to initialize a distinguisher. SimKeyGen executes the steps that $[\mathcal{F}_{\mathsf{ABS}}, \mathcal{S}]$ executes for the Key Generation activation by a party P_i with corrupted P_{Setup}, with the difference that SimKeyGen generates the secret keys on its own by using the KeyGen algorithm of the ABS scheme. This becomes more clear after the definition of our environment that acts just the same. SimSign simply runs the Signature activation steps of $\mathcal{F}_{\mathsf{ABS}}$ and outputs the generated signature. To make sure the three algorithms run the same instance of $\mathcal{S}^{\mathcal{A}}$, we let SimSetup choose the randomness of $\mathcal{S}^{\mathcal{A}}$ and use the same randomness, via an input, in SimKeyGen and SimSign to start $\mathcal{S}^{\mathcal{A}}$ again.

In the following definition of the algorithms we do not repeat the steps of the activations of $[\mathcal{F}_{\mathsf{ABS}}, \mathcal{S}]$ in detail, because of their complexity. For details of the

activations we refer to $\mathcal{F}_{\mathsf{ABS}}$ in Figure 4.3 and $\mathcal{S}$ in Figure 4.5. If we refer to activations, the following algorithms use fixed $sid = (P_{\mathsf{Setup}}, 0)$.

$\mathsf{SimSetup}(1^\lambda)$**:** On input security parameter 1^λ, SimSetup basically simulates party P_{Setup} and a setup activation as in $[\mathcal{F}_{\mathsf{ABS}}, \mathcal{S}^{\mathcal{A}}]$. It picks randomness r, starts $\mathcal{S}^{\mathcal{A}}$ with randomness r and sends $(\texttt{Respond}, (\texttt{Setup}, sid))$ to $\mathcal{S}^{\mathcal{A}}$.

Upon receiving $(\texttt{Setup}, sid, \mathcal{S}.\mathsf{Setup}, \mathcal{S}.\mathsf{KeyGen}, \mathcal{S}.\mathsf{Sign}, \mathcal{S}.\mathsf{Verify})$ from $\mathcal{S}^{\mathcal{A}}$, it generates $(pp, msk, td_{sim}) \leftarrow \mathcal{S}.\mathsf{Setup}$, records $\langle \texttt{SetupParams}, sid', P_{\mathsf{Setup}}, pp, msk, td_{sim}\rangle$ and $\langle \texttt{Algorithms}, \mathcal{S}.\mathsf{Setup}, \mathcal{S}.\mathsf{KeyGen}, \mathcal{S}.\mathsf{Sign}, \mathcal{S}.\mathsf{Verify}\rangle$. Then algorithm SimSetup simulates output $(\texttt{PublicParams}, sid, pp)$ for party P_{Setup}. After that, SimSetup outputs $(pp, msk, td_{sim} := r)$.

$\mathsf{SimKeyGen}(pp, msk, td_{sim}, \mathbb{A}_i, st_{\mathsf{KG}})$**:** The algorithm starts $\mathcal{S}^{\mathcal{A}}$ with randomness r encoded in td_{sim}. The remainder defined as in SimSetup except the last output step.

The steps result in the records $\langle \texttt{SetupParams}, sid', P_{\mathsf{Setup}}, pp, msk, td_{sim}\rangle$ and $\langle \texttt{Algorithms}, \mathcal{S}.\mathsf{Setup}, \mathcal{S}.\mathsf{KeyGen}, \mathcal{S}.\mathsf{Sign}, \mathcal{S}.\mathsf{Verify}\rangle$.

It restores the state from st_{KG}. Then, SimKeyGen executes the steps of the Key Generation activation $(\texttt{KeyGenRequest}, sid, kid, \mathbb{A}_i)$ as in $[\mathcal{F}_{\mathsf{ABS}}, \mathcal{S}^{\mathcal{A}}]$, for a new honest party P_i and an unique kid, including storing the KeyGen records. Therefore, SimKeyGen delegates public delayed output to $\mathcal{S}^{\mathcal{A}}$ including the request $(\texttt{KeyGenRequest}, sid, kid, pp, \mathbb{A}, P_i)$. On answer $(\texttt{KeyGen}, sid, kid)$ by $\mathcal{S}^{\mathcal{A}}$, SimKeyGen generates $sk_{\mathbb{A}_i} \leftarrow \mathsf{KeyGen}(pp, msk, \mathbb{A}_i)$ and records $\langle \texttt{KeyGen}, sid, kid, P_i, pp, \mathbb{A}_i, sk_{\mathbb{A}_i}\rangle$. It updates the state st_{KG} to include the KeyGen records. SimKeyGen outputs $sk_{\mathbb{A}_i}, st_{\mathsf{KG}}$.

$\mathsf{SimSign}(pp, msk, td_{sim}, m, \mathbb{P}, st_{\mathsf{KG}})$**:** It starts with the same steps as SimSetup (using randomness r in td_{sim}), but changes the last output step. The steps result in the same record $\langle \texttt{SetupParams}, sid', P_{\mathsf{Setup}}, pp, msk, td_{sim}\rangle$ and record $\langle \texttt{Algorithms}, \mathcal{S}.\mathsf{Setup}, \mathcal{S}.\mathsf{KeyGen}, \mathcal{S}.\mathsf{Sign}, \mathcal{S}.\mathsf{Verify}\rangle$.

SimSign restores the state (KeyGen records) from st_{KG}. Then, SimSign uses the Signature activation of $[\mathcal{F}_{\mathsf{ABS}}, \mathcal{S}^{\mathcal{A}}]$ to get the signature on $(m, \mathbb{P})$. On output $(\texttt{Signature}, sid, \mathbb{A}, m, \mathbb{P}, \sigma)$ algorithm SimSign outputs σ.

Note, that SimSetup, SimKeyGen and SimSign are getting the same algorithms from $\mathcal{S}^{\mathcal{A}}$, since they use the same randomness r to run $\mathcal{S}^{\mathcal{A}}$.

The specifics of the definition of the above algorithms is due to the circumstance that we have to wrap and split the (stateful) behavior of $[\mathcal{F}_{\mathsf{ABS}}, \mathcal{S}^{\mathcal{A}}]$ in three algorithms. However, this allows us to argue in the following with our environment that, if we execute SimSign, SimKeyGen, and SimSign as defined above and use the key generation state to essentially transfer the KeyGen records from SimKeyGen to SimSign we simulate the setting $[\mathcal{F}_{\mathsf{ABS}}, \mathcal{S}^{\mathcal{A}}]$ with these algorithms.

From the assumption we know ABS is not computationally simulation private. Hence for the simulation algorithms (SimSetup, SimKeyGen, SimSign) as defined

above, there exists a distinguisher $\mathcal{D}_{\mathcal{S}^{\mathcal{A}}}$ that distinguishes $\mathrm{Exp}^{\mathsf{sim\text{-}privacy},0}_{\mathcal{D}_{\mathcal{S}^{\mathcal{A}}},\mathsf{ABS}}(\lambda)$ and $\mathrm{Exp}^{\mathsf{sim\text{-}privacy},1}_{\mathcal{D}_{\mathcal{S}^{\mathcal{A}}},\mathsf{ABS}}(\lambda)$ with non-negligible advantage $\mathrm{Adv}^{\mathsf{sim\text{-}privacy}}_{\mathcal{D}_{\mathcal{S}^{\mathcal{A}}},\mathsf{ABS}}(\lambda)$.

Next, we define the environment $\mathcal{E}$, for adversary $\mathcal{A}$ as defined above and arbitrary but fixed $\mathcal{S}^{\mathcal{A}}$. Note, $\mathcal{E}$ takes the role of P_{Setup} as $\mathcal{A}$ will forward every message for P_{Setup} after the corruption of P_{Setup}. The setup party is corrupted only to get the master secret key and important is that $\mathcal{E}$ will act as an honest P_{Setup} and answers to requests as described in $\mathcal{F}_{\mathsf{ABS}}$.

1. $\mathcal{E}$ sends $(\texttt{Setup}, sid)$ to P_{Setup} and gets $(\texttt{PublicParams}, sid, pp)$ back. By the definition of our $\mathcal{A}$, from the corruption of P_{Setup} it gets (pp, msk) in return.

2. $\mathcal{E}$ starts $\mathcal{D}_{\mathcal{S}^{\mathcal{A}}}$ on input (pp, msk) and answers $\mathcal{D}_{\mathcal{S}^{\mathcal{A}}}$'s oracle queries as follows.

   $\mathcal{O}^{\mathsf{KeyGen}_b}(\mathbb{A}_i)$: $\mathcal{E}$ activates a new party P_i with $(\texttt{KeyGenRequest}, sid, kid, \mathbb{A}_i)$ and takes the role of corrupted P_{Setup}. Therefore, $\mathcal{E}$ eventually gets message $(\texttt{KeyGenRequest}, sid, kid, pp, \mathbb{A}_i, P_i)$ for party P_{Setup} from party P_i forwarded by adversary $\mathcal{A}$. Environment $\mathcal{E}$ then generates $sk_{\mathbb{A}_i} \leftarrow \mathsf{KeyGen}(pp, msk, \mathbb{A}_i)$, and sends message $(\texttt{KeyGen}, sid, kid, pp, \mathbb{A}_i, sk_{\mathbb{A}_i})$ to party P_i. After it is delivered, environment $\mathcal{E}$ outputs $sk_{\mathbb{A}_i}$ to $\mathcal{D}_{\mathcal{S}^{\mathcal{A}}}$.

   $\mathcal{O}^{\mathsf{Sign}_b}(i, m_j, \mathbb{P}_j)$: Environment $\mathcal{E}$ checks if $\mathbb{A}_i$ was queried and processed by a party P_i and that $\mathbb{P}_j(\mathbb{A}_i) = 1$ holds. If not, it ignores the query.

   Otherwise, environment $\mathcal{E}$ activates party P_i with $(\texttt{Signature}, sid, kid_j, pp, \mathbb{A}_i, m_j, \mathbb{P}_j,)$, eventually party P_i returns $(\texttt{Signature}, sid, kid_j, \mathbb{A}_i, m_j, \mathbb{P}_j, \sigma_j)$ and environment $\mathcal{E}$ outputs σ_j to $D_{\mathcal{S}^{\mathcal{A}}}$.

3. Eventually $\mathcal{D}_{\mathcal{S}^{\mathcal{A}}}$ outputs $\tilde{b}$ and environment $\mathcal{E}$ also outputs $\tilde{b}$.

Analysis. We will now relate the advantage of environment $\mathcal{E}$ to the advantage of distinguisher $\mathcal{D}_{\mathcal{S}^{\mathcal{A}}}$ in the simulation private experiment $\mathrm{Exp}^{\mathsf{sim\text{-}privacy},b}_{\mathcal{D}_{\mathcal{S}^{\mathcal{A}}},\mathsf{ABS}}(\lambda)$. We analyze the real setting $[\rho_{\mathsf{ABS}}, \mathcal{A}]$ and the ideal setting $[\mathcal{F}_{\mathsf{ABS}}, \mathcal{S}^{\mathcal{A}}]$ separately.

Case $[\rho_{\mathsf{ABS}}, \mathcal{A}]$**:** From the definition of ρ_{ABS} (Figure 4.4) we get the following. The tuple (pp, msk) is generated by $\mathsf{Setup}(1^{\lambda})$ and the secret keys $sk_{\mathbb{A}_i}$ are generated by the algorithm KeyGen (same in $\mathcal{E}$) with input pp, msk and attribute vector $\mathbb{A}_i$. The signatures σ_j are generated by an execution of $\mathsf{Sign}(pp, sk_{\mathbb{A}_i}, m_j, \mathbb{P}_j)$. Thus the view of $\mathcal{D}_{\mathcal{S}^{\mathcal{A}}}$ is equal to the view in $\mathrm{Exp}^{\mathsf{sim\text{-}privacy},1}_{\mathcal{D}_{\mathcal{S}^{\mathcal{A}}},\mathsf{ABS}}(\lambda)$, with algorithms Setup, KeyGen, Sign, and Verify as defined by ABS, which is used in ρ_{ABS}.

Case $[\mathcal{F}_{\mathsf{ABS}}, \mathcal{S}^{\mathcal{A}}]$**:** In this case, the tuple (pp, msk, td_{sim}) is generated by the algorithm $\mathcal{S}.\mathsf{Setup}$ output by $\mathcal{S}^{\mathcal{A}}$. The same algorithm is used in our $\mathsf{SimSetup}$ above. Further, the Signature activations from $\mathcal{E}$ result in signatures generated in $\mathcal{F}_{\mathsf{ABS}}$. This is the same process as specified by $\mathsf{SimSign}$ above. The Key Generation activations are answered by $\mathcal{E}$, with the use of KeyGen. In the setting $[\mathcal{F}_{\mathsf{ABS}}, \mathcal{S}^{\mathcal{A}}]$ with a corrupted P_{Setup} the secret keys are also generated

by $\mathcal{E}$, due to the definition of our adversary $\mathcal{A}$ that hands the responsibility for P_{Setup} to $\mathcal{E}$. Hence, the key generation is done as specified by $\mathsf{SimKeyGen}$ via KeyGen. Consequently, the view of $\mathcal{D}_{\mathcal{SA}}$ with the above defined algorithms $\mathsf{SimSetup}$, $\mathsf{SimKeyGen}$, and $\mathsf{SimSign}$ is equal to the view in $\mathrm{Exp}^{\mathsf{sim\text{-}privacy},0}_{\mathcal{D}_{\mathcal{SA}},\mathsf{ABS}}(\lambda)$.

From the analysis it follows that,

$$\begin{aligned}
&\Big|\Pr[\mathcal{E}[\mathcal{F}_{\mathsf{ABS}},\mathcal{S}^{\mathcal{A}}]=1]-\Pr[\mathcal{E}[\rho_{\mathsf{ABS}},\mathcal{A}]=1]\Big| \\
&=\Big|\Pr[\mathrm{Exp}^{\mathsf{sim\text{-}privacy},0}_{\mathcal{D}_{\mathcal{SA}},\mathsf{ABS}}(\lambda)=1]-\Pr[\mathrm{Exp}^{\mathsf{sim\text{-}privacy},1}_{\mathcal{D}_{\mathcal{SA}},\mathsf{ABS}}(\lambda)=1]\Big| \\
&=\mathrm{Adv}^{\mathsf{sim\text{-}privacy}}_{\mathcal{D}_{\mathcal{SA}},\mathsf{ABS}}(\lambda)\ .
\end{aligned}$$

□

Unforgeability. To show unforgeability we construct an environment $\mathcal{E}$ that uses a forger to distinguish the real and the ideal setting.

Lemma 4.5.13. *Let* ABS *be correct, consistent, computationally simulation private. If* ABS *is not unforgeable (Definition 3.2.6), then protocol* ρ_{ABS} *does not UC-realize the ideal functionality* $\mathcal{F}_{\mathsf{ABS}}$[adaptive, erasure, secure channels].

The proof technique of the lemma is based on the proof of the unforgeability of the blind signatures from a UC realization in [AO12] and is adapted to our ABS case. Adaptations include that we have to deal with the specifics of our ideal functionality $\mathcal{F}_{\mathsf{ABS}}$ definition and three oracles instead of one compared to [AO12].

Proof. If ABS is not unforgeable (Definition 3.2.6) but correct, consistent and computational simulation private, then there is a ppt forger F with non-negligible advantage $\mathrm{Adv}^{\mathsf{euf}}_{F,\mathsf{ABS}}(\lambda) = \Pr[\mathrm{Exp}^{\mathsf{euf}}_{F,\mathsf{ABS}}(\lambda)=1]$ (Definition 3.2.6). We construct environment $\mathcal{E}$ and $\mathcal{A}$ in the following. Let $\mathcal{A}$ be the dummy adversary (see Section 4.2.4). Note, if environment $\mathcal{E}$ wants to corrupts a party, $\mathcal{A}$ outputs the list of the secret keys of the corrupted party to $\mathcal{E}$. Hence, every simulator S has to do the same for corrupted parties. If not, the settings are easy to distinguish. Environment $\mathcal{E}$ first runs the setup through some party P_{Setup} with $(\texttt{Setup}, sid)$, where $sid := (P_{\mathsf{Setup}}, 0)$, and obtains as a result (output by P_{Setup}) the public parameters pp. Then, environment $\mathcal{E}$ runs forger F with pp. Let us describe the interaction of $\mathcal{E}$ and F, which involves the description of how $\mathcal{E}$ answers F's oracle queries.

$\mathcal{O}^{\mathsf{KeyGen}}_{pp,msk}(\mathbb{A}_i)$: On i-th query input $\mathbb{A}_i$, environment $\mathcal{E}$ activates a party P_i with a key generation $(\texttt{KeyGenRequest}, sid, kid, \mathbb{A}_i)$. Eventually $\mathcal{E}$ gets message $(\texttt{KeyGen}, sid, kid, \mathbb{A},)$ for P_i back after a finished key generation by P_{Setup} and stores it.

$\mathcal{O}^{\mathsf{Reveal}}(t)$: On input t, if there was a t-th KeyGen query, then $\mathcal{E}$ corrupts P_t, gets the secret key $sk_{\mathbb{A}_t}$, stores it, and outputs $sk_{\mathbb{A}_t}$ to F. If there was no t-th KeyGen query $\mathcal{E}$ ignores the reveal query.

$\mathcal{O}_{pp}^{\mathsf{Sign}}(t, m_j, \mathbb{P}_j)$: On j-th query, if the corresponding secret key $sk_{\mathbb{A}_t}$ was generated in a KeyGen query and not revealed, $\mathcal{E}$ activates P_t with ($\mathtt{Signature}$, sid, kid_j, $pp, \mathbb{A}_t, m_j, \mathbb{P}_j$). Eventually $\mathcal{E}$ gets ($\mathtt{Signature}$, $sid, kid_j, \mathbb{A}_j, m_j, \mathbb{P}_j, \sigma_j$) back and returns σ_j to F. If secret key $sk_{\mathbb{A}_t}$ was not generated in a KeyGen query ignore this signature query. In the case that $sk_{\mathbb{A}_t}$ was revealed, the corresponding party was corrupted and $\mathcal{E}$ generates the signature with $\sigma_j \leftarrow \mathsf{Sign}(pp, sk_{\mathbb{A}_t}, m_j, \mathbb{P}_j)$ and returns it to F.

Eventually F outputs a triple $(m^*, \mathbb{P}^*, \sigma^*)$. (1) If $(m^*, \mathbb{P}^*)$ was signed in a query or $\mathbb{P}^*(\mathbb{A}_t) = 1$ holds for any $\mathbb{A}_t$ where the corresponding secret key $sk_{\mathbb{A}_t}$ was revealed to forger F, environment $\mathcal{E}$ outputs 0. Else (2) $\mathcal{E}$ activates a new and not corrupted party P_V with ($\mathtt{Verify}$, sid, pp, m^*, $\mathbb{P}^*, \sigma^*$) and outputs the result.

Analysis. Overall, we have two cases to consider for the final output of environment $\mathcal{E}$ and they depend on forger F. In (1) the signature that F outputs can be valid, but nonetheless F does not win its unforgeability experiment, because the message-policy pair was already signed in an oracle query or one of the revealed attributes satisfy the output policy $\mathbb{P}^*$. In (2) the above does not hold and thus the F can win in its unforgeability experiment, if the signature is valid. In the following, we analyze environment $\mathcal{E}$ and its final output. We split up the analysis for the two settings (ideal, real) that $\mathcal{E}$ interacts with. We denote with "F wins" the event that the unforgeability experiment outputs 1 ($\mathrm{Exp}_{F,\mathsf{ABS}}^{\mathsf{euf}}(\lambda) = 1$) and with "$F$ fails" the event that it outputs 0 ($\mathrm{Exp}_{F,\mathsf{ABS}}^{\mathsf{euf}}(\lambda) = 0$).

Case $[\mathcal{F}_{\mathsf{ABS}}, \mathcal{S}^{\mathcal{A}}]$**:** Here, $\mathcal{E}$ interacting with $[\mathcal{F}_{\mathsf{ABS}}, \mathcal{S}^{\mathcal{A}}]$. Note that in this case the forger F gets inputs (pp and answers from the queries) generated by algorithms provided by the simulator. Intuitively, this means the advantage of F is influenced, if it *recognizes* the simulation. More formally, the advantage of F is at least $\mathrm{Adv}_{F,\mathsf{ABS}}^{\mathsf{euf}}(\lambda) - \mathrm{Adv}_{\mathcal{F}',\mathsf{ABS}}^{\mathsf{sim\text{-}privacy}}(\lambda)$. Where the advantage $\mathrm{Adv}_{\mathcal{F}',\mathsf{ABS}}^{\mathsf{sim\text{-}privacy}}(\lambda) = \mathsf{negl}_{\mathsf{sp}}(\lambda)$ of on adversary $\mathcal{F}'$ that uses forger F internally is negligible, since we assume that ABS is computationally simulation private.

If environment $\mathcal{E}$ activates a party P_V in (2), we know that the signature in the output of F was not signed by the ideal functionality, otherwise we are in case (1). From the definition of $\mathcal{F}_{\mathsf{ABS}}$ we know that every Verification activation for a signature under recorded and honest public parameter pp (honest setup), where the signature was not generated by the ideal functionality, results in output 0, because $\mathcal{F}_{\mathsf{ABS}}$ guarantees unforgeability (see Verify, Step 2c in Figure 4.3 and Section 4.3.2). This is exactly the case, if environment $\mathcal{E}$ activates a party P_V in (2). Thus, regardless of the success of F the ideal functionality $\mathcal{F}_{\mathsf{ABS}}$ outputs 0 at the end of the activation of party P_V and therefore also environment $\mathcal{E}$. Putting all together, $\mathcal{E}$ as defined above outputs 0 in both output cases (1) where the F was not successful and (2) because

of the unforgeability guarantee of $\mathcal{F}_{\mathsf{ABS}}$. Hence, we get the following.

$$\begin{aligned}
\Pr\Big[\mathcal{E}[\mathcal{F}_{\mathsf{ABS}}, \mathcal{S}^{\mathcal{A}}] = 0\Big] &= \Pr[\mathcal{E} \text{ in (1) outputs } 0] + \Pr[\mathcal{E} \text{ in (2) outputs } 0] \\
&= \Pr[F \text{ fails in case (1)}] + \Pr[F \text{ fails in case (2)}] \\
&\quad + \Pr[F \text{ wins in case (2)}] \\
&= \Pr[F \text{ fails}] + \Pr[F \text{ wins}] \\
&\geq 1 - (\mathrm{Adv}^{\mathsf{euf}}_{F,\mathsf{ABS}}(\lambda) - \mathsf{negl}_{\mathsf{sp}}(\lambda)) \\
&\quad + \mathrm{Adv}^{\mathsf{euf}}_{F,\mathsf{ABS}}(\lambda) - \mathsf{negl}_{\mathsf{sp}}(\lambda) \\
&= 1
\end{aligned}$$

Which gives us that $\Pr[\mathcal{E}[\mathcal{F}_{\mathsf{ABS}}, \mathcal{S}^{\mathcal{A}}] = 1] = 0$.

Case $[\rho_{\mathsf{ABS}}, \mathcal{A}]$**:** Here, environment $\mathcal{E}$ interacts with $[\rho_{\mathsf{ABS}}, \mathcal{A}]$. The forger F gets as inputs (pp and query answers) generated by the algorithms $\mathsf{Setup}, \mathsf{KeyGen}$, and Sign of the ABS scheme ABS. Hence, environment $\mathcal{E}$ simulates the experiment $\mathrm{Exp}^{\mathsf{euf}}_{F,\mathsf{ABS}}(\lambda)$ for forger F perfectly. The output of the Verification activation in (2) and therefore the output of $\mathcal{E}$ is just the output of $\mathsf{Verify}(pp, m^*, \mathbb{P}^*, \sigma^*)$, see Figure 4.4. This is the same in the winning condition of experiment $\mathrm{Exp}^{\mathsf{euf}}_{F,\mathsf{ABS}}(\lambda)$. Accordingly, the following holds.

$$\begin{aligned}
\Pr[\mathcal{E}[\rho_{\mathsf{ABS}}, \mathcal{A}](\lambda) = 1] &= \Pr[\mathcal{E} \text{ outputs } 1 \mid F \text{ wins}] \cdot \Pr[F \text{ wins}] \\
&\quad + \Pr[\mathcal{E} \text{ outputs } 1 \mid F \text{ fails}] \cdot \Pr[F \text{ fails}] \\
&= 1 \cdot \mathrm{Adv}^{\mathsf{euf}}_{F,\mathsf{ABS}}(\lambda) + 0 \cdot (1 - \mathrm{Adv}^{\mathsf{euf}}_{F,\mathsf{ABS}}(\lambda)) \\
&= \mathrm{Adv}^{\mathsf{euf}}_{F,\mathsf{ABS}}(\lambda)
\end{aligned}$$

Overall, we get that

$$|\Pr[\mathcal{E}[\rho_{\mathsf{ABS}}, \mathcal{A}](\lambda) = 1] - \Pr[\mathcal{E}[\mathcal{F}_{\mathsf{ABS}}, \mathcal{S}^{\mathcal{A}}] = 1]| = \mathrm{Adv}^{\mathsf{euf}}_{F,\mathsf{ABS}}(\lambda).$$

□

In conclusion, Lemma 4.5.9 follows from Lemmas 4.5.10, 4.5.11, 4.5.12 and 4.5.13. This shows that an ABS scheme that is used in the protocol ρ_{ABS} is correct, consistent, computationally simulation private, and unforgeable, if protocol ρ_{ABS} UC-realizes the ideal functionality $\mathcal{F}_{\mathsf{ABS}}$. Overall, Theorem 4.5.1 follows from Lemma 4.5.1 proven in Section 4.5.1 and Lemma 4.5.9 proven in Section 4.5.2. Hence, the security model of our ideal functionality $\mathcal{F}_{\mathsf{ABS}}$ is equivalent to our experiment-based security model presented in Chapter 3.

Part II

Anonymous Credentials

Updatable Anonymous Credentials

5

Summary. In this chapter we present the results of the CCS'19 paper [BBDE19a] and the corresponding full version [BBDE19b]. The main contribution of the paper that we present in this thesis are updatable anonymous credentials UAC (Section 5.2). UAC allows users to update their attributes of existing credentials in a privacy-preserving way, i.e., the updates can be hidden. Additionally, UAC allows users to get a credential on hidden attributes. To this end, UAC incorporates hidden parameters and an update function directly in the issuing and update process of credentials. The update function allows users to prove that their hidden and public attributes satisfy a policy under which the issuer is willing to issue credentials. In this thesis we revise the security model of UAC. Enabling us to precisely state the requirements on the argument system that we use to present a generic construction of UAC (Section 5.3), i.e., zero-knowledge and online extractability. As a consequence, we change the used argument system from interactive as presented in [BBDE19a; BBDE19b] to non-interactive. Together with online extractability, this change simplifies the security proofs of the generic UAC construction presented in this thesis. Further, we present an efficient instantiation of the generic UAC construction based on blind signatures and zero-knowledge argument systems with online extractability (Section 5.5). In addition to the results of [BBDE19a; BBDE19b] we discuss the connection of UAC and ABS and sketch an instantiation of UAC from ABS.

We start with an introduction to anonymous credentials and existing extensions from the literature (Section 5.1). We then formulate research questions and answer them by defining updatable anonymous credentials UAC and their security in Section 5.2 and by presenting a generic UAC construction in Section 5.3. To show that the generic UAC construction can be efficiently instantiated, we present a concrete instantiation in the random oracle model (ROM) and extend this to an instantiation with a common reference string (CRS) (Section 5.5). We end this chapter with a discussion of the connection between UAC and ABS in Section 5.6.

5.1 Introduction

In usual authentication systems personal information (e.g., mail address, user id) and a secret (e.g., a password) is provided by users to authenticate at a service provider. This allows multiple service providers that pool their data to identify users, build comprehensive user profiles, and to track users across multiple services. Anonymous credentials (AC) mitigate such problems by supporting authentication without identification via attributes that are not revealed and by supporting access policies. In AC *users* receive *credentials* that certify personal information (*attributes*) from a service provider (*issuer*). A *credential* encodes a vector of attributes, e.g., `date of birth`, `favorite item`, `number of purchases`. A user can then anonymously authenticate to a service provider (*verifier*) via a show protocol by proving possession of a credential that satisfies an access policy (e.g., "`favorite item` $\in \{$*soda*, *chocolate*$\}$ or `number of purchases` > 5") without revealing anything about her attributes except that they satisfy the access policy (*predicate*). Typical examples of *predicates* includes that a user shows that she is old enough, citizen of an EU member state, or that she has certain rights in an organization. AC guarantees for the verifier of an anonymous authentication that the user has indeed satisfying attributes (with respect to the predicate) encoded in a valid credential. Here, the verifier has to trust the issuer of the credential since a dishonest issuer can certify arbitrary attributes. Other examples of attributes of users are medical records, subscription status, affiliations and roles in an organization, or biometric data that can then be used to anonymously get physical access to locations. All of these attributes correspond to a scenario where users naturally do not want to reveal their attributes, however they want to use them to get access to services, buildings, or archives.

In typical AC systems from the literature [BCKL08; CL01; CL04; ILV11; PS16], to get a credential on her attributes a user has to reveal them to the issuer or the issuer exclusively determines the attributes. This limits the capabilities of AC.

Consider the `number of purchases` attribute of our example. If a user wants to increment this attribute in an AC system she has to reveal all of her attribute values to the issuer. Only then the issuer is able to issue a new credential on her old attributes with the attribute `number of purchases` incremented by 1. This means an issuer can identify the user on the basis of her `date of birth` and `number of purchases`. Hence, the user is anonymous with respect to all users that are born on the same date and that have made equally many purchases. Hence, updating attributes is not privacy-preserving in typical AC systems.

We present updatable anonymous credentials (UAC) as a solution to this problem. In UAC a user can get updates on attributes encoded in a credential from the original issuer of the credential without revealing the attributes' values. Furthermore, a user can also directly get a new credential on hidden and public attributes. Previously, the latter was limited to include a hidden user secret (corresponding to a pseudonym) in a credential and to transfer attributes of existing credentials to a new one [Cam+16b]. Hence, UAC generalizes this to update functions for the issue protocol that encode the public attributes and take hidden parameters α (e.g., hidden attributes) as input. The output of an update function is the attributes to be issued. Since issuance of credentials with an update function is a special case of an update of credentials in UAC we start with the description of an update. For an *update* of attributes of an existing credential the user and issuer engage in an update protocol with an update function υ as a common input. The user then contributes her existing credential on attributes $\mathbb{A}$ and a hidden parameter α to the protocol. The result of the update protocol is then a new credential on updates attributes $\mathbb{A}^* = \upsilon(\mathbb{A}, \alpha)$ for the user. The issuer on the other hand only learns that the update function υ was applied, but not hidden parameter α and not necessarily the resulting attributes $\mathbb{A}^*$. In our purchase example, the user can now run the update protocol with an update function υ defined as $\upsilon((\mathbb{A}', \texttt{number of purchases}), \alpha) = (\mathbb{A}', \texttt{number of purchases} + 1)$ where $\mathbb{A}'$ are the unchanged attributes. Here the value of `number of purchases` stays hidden from the issuer. The issuer only knows that it applied a "+1" update. Furthermore, if the user also wants to change her favorite item to a gaming console, the update function can be defined as $\upsilon((\mathbb{A}'', \texttt{favorite item}, \texttt{number of purchases}), \alpha \coloneqq \textit{gaming console}) = (\mathbb{A}'', \texttt{favorite item} = \alpha, \texttt{number of purchases} + 1)$ where $\mathbb{A}''$ are the unchanged attributes. In this example the hidden parameter α stays hidden from the issuer and as a result the issuer does not learn the user's favorite item. The same techniques are applied to the issuance of credentials. Hence, for the issuance of a new credential the user and issuer engage in an issue protocol with an update function υ that only takes a hidden parameter α as input and no existing attributes. For example, if an issuer just wants to issue a credential on public attributes $\mathbb{A}$ without any involvement of a hidden parameter the update function υ is defined such that $\upsilon(\perp, \alpha) = \mathbb{A}$. Here, the hidden parameter α is ignored and the update function υ always maps to the attributes $\mathbb{A}$. For example the attributes $\mathbb{A}$ are ($\texttt{favorite item} = \text{-}, \texttt{number of purchases} = 0$) modeling a `favorite item` attribute value that is not yet set and a start value of 0 for `number of purchases`.

More interesting is the issuance of a credential with hidden attributes encoded in the hidden parameter. If the user wants hide some of the attributes in an issue protocol the update function υ is set to $\upsilon(\perp, \alpha = (\mathit{chocolate})) = (\texttt{favorite item} = \mathit{chocolate}, \texttt{number of purchases} = 0)$ in our example. Here the attribute value 0 is known to the issuer and user, however *chocolate* is hidden from the issuer.

We show in this chapter an efficient instantiation of UAC based only on building blocks that are commonly used to construct AC, e.g., argument systems, commitments, and blind signatures. As mentioned before typical AC systems from the literature limit the applicability of AC in scenarios where attributes are updated regularly, e.g., the end date of a subscription or a counter for points or number of purchases. The above purchase example is extended in [BBDE19a; BBDE19b] to an incentive system based on UAC. An incentive system lets users collect points according to their purchases, e.g., 5 points for every €10 and lets users spend their points to get discounts or special items. The privacy-preserving incentive system presented in [BBDE19a; BBDE19b] provides anonymity for the user, and double-spending protection for the store. For more details we refer to [BBDE19a; BBDE19b] and focus in the following on UAC.

5.1.1 Related Work

Anonymous credentials (AC) originally presented by Chaum [Cha81; Cha85] is a broad research area covering AC constructions such as [BCKL08; BL13; CL01; CL03; CL04; HP22; ILV11; RVH17] and UC security for AC [CDD17; CDHK15] among others. These AC constructions are build from a versatile set of building blocks. Namely commitments, (blind) signatures and argument systems. These building blocks led to the addition of many practical features to AC over the years such as delegation of credentials [BB18; Bel+09; CDD17; CL19; MSBM22], revocation [CKS10; CL01; CL02], hidden predicates [Deu+18], auditing [CLNR14], and efficient support of a broad class of predicates [Bem+18; CG08]. AC systems that deviate from the above building blocks are systems build from structure-preserving signatures on equivalence classes (SPS-EQ) [CLP22; FHS19; MSBM22] or mercurial signatures [CL19]. Such AC systems are more efficient, since they do not use commitment schemes in the case of mercurial signatures [CL19] which simplifies protocols or they have more efficient show protocols (constant-size) in the case of SPS-EQ. Furthermore, symmetric AC contrary to the typical asymmetric notion of AC were introduced in [CMZ14] with efficient protocols. Motivated by real-world applications the link between credential showings and the issuer's identity in asymmetric AC systems was loosened by the concept of issuer hiding in [BFGP22; Bob+21; CLP22]. Without hiding the issuer, a show protocol is executed with respect to the issuer's public key of the credential to be shown, e.g., showing a credential issued by a user's country does not allow to hide the country since it is encoded in the public key. The AC model was also adapted to more practical and modern scenarios like having access to a cloud [HK19] and hardware structures present in modern devices [HS21] where one part of the hardware is more powerful

(e.g., smartphone) than the other (e.g., SIM card). Implementations and prototypes of AC are also featured in the literature. The most well known implementations of classical AC systems are U-Prove [Bra99; PZ13] and Identity Mixer [CL01; CV02]. Furthermore, a modern approach to AC with a link to reputation systems was implemented and presented in [Bem+18]. Considering the general implementation task, a prototyping library for AC and other privacy-preserving schemes called Cryptimeleon was presented in [BEHF21] and features a framework for argument systems in the form of Sigma protocols.

The concept of allowing hiding attributes during the issuance of credentials was also considered in [Cam+16b] with the support that attributes carry over from existing credentials to a new credential, if the attribute values stay the same. In detail, the user proves for each hidden attribute value that it is equal to an attribute value certified in an existing credential. Basic updates on credentials were also considered in the literature [CKS10; DDD05; NDD06]. In [DDD05] privacy-preserving updates of credentials in the form of addition and subtraction of attribute values certified in an existing credential are presented for two concrete AC system based on Brands' system [Bra99] and CL-signatures [CL01]. However without a corresponding security model. In [NDD06] the authors first formalized an update protocol in the syntax of AC that considers an unspecified update operation. However, the authors require that a show protocol has to be executed before every update protocol. Further, they do not define any security properties. The AC system in [CKS10] supports update of credentials by their original issuer. In contrast to UAC, the issuer learns all the attributes involved and the user cannot contribute a hidden parameter.

Our modeling of the update function generalizes the existing approaches in three aspects. First, an update function can perform arbitrary checks on attributes and the hidden parameter, i.e., not limited to equality checks of attribute values. Second, an update function involves an arbitrary hidden parameter, i.e., it does not have to correspond to attributes certified in existing credentials. Third, the issuer does not necessarily learn the attributes or hidden parameter of an update function. Only if the user and issuer agrees on an update function that reveals (parts of) them, e.g., with an equality check. Hence, with UAC users have the choice of adding private information to their credentials and the issuer is able to limit this by using a check on the hidden parameter. For example, if a user wants to add her favorite item of a store as an attribute in an issue protocol, she sets the favorite item as the hidden parameter and the store (issuer) adds to the update function a check that the favorite item is on the inventory list. In UAC it is then guaranteed that the store does not learn the user's favorite item.

From a technical perspective closer related to our UAC are stateful anonymous credentials [CGH11; GGM14]. There, a credential of a user encodes a state that gets updated according to a public state machine model that then determines a successor state. Similar to UAC, the old state stays private during the update. The same functionality can be achieved using UAC by encoding the state transition in the update function.

Since we also consider ABS in this thesis, we want to note the difference between ABS and AC in general. Most of the existing AC systems are build using signature schemes (or blind signature schemes), commitment schemes, and argument systems, e.g., [BB18; BCKL08; BL13; CL01; CL04; ILV11; PS16; RVH17] and an ABS scheme is indeed an extended signature scheme. Hence, intuitively one can try to build an AC system using an ABS scheme as the signature scheme. However, the obstacle is that an ABS is generated on a message-policy pair. This means, if a user gets a credential from an issuer, here an ABS signature, the policy (predicate) has be fixed during the issuing of the credential and therefore known to the issuer. This makes it impossible for the user to choose the predicate for the show protocol independently from the issuance of the credential. Furthermore, the issuer can recognize the predicate and link showings to the issuing of credentials. To circumvent this, we have to change the definition of ABS and allow that the policy is not fixed during signing. To this end, an instantiation of an AC system from a so called threshold attribute-based signatures (t-ABS) was shown in [SS09]. The key feature of t-ABS is that the policy is fixed during verification of the signature. This feature comes with the disadvantage of very limited policies, i.e., policies that define a set of attributes and a valid t-ABS shows that the user's secret key has at least t attributes in common with the policy. In Section 5.6 we show a different approach to build a UAC system from an ABS scheme.

5.1.2 Research Questions

We identify the following questions based on the related work and described limits of existing AC systems.

- Q1 How to define security for AC, i.e., anonymity and soundness, with hidden attributes and an update function in the issue und update protocols?
- Q2 Is there a generic construction of UAC with the sketched features (issuance of hidden attributes and update of attributes) constructed from building blocks that are commonly used in the AC literature?
- Q3 Is there an efficient instantiation of the generic construction of UAC from those building blocks?
- Q4 What is the connection between ABS and UAC?

5.1.3 Our Contribution

We answer the questions in this chapter affirmatively in the following way.

Research question Q1. The challenge to define security for an AC with an update function that includes hidden parameters, such as hidden attributes, is that no issuer necessarily knows what kind of attributes (and credentials) it issued to

users. In detail, in a soundness experiment we typically define that an adversary, acting as dishonest users, outputs a forged credential on some attributes, if a credential on these attributes was never issued by an honest issuer. However, in our case we cannot define soundness in this way, because we as the experiment do not know the hidden parameters (attributes) of the issued credentials. Therefore, we define soundness differently with an extractor that outputs an explanation list that includes the hidden parameters of every issue and update protocol in Section 5.2. Regarding anonymity we use a simulation-based security definition that requires that there is a simulator that can run the user side of the issue, update and show protocols without having access to a user's credentials, attributes, or hidden parameters. This captures that adversarial issuers and verifiers do not learn these private information by interacting with a user in the protocols.

Compared to [BBDE19a; BBDE19b] we revise the security model in two ways. First, the soundness definition in this thesis is with respect to a black-box extractor (also called online or straight-line extractor in the literature) that does not require rewinding access to an adversary. Second, the anonymity definition presented in [BBDE19a; BBDE19b] is formalized as an experiment in this thesis.

Research question Q2. We answer the second question by presenting a generic UAC construction from a blind signature scheme and a zero-knowledge argument of knowledge system in the ROM (Section 5.3). We prove the security of the generic UAC construction using our security definitions in Section 5.4. Additionally, we discuss the generic UAC construction and its security in the CRS model in Section 5.4.3.

Research question Q3. We present an efficient instantiation of the generic UAC construction in Section 5.5 to answer the third question. For this we use the blind signature scheme by Pointcheval and Sanders [PS16] and a zero-knowledge argument of knowledge system in the ROM construction with the Fischlin transformation [BFW15; Fis05].

Research question Q4. This question is interesting since ABS is a signature schemes and in the literature signature schemes are used to build AC systems. However, as mentioned in Section 5.1.1 this direct approach used in [SS09] is limited to very specific ABS schemes for which there is only one instantiation presented in the same work [SS09]. We first discuss the high level similarities of ABS and UAC in Section 5.6 and then describe an instantiation of our generic UAC construction from an ABS scheme, where we use the ABS scheme for the showing of credentials. For the latter we have to make requirements on how the ABS scheme is constructed, i.e., that it uses a signature scheme to generate ABS secret keys and that the signature scheme can be transformed to a blind signature scheme.

5.2 Updatable Anonymous Credentials (UAC) Definitions

In the following, we formally define updatable anonymous credential (UAC) systems and their security. In UAC there are three roles: issuers, users, and verifiers. An issuer is responsible for issuing credentials on user attributes under its own key, users can contact issuers to get a credential for their attributes and prove possession of a credential to verifiers. Additionally, users can contact the issuer of a credential to update its attributes. For the interaction of the roles we assume that the communication is done via secure channels. Note, in practical applications it is also necessary to hide identifying information of the communication network to preserve anonymity of the users, e.g., the IP addresses, if UAC is used over an IP-based network, or Media Access Control (MAC) address, if we use UAC in a system with short-range communication (e.g., Bluetooth, NFC). However, these are concerns on a higher level than the definition of UAC and are not considered further in this thesis. Next, we define the syntax of UAC and mention the corresponding roles.

Definition 5.2.1 (Updatable Anonymous Credentials)
An updatable anonymous credential (UAC) system denoted as UAC consists of algorithms Setup, $\mathsf{IssuerKeyGen}$, $\mathsf{IssueCred}$, $\mathsf{ReceiveCred}$, $\mathsf{UpdateCred}$, $\mathsf{ReceiveUpd}$, $\mathsf{ShowCred}$, and $\mathsf{ShowVerify}$.

- $pp \leftarrow \mathsf{Setup}(1^\lambda)$ is a ppt setup algorithms that on input security parameter 1^λ outputs public parameters pp. We assume that the public parameters pp define the attribute universe $\mathcal{U}_{\mathsf{Attr}}$.
- $(pk, sk) \leftarrow \mathsf{IssuerKeyGen}(pp, 1^n)$ is a ppt issuer key generation algorithm that on input public parameters pp and attribute vector length 1^n outputs an issuer key pair (pk, sk). We assume that the issuer public key pk defines the attribute space $\mathcal{U}_{\mathsf{Attr}}^n$ of the issuer, and hence the update function universe Υ

$$\begin{aligned}\Upsilon \coloneqq \{\upsilon \mid \upsilon : \mathcal{U}_{\mathsf{Attr}}^n \times \{0,1\}^* \to \mathcal{U}_{\mathsf{Attr}}^n \cup \{\bot\} \\ \vee\ \upsilon : \{\bot\} \times \{0,1\}^* \to \mathcal{U}_{\mathsf{Attr}}^n \cup \{\bot\}\},\end{aligned}$$

  and the predicate universe $\mathcal{X}$

$$\mathcal{X} \coloneqq \{\chi \mid \chi : \mathcal{U}_{\mathsf{Attr}}^n \times \{0,1\}^* \to \{0,1\}\}.$$

- $cred \leftarrow \mathsf{ReceiveCred}(pp, pk, ctx, \upsilon, \alpha) \leftrightarrow \mathsf{IssueCred}(pp, pk, ctx, \upsilon, sk)$ is an interactive protocol with common input public parameters pp, issuer public key pk, context ctx, and update function $\upsilon : \{\bot\} \times \{0,1\}^* \to \mathcal{U}_{\mathsf{Attr}}^n \cup \{\bot\}$ between a user running ppt algorithm $\mathsf{ReceiveCred}$ and issuer running ppt algorithm $\mathsf{IssueCred}$. Algorithm $\mathsf{IssueCred}$ gets as additional input an issuer secret key sk. Algorithm $\mathsf{ReceiveCred}$ with additional input a hidden parameter α outputs a credential

cred or failure symbol $\perp$. We assume that a credential *cred* includes the issued attributes $\mathbb{A}$.

- $cred^* \leftarrow \mathsf{ReceiveUpd}(pp, pk, ctx, \upsilon, \alpha, cred) \leftrightarrow \mathsf{UpdateCred}(pp, pk, ctx, \upsilon, sk) \rightarrow b$ is an interactive protocol with common input public parameters pp, issuer public key pk, context ctx, and update function $\upsilon : \mathcal{U}^n_{\mathsf{Attr}} \times \{0,1\}^* \rightarrow \mathcal{U}^n_{\mathsf{Attr}} \cup \{\perp\}$ between a user running ppt algorithm ReceiveUpd and issuer running ppt algorithm UpdateCred. Algorithm UpdateCred with additional input an issuer secret key sk outputs a bit $b \in \{0,1\}$. Algorithm ReceiveUpd with additional input a hidden parameter α and a credential *cred* outputs a credential $cred^*$ or failure symbol $\perp$.

- $\mathsf{ShowCred}(pp, pk, ctx, \chi, \alpha, cred) \leftrightarrow \mathsf{ShowVerify}(pp, pk, ctx, \chi) \rightarrow b$ is an interactive protocol with common input public parameters pp, issuer public key pk, context ctx, and show predicate $\chi : \mathcal{U}^n_{\mathsf{Attr}} \times \{0,1\}^* \rightarrow \{0,1\}$ between a user running ppt algorithm ShowCred and verifier running ppt algorithm ShowVerify. Algorithm ShowCred gets as additional input a hidden parameter α and a credential *cred*. Algorithm ShowVerify outputs bit $b \in \{0,1\}$.

$\diamond$

Intuitively, correctness for an UAC system means that all credentials output by algorithm ReceiveCred and ReceiveUpd are valid on the attributes determined by the corresponding update function υ and hidden parameter α. We give a description of the usage of an UAC system, before we formally define correctness and security in the form of anonymity and soundness for UAC systems.

5.2.1 Description of an UAC System

Let us describe how the roles issuer, user, and verifier use an UAC system UAC as defined above. In UAC each of the roles can be instantiated multiple times, hence there are multiple issuers, users and verifiers in an UAC system.

Setup. We use an explicit setup algorithm Setup run by a trusted party that generates public parameters such as a group description to make our setup compatible with other schemes and applications that run on the same public parameters. For example the public parameters can include parameters for the commitment. With that in place, everyone knows also the attribute universe $\mathcal{U}_{\mathsf{Attr}}$, e.g., $\mathcal{U}_{\mathsf{Attr}} = \mathbb{Z}_p$.

Key generation. Each issuer generates its own public and secret key by running $(pk, sk) \leftarrow \mathsf{IssuerKeyGen}(pp, 1^n)$, where each issuer chooses n to be length of the attribute vectors that it uses in the issuing of credentials, e.g., the attribute space of an issuer is $\mathcal{U}^n_{\mathsf{Attr}} = \mathbb{Z}^n_p$. An issuer with key-pair (pk, sk) uses its secret key sk to issue credentials on attributes $\mathbb{A} \in \mathcal{U}^n_{\mathsf{Attr}}$ and to update previously issued credentials.

Issuing and updating credentials. User can either get a fresh credential issued by an issuer or get an update of an existing credential. We first describe the update protocol since issuing is a special case of an update. Let credential *cred* be the existing credential of a user with attributes $\mathbb{A}$ that it wants to update with an issuer identified by public key *pk*. To run the update protocol the user and issuer agree on an update function $\upsilon : \mathcal{U}^n_{\mathsf{Attr}} \times \{0,1\}^* \to \mathcal{U}^n_{\mathsf{Attr}} \cup \{\perp\}$ that takes attributes as its first argument and a hidden parameter α chosen by (and only known to) the user as its second argument such that $\upsilon(\mathbb{A}, \alpha) \neq \perp$, otherwise the update ends with a failure symbol. Note. a user should never start an update where $\upsilon(\mathbb{A}, \alpha) = \perp$. Informally, we consider appropriately chosen update function υ to include a check of the attributes and hidden parameter before applying the update. With the common input fixed, the user and issuer run the update protocol where the user runs ppt algorithm $\mathsf{ReceiveUpd}(pp, pk, ctx, \upsilon, \alpha, cred)$ and the issuer runs ppt algorithm $\mathsf{UpdateCred}(pp, pk, ctx, \upsilon, sk)$. The result of this protocol is that the user gets an updated credential $cred^*$ on attributes $\mathbb{A}^*$ determined by the update function υ, i.e., $\mathbb{A}^* = \upsilon(\mathbb{A}, \alpha)$, or the failure symbol $\perp$, e.g., if $\upsilon(\mathbb{A}, \alpha) = \perp$ or the protocol aborted otherwise. On the issuer side algorithm $\mathsf{UpdateCred}$ produces as an output a bit *b*. Intuitively, the output bit *b* indicates for the issuer, if the update was successful or not. The correctness of UAC guarantees that, if the bit *b* is 1, then we have that the update function "worked", i.e., $\upsilon(\mathbb{A}, \alpha) \neq \perp$.

As already mentioned the issuing of credentials is a special case of an update. Intuitively, the issuer updates an empty credential that has no attributes and during the update the issuer adds attributes to the credential. First the user and issuer agree on an update function $\upsilon : \{\perp\} \times \{0,1\}^* \to \mathcal{U}^n_{\mathsf{Attr}} \cup \{\perp\}$ that takes no existing attributes as its first argument, we use $\perp$ for this, and as a second argument a hidden attribute α chosen by the user, such that $\upsilon(\perp, \alpha) \neq \perp$. By this the first input is hardcoded in the update function υ. Then user and issuer run the protocol where the user runs the ppt algorithm $\mathsf{ReceiveCred}(pp, pk, ctx, \upsilon, \alpha)$ and the issuer runs the ppt algorithm $\mathsf{IssueCred}(pp, pk, ctx, \upsilon, sk)$. To finish the protocol algorithm $\mathsf{ReceiveCred}$ outputs a new credential *cred* on attributes $\mathbb{A} = \upsilon(\perp, \alpha)$ on the user side. If the protocol aborts the algorithm $\mathsf{ReceiveCred}$ outputs the failure symbol, e.g., if $\upsilon(\perp, \alpha) = \perp$.

Let us describe the role of the update function in more detail. In the simplest case an issuer just wants to issue a credential on some known attributes, e.g., $\mathbb{A} = (a, b, c, d)$. Hence no attributes are hidden to the issuer. In this case the update function is set to $\upsilon(\perp, \alpha) = \mathbb{A}$ where the hidden parameter α is ignored. If for example the user wants to hide some of the attributes from the issuer, then the update function is set to $\upsilon(\perp, \alpha = (u, v)) = (a, b, c + u, u + v)$ where the attributes a, b, and c are known to the issuer, however attributes u and v are only known to the user. The update function is very versatile and can also include further checks of the user's attributes and hidden parameter. For example the update function υ can include a check of the hidden parameter α to restrict the choice of the user, e.g., we can define that $\upsilon(\perp, \alpha = (u, v))$ outputs the failure symbol $\perp$ if $u + v > 50$. With this modeling we capture scenarios where only specific

updates are allowed. To look ahead, the checks in the update function can be instantiated by an argument system, e.g., a range proof that $u + v < 50$ holds. If a user and issuer wants to update a credentials they also agree on an update function υ that includes the existing attributes. For example, consider for any attributes $\mathbb{A} = (a, b, c, d)$ an update function $\upsilon(\mathbb{A}, \alpha) = (a + 8, b, c + 20, d + 9 + \alpha)$ meaning that the first element of an attribute vector is increased by 8, the third by 20, and the last by $9 + \alpha$. Here, the values +8, +20, and +9 are hardcoded in the update function υ and known to the user and issuer. However, the hidden parameter α is only known to the user.

Showing credentials. Users can show credentials and prove predicates over their attributes certified by credentials to any verifier in the system. To show a credential *cred* on attributes $\mathbb{A}$ issued by an issuer with public key *pk* a user and a verifier agree on a predicate $\chi : \mathcal{U}_{\mathsf{Attr}}^n \times \{0,1\}^* \to \{0,1\}$. For this, the user chooses a hidden parameter α such that the predicate evaluates to 1, i.e., $\chi(\mathbb{A}, \alpha) = 1$. Next, the user and verifier execute the protocol where the user runs the ppt algorithms $\mathsf{ShowCred}(pp, pk, ctx, \chi, \alpha, cred)$ and the verifier runs ppt algorithm $\mathsf{ShowVerify}(pp, pk, ctx, \chi)$. The algorithm $\mathsf{ShowVerify}$ outputs a bit b at the end of the protocol on the verifier side. If $b = 1$ it indicates that the user has a valid credential under issuer public key *pk* on some attributes $\mathbb{A}'$ such that there is a hidden parameter α' where $\chi(\mathbb{A}', \alpha') = 1$. Hence, the verifier gets information, if the predicate χ does hold or not with respect to the user's attributes and hidden parameter. Note, that predicate χ can perform checks on the attributes *and* the hidden parameter.

In above description of the usage of an UAC system the instantiation of the update functions and predicates are left open. In Section 5.3 we show a generic UAC construction that relies on non-interactive argument systems and blind signatures to instantiate the update functions and predicates in a meaningful way.

A note on pseudonyms in AC. Anonymous credential (AC) are usually defined with users having a user secret *usk* and the ability for users to generate pseudonyms N from *usk*, e.g., [BB18; BCKL08; BL13; CL01; CL03; CL04; FHS19; PS16]. Each credential then includes the user secret *usk* as one of its attributes where the issuer of the credential does not learn *usk*, but it learns the pseudonym N. This means, users can have multiple pseudonyms under which different issuers can recognize them. Our UAC definition generalizes this by supporting hidden attributes during issuing and update via the hidden parameter α that the user chooses. In detail, a user with user secret *usk* computes a pseudonym N as a commitment to *usk*, i.e., $N \leftarrow \mathsf{Commit}(pp, pk, usk, o)$ with randomness o. Thus, the opening of the commitment (pseudonym N) is (usk, o). Then, in an issue protocol for attributes $\mathbb{A}$ and pseudonym N the user sets its hidden parameter $\alpha := (usk, o)$ and agrees with the issuer on an update function υ that verifies if (usk, o) is the opening of pseudonym N and that sets one of the attribute values

to the user secret usk. In detail the update function υ is defined like this:

$$\upsilon(\perp, \alpha) = \begin{cases} (usk, \mathbb{A}), \text{ if } N = \mathsf{Commit}(pp, pk, usk, o) \\ \perp, \text{ otherwise} \end{cases}$$

By this, the user proves during the issuing that it is the user that created the pseudonym N. In the update protocol, user and issuer have to make sure that the used update function does not update the user secret usk. To support pseudonyms in the show protocol with a credential $cred$ the predicate χ is also modified such that it checks if (usk, o) is the opening of pseudonym N where the user sets the hidden parameter $\alpha := o$ and the user secret usk is checked against the user secret of the credential $cred$. This ensures that the credential $cred$ belongs to the user known under pseudonym N. Concerning security, if the commitment is computational hiding, then our simulation-based anonymity definition for UAC captures that the commitment to a user secret can be simulated without knowing the opening. Also, if the commitment is computationally binding, then the extractor Extr of the soundness definition can extract an opening (usk, o) of the commitment (pseudonym) N. Hence, if a user knows two openings of a pseudonym N, then it holds that a consistent explanation list L contains two openings of the same pseudonym N which breaks the binding property.

Next, we focus on the correctness and security definitions of UAC. To define correctness formally we have to introduce the notion of valid credentials. For this, we define an algorithm $\mathsf{ValidCred}(pp, pk, cred, \mathbb{A})$ that determines whether a credential $cred$ is valid on attributes $\mathbb{A}$ under an issuer public key pk and public parameters pp.

Definition 5.2.2 (Valid Credentials)
Let $\mathsf{ValidCred}(pp, pk, cred, \mathbb{A})$ be an algorithm for an updatable anonymous credential system UAC that on input public parameters pp, issuer public key pk, credential $cred$, and attributes $\mathbb{A}$ works as follows.

- if $cred \in \mathsf{ReceiveCred}(pp, pk, ctx, \upsilon, \alpha)$, where $ctx \in \{0,1\}^*$, and $cred \neq \perp$, then it holds that

$$\mathsf{ValidCred}(pp, pk, cred, \mathbb{A} := \upsilon(\perp, \alpha)) = 1,$$

- if $cred^* \in \mathsf{ReceiveUpd}(pp, pk, ctx, \upsilon, \alpha, cred)$, where $ctx \in \{0,1\}^*$, and $cred^* \neq \perp$ and $\mathsf{ValidCred}(pp, pk, cred, \mathbb{A}) = 1$ then it holds that

$$\mathsf{ValidCred}(pp, pk, cred^*, \mathbb{A}^* := \upsilon(\mathbb{A}, \alpha)) = 1,$$

- in every other case it holds that $\mathsf{ValidCred}$ outputs 0.

$\diamond$

Note, algorithm $\mathsf{ValidCred}$ is not required to be efficient and serves as a tool in the definitions of correctness and anonymity. Intuitively, a UAC system is correct, if under an honest setup all honestly generated credentials are valid (according to algorithm $\mathsf{ValidCred}$) and a corresponding run of the verify protocol (update protocol) end with output 1 on the verifier side (issuer side). We require that an UAC system is correct and define it formally in the following.

Definition 5.2.3 (Correctness)
An updatable anonymous credential system UAC is correct, if for all $\lambda, n \in \mathbb{N}$, all $pp \in \mathsf{Setup}(1^\lambda)$, all issuer key pairs $(pk, sk) \in \mathsf{IssuerKeyGen}(pp, 1^n)$, all contexts $ctx \in \{0,1\}^*$, all credentials $cred$, all attributes $\mathbb{A} \in \mathcal{U}_{\mathsf{Attr}}$, whenever it holds that $\mathsf{ValidCred}(pp, pk, cred, \mathbb{A}) = 1$, we have that

- for all predicates $\chi \in \mathcal{X}$, and all hidden parameters $\alpha \in \{0,1\}^*$ such that $\chi(\mathbb{A}, \alpha) = 1$, it holds that

$$\Pr[\mathsf{ShowCred}(pp, pk, ctx, \chi, \alpha, cred) \\ \leftrightarrow \mathsf{ShowVerify}(pp, pk, ctx, \chi) \rightarrow 0] = \mathsf{negl}(\lambda)$$

- for all update functions $\upsilon \in \Upsilon$, all hidden parameters $\alpha \in \{0,1\}^*$ such that $\upsilon(\mathbb{A}, \alpha) \neq \perp$, it holds that

$$\Pr[\perp = cred^* \leftarrow \mathsf{ReceiveUpd}(pp, pk, ctx, \upsilon, \alpha, cred) \\ \leftrightarrow \mathsf{UpdateCred}(pp, pk, ctx, \upsilon, sk) \rightarrow 0] = \mathsf{negl}(\lambda).$$

$\diamond$

Next, we define the security of UAC with the notion of anonymity for the user and soundness for the issuer and verifier.

5.2.2 Anonymity

Intuitively, the anonymity notion captures that a user does not leak private information by interacting with an adversarial issuer or verifier. Hence, we require for anonymity that there are simulators for the user side of the issue, update, and show protocols. Further, we require that they simulate the interaction with an adversary without having access to private user information, i.e., without access to attributes, hidden parameters, and credentials, in such a way that the adversary cannot distinguish if it interacts with the real protocol algorithms or the simulators.

In the following experiment the adversary acts as dishonest issuers and verifiers, where the experiment executes the user side of the protocols honestly with the given parameters. The adversary can ask oracles to issue a credential with the adversary, to let the adversary update an existing credential, and to show an existing credential to the adversary.

Definition 5.2.4 (Anonymity)
Let UAC be an updatable anonymous credential system and let $\mathcal{D}$ be a distinguisher. We define the advantage in the experiment $\mathrm{Exp}^{\mathsf{anon},b}_{\mathcal{D},\mathsf{UAC}}(\lambda)$ (Experiment 5.1) as

$$\mathrm{Adv}^{\mathsf{anon}}_{\mathcal{D},\mathsf{UAC}}(\lambda) \coloneqq \left|\Pr\left[\mathrm{Exp}^{\mathsf{anon},0}_{\mathcal{D},\mathsf{UAC}}(\lambda) = 1\right] - \Pr\left[\mathrm{Exp}^{\mathsf{anon},1}_{\mathcal{D},\mathsf{UAC}}(\lambda) = 1\right]\right| \ .$$

and call UAC (computational) *anonymous*, if there exists a (stateful) ppt simulator $\mathsf{Sim} \coloneqq (\mathsf{SimReceiveCred}, \mathsf{SimReceiveUpd}, \mathsf{SimShowCred})$ such that for every ppt distinguishers $\mathcal{D}$ it holds that

$$\mathrm{Adv}^{\mathsf{anon}}_{\mathcal{D},\mathsf{UAC}}(\lambda) = \mathsf{negl}(\lambda) \ .$$

◇

Similar to argument systems in the ROM Section 2.3.5, if the simulator Sim in the above definition is stateful then we require that the state is shared between the algorithms $\mathsf{SimReceiveCred}$, $\mathsf{SimReceiveUpd}$, $\mathsf{SimShowCred}$, i.e., they share a common state that gets implicitly updated by each run of the algorithms.

5.2.3 Soundness

Formally defining soundness for UAC is not straightforward, since issuers (and verifiers) do not necessarily know the attributes that are issued or are a result of an update. Usually in AC one defines soundness such that it should be infeasible for an adversary (acting as dishonest users) to show a credentials that it did not get from an issuer. Meaning the adversary should not be able to forge a valid credential on some attributes. However, in an UAC system with its hidden parameter for issuing and updating credentials the issuers, verifiers and therefore also the experiment do not know the hidden attributes (and credentials). Thus, we cannot define a typical winning condition that checks if the forgery is a valid credential on attributes that the experiment (the oracles) never generated. Hence, we have to define soundness differently. Intuitively, we require that it is infeasible for an adversary to run issue, update and show protocols such that there is no explanation of what happened given the involved update functions, attributes, and hidden parameters of the protocols. In detail, there has to be an efficient extraction algorithm that extracts the used hidden parameters and attributes and generates a consistent explanation of the state before and after each protocol run. Thus, our soundness notion is comparable to extractability definitions as it is the case in argument systems (Section 2.3.5).

Before defining the soundness of UAC we have to define explanation lists. An explanation list L contains which attributes and hidden parameters were used in protocols. To explain the usability of this, let us consider a series of protocol runs (issue, update, show) where we are the issuer and verifier. This means, we do not know the attributes and hidden parameters of any of the protocols. Further, let the

$\mathrm{Exp}^{\mathsf{anon},b}_{\mathcal{D},\mathsf{UAC}}(\lambda)$

1 : $pp \leftarrow \mathsf{Setup}(1^\lambda), L_{\mathsf{cred}} := \emptyset$
2 : $(pk, st) \leftarrow \mathcal{D}(pp)$
3 : $\tilde{b} \leftarrow \mathcal{D}^{\mathcal{O}^{\mathsf{ReceiveCred}_b}_{pp,pk}, \mathcal{O}^{\mathsf{ReceiveUpd}_b}_{pp,pk}, \mathcal{O}^{\mathsf{ShowCred}_b}_{pp,pk}}(st)$
4 : **return** $\tilde{b}$

$\mathcal{O}^{\mathsf{ReceiveCred}_1}_{pp,pk}(\upsilon, \alpha, ctx)$

1 : **if** $\upsilon \notin \Upsilon \vee \upsilon(\bot, \alpha) = \bot$ **then**
2 : **ignore**
3 : $cred \leftarrow \mathsf{ReceiveCred}(pp, pk, ctx, \upsilon, \alpha) \leftrightarrow \mathcal{D}$
4 : **if** $cred = \bot$ **then**
5 : **return** $\bot$
6 : **else**
7 : $L_{\mathsf{cred}} := L_{\mathsf{cred}} \cup cred$

$\mathcal{O}^{\mathsf{ShowCred}_1}_{pp,pk}(j, \chi, \alpha, \mathbb{A}, ctx)$

1 : **parse** $L_{\mathsf{cred}}[j] = cred$
2 : **if** $cred = \bot \vee$
3 : $\chi \notin \mathcal{X} \vee \chi(\mathbb{A}, \alpha) = 0$ **then**
4 : **ignore**
5 : **if** $\mathsf{ValidCred}(pp, pk, ctx, cred, \mathbb{A}) = 0$ **then**
6 : **ignore**
7 : $\mathsf{ShowCred}(pp, pk, \chi, \alpha, cred) \leftrightarrow \mathcal{D}$

$\mathcal{O}^{\mathsf{ReceiveUpd}_1}_{pp,pk}(j, \upsilon, \alpha, \mathbb{A}, ctx)$

1 : **parse** $L_{\mathsf{cred}}[j] = cred$
2 : **if** $cred = \bot \vee \upsilon \notin \Upsilon \vee \upsilon(\mathbb{A}, \alpha) = \bot$ **then**
3 : **ignore**
4 : **if** $\mathsf{ValidCred}(pp, pk, cred, \mathbb{A}) = 0$ **then**
5 : **ignore**
6 : $cred^* \leftarrow \mathsf{ReceiveUpd}(pp, pk, ctx, \upsilon, \alpha, cred) \leftrightarrow \mathcal{D}$
7 : **if** $cred^* = \bot$ **then**
8 : **return** $\bot$
9 : **else**
10 : $L_{\mathsf{cred}} := L_{\mathsf{cred}} \cup cred^*$

$\mathcal{O}^{\mathsf{ReceiveCred}_0}_{pp,pk}(\upsilon, \alpha, ctx)$

1 : **if** $\upsilon \notin \Upsilon \vee \upsilon(\bot, \alpha) = \bot$ **then**
2 : **ignore**
3 : $cred \leftarrow \mathsf{SimReceiveCred}(pp, pk, ctx, \upsilon) \leftrightarrow \mathcal{D}$
4 : **if** $cred = \bot$ **then**
5 : **return** $\bot$
6 : **else**
7 : $L_{\mathsf{cred}} := L_{\mathsf{cred}} \cup cred$

$\mathcal{O}^{\mathsf{ReceiveUpd}_0}_{pp,pk}(j, \upsilon, \alpha, \mathbb{A}, ctx)$

1 : **parse** $L_{\mathsf{cred}}[j] = cred$
2 : **if** $cred = \bot \vee \upsilon \notin \Upsilon \vee \upsilon(\mathbb{A}, \alpha) = \bot$ **then**
3 : **ignore**
4 : **if** $\mathsf{ValidCred}(pp, pk, cred, \mathbb{A}) = 0$ **then**
5 : **ignore**
6 : $cred^* \leftarrow \mathsf{SimReceiveUpd}(pp, pk, ctx, \upsilon) \leftrightarrow \mathcal{D}$
7 : **if** $cred^* = \bot$ **then**
8 : **return** $\bot$
9 : **else**
10 : $L_{\mathsf{cred}} := L_{\mathsf{cred}} \cup cred^*$

$\mathcal{O}^{\mathsf{ShowCred}_0}_{pp,pk}(j, \chi, \alpha, \mathbb{A}, ctx)$

1 : **parse** $L_{\mathsf{cred}}[j] = cred$
2 : **if** $cred = \bot \vee$
3 : $\chi \notin \mathcal{X} \vee \chi(\mathbb{A}, \alpha) = 0$ **then**
4 : **ignore**
5 : **if** $\mathsf{ValidCred}(pp, pk, cred, \mathbb{A}) = 0$ **then**
6 : **ignore**
7 : $\mathsf{SimShowCred}(pp, pk, ctx, \chi) \leftrightarrow \mathcal{D}$

Experiment 5.1: Anonymity experiment for updatable anonymous credentials.

series of protocol runs end with a ShowVerify run that outputs $b = 1$. We would like to know how this output came to be. For this we need to know which attributes and hidden parameters were used such that ShowVerify outputs $b = 1$. We require that if an explanation list L is consistent (with the series of protocol runs), then it contains a series of entries consisting of attributes and hidden parameters starting from an issue of a credential, over potentially several updates, to the final show protocol run. Intuitively, the entries (attributes, hidden parameters) explain the state before and after a protocol run. This means an explanation list L explains all attribute vectors that are issued or are a result of an update. We build the explanation list L like this, because old credentials stay valid even after an update or issuance of new credentials to a user.

Definition 5.2.5 (Explanation List)
Let UAC be an updatable anonymous credential system, let $\mathcal{A}$ be an adversary, let $pp \leftarrow \mathsf{Setup}(1^\lambda)$, and let $(pk, sk) \leftarrow \mathsf{IssuerKeyGen}(pp, 1^n)$ for some $n \in \mathbb{N}$. We define an explanation list L for UAC that lists which attributes and hidden parameters adversary $\mathcal{A}$ running the user side of protocols used in a series of issue, update, and show protocols as follows. Explanation list L consists of one entry per protocol where adversary $\mathcal{A}$ runs the user side, i.e., ReceiveCred, ReceiveUpd, or ShowVerify respectively. The other side of the protocols (i.e., IssueCred, UpdateCred, ShowVerify) is executed honestly (by an experiment). Hence, the i-th entry of explanation list L is

- a tuple $(\mathbb{A}_i, \alpha_i)$, if algorithm ShowVerify or UpdateCred was executed.
- a hidden parameter α_i, if IssueCred was executed.

Next, we inductively define explanation sets E_i of attributes that correspond to the explanation list L and therefore to the attributes that adversary $\mathcal{A}$ is expected to have *after* the i-th protocol finished. The sets start empty, hence $E_0 := \emptyset$. Then:

- if i-th protocol algorithm is ShowVerify, we set $E_i := E_{i-1}$. Here, no credentials are issued.
- if the i-th protocol executes $\mathsf{IssueCred}(pp, pk, ctx, \upsilon, sk)$ with update function υ, we set $E_i := E_{i-1} \cup \{\upsilon(\bot, \alpha_i)\}$ if $\upsilon(\bot, \alpha_i) \neq \bot$. Here, we expect that the output for adversary $\mathcal{A}$ is a credential with attributes $\mathbb{A}' = \upsilon(\bot, \alpha_i)$.
- if the i-th protocol executes $\mathsf{UpdateCred}(pp, pk, ctx, \upsilon, sk)$ and the output of algorithm UpdateCred is 1 we set $E_i := E_{i-1} \cup \{\upsilon(\mathbb{A}_i, \alpha_i)\}$ if $\upsilon(\mathbb{A}_i, \alpha_i) \neq \bot$. Here, we expect that the output for adversary $\mathcal{A}$ is an updated credential with attributes $\mathbb{A}^* = \upsilon(\mathbb{A}_i, \alpha_i)$. If the output of algorithm UpdateCred is 0 we set $E_i := E_{i-1}$. Here, we expect that no credential was output to the adversary $\mathcal{A}$.

We call an explanation list L *consistent*, if it includes entries such that

- if the i-th protocol was a show protocol and algorithm $\mathsf{ShowVerify}(pp, pk, ctx, \chi)$ output 1, then the i-th list entry $(\mathbb{A}_i, \alpha_i)$ satisfies χ, i.e., $\chi(\mathbb{A}_i, \alpha_i) = 1$ and attributes $\mathbb{A}_i$ is a result of a previous issue or update protocol run, i.e., it holds that $\mathbb{A}_i \in E_{i-1}$.
- if the i-th protocol was an update protocol and algorithm $\mathsf{UpdateCred}(pp, pk, ctx, \upsilon, sk)$ output 1, then for the i-th list entry $(\mathbb{A}_i, \alpha_i)$ it holds $\upsilon(\mathbb{A}_i, \alpha_i) \neq \bot$ and attributes $\mathbb{A}_i$ are a result of a previous issue or update protocol run, i.e., it holds that $\mathbb{A}_i \in E_{i-1}$.

$\diamond$

With this in place we can formally define soundness for UAC. Here, we require the existence of an efficient extractor that generates an explanation list L. This means for every protocol of an adversary with an honest issuer and verifier there is an explanation what happened, i.e., which attributes and hidden parameters were used to start the protocol and what was the corresponding outcome of the protocol. The soundness notion then requires that there is an extractor, such that for all adversaries it should be infeasible to produce a series of protocol runs, such that the explanation list generated by the extractor is not consistent. If an adversary is indeed successful then it forged a credential on attributes (and hidden parameters) that it did not get from the series of protocols. The following definition also considers soundness in the ROM. Note, if we do not consider soundness in the ROM the adversary does not get oracles access to the random oracle and the same definition can be used.

Definition 5.2.6 (Soundness)
Let UAC be an updatable anonymous credential system and let $\mathcal{A}$ be an adversary (and let RO a random oracle). We define the advantage in the experiment $\mathrm{Exp}^{\mathsf{sound}}_{\mathcal{A},\mathsf{UAC}}(\lambda)$ (Experiment 5.2) as

$$\mathrm{Adv}^{\mathsf{sound}}_{\mathcal{A},\mathsf{UAC}}(\lambda) \coloneqq \Pr\Big[\mathrm{Exp}^{\mathsf{sound}}_{\mathcal{A},\mathsf{UAC}}(\lambda) = 1\Big] \ .$$

and call UAC *sound* (in the ROM), if there exists a ppt extractor Extr such that for all ppt adversaries $\mathcal{A}$ it holds that

$$\mathrm{Adv}^{\mathsf{sound}}_{\mathcal{A},\mathsf{UAC}}(\lambda) = \mathsf{negl}(\lambda) \ .$$

$\diamond$

5.3 Generic UAC Construction

In this section we define our generic UAC construction based on blind signatures and an argument system and show a concrete instantiation. We present here

$\mathrm{Exp}^{\mathsf{sound}}_{\mathcal{A},\mathsf{UAC}}(\lambda)$

$Q_I, Q_U, Q_V, Q_H = \emptyset$
$pp \leftarrow \mathsf{Setup}(1^\lambda)$
$(1^n, st) \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{RO}}}(pp)$
$(pk, sk) \leftarrow \mathsf{IssuerKeyGen}(pp, 1^n)$
$\mathsf{halt} \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{IssueCred}}_{pp,pk,sk}, \mathcal{O}^{\mathsf{UpdateCred}}_{pp,pk,sk}, \mathcal{O}^{\mathsf{ShowVerify}}_{pp,pk}, \mathcal{O}^{\mathsf{RO}}}(pk, st)$
$L \leftarrow \mathsf{Extr}(pp, pk, sk, Q_I, Q_U, Q_V, Q_H)$
if L is *not* consistent **then**
  return 1
else return 0

$\mathcal{O}^{\mathsf{IssueCred}}_{pp,pk,sk}(\upsilon, ctx)$

if $\upsilon \notin \Upsilon$ **then**
  ignore
$\mathcal{A} \leftrightarrow \mathsf{IssueCred}(pp, pk, ctx, \upsilon, sk)$
Let st^I be the exchanged messages
$Q_I \coloneqq Q_I \cup \{(\upsilon, ctx, st^I)\}$

$\mathcal{O}^{\mathsf{UpdateCred}}_{pp,pk,sk}(\upsilon, ctx)$

if $\upsilon \notin \Upsilon$ **then**
  ignore
$\mathcal{A} \leftrightarrow \mathsf{UpdateCred}(pp, pk, ctx, \upsilon, sk) \to b$
Let st^U be the exchanged messages
$Q_U \coloneqq Q_U \cup \{(\upsilon, b, ctx, st^U)\}$

$\mathcal{O}^{\mathsf{ShowVerify}}_{pp,pk}(\chi, ctx)$

if $\chi \notin \mathcal{X}$ **then** **ignore**
$\mathcal{A} \leftrightarrow \mathsf{ShowVerify}(pp, pk, ctx, \chi) \to b$
Let st^V be the exchanged messages
$Q_V \coloneqq Q_V \cup \{(\chi, b, ctx, st^V)\}$

$\mathcal{O}^{\mathsf{RO}}(v_i)$

$y_i \leftarrow \mathsf{RO}(v_i)$
$Q_H = Q_H \cup \{(v_i, y_i)\}$
return y_i

Experiment 5.2: Soundness experiment for updatable anonymous credential

the generic construction in the random oracle model (ROM), because we rely on argument systems in the ROM to allow efficient instantiations. To this end, we present a concrete and efficient instantiation based on the blind signature scheme by Pointcheval and Sanders [PS16] and non-interactive argument systems based on the Fischlin transform [BFW15; Fis05] in Section 5.5. We also discuss the generic UAC construction in the CRS model in Section 5.4.3.

Let us start with an intuitive description of our generic UAC construction called $\mathsf{UAC}^{\mathsf{gc}}$. On a high level, the issuer uses the blind signature scheme to issue credentials on attributes and the argument system is used by the users to prove statements about their attributes and credentials. In more detail, consider a user that wants

to get a credential by an issuer (known under its public key pk) on attributes $\mathbb{A} = (u, a, c)$, where u and a should be private and c is public, e.g., the issuer gives such an attribute to all users like c being a counter of how many purchases a user has made (initially 0). For the issue protocol between the user and the issuer both agree on the corresponding update function υ, i.e., $\upsilon(\perp, \alpha \coloneqq (u, a)) \coloneqq (u, a, c)$. To get a credential on the attributes $\mathbb{A} = (u, a, c)$ the user computes a commitment *com* to attributes $\mathbb{A}$ and proves that it is well-formed, i.e., the user knows the hidden parameters α such that $\upsilon(\perp, \alpha) = \mathbb{A}$ and the commitment randomness r that were used to compute *com*. Together attributes $\mathbb{A}$ and *com* form the opening of the commitment. The issuer gets this proof and by checking that it is valid, the issuer can be sure that, even if it signs the commitment *com* blindly (without knowing the opening), the committed attributes are valid with respect to the update function υ. This is because, if the issuer does not agree with the update function υ, it does not start the protocol with the user. Thus, the issuing proceeds by running the blind signing protocol that the user initiates. This means, the issuer blindly signs the commitment *com* and the user receives a signature σ on attributes $\mathbb{A} = (u, a, c)$. The user checks the validity of the signature σ. If it is valid the user sets the credential $cred \coloneqq (\mathbb{A}, \sigma)$ and outputs it such that it can be stored. If the signature σ is not valid the user outputs the failure symbol $\perp$. For example this can be a trigger to retry the issue protocol.

After the user has made a purchase the issuer offers the user to update the user's credential $cred = (\mathbb{A} = (u, a, c), \sigma)$ by increasing the purchases counter attribute c by 1. For this example let attribute a be the users favorite item and suppose this has changed and has to be updated as well. To do this, both interact in an update protocol, where the update function υ' is defined such that $\upsilon'(\mathbb{A} = (u, a, c), \alpha \coloneqq favorite) = (u, a \coloneqq favorite, c + 1)$. Here *favorite* is the user's favorite item of the last purchase and is for privacy reasons hidden to the issuer. Then, the user computes commitment *com* to the updated attributes $\mathbb{A}^* \coloneqq (u, a = favorite, c+1)$. Then the user proves that the commitment *com* is well-formed (as in the issue protocol). Additionally, the user proves that its credentials $cred = (\mathbb{A} = (u, a, c), \sigma)$ is valid, i.e., that the signature σ on attributes $\mathbb{A}$ is valid. The remainder of the protocol proceeds as the issue protocol, i.e., the issuer checks the proof, then generates a blind signature σ^* on the commitment *com*, and the user checks the signature and generates the output accordingly. If all checks are valid, the user receives at the end of the issue protocol a credential $cred^*$ on the updated attributes $\mathbb{A}^*$, i.e., $cred^* \coloneqq (\mathbb{A}^*, \sigma^*)$. The user can now show this credential to any verifier, e.g., to get a discount, because the purchase counter is higher than a threshold k and the user's favorite item is contained in a discount list of the verifier. These conditions are then used to define a predicate χ for the show protocol. In the example the predicate χ checks if the purchase counter attribute c is higher than the threshold k and checks that the favorite item attribute a is equal to one of the items on the discount list. In the show protocol, the user then proves that its credential $cred = (\mathbb{A}, \sigma)$ is valid, i.e., that σ is a valid signature on attributes $\mathbb{A}$, and that the predicate evaluates to 1, i.e., $\chi(\mathbb{A}, \alpha \coloneqq \perp) = 1$. To extend the example

to also use the hidden parameter α, let us consider that the user additionally has to prove that it has access to an accepted payment option. Hence, let the hidden attribute α be a signature by an accepted payment provider. Then, the predicate χ also checks that the signature is valid under a public key of an accepted payment provider. The user then sends the proof to the verifier which gives the discount to the user, if the proof is valid. In general the user can authenticate towards a verifier without revealing any of its attributes. The expressiveness comes from the predicate in the show protocol that allows the user to prove the validity of statements over its attributes and hidden parameter without revealing the concrete attributes or the hidden parameter.

Next, we formally define the generic UAC construction $\mathsf{UAC}^{\mathsf{gc}}$. For this we require that the setups of the blind signature scheme and argument system are compatible, i.e., they both work with same public parameters (e.g., a group description).

Construction 5.3.1 (Generic UAC construction)
Let $\mathsf{BS} = (\mathsf{BS.Setup}, \mathsf{BS.KeyGen}, \mathsf{BS.Commit}, \mathsf{BS.BlindSign}, \mathsf{BS.BlindRcv}, \mathsf{BS.Verify})$ be a blind signature scheme (Definition 2.3.7). Further, let $\mathsf{Arg} = (\mathsf{Arg.Prove}^{\mathsf{RO}}, \mathsf{Arg.Verify}^{\mathsf{RO}})$ be an argument system in the ROM (Definition 2.3.27). We define the generic UAC construction $\mathsf{UAC}^{\mathsf{gc}}$ as follows.

- $pp \leftarrow \mathsf{Setup}(1^\lambda)$ outputs public parameters $pp \leftarrow \mathsf{BS.Setup}(1^\lambda)$ where the message space parameters included in pp define the attribute universe $\mathcal{U}_{\mathsf{Attr}}$.
- $(pk, sk) \leftarrow \mathsf{IssuerKeyGen}(pp, 1^n)$ outputs a blind signature key pair $(pk, sk) \leftarrow \mathsf{BS.KeyGen}(pp, 1^n)$. Public key pk defines the message space $\mathcal{M}^n$ of the blind signature scheme and hence the attribute space $\mathcal{U}^n_{\mathsf{Attr}} := \mathcal{M}^n$ of the issuer. The update function universe Υ is defined as

$$\begin{aligned}\Upsilon := \{\upsilon \mid \upsilon : \mathcal{U}^n_{\mathsf{Attr}} \times \{0,1\}^* \to \mathcal{U}^n_{\mathsf{Attr}} \cup \{\bot\} \\ \vee\ \upsilon : \{\bot\} \times \{0,1\}^* \to \mathcal{U}^n_{\mathsf{Attr}} \cup \{\bot\}\},\end{aligned}$$

  and the predicate universe is defined as

$$\mathcal{X} := \{\chi \mid \chi : \mathcal{U}^n_{\mathsf{Attr}} \times \{0,1\}^* \to \{0,1\}\}$$

  and we require that they are supported by the argument system Arg.

- $cred \leftarrow \mathsf{ReceiveCred}(pp, pk, ctx, \upsilon, \alpha) \leftrightarrow \mathsf{IssueCred}(pp, pk, ctx, \upsilon, sk)$ for $\upsilon \in \Upsilon$ proceeds as follows.
  - The user computes its attributes $\mathbb{A} = \upsilon(\bot, \alpha)$ and then a commitment to the attributes $\mathbb{A}$ by $com \leftarrow \mathsf{BS.Commit}(pp, pk, \mathbb{A}, r)$ where $r \leftarrow \mathcal{R}^{\mathsf{Com}}$
  - Next, the user computes the proof

$$\pi = \mathsf{Arg}[(\alpha, r) : com = \mathsf{BS.Commit}(pp, pk, \upsilon(\bot, \alpha), r)](ctx)$$

    and sends the proof π together with the commitment com to the issuer.

  - Then, the user starts the blind signing protocol with the issuer by running algorithm $\mathsf{BS.BlindRcv}(pp, pk, \mathbb{A}, r)$
  - The issuer aborts, if the proof π is invalid, i.e., if it holds that

$$\mathsf{Verify}^{\mathsf{RO}}(pp, x := (pk, \upsilon, com), \pi, ctx) = 0 \ .$$

    Otherwise, the issuer runs algorithm $\mathsf{BS.BlindSign}(pp, pk, sk, com)$ in the blind signing protocol.
  - Finally, the user on signature output $\sigma \leftarrow \mathsf{BS.BlindRcv}(pp, pk, \mathbb{A}, r)$ it checks if the signature σ is valid by $b \leftarrow \mathsf{BS.Verify}(pp, pk, \mathbb{A}, \sigma)$. If $b = 1$ it outputs the credential $cred = (\mathbb{A}, \sigma)$, otherwise it outputs the failure symbol $\perp$.

- $cred^* \leftarrow \mathsf{ReceiveUpd}(pp, pk, ctx, \upsilon, \alpha, cred) \leftrightarrow \mathsf{UpdateCred}(pp, pk, ctx, \upsilon, sk) \rightarrow b$ for $\upsilon \in \Upsilon$ proceeds as follows.
  - The user parses the credential $cred = (\mathbb{A}, \sigma)$
  - Then the user computes the updated attributes $\mathbb{A}^*$ as $\mathbb{A}^* = \upsilon(\mathbb{A}, \alpha)$ and commits to the attributes $\mathbb{A}^*$ by $com \leftarrow \mathsf{BS.Commit}(pp, pk, \mathbb{A}^*, r)$ where $r \leftarrow \mathcal{R}^{\mathsf{Com}}$.
  - Next, the user computes the proof

$$\begin{aligned}\pi = &\mathsf{Arg}[(\mathbb{A}, \sigma, \alpha, r) : com = \mathsf{BS.Commit}(pp, pk, \upsilon(\mathbb{A}, \alpha), r) \\ &\wedge \mathsf{BS.Verify}(pp, pk, \mathbb{A}, \sigma) = 1](ctx)\end{aligned}$$

    and sends the proof π together with the commitment com to the issuer.
  - Then, the user starts the blind signing protocol with the issuer by running algorithm $\mathsf{BS.BlindRcv}(pp, pk, \mathbb{A}^*, r)$
  - The issuer outputs 0 and aborts, if the proof π is invalid, i.e., if it holds that

$$\mathsf{Verify}^{\mathsf{RO}}(pp, x := (pk, \upsilon, com), \pi, ctx) = 0 \ .$$

    Otherwise, the issuer runs $\mathsf{BS.BlindSign}(pp, pk, sk, com)$ in the blind signing protocol.
  - The user on signature output $\sigma^* \leftarrow \mathsf{BS.BlindRcv}(pp, pk, \mathbb{A}^*, r)$ it checks if the signature σ^* is valid by $b \leftarrow \mathsf{BS.Verify}(pp, pk, \mathbb{A}^*, \sigma^*)$. If $b = 1$ it outputs $cred^* = (\mathbb{A}^*, \sigma^*)$, otherwise it outputs the failure symbol $\perp$.
  - The issuer outputs 1 after the blind signing protocol finished.

- $\mathsf{ShowCred}(pp, pk, ctx, \chi, \alpha, cred) \leftrightarrow \mathsf{ShowVerify}(pp, pk, ctx, \chi) \rightarrow b$ for $\chi \in \mathcal{X}$ proceeds as follows.
  - The user parses $cred = (\mathbb{A}, \sigma)$
  - Then, the user aborts if $\chi(\mathbb{A}, \alpha) = 0$ and informs the issuer

- The issuer on abort by the user outputs 0
- The user, if $\chi(\mathbb{A}, \alpha) = 1$, computes the proof

$$\pi = \mathsf{Arg}[(\mathbb{A}, \sigma, \alpha) : \mathsf{BS.Verify}(pp, pk, \mathbb{A}, \sigma) = 1 \wedge \chi(\mathbb{A}, \alpha) = 1](ctx)$$

  and sends the proof π to the issuer.
- The issuer outputs 1, if the proof π is valid, i.e., if it holds that $\mathsf{Verify}^{\mathsf{RO}}(pp, x := (pk, \chi), \pi, ctx) = 1$, otherwise it outputs 0.

Correctness. The correctness of $\mathsf{UAC}^{\mathsf{gc}}$ follows from the perfect correctness of the blind signature scheme BS (Definition 2.3.8) and the correctness of the argument system Arg (Definition 2.3.28). Here, we instantiate the algorithm $\mathsf{ValidCred}$ with the verification algorithm of the blind signature scheme BS. In detail, we define algorithm $\mathsf{ValidCred}(pp, pk, cred, \mathbb{A})$ on input public parameters pp, issuer public key pk, credential $cred = (\mathbb{A}', \sigma)$, and attributes $\mathbb{A}$ to output $\mathsf{BS.Verify}(pp, pk, \mathbb{A}, \sigma)$. Hence, the perfect correctness of BS, that algorithm $\mathsf{BS.Verify}$ is deterministic, and that the issue, update, and show protocols rely on BS (credentials are blind signatures) gives us a way to verify credentials according to Definition 5.2.2. Together with the correctness of the argument system Arg (in the ROM) the correctness of the generic UAC construction $\mathsf{UAC}^{\mathsf{gc}}$ (Definition 5.2.3) follows directly.

5.4 Security of the Generic UAC Construction

In the following we show that the generic UAC construction $\mathsf{UAC}^{\mathsf{gc}}$ is perfectly anonymous and computationally sound. We show this in two separate theorems and we start with the anonymity of $\mathsf{UAC}^{\mathsf{gc}}$.

5.4.1 Anonymity

The anonymity of $\mathsf{UAC}^{\mathsf{gc}}$ follows from the perfect message privacy of the blind signature scheme BS and that the argument system Arg is zero-knowledge. Intuitively, that Arg is zero-knowledge gives us that the proofs in the protocols of $\mathsf{UAC}^{\mathsf{gc}}$ do not leak information to an adversarial issuer (verifier), that it did not already know before the protocol run. Furthermore, from the message privacy of BS it follows that (1) the commitments to the attributes in the protocols perfectly hide the attributes from an adversarial issuer (verifier) and (2) the blind signing protocol does not leak the message to be signed.

Theorem 5.4.1. *If the blind signature scheme* BS *is perfectly message private (Definition 2.3.10) and the argument system* Arg *is a non-interactive zero-knowledge argument system in the ROM (Definition 2.3.29), then the generic UAC construction* $\mathsf{UAC}^{\mathsf{gc}}$ *(Construction 5.3.1) is computational anonymous (Definition 5.2.4) in the ROM.*

Proof. We show the anonymity of $\mathsf{UAC}^{\mathsf{gc}}$ in the ROM and therefore we implicitly give the algorithms oracles access to the random oracle in the following . To show that $\mathsf{UAC}^{\mathsf{gc}}$ is computational anonymous, we have to define a ppt simulator $\mathsf{Sim} := (\mathsf{SimReceiveCred}, \mathsf{SimReceiveUpd}, \mathsf{SimShowCred})$ consisting of three ppt algorithms (simulators). Let us go through the three simulators step by step and show their goal and define how they work. The goal for simulator SimReceiveCred is to use the protocol $\mathsf{SimReceiveCred}(pp, pk, ctx, \upsilon) \leftrightarrow \mathsf{IssueCred}(pp, pk, ctx, \upsilon, sk)$ instead of $\mathsf{ReceiveCred}(pp, pk, ctx, \upsilon, \alpha) \leftrightarrow \mathsf{IssueCred}(pp, pk, ctx, \upsilon, sk)$ where SimReceiveCred simulates to interaction without the hidden parameter α as an input. The simulator SimReceiveCred works like ReceiveCred, but it commits to a random attribute vector instead of the actual attribute vector and simulates every proof by using the zero-knowledge simulator of the argument system Arg. By this, the commitment and proof is independent of the attributes and hidden parameter. This change is not recognizable by an adversary, because of the message privacy of the blind signature and the zero-knowledge property of the argument system. Next, the simulator SimReceiveUpd is used in the protocol $\mathsf{SimReceiveUpd}(pp, pk, ctx, \upsilon) \leftrightarrow \mathsf{UpdateCred}(pp, pk, ctx, \upsilon, sk)$ instead of $\mathsf{ReceiveUpd}(pp, pk, ctx, \upsilon, \alpha, cred) \leftrightarrow \mathsf{UpdateCred}(pp, pk, ctx, \upsilon, sk)$ where simulator SimReceiveUpd does not get the hidden parameter α and the credential $cred$ as input. Simulator SimReceiveUpd works like algorithm ReceiveUpd, but similar to simulator SimReceiveCred it commits to a fixed value (0^n) instead of the updated attributes and simulates the proof. The third and last simulator SimShowCred is used in the protocol $\mathsf{SimShowCred}(pp, pk, ctx, \chi) \leftrightarrow \mathsf{ShowVerify}(pp, pk, ctx, \chi)$ instead of $\mathsf{ShowCred}(pp, pk, ctx, \chi, \alpha, cred) \leftrightarrow \mathsf{ShowVerify}(pp, pk, ctx, \chi)$ where simulator SimShowCred simulates without the hidden parameter α and the credential $cred$ as input. Again, simulator SimShowCred works like algorithm ShowCred but it simulates the proof.

With the description of the simulators in place, let us outline the details of the proof. Let $\mathcal{D}$ be an distinguisher against the anonymity (Definition 5.2.4) of $\mathsf{UAC}^{\mathsf{gc}}$. We prove the anonymity in a sequence of experiments (games), where we start with $\mathrm{Exp}^{\mathsf{anon},1}_{\mathcal{D},\mathsf{UAC}^{\mathsf{gc}}}(\lambda)$ and modify the used protocol algorithms ReceiveCred, ReceiveUpd, and ShowCred in each step of the sequence. The sequence ends with the above described simulation algorithms replacing the protocol algorithms, where the commitments, proofs and blind signing protocols are independent of the attributes and hidden parameters. Hence, at the end of the sequence we are in the experiment $\mathrm{Exp}^{\mathsf{anon},0}_{\mathcal{D},\mathsf{UAC}^{\mathsf{gc}}}(\lambda)$ that uses the simulators to answer the oracle queries. The first step in the sequence is to simulate all of the proofs generated by the argument system Arg. This includes that the random oracle gets simulated. Note that there is nothing left to do for the show protocol after this step, since simulating the proofs is all that the simulator SimShowCred does. The second step is to commit to a random attribute instead of the attribute determined by the update function and the hidden attributes in the corresponding issue and update protocols. This also means that the blind signing protocol ($\mathsf{BS.BlindRcv} \leftrightarrow \mathsf{BS.BlindSign}$) runs on the commitment to the random attributes. Note, for the blind signing protocol this is just a commitment as any other commitment and hence does not change its

behavior. The result of a blind signing protocol is then a credential on random attributes. Since all the proofs are simulated (step 1) and therefore do not use the credentials, this is not a problem for the other protocols (issue, update, show).

Step 1 simulating proofs. To simulate the proofs let $\mathsf{Sim} := (\mathsf{SimRO}, \mathsf{SimProve})$ be the *stateful* ppt simulator of the a non-interactive zero-knowledge argument system Arg. Note, since Arg is a non-interactive zero-knowledge argument system in the ROM there is a simulator for the random oracle (SimRO) and for the proofs ($\mathsf{SimProve}$) with shared state.

Let experiment $\mathrm{Exp}^{\mathsf{anon\text{-}step\text{-}1}}_{\mathcal{D},\mathsf{UAC}^{\mathsf{gc}}}(\lambda)$ be defined as $\mathrm{Exp}^{\mathsf{anon},1}_{\mathcal{D},\mathsf{UAC}^{\mathsf{gc}}}(\lambda)$, but the protocol algorithms $\mathsf{ReceiveCred}$, $\mathsf{ReceiveUpd}$, and $\mathsf{ShowCred}$ simulate the corresponding Arg proofs by using simulator SimRO to simulate the random oracle (and answer queries to it) and simulator $\mathsf{SimProve}$ to simulate the proofs. To show that distinguisher $\mathcal{D}$ cannot recognize this change, i.e., $\mathrm{Adv}^{\mathsf{step\text{-}1}}_{\mathcal{D},\mathsf{UAC}^{\mathsf{gc}}}(\lambda) = |\Pr[\mathrm{Exp}^{\mathsf{anon},1}_{\mathcal{D},\mathsf{UAC}^{\mathsf{gc}}}(\lambda) = 1] - \Pr[\mathrm{Exp}^{\mathsf{anon\text{-}step\text{-}1}}_{\mathcal{D},\mathsf{UAC}^{\mathsf{gc}}}(\lambda) = 1]|$ is negligible, we show a reduction to the zero-knowledge property of the argument system Arg.

In detail, we use distinguisher $\mathcal{D}$ to define an adversary $\mathcal{A}$ against the zero-knowledge property of the argument system Arg in the following. Hence, adversary $\mathcal{A}$ has access to two oracles $\mathcal{O}^{\mathsf{RO}_b}$ and $\mathcal{O}^{\mathsf{Prove}_b}$ in the zero-knowledge experiment $\mathrm{Exp}^{\mathsf{zk\text{-}rom}}_{\mathcal{A},\mathsf{Arg}}(\lambda)$. Depending on the bit b they either use the random oracle RO and algorithm $\mathsf{Arg.Prove}$ to generate proofs if $b = 1$, or they use the simulator $\mathsf{Arg.SimRO}$ and simulator $\mathsf{Arg.SimProve}$ if $b = 0$. Distinguisher $\mathcal{D}$ expects in the UAC anonymity experiments three oracles, namely $\mathcal{O}^{\mathsf{ReceiveCred}_b}$, $\mathcal{O}^{\mathsf{ReceiveUpd}_b}$, and $\mathcal{O}^{\mathsf{ShowCred}_b}$. We describe how queries to these oracles are answered in the following construction of adversary $\mathcal{A}$. Note, in the following the adversary $\mathcal{A}$ provides distinguisher $\mathcal{D}$ access to the random oracle via $\mathcal{O}^{\mathsf{RO}_b}$. Next, we construct adversary $\mathcal{A}$ in experiment $\mathrm{Exp}^{\mathsf{zk\text{-}rom}}_{\mathcal{A},\mathsf{Arg}}(\lambda)$ using distinguisher $\mathcal{D}$.

1. $\mathcal{A}$ receives as input public parameters pp and runs $\mathcal{D}^{\mathcal{O}^{\mathsf{RO}_b}}(pp)$. On output (pk, st) by $\mathcal{D}$ adversary $\mathcal{A}$ runs $\mathcal{D}^{\mathcal{O}^{\mathsf{RO}_b}}(st)$ and

2. $\mathcal{A}$ answers oracle queries by $\mathcal{D}$ as follows:

   $\mathcal{O}^{\mathsf{ReceiveCred}_b}_{pp,pk}$**:** executes the same steps as $\mathcal{O}^{\mathsf{ReceiveCred}_1}_{pp,pk}$ in Definition 5.2.4 except that in $\mathsf{ReceiveCred}$, instead of computing the proof $\pi = \mathsf{Arg}[(\alpha, r) : com = \mathsf{BS.Commit}(pp, pk, \upsilon(\bot, \alpha), r)](ctx)$ (see Construction 5.3.1), adversary $\mathcal{A}$ queries its own oracle to compute the proof $\pi \leftarrow \mathcal{O}^{\mathsf{Prove}_b}_{pp}(x := (pk, \upsilon, com), w := (\alpha, r), ctx)$. It then proceeds without any further changes.

   $\mathcal{O}^{\mathsf{ReceiveUpd}_b}_{pp,pk}$**:** executes the same steps as $\mathcal{O}^{\mathsf{ReceiveUpd}_1}_{pp,pk}$ in Definition 5.2.4 except that in $\mathsf{ReceiveUpd}$, instead of computing the proof $\pi = \mathsf{Arg}[(\mathbb{A}, \sigma, \alpha, r) : com = \mathsf{BS.Commit}(pp, pk, \upsilon(\mathbb{A}, \alpha), r) \wedge \mathsf{BS.Verify}(pp, pk, \mathbb{A}, \sigma) = 1](ctx)$ (see Construction 5.3.1), adversary $\mathcal{A}$ queries its own oracle to compute the proof $\pi \leftarrow \mathcal{O}^{\mathsf{Prove}_b}_{pp}(x := (pk, \upsilon, com), w := (\mathbb{A}, \sigma, \alpha, r), ctx)$. It then proceeds without any further changes.

$\mathcal{O}_{pp,pk}^{\mathsf{ShowCred}_b}$: executes the same steps as $\mathcal{O}_{pp,pk}^{\mathsf{ShowCred}_1}$ in Definition 5.2.4 except that in $\mathsf{ShowCred}$, instead of computing the proof $\pi = \mathsf{Arg}[(\mathbb{A}, \sigma, \alpha) : \mathsf{BS.Verify}(pp, pk, \mathbb{A}, \sigma) = 1 \wedge \chi(\mathbb{A}, \alpha) = 1](ctx)$ (see Construction 5.3.1), adversary $\mathcal{A}$ queries its oracle to compute the proof $\pi \leftarrow \mathcal{O}_{pp}^{\mathsf{Prove}_b}(x := (pk, \chi), w := (\mathbb{A}, \sigma, \alpha), ctx)$. It then proceeds without any further changes.

3. On $\mathcal{D}$'s output b' adversary $\mathcal{A}$ also outputs b'

From inspection it follows that adversary $\mathcal{A}$ is a ppt algorithm, since it uses the ppt distinguisher $\mathcal{D}$, the ppt simulators and ppt algorithms of $\mathsf{UAC}^{\mathsf{gc}}$ as subroutines. Let us analyze the advantage of adversary $\mathcal{A}$. If the internal bit b of $\mathcal{A}$'s experiment $\mathrm{Exp}_{\mathcal{A},\mathsf{Arg}}^{\mathsf{zk\text{-}rom}}(\lambda)$ is 1, then adversary $\mathcal{A}$ gives the distinguisher $\mathcal{D}$ access to the random oracle RO since oracle $\mathcal{O}^{\mathsf{RO}_1}$ just uses RO. Further, the proof oracle $\mathcal{O}_{pp}^{\mathsf{Prove}_1}$ then internally uses the algorithm $\mathsf{Arg.Prove}$ to generate proofs. This means the proofs are generated as in $\mathrm{Exp}_{\mathcal{D},\mathsf{UAC}^{\mathsf{gc}}}^{\mathsf{anon},1}(\lambda)$. However, if $b = 0$ in $\mathrm{Exp}_{\mathcal{A},\mathsf{Arg}}^{\mathsf{zk\text{-}rom}}(\lambda)$, then adversary $\mathcal{A}$ gives the distinguisher $\mathcal{D}$ access to the simulated random oracle via SimRO. The proofs are then generated by the oracle $\mathcal{O}_{pp}^{\mathsf{Prove}_0}$ which uses the simulation algorithm $\mathsf{Arg.SimProve}$ to simulate proofs. Hence, the random oracle queries are answered and proofs are generated as in $\mathrm{Exp}_{\mathcal{D},\mathsf{UAC}^{\mathsf{gc}}}^{\mathsf{anon\text{-}step\text{-}1}}(\lambda)$. In both cases adversary $\mathcal{A}$ simulates the experiments for distinguisher $\mathcal{D}$ perfectly. In conclusion we have the following.

$$
\begin{aligned}
\Pr\left[\mathrm{Exp}_{\mathcal{A},\mathsf{Arg}}^{\mathsf{zk\text{-}rom}}(\lambda) = 1\right] &= \Pr[\mathcal{D} \text{ outputs } 1 \mid b = 1] + \Pr[\mathcal{D} \text{ outputs } 0 \mid b = 0] \\
&= \frac{1}{2} \cdot \left(\Pr\left[\mathrm{Exp}_{\mathcal{D},\mathsf{UAC}^{\mathsf{gc}}}^{\mathsf{anon},1}(\lambda) = 1\right] + \Pr\left[\mathrm{Exp}_{\mathcal{D},\mathsf{UAC}^{\mathsf{gc}}}^{\mathsf{anon\text{-}step\text{-}1}}(\lambda) = 0\right] \right) \\
&= \frac{1}{2} \cdot \left(\Pr\left[\mathrm{Exp}_{\mathcal{D},\mathsf{UAC}^{\mathsf{gc}}}^{\mathsf{anon},1}(\lambda) = 1\right] \right. \\
&\qquad \left. + \left(1 - \Pr\left[\mathrm{Exp}_{\mathcal{D},\mathsf{UAC}^{\mathsf{gc}}}^{\mathsf{anon\text{-}step\text{-}1}}(\lambda) = 1\right]\right) \right) \\
&= \frac{1}{2} \cdot \left(\Pr\left[\mathrm{Exp}_{\mathcal{D},\mathsf{UAC}^{\mathsf{gc}}}^{\mathsf{anon},1}(\lambda) = 1\right] \right. \\
&\qquad \left. - \Pr\left[\mathrm{Exp}_{\mathcal{D},\mathsf{UAC}^{\mathsf{gc}}}^{\mathsf{anon\text{-}step\text{-}1}}(\lambda) = 1\right] \right) + \frac{1}{2} \\
&= \frac{1}{2} \cdot \mathrm{Adv}_{\mathcal{D},\mathsf{UAC}^{\mathsf{gc}}}^{\mathsf{step\text{-}1}}(\lambda) + \frac{1}{2}
\end{aligned}
$$

By this we get for the advantage of adversary $\mathcal{A}$

$$
\begin{aligned}
\mathrm{Adv}_{\mathcal{A},\mathsf{Arg}}^{\mathsf{zk\text{-}rom}}(\lambda) &= \left| 2 \cdot \left(\frac{1}{2} \cdot \mathrm{Adv}_{\mathcal{D},\mathsf{UAC}^{\mathsf{gc}}}^{\mathsf{step\text{-}1}}(\lambda) + \frac{1}{2} \right) - 1 \right| \\
&= \mathrm{Adv}_{\mathcal{D},\mathsf{UAC}^{\mathsf{gc}}}^{\mathsf{step\text{-}1}}(\lambda) \ .
\end{aligned}
$$

This ends the proof of the first step in the sequence since by assumption adversary $\mathcal{A}$'s advantage $\mathrm{Adv}_{\mathcal{A},\mathsf{Arg}}^{\mathsf{zk\text{-}rom}}(\lambda)$ is negligible. □

Step 2 commitments to random attributes. To introduce commitments to random attributes, we start with the experiment of step 1, i.e., $\mathrm{Exp}^{\mathsf{anon\text{-}step\text{-}1}}_{\mathcal{D},\mathsf{UAC}^{\mathsf{gc}}}(\lambda)$, and modify it. Hence, let experiment $\mathrm{Exp}^{\mathsf{anon\text{-}step\text{-}2}}_{\mathcal{D},\mathsf{UAC}^{\mathsf{gc}}}(\lambda)$ be $\mathrm{Exp}^{\mathsf{anon\text{-}step\text{-}1}}_{\mathcal{D},\mathsf{UAC}^{\mathsf{gc}}}(\lambda)$ with the difference that whenever a commitment is computed to a message m' we instead choose uniformly at random a message m and commit to message m instead. For example, if commitment com is originally computed as $com \leftarrow \mathsf{BS.Commit}(pp, pk, m' := \mathbb{A}, r)$ for attributes $\mathbb{A} \in \mathcal{U}^n_{\mathsf{Attr}}$ and randomness $r' \leftarrow \mathcal{R}^{\mathsf{Com}}$, we compute $com \leftarrow \mathsf{BS.Commit}(pp, pk, m, r)$ for message $m \leftarrow \mathcal{U}^n_{\mathsf{Attr}}$ and randomness $r \leftarrow \mathcal{R}^{\mathsf{Com}}$. The rest of the protocols run in experiment $\mathrm{Exp}^{\mathsf{anon\text{-}step\text{-}2}}_{\mathcal{D},\mathsf{UAC}^{\mathsf{gc}}}(\lambda)$ as defined in experiment $\mathrm{Exp}^{\mathsf{anon\text{-}step\text{-}1}}_{\mathcal{D},\mathsf{UAC}^{\mathsf{gc}}}(\lambda)$. This means, if the commitment com is used in a blind signing protocol, the corresponding opening value (m, r) is used by the user. In the following we show a reduction from a distinguisher $\mathcal{D}$ that recognizes this change to the message privacy (Definition 2.3.10) of the blind signature scheme BS. Hence, by the assumption that BS is message private we show that $\mathrm{Adv}^{\mathsf{step\text{-}2}}_{\mathcal{D},\mathsf{UAC}^{\mathsf{gc}}}(\lambda) = |\Pr[\mathrm{Exp}^{\mathsf{anon\text{-}step\text{-}1}}_{\mathcal{D},\mathsf{UAC}^{\mathsf{gc}}}(\lambda) = 1] - \Pr[\mathrm{Exp}^{\mathsf{anon\text{-}step\text{-}2}}_{\mathcal{D},\mathsf{UAC}^{\mathsf{gc}}}(\lambda) = 1]|$ is negligible.

Concretely, using distinguisher $\mathcal{D}$ we construct an adversary $\mathcal{B}$ against the message privacy of the blind signature scheme (Definition 2.3.10). Let us start with the description of the oracles before we show the detailed construction of adversary $\mathcal{B}$. Adversary $\mathcal{B}$ in experiment $\mathrm{Exp}^{\mathsf{msg\text{-}priv\text{-}bs}}_{\mathcal{B},\mathsf{BS}}(\lambda)$ has access to two oracles $\mathcal{O}^{\mathsf{Commit}}$ and $\mathcal{O}^{\mathsf{BlindRcv}}$. The experiment $\mathrm{Exp}^{\mathsf{msg\text{-}priv\text{-}bs}}_{\mathcal{B},\mathsf{BS}}(\lambda)$ chooses a bit $b \in \{0, 1\}$ internally by which it determines if the commitment oracle $\mathcal{O}^{\mathsf{Commit}}(m_i)$ commits to message m_i (if $b = 1$) or if it commits to a random message $m \leftarrow \mathcal{U}^n_{\mathsf{Attr}} = \mathcal{M}^n$ (if $b = 0$). As in step 1 the distinguisher $\mathcal{D}$ expects its anonymity experiments three oracles, namely $\mathcal{O}^{\mathsf{ReceiveCred}_b}$, $\mathcal{O}^{\mathsf{ReceiveUpd}_b}$, and $\mathcal{O}^{\mathsf{ShowCred}_b}$. Next, we construct adversary $\mathcal{B}$ in experiment $\mathrm{Exp}^{\mathsf{msg\text{-}priv\text{-}bs}}_{\mathcal{B},\mathsf{BS}}(\lambda)$ using distinguisher $\mathcal{D}$. Note, we use as in step 1 the simulators of the argument system Arg, i.e., random oracle simulator SimRO and proof simulator $\mathsf{Arg.Prove}^{\mathsf{RO}}$ with shared state st. Also, we do not explicitly define the steps that we introduces in step 1 to shorten the following description.

1. $\mathcal{B}$ receives as input public parameters pp and runs $\mathcal{D}^{\mathsf{SimRO}}(pp)$. On output (pk, st) by $\mathcal{D}$ adversary $\mathcal{B}$ runs $\mathcal{D}(st)$ and

2. $\mathcal{B}$ answers oracle queries by $\mathcal{D}$ as follows:

   $\mathcal{O}^{\mathsf{ReceiveCred}_b}_{pp,pk}$**:** executes the same steps as $\mathcal{O}^{\mathsf{ReceiveCred}_1}_{pp,pk}$ in Definition 5.2.4 except that in $\mathsf{ReceiveCred}$, instead of committing to the attributes $\mathbb{A} = \upsilon(\bot, \alpha)$ for hidden parameter α by computing $com \leftarrow \mathsf{BS.Commit}(pp, pk, \mathbb{A}, r)$ where $r \leftarrow \mathcal{R}^{\mathsf{Com}}$ (see Construction 5.3.1), adversary $\mathcal{B}$ queries its own commit oracle $com \leftarrow \mathcal{O}^{\mathsf{Commit}}_{pp,pk,b}(m_i := \mathbb{A})$. Also, adversary $\mathcal{B}$ then simulates the proof π by using $\mathsf{Arg.SimProve}(pp, x := (pk, \upsilon, com), ctx, st)$. Let the above query to the commit query be the j-th query. Then, adversary $\mathcal{B}$ instead of running $\mathsf{BS.BlindRcv}(pp, pk, \mathbb{A}, r)$ it uses its own oracle $\mathcal{O}^{\mathsf{BlindRcv}}_{pp,pk}(j)$ to run the blind signing protocol and forwards any messages of the oracle to distinguisher $\mathcal{D}$ and vice versa. Adversary $\mathcal{B}$ then proceeds without any further changes.

$\mathcal{O}_{pp,pk}^{\mathsf{ReceiveUpd}_b}$**:** executes the same steps as $\mathcal{O}_{pp,pk}^{\mathsf{ReceiveUpd}_1}$ in Definition 5.2.4 except that in $\mathsf{ReceiveUpd}$, instead of committing to the updated attributes $\mathbb{A}^* = \upsilon(\mathbb{A}, \alpha)$ for hidden parameter α by computing $com \leftarrow \mathsf{BS.Commit}(pp, pk, \mathbb{A}^*, r)$ where $r \leftarrow \mathcal{R}^{\mathsf{Com}}$ (see Construction 5.3.1), adversary $\mathcal{B}$ queries its own oracle $com \leftarrow \mathcal{O}_{pp,pk,b}^{\mathsf{Commit}}(m_i := \mathbb{A}^*)$. Also, adversary $\mathcal{B}$ then simulates the proof π by using $\mathsf{Arg.SimProve}(pp, x := (pk, \upsilon, com), ctx, st)$. Let the above query to the commit query be the l-th query. Then, adversary $\mathcal{B}$, instead of running $\mathsf{BS.BlindRcv}(pp, pk, \mathbb{A}^*, r)$, it uses its own oracle $\mathcal{O}_{pp,pk}^{\mathsf{BlindRcv}}(l)$ to run the blind signing protocol and forwards any messages of the oracle to distinguisher $\mathcal{D}$ and vice versa. Adversary $\mathcal{B}$ then proceeds without any further changes.

$\mathcal{O}_{pp,pk}^{\mathsf{ShowCred}_b}$**:** executes the same steps as $\mathcal{O}_{pp,pk}^{\mathsf{ShowCred}_1}$ in Definition 5.2.4 except that in $\mathsf{ShowCred}$ it simulates the proof π by using $\mathsf{Arg.SimProve}(pp, x := (pk, \chi), ctx, st)$. Adversary $\mathcal{B}$ then proceeds without any further changes.

3. On $\mathcal{D}$'s output b' adversary $\mathcal{B}$ also outputs b'

Note that adversary $\mathcal{B}$ is a ppt algorithm, since it uses the ppt distinguisher $\mathcal{D}$, the ppt simulators of the argument system Arg and ppt algorithms of $\mathsf{UAC}^{\mathsf{gc}}$ as subroutines. Next, let us analyze the advantage of adversary $\mathcal{B}$. For this let us look at the experiment that adversary $\mathcal{B}$ simulates for distinguisher $\mathcal{D}$. If the internal bit b of $\mathcal{B}$'s experiment $\mathrm{Exp}_{\mathcal{B},\mathsf{BS}}^{\mathsf{msg\text{-}priv\text{-}bs}}(\lambda)$ is 1, then adversary $\mathcal{B}$ provides distinguisher $\mathcal{D}$ access to the commit oracle $\mathcal{O}_{pp,pk,1}^{\mathsf{Commit}}(m_j)$ that outputs a commitment $com \leftarrow \mathsf{BS.Commit}(pp, pk, m_j, r_j)$ to a given message m_j (attributes) and $r_j \leftarrow \mathcal{R}^{\mathsf{Com}}$. Hence, if $\mathcal{B}$ queries the blind receive oracle $\mathcal{O}_{pp,pk}^{\mathsf{BlindRcv}}(j)$ the blind signing protocol is run with the corresponding opening value (m_j, r_j) of commitment com. Therefore, the commitments are generated and the blind signing protocol is executed as in experiment $\mathrm{Exp}_{\mathcal{D},\mathsf{UAC}^{\mathsf{gc}}}^{\mathsf{anon\text{-}step\text{-}1}}(\lambda)$. If the internal bit b is 0, then adversary $\mathcal{B}$ provides distinguisher $\mathcal{D}$ access to the commit oracle $\mathcal{O}_{pp,pk,0}^{\mathsf{Commit}}(m_j)$ that chooses a message $m \leftarrow \mathcal{U}_{\mathsf{Attr}}^n$ uniformly at random and outputs a commitment $com \leftarrow \mathsf{BS.Commit}(pp, pk, m, r)$ to message m and $r \leftarrow \mathcal{R}^{\mathsf{Com}}$. The commit oracle stores the opening value (m, r) under index j. This means, that also the blind receive oracle $\mathcal{O}_{pp,pk}^{\mathsf{BlindRcv}}(j)$ runs the blind signing protocol with the corresponding opening value (m, r). Therefore, if $b = 0$ we have that the commitments are generated and the blind signing protocol is executed as in experiment $\mathrm{Exp}_{\mathcal{D},\mathsf{UAC}^{\mathsf{gc}}}^{\mathsf{anon\text{-}step\text{-}2}}(\lambda)$. Hence, in both cases the adversary $\mathcal{B}$ simulates the experiments for distinguisher $\mathcal{D}$ perfectly. Thus, we have the following.

$$\begin{aligned}\Pr\left[\mathrm{Exp}_{\mathcal{B},\mathsf{BS}}^{\mathsf{msg\text{-}priv\text{-}bs}}(\lambda) = 1\right] &= \Pr[\mathcal{D} \text{ outputs } 1 \mid b = 1] + \Pr[\mathcal{D} \text{ outputs } 0 \mid b = 0] \\ &= \frac{1}{2} \cdot \Big(\Pr\left[\mathrm{Exp}_{\mathcal{D},\mathsf{UAC}^{\mathsf{gc}}}^{\mathsf{anon\text{-}step\text{-}1}}(\lambda) = 1\right] \\ &\qquad + \Pr\left[\mathrm{Exp}_{\mathcal{D},\mathsf{UAC}^{\mathsf{gc}}}^{\mathsf{anon\text{-}step\text{-}2}}(\lambda) = 0\right]\Big)\end{aligned}$$

$$
\begin{aligned}
=&\frac{1}{2}\cdot\Bigg(\Pr\Big[\mathrm{Exp}^{\mathsf{anon\text{-}step\text{-}1}}_{\mathcal{D},\mathsf{UAC}^{\mathsf{gc}}}(\lambda)=1\Big] \\
&\qquad\qquad +\Big(1-\Pr\Big[\mathrm{Exp}^{\mathsf{anon\text{-}step\text{-}2}}_{\mathcal{D},\mathsf{UAC}^{\mathsf{gc}}}(\lambda)=1\Big]\Big)\Bigg) \\
=&\frac{1}{2}\cdot\Bigg(\Pr\Big[\mathrm{Exp}^{\mathsf{anon\text{-}step\text{-}1}}_{\mathcal{D},\mathsf{UAC}^{\mathsf{gc}}}(\lambda)=1\Big] \\
&\qquad\qquad -\Pr\Big[\mathrm{Exp}^{\mathsf{anon\text{-}step\text{-}2}}_{\mathcal{D},\mathsf{UAC}^{\mathsf{gc}}}(\lambda)=1\Big]\Bigg)+\frac{1}{2} \\
=&\frac{1}{2}\cdot\mathrm{Adv}^{\mathsf{step\text{-}2}}_{\mathcal{D},\mathsf{UAC}^{\mathsf{gc}}}(\lambda)+\frac{1}{2}
\end{aligned}
$$

Therefore, the advantage of adversary $\mathcal{B}$ is

$$
\begin{aligned}
\mathrm{Adv}^{\mathsf{msg\text{-}priv\text{-}bs}}_{\mathcal{B},\mathsf{BS}}(\lambda)=&\left|2\cdot\left(\frac{1}{2}\cdot\mathrm{Adv}^{\mathsf{step\text{-}2}}_{\mathcal{D},\mathsf{UAC}^{\mathsf{gc}}}(\lambda)+\frac{1}{2}\right)-1\right| \\
=&\mathrm{Adv}^{\mathsf{step\text{-}2}}_{\mathcal{D},\mathsf{UAC}^{\mathsf{gc}}}(\lambda)\ .
\end{aligned}
$$

This finishes the proof of the second step in the sequence, since by assumption that the blind signature scheme BS is message private we have that the advantage $\mathrm{Adv}^{\mathsf{msg\text{-}priv\text{-}bs}}_{\mathcal{B},\mathsf{BS}}(\lambda)$ of adversary $\mathcal{B}$ is negligible. The final step does not introduce any changes, since with the changes of step 2 the protocol algorithms $\mathsf{ReceiveCred}$, $\mathsf{ReceiveUpd}$, and $\mathsf{ShowCred}$ are changed to what we described at the beginning of the proof as the simulators for $\mathsf{UAC}^{\mathsf{gc}}$, i.e., $\mathsf{SimReceiveCred}$, $\mathsf{SimReceiveUpd}$, and $\mathsf{SimShowCred}$. Hence, we use the changed protocol algorithms as the simulation algorithms and thus experiment $\mathrm{Exp}^{\mathsf{anon\text{-}step\text{-}2}}_{\mathcal{D},\mathsf{UAC}^{\mathsf{gc}}}(\lambda)$ is equal to $\mathrm{Exp}^{\mathsf{anon},0}_{\mathcal{D},\mathsf{UAC}^{\mathsf{gc}}}(\lambda)$. Therefore it holds that

$$
\Pr\Big[\mathrm{Exp}^{\mathsf{anon},0}_{\mathcal{D},\mathsf{UAC}^{\mathsf{gc}}}(\lambda)=1\Big]=\Pr\Big[\mathrm{Exp}^{\mathsf{anon\text{-}step\text{-}2}}_{\mathcal{D},\mathsf{UAC}^{\mathsf{gc}}}(\lambda)=1\Big]\ .
$$

Overall we get that

$$
\mathrm{Adv}^{\mathsf{anon}}_{\mathcal{D},\mathsf{UAC}^{\mathsf{gc}}}(\lambda)=\mathrm{Adv}^{\mathsf{zk\text{-}rom}}_{\mathcal{A},\mathsf{Arg}}(\lambda)+\mathrm{Adv}^{\mathsf{msg\text{-}priv\text{-}bs}}_{\mathcal{B},\mathsf{BS}}(\lambda)
$$

which is negligible by the assumption that the argument system Arg is a non-interactive zero-knowledge argument system in the ROM and the blind signature scheme BS is perfectly message private.

5.4.2 Soundness

In the following we show that soundness of the generic UAC construction $\mathsf{UAC}^{\mathsf{gc}}$ follows from the unforgeability of the blind signature scheme BS and the extractability of the argument system Arg. Intuitively, the extractability of the argument system Arg gives us an efficient extractor to generate the explanation list required of the soundness definition of UAC. Then, if the explanation list is not consistent, this means there is a credential on attributes that an adversary should not have, i.e.,

a forgery, we construct a reduction to the unforgeability of the blind signature scheme BS.

Theorem 5.4.2. *If the blind signature scheme* BS *is unforgeable (Definition 2.3.9) and the argument system* Arg *is extractable in the ROM (Definition 2.3.30), then the generic UAC construction* $\mathsf{UAC}^{\mathsf{gc}}$ *(Construction 5.3.1) is sound (Definition 5.2.6) in the ROM.*

Proof. To prove soundness of $\mathsf{UAC}^{\mathsf{gc}}$, we need to define an efficient extractor Extr that outputs an explanation list L (Definition 5.2.5) as required by the soundness definition. From the assumption that argument system Arg is extractable we know that there is an extractor Arg.Extr that online extracts proofs, i.e., without rewinding. Further, let $\mathcal{A}$ be an adversary against the soundness of $\mathsf{UAC}^{\mathsf{gc}}$, i.e., in the experiment $\mathrm{Exp}^{\mathsf{sound}}_{\mathcal{A},\mathsf{UAC}^{\mathsf{gc}}}(\lambda)$ (Definition 5.2.6). We start the proof by presenting the extractor Extr that internally uses the proof extractor Arg.Extr. Then we use the extractor Extr and adversary $\mathcal{A}$ to construct an adversary against the unforgeability of the blind signature scheme BS. Hence, we define extractor Extr for the soundness experiment $\mathrm{Exp}^{\mathsf{sound}}_{\mathcal{A},\mathsf{UAC}^{\mathsf{gc}}}(\lambda)$ (Definition 5.2.6) that outputs an explanation list L as follows.

1. Extr receives as input $(pp, pk, sk, Q_I, Q_U, Q_V, Q_H)$ and sets explanation list $L := \emptyset$

2. For the i-th IssueCred under (pk, sk) there is an entry $(\upsilon_i, ctx_i, st_i^I) \in Q_I$, where st_i^I contains the exchanged messages of the issue protocol with adversary $\mathcal{A}$. Hence, st_i^I contains a commitment com_i and a Arg proof π_i.
   a) if the proof π_i is valid, let $x_i := (pk, \upsilon_i, com_i)$ be the instance of the proof.
   b) Extr uses the extractor $\mathsf{Arg.Extr}(pp, x_i, \pi_i, ctx_i, Q_H)$ to extract a witness (α_i, r_i).
   c) Extr adds the hidden parameter α_i to the explanation list L
   d) If the proof π_i is invalid, it adds a random hidden parameter α to explanation List L.

3. For the i-th UpdateCred under (pk, sk) there is an entry $(\upsilon_i, b_i, ctx_i, st_i^U) \in Q_U$, where st_i^U contains the exchanged messages of the update protocol with adversary $\mathcal{A}$. Hence, st_i^U contains a commitment com_i and a Arg proof π_i.
   a) if the proof π_i is valid, let $x_i := (pk, \upsilon_i, com_i)$ be the instance of the proof.
   b) Extr uses the extractor $\mathsf{Arg.Extr}(pp, x_i, \pi_i, ctx_i, Q_H)$ to extract a witness $(\mathbb{A}_i, \sigma_i, \alpha_i, r_i)$, , where $(\mathbb{A}_i, \sigma_i)$ forms a credential
   c) Extr adds the attributes $\mathbb{A}_i$ and the hidden parameter α_i to the explanation list L

d) If the proof π_i is invalid, it adds nothing to the explanation List L.

4. For the i-th $\mathsf{ShowCred}$ under pk there is an entry $(\chi_i, b_i, ctx_i, st_i^V) \in Q_U$, where st_i^V contains the exchanged messages of the show protocol with adversary $\mathcal{A}$. Hence, st_i^V contains a Arg proof π_i or an abort information.
   a) if there is an abort information Extr does nothing
   b) if the proof π_i is valid, let $x_i := (pk, \chi_i)$ be the instance of the proof.
   c) Extr uses the extractor $\mathsf{Arg.Extr}(pp, x_i, \pi_i, ctx_i, Q_H)$ to extract a witness $(\mathbb{A}_i, \sigma_i)$, where $(\mathbb{A}_i, \sigma_i)$ forms a credential
   d) Extr adds the attributes $\mathbb{A}_i$ and signature σ_i to the explanation list L
   e) If the proof π_i is invalid, it adds nothing to the explanation List L.
5. Finally, Extr outputs explanation list L

Observe that since the proof extractor $\mathsf{Arg.Extr}$ is a ppt algorithm the extractor Extr is also a ppt algorithm.

With the extractor Extr for $\mathsf{UAC}^{\mathsf{gc}}$ in place we construct in the following an adversary $\mathcal{B}$ against the unforgeability of the blind signature scheme BS ($\mathrm{Exp}^{\mathsf{euf\text{-}cma\text{-}bs}}_{\mathcal{B},\mathsf{BS}}(\lambda)$) that uses adversary $\mathcal{A}$ against the soundness of $\mathsf{UAC}^{\mathsf{gc}}$ ($\mathrm{Exp}^{\mathsf{sound}}_{\mathcal{A},\mathsf{UAC}^{\mathsf{gc}}}(\lambda)$) and the extractor Extr. Note, that the extractor Extr internally uses the proof extractor $\mathsf{Arg.Extr}$ of the argument system Arg in the ROM and therefore we give the adversaries oracle access to a random oracle RO.

1. $\mathcal{B}$ receives as input public parameters pp and runs $\mathcal{A}^{\mathcal{O}^{\mathsf{RO}}}$. On output $(1^n, st)$ by $\mathcal{A}$ adversary $\mathcal{B}$ outputs $(1^n, st')$ to its unforgeability experiment, where st' is $\mathcal{B}$'s state.
2. On input (pk, st) adversary $\mathcal{B}$ runs $\mathcal{A}(pk, st)$ and
3. $\mathcal{B}$ answers oracle queries by $\mathcal{A}$ as follows:

   $\mathcal{O}^{\mathsf{IssueCred}}_{pp,pk}(\upsilon, ctx)$**:** $\mathcal{B}$ executes the same steps of the oracle in $\mathrm{Exp}^{\mathsf{sound}}_{,\mathsf{UAC}^{\mathsf{gc}}}(\lambda)$ (Definition 5.2.6) except that in the issue protocol with adversary $\mathcal{A}$, if $\mathcal{A}$ sends a proof π adversary $\mathcal{B}$ checks the proof. If the proof π is valid, then $\mathcal{B}$ uses the extractor $\mathsf{Arg.Extr}$ to extract a witness (α, r). Then, $\mathcal{B}$ sets $m := \upsilon(\bot, \alpha)$ and queries its own blind signing oracle $\mathcal{O}^{\mathsf{BlindSign}}_{pp,pk,sk}(m, r)$ and forwards any messages of the oracle in the blind signing protocol to adversary $\mathcal{A}$ and vice versa

   $\mathcal{O}^{\mathsf{UpdateCred}}_{pp,pk}(\upsilon, ctx)$**:** $\mathcal{B}$ executes the same steps of the oracle in $\mathrm{Exp}^{\mathsf{sound}}_{,\mathsf{UAC}^{\mathsf{gc}}}(\lambda)$ (Definition 5.2.6) except that in the update protocol with adversary $\mathcal{A}$, if $\mathcal{A}$ sends a proof π adversary $\mathcal{B}$ checks the proof. If the proof π is valid, then $\mathcal{B}$ uses the proof extractor $\mathsf{Arg.Extr}$ to extract a witness $(\mathbb{A}, \sigma, \alpha, r)$. If $\mathcal{B}$ did not query its blind signing oracle $\mathcal{O}^{\mathsf{BlindSign}}_{pp,pk,sk}(m)$ for $m := \mathbb{A}$, then $\mathcal{B}$ outputs (m, σ) as a *forgery* and halts. Otherwise, $\mathcal{B}$ sets $m := \upsilon(\mathbb{A}, \alpha)$

and queries its own blind signing oracle $\mathcal{O}^{\mathsf{BlindSign}}_{pp,pk,sk}(m,r)$ and forwards any messages of the oracle in the blind signing protocol to adversary $\mathcal{A}$ and vice versa

$\mathcal{O}^{\mathsf{ShowVerify}}_{pp,pk}(\chi, ctx)$: $\mathcal{B}$ executes the same steps of the oracle in $\mathrm{Exp}^{\mathsf{sound}}_{\cdot,\mathsf{UAC}^{\mathsf{gc}}}(\lambda)$ (Definition 5.2.6) except that in the show protocol with adversary $\mathcal{A}$, if $\mathcal{A}$ sends a proof π adversary $\mathcal{B}$ checks the proof. If the proof π is valid, then $\mathcal{B}$ uses the proof extractor $\mathsf{Arg.Extr}$ to extract a witness $(\mathbb{A}, \sigma, \alpha)$. If $\mathcal{B}$ did not query its blind signing oracle $\mathcal{O}^{\mathsf{BlindSign}}_{pp,pk,sk}(m)$ for $m := \mathbb{A}$, then $\mathcal{B}$ outputs (m, σ) as a *forgery* and halts

$\mathcal{O}^{\mathsf{RO}}(v)$: $\mathcal{B}$ computes $y \leftarrow \mathsf{RO}(v)$ and sets $Q_H = Q_H \cup \{(v, y)\}$ and returns y (Q_H is initially empty)

4. On output halt by adversary $\mathcal{A}$, adversary $\mathcal{B}$ computes explanation list L as $L \leftarrow \mathsf{Extr}(pp, pk, sk, Q_I, Q_U, Q_V, Q_H)$

5. Then adversary $\mathcal{B}$ halts

Observe that adversary $\mathcal{B}$ is a ppt algorithm, since it uses the ppt adversary $\mathcal{A}$ and ppt extractors $\mathsf{Arg.Extr}$ and Extr as subroutines and besides that it runs the ppt algorithms of $\mathsf{UAC}^{\mathsf{gc}}$. Adversary $\mathcal{B}$ simulates the soundness experiment for adversary $\mathcal{A}$ perfectly, since $\mathcal{B}$ answers $\mathcal{A}$'s oracles queries as defined in the soundness experiment by using its own blind signing oracles.

Let us analyze the advantage of adversary $\mathcal{B}$. For this let us consider $\mathcal{B}$'s winning condition in the unforgeability experiment $\mathrm{Exp}^{\mathsf{euf\text{-}cma\text{-}bs}}_{\mathcal{B},\mathsf{BS}}(\lambda)$. There, the output of $\mathcal{B}$ is checked if it is a valid signature σ on message m and that a signature on m was never queried to the blind signing oracle $\mathcal{O}^{\mathsf{BlindSign}}\, pp, pk, sk$. That the output (m, σ) of $\mathcal{B}$ is a valid signature is guaranteed by the valid Arg proofs that are extracted by $\mathcal{B}$, since the proofs indeed prove that the signature is valid. That the message m of $\mathcal{B}$'s output was not queried before is guaranteed by the construction of $\mathcal{B}$, i.e., it only outputs a forgery, if the message was not queried before. If adversary $\mathcal{B}$ does not output a forgery and therefore halts after adversary $\mathcal{A}$ halts, we have that the explanation list L output by the extractor Extr is consistent (Definition 5.2.5). Suppose that explanation list L is not consistent, then there is an index i in the list such that L is not consistent for that index. Let the corresponding explanation set be E_i. In the following we consider index i and the corresponding i-th query of adversary $\mathcal{A}$ that adversary $\mathcal{B}$ answers. This means up to that point adversary $\mathcal{B}$ has queried its own blind signing oracle $\mathcal{O}^{\mathsf{BlindSign}}_{pp,pk,sk}$ only for attributes $\mathbb{A}'$ in the explanation set E_{i-1}, i.e., $\mathbb{A}' \in E_{i-1}$. We have to consider three types of queries for the i-th query that adversary $\mathcal{B}$ answers, namely queries to $\mathcal{O}^{\mathsf{IssueCred}}_{pp,pk}(v, ctx)$, $\mathcal{O}^{\mathsf{UpdateCred}}_{pp,pk}(v, ctx)$, and $\mathcal{O}^{\mathsf{ShowVerify}}_{pp,pk}(\chi, ctx)$.

- If i-th query is to oracle $\mathcal{O}^{\mathsf{IssueCred}}_{pp,pk}(v_i, ctx_i)$, then by definition of the consistency of the explanation list L, it holds that this query cannot have caused that L is not consistent.

- If i-th query is to oracle $\mathcal{O}_{pp,pk}^{\mathsf{UpdateCred}}(\upsilon_i, ctx_i)$, then the corresponding entry in the explanation list L is a tuple $(\mathbb{A}_i, \alpha_i)$ consisting of attributes $\mathbb{A}_i$ and hidden parameter α_i that were used in the show protocol. Now, since the i-th query leads to that L is not consistent, we have that $\mathsf{UpdateCred}(pp, pk, ctx_i, \upsilon_i, sk)$ output 1 and it holds that (1) $\upsilon_i(\mathbb{A}_i, \alpha_i) = \perp$ or (2) attribute set $\mathbb{A}_i$ is not in the explanation set E_{i-1} (i.e., $\mathbb{A}_i \notin E_{i-1}$). However output 1 means adversary $\mathcal{B}$ has extracted the corresponding proof of the update protocol to get $(\mathbb{A}, \sigma, \alpha, r)$. Since the proofs guarantee that $\upsilon_i(\mathbb{A}_i, \alpha_i) \neq \perp$ holds, we conclude that (1) cannot hold. If $\mathbb{A}_i \notin E_{i-1}$, this means the attributes $\mathbb{A}_i$ were never queried and adversary $\mathcal{B}$ outputs it as part of its forgery $(m \coloneqq \mathbb{A}_i, \sigma)$ and halts. However, this contradicts that adversary $\mathcal{B}$ does not halt before adversary $\mathcal{A}$.
- If i-th query is to oracle $\mathcal{O}_{pp,pk}^{\mathsf{ShowVerify}}(\chi_i, ctx_i)$ the argument is identical to the argument for the query to $\mathcal{O}^{\mathsf{UpdateCred}}$.

We conclude that, if adversary $\mathcal{B}$ does not halt before adversary $\mathcal{A}$ and therefore does not output a forgery, adversary $\mathcal{B}$ computes a consistent explanation list L using extractor Extr. Note, the soundness experiment $\mathrm{Exp}_{\mathcal{A},\mathsf{UAC}^{\mathsf{gc}}}^{\mathsf{sound}}(\lambda)$ outputs 0, if explanation list L is consistent and 1, if it is not. Hence,

$$\Pr\left[\mathrm{Exp}_{\mathcal{B},\mathsf{BS}}^{\mathsf{euf\text{-}cma\text{-}bs}}(\lambda) = 1\right] \geq \Pr\left[\mathrm{Exp}_{\mathcal{A},\mathsf{UAC}^{\mathsf{gc}}}^{\mathsf{sound}}(\lambda) = 1\right]$$

holds and we have that $\mathrm{Adv}_{\mathcal{B},\mathsf{BS}}^{\mathsf{euf\text{-}cma\text{-}bs}}(\lambda) \geq \mathrm{Adv}_{\mathcal{A},\mathsf{UAC}^{\mathsf{gc}}}^{\mathsf{sound}}(\lambda)$. By assumption that the blind signature scheme BS is unforgeable and that the argument system Arg is extractable in the ROM the advantage of $\mathcal{B}$ is negligible. □

5.4.3 Security in the CRS Model

In the following we briefly describe the generic UAC construction $\mathsf{UAC}^{\mathsf{gc}}$ (Construction 5.3.1) in the CRS model instead of the random oracle model. That the generic construction $\mathsf{UAC}^{\mathsf{gc}}$ is presented in the ROM is due to the used argument system in the ROM. However the switch to argument systems with a CRS that are not in the ROM only requires minor changes.

Changes. Let us describe the necessary changes to the generic construction in the following. First observe that the blind signature scheme is unaffected by the change to a CRS. Hence, we focus on an argument system with a CRS which we call $\mathsf{Arg}^{\mathsf{crs}}$ in the following. Let $\mathsf{Arg}^{\mathsf{crs}}$ be an argument system according to Definition 2.3.17. Further, let it be a non-interactive zero-knowledge black-box argument of knowledge, i.e., a NIZK. Comparing it to the argument system in the ROM the NIZK $\mathsf{Arg}^{\mathsf{crs}}$ is also zero-knowledge and black-box extractable. Intuitively, zero-knowledge means here that ppt algorithm $\mathsf{Arg}^{\mathsf{crs}}.\mathsf{SimProve}$ simulates proofs given a simulation trapdoor td_{sim} (Definition 2.3.20). Further, black-box extractable means that extractor $\mathsf{Arg}^{\mathsf{crs}}.\mathsf{Extr}$ extracts witnesses using an extraction

trapdoor td_{extr}, where extractor $\mathsf{Arg}^{\mathsf{crs}}.\mathsf{Extr}$ does not depend on the adversary (Definition 2.3.21). Next, we describe how the construction $\mathsf{UAC}^{\mathsf{gc}}$ has to be changed. We refer to the construction with a CRS as $\mathsf{UAC}^{\mathsf{crs}}$. For the change to the argument system $\mathsf{Arg}^{\mathsf{crs}}$ we first have to modify the setup algorithm Setup of $\mathsf{UAC}^{\mathsf{gc}}$, see Construction 5.3.1. Hence, in addition to the setup of the blind signature scheme $pp' \leftarrow \mathsf{BS}.\mathsf{Setup}(1^\lambda)$ it runs the setup of the argument system $\mathsf{Arg}^{\mathsf{crs}}$, i.e., $(crs, td_{sim}) \leftarrow \mathsf{Arg}^{\mathsf{crs}}.\mathsf{Setup}(pp')$, where we require that the public parameters pp' of the blind signature scheme determines the relation, e.g., parameter of the message space. Then $\mathsf{UAC}^{\mathsf{crs}}.\mathsf{Setup}$ outputs public parameters $pp := crs$. Furthermore, in $\mathsf{UAC}^{\mathsf{gc}}$ the protocol algorithms use the argument system $\mathsf{Arg}^{\mathsf{crs}}$ to generate proofs and to verify them. These are the changes that are needed. The result is that the proofs are now generated by an argument system with a CRS.

Security. Considering the security of the construction $\mathsf{UAC}^{\mathsf{crs}}$ the proofs for $\mathsf{UAC}^{\mathsf{gc}}$ only have to modified slightly. Therefore, we omit detailed proofs and describe the changes in the following. Regarding anonymity of the construction $\mathsf{UAC}^{\mathsf{crs}}$ we present the following theorem.

Theorem 5.4.3. *If the blind signature scheme* BS *is perfectly message private (Definition 2.3.10) and the argument system* $\mathsf{Arg}^{\mathsf{crs}}$ *is a non-interactive zero-knowledge argument system (Definition 2.3.20), then the generic UAC construction* $\mathsf{UAC}^{\mathsf{crs}}$ *is computational anonymous (Definition 5.2.4).*

The proof is analogous to the proof of Theorem 5.4.1 with the difference that the random oracle is removed from the proof, since the random oracle is no longer part of any of the building blocks. Furthermore, simulation trapdoor td_{sim} generated in the setup algorithm of $\mathsf{UAC}^{\mathsf{crs}}$ is used to simulate the proofs, i.e., via the proof simulator $\mathsf{Arg}^{\mathsf{crs}}.\mathsf{SimProve}$ of the argument system $\mathsf{Arg}^{\mathsf{crs}}$. The remaining arguments are analogous.

Next, let us consider the soundness of the construction $\mathsf{UAC}^{\mathsf{crs}}$.

Theorem 5.4.4. *If the blind signature scheme* BS *is unforgeable (Definition 2.3.9) and the argument system* Arg *is extractable (Definition 2.3.21), then the generic UAC construction* $\mathsf{UAC}^{\mathsf{crs}}$ *is sound (Definition 5.2.6).*

This proof is analogous to the proof of Theorem 5.4.2. Here, we modify the setup algorithm of $\mathsf{UAC}^{\mathsf{crs}}$ to execute the extraction setup $\mathsf{Arg}^{\mathsf{crs}}.\mathsf{ExtrSetup}$ that outputs an extraction trapdoor td_{extr} instead of the simulation trapdoor td_{sim}. In the remainder of the proof we then use the extractor $\mathsf{Arg}^{\mathsf{crs}}.\mathsf{Extr}$ with extraction trapdoor td_{extr} to extract witnesses from the proofs.

5.5 Concrete Instantiation

In the following we discuss the instantiations of the generic UAC construction $\mathsf{UAC}^{\mathsf{gc}}$ (Construction 5.3.1) and its CRS based version $\mathsf{UAC}^{\mathsf{crs}}$ (Section 5.4.3).

Further, we present a concrete instantiation based on the blind signature scheme by Pointcheval and Sanders [PS16] and non-interactive argument systems based on the Fischlin transform [BFW15; Fis05].

To instantiate construction $\mathsf{UAC}^{\mathsf{gc}}$ we need to instantiate the blind signature scheme BS and argument system Arg. Regarding the argument system Arg, we can rely on the result that there are NIZK for all languages in NP [GMW91]. However, this generic approach leads to inefficient instantiations. For practical reasons we presented the construction $\mathsf{UAC}^{\mathsf{gc}}$ in the ROM. Thus, we get a wide support of update functions and predicates by using argument systems in the ROM. In detail, argument systems in the ROM that we use are based on Sigma protocols (also called Σ-protocols) [Dam10; FKMV12]. They are a generalization of Schnorr's protocol [Sch91]. Sigma protocols are then transformed to a NIZK in the ROM via the Fiat-Shamir transformation [FKMV12; FS87] or the Fischlin transformation [BFW15; Fis05]. Sigma protocols are interactive protocols with three messages where the prover sends the first message (announcement) $\mathtt{com}$ to the verifier, the verifier answers with a random challenge $\mathtt{ch}$ (independent of other messages, i.e., public-coin), and then the prover sends the third message (response) $\mathtt{res}$. For a formal definition we refer to [FKMV12]. The Fiat-Shamir transformation makes Sigma protocols non-interactive by replacing the random challenge $\mathtt{ch}$ of the verifier with a hash value of the announcement $\mathtt{com}$, the context (e.g., verifier identity), and the corresponding instance of the proof. Here, the hash function is modeled as a random oracle. Faust et al. [FKMV12] showed that an argument system resulting from the Fiat-Shamir transformation is zero-knowledge and white-box simulation-extractable (i.e., with rewinding) in the ROM. For this, they have the additional requirement that the responses are quasi unique. This means, for a valid proof $(\mathtt{com},\mathtt{ch},\mathtt{res})$, consisting of an announcement $\mathtt{com}$, a challenge $\mathtt{ch}$, and a response $\mathtt{res}$, it is infeasible for an adversary to find a second response $\mathtt{res}'$ such that $(\mathtt{com},\mathtt{ch},\mathtt{res}')$ is also a valid proof. For example Schnorr-based protocols have unique responses. Since extractions with rewinding can be problematic as described in [BFW15; Fis05], we opt to use extractable argument systems for UAC that do not use the rewinding technique. In detail, we use argument systems resulting from the Fischlin transformation on Sigma protocols with quasi unique responses originally presented in [Fis05]. Fischlin [Fis05] shows that the resulting argument system is zero-knowledge (Definition 2.3.29) and extractable (Definition 2.3.30) in the ROM. Fischlin calls this online extractable and we refer to them as black-box extractors. This is already sufficient to construct UAC. Nonetheless, we still want to mention that Bernhard, Fischlin and Warinschi [BFW15] show that the Fischlin transformation even results in simulation-extractible argument systems (Definition 2.3.31) in the ROM. Intuitively, the Fischlin transformation forces the prover to rewind itself. This is achieved by requiring that the hash value, used as the challenge $\mathtt{ch}$, has a predetermined number of 0's at the end. Since this is very unlikely to happen in few hash attempts, the hash function modeled as a random oracle is queried often enough on tuples consisting of announcement, challenge, and response, such that an extractor can extract without rewinding the

prover. For this the extractor need two valid proofs (com,ch,res) and (com,ch′,res′) with the same announcement com, but different challenges ch and ch′. This is the so called Schnorr-trick and is originally used to extract witnesses from Schnorr protocol proofs. For further details we refer to [Fis05]. To summarize, the Fischlin transformation gives the extractor access to two valid proofs as described above and therefore is able to extract without rewinding the prover.

Considering our generic UAC construction $\mathsf{UAC}^{\mathsf{gc}}$ it is important to note that there is a wide range of Sigma protocols to prove rich statements, e.g., equality (of dlog) proofs, set membership proofs, range proofs [CCs08; MKWK19] Or-proofs, proofs of partial knowledge [CDS94; Ks22].

The other building block, that we need to instantiate, is the blind signature scheme BS used in the construction $\mathsf{UAC}^{\mathsf{gc}}$. As mentioned before Pointcheval and Sanders [PS16] present a blind signature scheme (Definition 2.3.7) using bilinear groups that we use to instantiate BS. Here, the algorithm Commit computes a perfectly hiding generalized Pedersen commitment [FHS15] to a message vector. Perfectly hiding means that the output distribution of algorithm Commit is independent of the message vector. Furthermore, mapped to our commit-then-sign structure of blind signature schemes, i.e., with an externalized proof over the commitment via the argument system, the user algorithm BlindRcv presented in [PS16] does not send any messages to the signer. This means an (adversarial) signer running BlindSign does only get the perfectly hiding commitment and then signs the commitment for the user. Hence, the message privacy of the blind signature scheme follows.

The described instantiation of $\mathsf{UAC}^{\mathsf{gc}}$ is also efficient, since the signature scheme by Pointcheval and Sanders [PS16] is defined for type 3 bilinear groups (efficient pairing and group operations). Additionally, we use argument systems in the ROM which enables efficient generation of proofs by design. Furthermore, the argument system in the ROM is non-interactive, which means a proof only consists of one message sent from the prover to the verifier.

We omit presenting the concrete instantiation in detail, since it is very similar to the generic construction only with the commitment and signature parts presented in more detail. The proofs of the argument system still use the Camenisch-Stadler notation Section 2.3.5, because the update functions and predicates are generic, since they depend on the choice of the user and issuer.

Regarding a concrete instantiation of the generic UAC construction $\mathsf{UAC}^{\mathsf{crs}}$ with a CRS we only have to instantiate the argument system with an argument system $\mathsf{Arg}^{\mathsf{crs}}$ with a CRS and the rest stays the same. Depending on the argument system $\mathsf{Arg}^{\mathsf{crs}}$ the supported predicates and update functions change, but also the efficiency of $\mathsf{UAC}^{\mathsf{crs}}$. A candidate for an instantiation is the Groth-Sahai proof system [GS12]. Note that the Groth-Sahai proof system can efficiently prove the validity of the signatures generated by the pairing-based signature scheme by Pointcheval and Sanders [PS16].

5.6 Anonymous Credentials and Attribute-based Signatures

We discuss the connection and highlight the differences between attribute-based Signatures ABS and anonymous credentials AC (and UAC) in this section. We first list the roles and functionality of ABS as a reminder and then start to build a variant of the generic UAC construction $\mathsf{UAC}^{\mathsf{gc}}$ (Construction 5.3.1) from an ABS scheme.

In an ABS scheme $\mathsf{ABS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ we have users, an authority, and verifiers as roles. The authority generates public parameters of the scheme via algorithm $\mathsf{ABS.Setup}$ and issues secret keys on attributes to users via algorithm $\mathsf{ABS.KeyGen}$. Then, a user can sign message-policy pairs with such a secret key via algorithm $\mathsf{ABS.Sign}$, if the attributes of the secret key satisfy the policy. Everybody can verify an attribute-based signature via the algorithm $\mathsf{ABS.Verify}$ under the public parameters. As we have seen in the UC model of ABS the key generation can be modeled as an interaction between the user and authority, where the user and authority agree on attributes for the secret key to be issued. Note, in ABS this key generation protocol has no (attribute) anonymity. The attributes are communicated to the authority in the clear.

Let us map the roles (users, authority, verifiers) of ABS to the roles (users, issuers, and verifiers) of an UAC system. Users get credentials on attributes from the issuers and users can show them to verifiers in a privacy-preserving way. On a high level there are two key similarities between UAC and ABS.

1. In ABS a user gets a secret key on attributes from an authority and in UAC a user gets a credential on attributes from an issuer.

2. In ABS a user signs a message-policy pair, such that the validity of the signature convinces a verifier that the user has a secret key on attributes that satisfy the policy. In UAC a user proves that it has a valid credential on attributes that satisfy a predicate (policy).

Next, we use these similarities for an intermediate variant of the generic UAC construction $\mathsf{UAC}^{\mathsf{gc}}$ build from an ABS scheme $\mathsf{ABS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$. Let pp_{ABS} be the public parameters and msk the master secret key for ABS generated by the issuer in $\mathsf{IssuerKeyGen}$. Then, we let the issuer generate an ABS key on attributes $\mathbb{A}$ as a credential for the users, i.e., $cred \leftarrow \mathsf{ABS.KeyGen}(pp_{\mathsf{ABS}}, msk, \mathbb{A})$. Furthermore, instead of computing a proof in the show protocol a user shows its credential with respect to a predicate χ by computing and sending an ABS signature on the context ctx of the show protocol, i.e., $\pi \leftarrow \mathsf{ABS.Sign}(pp_{\mathsf{ABS}}, sk_{\mathbb{A}}, m \coloneqq ctx, \chi)$. The corresponding verifier of the show protocol then verifies the ABS signature, i.e., $b \leftarrow \mathsf{ABS.Verify}(pp_{\mathsf{ABS}}, m \coloneqq ctx, \chi, \pi)$. Note, that the ABS scheme ABS now determines the predicate universe $\mathcal{X}$ corresponding to the supported policy class of the ABS scheme.

We do not consider updates of credentials for now and rather inspect what we have achieved. Observe that the above approach results in an issue protocol without any anonymity (considering UAC), since the key generation of ABS gives the issuer (authority) the attributes in plain. We call this semi-anonymous credentials, where the issuing of credentials does reveal the attributes, however any showing of the credentials provide anonymity. The advantage here is that the issuing protocol is more efficient, since it does not use any argument system proofs. However, we lose the feature to issue credentials on hidden attributes.

To get issuing of hidden attributes and updates of credentials, we have to make some assumptions on how an ABS scheme ABS is build. The assumptions are:

- We only consider ABS schemes that use a signature scheme Sig to generate secret keys, similar to our generic ABS construction $\mathsf{ABS}^{\mathsf{gc}}$ (Section 3.3).

- Further, the ABS scheme has to be either build directly from SoK (Definition 2.3.11) to generate the attribute-based signatures, again similar to $\mathsf{ABS}^{\mathsf{gc}}$, or it is based on a NIZK that with some technique binds the message to the proof (which again can be seen as a SoK), e.g., the ABS schemes by Sakai et al. [SAH16] and by Maji et al. [MPR11]. We need this to be able to extract ABS signatures. Alternatively, we require that the ABS scheme is simulation-extractable according to Definition 3.2.7 or Definition 3.2.8.

- Then, we require that the setup algorithm $\mathsf{ABS.Setup}$ outputs ABS specific public parameters pp_{ABS} on input the public parameters of the credential system pp instead of the security parameter λ as input.

- We further require that the signature scheme Sig, used to generate ABS secret keys, can be modified to a blind signature scheme that follow the commit-then-sign structure. Examples for such signature schemes are CL-Signatures [CL04], P-Signatures [BCKL07; BCKL08; ILV11], the signature scheme by Pointcheval and Sanders [PS16], and structure-preserving blind signatures [FHKS16; FHS15].

As a result of these assumptions and with the above mapping of roles and algorithms of the simplified semi-anonymous credential system, we have that the key generation of ABS is realized via a blind signature scheme and can directly be used as the blind signature scheme in the issue and update protocol. This means, a successful issue (update) protocol outputs a credential $cred = (\mathbb{A}, sk_{\mathbb{A}})$ for the user. Here, the credentials $cred$ contains an ABS secret key $sk_{\mathbb{A}}$ for attributes $\mathbb{A}$.

We are ready to present the variant of the generic UAC construction $\mathsf{UAC}^{\mathsf{gc}}$ using an ABS scheme ABS that fulfills the above requirements. We call this variant $\mathsf{UAC}^{\mathsf{ABS}}$ and present the changes compared to $\mathsf{UAC}^{\mathsf{gc}}$ (Construction 5.3.1) in the following. Let BS be the blind signature scheme used in ABS. Note, we still use the argument system used in $\mathsf{UAC}^{\mathsf{gc}}$ during the issue and update protocols.

- $(pk, sk) \leftarrow \mathsf{IssuerKeyGen}(pp, 1^n)$ runs the setup of the ABS scheme ABS, i.e., $(pp_{\mathsf{ABS}}, msk) \leftarrow \mathsf{ABS.Setup}(pp)$, and sets $pk := pp_{\mathsf{ABS}}$ and $sk := msk$. The attribute space for the credentials is the attribute space of ABS.
- The issue protocol works as before by using the blind signature scheme BS used in ABS. The output for the user is a credential $cred = (\mathbb{A}, sk_{\mathbb{A}})$ where $sk_{\mathbb{A}}$ includes a blind signature σ on $\mathbb{A}$.
- The changes in the update protocol are analogous.
- In the show protocol $\mathsf{ShowCred}(pp, pk, ctx, \chi, \alpha, cred) \leftrightarrow \mathsf{ShowVerify}(pp, pk, ctx, \chi)$ instead of using the argument system to generate the proof the user in $\mathsf{UAC}^{\mathsf{ABS}}$ computes an ABS, i.e., $\theta \leftarrow \mathsf{Sign}(pp_{\mathsf{ABS}}, sk_{\mathbb{A}}, m := ctx, \mathbb{P} := \chi_\alpha)$ where χ_α is the predicate χ with the hidden parameter α fixed. The verifier, instead of verifying the argument system proof, it runs the verification algorithm of ABS, i.e., $\mathsf{ABS.Verify}(pp_{\mathsf{ABS}}, m := ctx, \mathbb{P} := \chi, \theta)$.

For concreteness and to see that this is correct, let us consider the proof π that the user in the show protocol originally computes in $\mathsf{UAC}^{\mathsf{gc}}$, i.e.,

$$\pi = \mathsf{Arg}[(\mathbb{A}, \sigma, \alpha) : \mathsf{BS.Verify}(pp, pk, \mathbb{A}, \sigma) = 1 \wedge \chi(\mathbb{A}, \alpha) = 1](ctx).$$

The explicit modeling of the hidden parameter α aside, this is the ABS relation $R^{\mathrm{ABS}}_{\mathsf{Sig}}$ (Definition 3.3.1). In detail, the first statement $\mathsf{BS.Verify}(pp, pk, \mathbb{A}, \sigma) = 1$ shows that the ABS secret key $sk_{\mathbb{A}}$ is valid and the second statement $\chi(\mathbb{A}, \alpha) = 1$ shows that the attributes satisfy the policy (predicate). Intuitively, these two statement are what a SoK or NIZK based ABS scheme proves to generate a signature. Regarding security of $\mathsf{UAC}^{\mathsf{ABS}}$, for anonymity we use the simulator from the simulation privacy of ABS to argue that the show protocol is anonymous. For the issue and update protocol the argument is the same that we used in the anonymity proof of $\mathsf{UAC}^{\mathsf{gc}}$ (Section 5.4.1), i.e., the message privacy of the blind signature scheme and the zero-knowledge property of the argument system. To show the soundness of $\mathsf{UAC}^{\mathsf{ABS}}$, the proof is similar to the soundness proof of $\mathsf{UAC}^{\mathsf{gc}}$ (Section 5.4.2). The difference is how we handle the extraction in a show protocol. Since the verifier does now get an ABS signature instead of a proof of the argument system, we have two cases to consider according to our requirements on the ABS scheme. First, we use the extractor from the SoK or NIZK, if the used ABS scheme generates its signatures using either of these building blocks. Second, we use the ABS extractor, if the ABS scheme is simulation-extractable. Since the changes to get $\mathsf{UAC}^{\mathsf{ABS}}$ and the adaptations to an ABS like our generic ABS construction $\mathsf{ABS}^{\mathsf{gc}}$ (Section 3.3) are straightforward, we do not go into further details. Besides that, a similar transformation based on anonymous key generation for ABS schemes, but for anonymous credential systems without updates was presented in [Kaa+17] with a concrete instantiation.

6 Conclusion

Privacy-preserving cryptography and its techniques are becoming more relevant for real-world applications. Therefore, security definitions that correspond to real-world threats are needed. Defining security for privacy-preserving cryptography is a delicate task that, as seen in this thesis, can lead to precise security definitions allowing schemes to be UC secure, to be more efficient and to support practical features such as updates.

We focussed in Part I on attribute-based signature (ABS), their security definitions (experiment-based and in UC), and on a generic ABS construction from SoK. We started with a detailed discussion of existing security definitions, ABS constructions, and support of expressive policies. We then defined a strengthened experiment-based security model that does not make any assumptions on the policy classes. In detail, we presented a stronger simulation-based privacy definition, introduced simulation-extractability for ABS and discussed in general the relations between the security definitions. With respect to our experiment-based security model, we presented a simulation private and simulation-extractable generic ABS construction and an instantiation with constant-size signatures consisting only of three group elements. Our next contribution for ABS was then an ideal ABS functionality in UC. The main result was that we showed that simulation private and unforgeable ABS schemes are secure in UC and vice versa. With regard to that, another interesting contribution was that we showed that two major generic ABS constructions from the literature [MPR11; SAH16] are simulation private. Hence, we showed that the two ABS constructions [MPR11; SAH16] are UC secure. Our results regarding the experiment-based and UC security models for ABS are an enhancement of the ABS security model.

Considering future work, we identified in Section 3.3, motivated by our generic ABS construction from SoK, the open question, if the results of SNARKs in the presence of oracles (O-SNARK) by Fiore and Nitulescu [FN16a; FN16b] extend to simulation-extractable SNARKS (SE-SNARKs). Additionally, extending our ideal ABS functionality to multi-authority ABS and hierarchical ABS is an interesting direction for future work.

In Part II we considered practical features of anonymous credential systems. We focussed on privacy-preserving updates of credentials and presented this as the

main part of updatable anonymous credentials. We motivated UAC by showing that it allows more versatile applications than typical AC. In detail, we achieved this by adding the feature of privacy-preserving updates of attributes certified by a credential. This feature was modeled using an update function that takes attributes $\mathbb{A}$ (of a credential) and a hidden parameter α as input and outputs the updated attributes. This modeling allowed us to encode arbitrary checks on the inputs in an update function. Furthermore, we allowed updates of attribute values using the hidden parameters. Update functions were also integrated and generalized to support issuing of credentials with respect to the hidden parameter, e.g., issuing of hidden attributes. Regarding security, we presented a security model for UAC covering privacy-preserving update including the special case of issuing credentials on hidden attributes chosen by the user. To this end, we defined anonymity and soundness for UAC, revised the security definitions in this thesis and showed that the generic UAC construction $\mathsf{UAC}^{\mathsf{gc}}$ in the ROM is secure with respect to this model. The generic UAC construction $\mathsf{UAC}^{\mathsf{gc}}$ was built using common building blocks of AC in the literature, i.e., a blind signature scheme and an argument system in the ROM. We then discussed a version of the generic UAC construction $\mathsf{UAC}^{\mathsf{gc}}$ with a CRS. Showing that $\mathsf{UAC}^{\mathsf{gc}}$ and UAC does not implicitly rely on the ROM, but rather that it inherits this from the used argument system. Therefore, we also sketched an instantiation using an argument system with a CRS. We closed Part II with the connection of ABS and AC in general. By analyzing the connection we were able to state requirements on ABS schemes that allowed us to use ABS schemes as a building block for UAC.

Regarding future work in the area of UAC, adding further practical features to UAC, such as revocation and the concept of issuer-hiding [Bob+21], is an interesting avenue. Additionally, using UAC for more real-world applications besides incentive systems, as presented in [BBDE19a], is a possible direction for future work.

Working with privacy-preserving techniques and constructing privacy-preserving attribute-based schemes, like ABS, UAC and others, showed me that there are many opportunities to build modern systems while protecting the privacy of users.

Bibliography

[AAS16] Hiroaki Anada, Seiko Arita, and Kouichi Sakurai. *Proofs of Knowledge on Monotone Predicates and its Application to Attribute-Based Identifications and Signatures*. Cryptology ePrint Archive, Report 2016/483. `https://eprint.iacr.org/2016/483`. 2016 (cit. on pp. 39, 67).

[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. Springer, Heidelberg, Aug. 2010, pp. 209–236. DOI: `10.1007/978-3-642-14623-7_12` (cit. on p. 40).

[Abe+16] Masayuki Abe, Georg Fuchsbauer, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. "Structure-Preserving Signatures and Commitments to Group Elements." In: *J. Cryptol.* 29.2 (2016), pp. 363–421. DOI: `10.1007/s00145-014-9196-7` (cit. on p. 90).

[ACHM05] Giuseppe Ateniese, Jan Camenisch, Susan Hohenberger, and Breno de Medeiros. *Practical Group Signatures without Random Oracles*. Cryptology ePrint Archive, Report 2005/385. `https://eprint.iacr.org/2005/385`. 2005 (cit. on pp. 106, 107).

[AGHO11] Masayuki Abe, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. "Optimal Structure-Preserving Signatures in Asymmetric Bilinear Groups." In: *Advances in Cryptology – CRYPTO 2011*. Ed. by Phillip Rogaway. Vol. 6841. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2011, pp. 649–666. DOI: `10.1007/978-3-642-22792-9_37` (cit. on p. 95).

[AHO10] Masayuki Abe, Kristiyan Haralambiev, and Miyako Ohkubo. *Signing on Elements in Bilinear Groups for Modular Protocol Design*. Cryptology ePrint Archive, Report 2010/133. `https://eprint.iacr.org/2010/133`. 2010 (cit. on p. 40).

[AHY15] Nuttapong Attrapadung, Goichiro Hanaoka, and Shota Yamada. "Conversions Among Several Classes of Predicate Encryption and Applications to ABE with Various Compactness Tradeoffs." In: *Advances in Cryptology – ASIACRYPT 2015, Part I*. Ed. by Tetsu

Iwata and Jung Hee Cheon. Vol. 9452. Lecture Notes in Computer Science. Springer, Heidelberg, 2015, pp. 575–601. DOI: 10.1007/978-3-662-48797-6_24 (cit. on pp. 38–40, 49, 72, 87).

[AKS19] Behzad Abdolmaleki, Hamidreza Khoshakhlagh, and Daniel Slamanig. "A Framework for UC-Secure Commitments from Publicly Computable Smooth Projective Hashing." In: *17th IMA International Conference on Cryptography and Coding*. Ed. by Martin Albrecht. Vol. 11929. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 2019, pp. 1–21. DOI: 10.1007/978-3-030-35199-1_1 (cit. on p. 31).

[AO09] Masayuki Abe and Miyako Ohkubo. "A Framework for Universally Composable Non-committing Blind Signatures." In: *Advances in Cryptology – ASIACRYPT 2009*. Ed. by Mitsuru Matsui. Vol. 5912. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 2009, pp. 435–450. DOI: 10.1007/978-3-642-10366-7_26 (cit. on p. 40).

[AO12] Masayuki Abe and Miyako Ohkubo. "A framework for universally composable non-committing blind signatures." In: *Int. J. Appl. Cryptogr.* 2.3 (2012), pp. 229–249. DOI: 10.1504/IJACT.2012.045581 (cit. on pp. 40, 107, 108, 130, 141, 145).

[AOS02] Masayuki Abe, Miyako Ohkubo, and Koutarou Suzuki. "1-out-of-n Signatures from a Variety of Keys." In: *Advances in Cryptology – ASIACRYPT 2002*. Ed. by Yuliang Zheng. Vol. 2501. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 2002, pp. 415–432. DOI: 10.1007/3-540-36178-2_26 (cit. on p. 88).

[Bad+15] Christoph Bader, Dennis Hofheinz, Tibor Jager, Eike Kiltz, and Yong Li. "Tightly-Secure Authenticated Key Exchange." In: *TCC 2015: 12th Theory of Cryptography Conference, Part I*. Ed. by Yevgeniy Dodis and Jesper Buus Nielsen. Vol. 9014. Lecture Notes in Computer Science. Springer, Heidelberg, Mar. 2015, pp. 629–658. DOI: 10.1007/978-3-662-46494-6_26 (cit. on p. 88).

[Bag19] Karim Baghery. "On the Efficiency of Privacy-Preserving Smart Contract Systems." In: *AFRICACRYPT 19: 11th International Conference on Cryptology in Africa*. Ed. by Johannes Buchmann, Abderrahmane Nitaj, and Tajje eddine Rachidi. Vol. 11627. Lecture Notes in Computer Science. Springer, Heidelberg, July 2019, pp. 118–136. DOI: 10.1007/978-3-030-23696-0_7 (cit. on p. 108).

[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. Springer, Heidelberg, May 2004, pp. 56–73. DOI: 10.1007/978-3-540-24676-3_4 (cit. on p. 96).

[BB08] Dan Boneh and Xavier Boyen. "Short Signatures Without Random Oracles and the SDH Assumption in Bilinear Groups." In: *Journal of Cryptology* 21.2 (Apr. 2008), pp. 149–177. DOI: 10.1007/s00145-007-9005-7 (cit. on p. 96).

[BB18] Johannes Blömer and Jan Bobolz. "Delegatable Attribute-Based Anonymous Credentials from Dynamically Malleable Signatures." In: *ACNS 18: 16th International Conference on Applied Cryptography and Network Security*. Ed. by Bart Preneel and Frederik Vercauteren. Vol. 10892. Lecture Notes in Computer Science. Springer, Heidelberg, July 2018, pp. 221–239. DOI: 10.1007/978-3-319-93387-0_12 (cit. on pp. 40, 72, 154, 156, 161).

[BBDE19a] Johannes Blömer, Jan Bobolz, Denis Diemert, and Fabian Eidens. "Updatable Anonymous Credentials and Applications to Incentive Systems." In: *ACM CCS 2019: 26th Conference on Computer and Communications Security*. Ed. by Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz. ACM Press, Nov. 2019, pp. 1671–1685. DOI: 10.1145/3319535.3354223 (cit. on pp. iii, iv, 151, 154, 157, 190).

[BBDE19b] Johannes Blömer, Jan Bobolz, Denis Diemert, and Fabian Eidens. *Updatable Anonymous Credentials and Applications to Incentive Systems*. Cryptology ePrint Archive, Report 2019/169. https://eprint.iacr.org/2019/169. 2019 (cit. on pp. iii, iv, 151, 154, 157).

[BBFR15] Michael Backes, Manuel Barbosa, Dario Fiore, and Raphael M. Reischuk. "ADSNARK: Nearly Practical and Privacy-Preserving Proofs on Authenticated Data." In: *2015 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2015, pp. 271–286. DOI: 10.1109/SP.2015.24 (cit. on p. 60).

[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. Springer, Heidelberg, Aug. 2004, pp. 41–55. DOI: 10.1007/978-3-540-28628-8_3 (cit. on pp. 54, 71).

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. "From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again." In: *ITCS 2012: 3rd Innovations in Theoretical Computer Science*. Ed. by Shafi Goldwasser. Association for Computing Machinery, Jan. 2012, pp. 326–349. DOI: 10.1145/2090236.2090263 (cit. on pp. 19, 59, 66).

[BCKL07] Mira Belenkiy, Melissa Chase, Markulf Kohlweiss, and Anna Lysyanskaya. *Non-Interactive Anonymous Credentials*. Cryptology ePrint Archive, Report 2007/384. https://eprint.iacr.org/2007/384. 2007 (cit. on p. 187).

[BCKL08] Mira Belenkiy, Melissa Chase, Markulf Kohlweiss, and Anna Lysyanskaya. "P-signatures and Noninteractive Anonymous Credentials." In: *TCC 2008: 5th Theory of Cryptography Conference*. Ed. by Ran Canetti. Vol. 4948. Lecture Notes in Computer Science. Springer, Heidelberg, Mar. 2008, pp. 356–374. DOI: 10.1007/978-3-540-78524-8_20 (cit. on pp. 40, 54, 152, 154, 156, 161, 187).

[BEHF21] Jan Bobolz, Fabian Eidens, Raphael Heitjohann, and Jeremy Fell. *Cryptimeleon: A Library for Fast Prototyping of Privacy-Preserving Cryptographic Schemes*. Cryptology ePrint Archive, Report 2021/961. https://eprint.iacr.org/2021/961. 2021 (cit. on p. 155).

[BEJ18a] Johannes Blömer, Fabian Eidens, and Jakob Juhnke. *Enhanced Security of Attribute-Based Signatures*. Cryptology ePrint Archive, Report 2018/874. https://eprint.iacr.org/2018/874. 2018 (cit. on pp. iii, 35, 43, 103).

[BEJ18b] Johannes Blömer, Fabian Eidens, and Jakob Juhnke. "Enhanced Security of Attribute-Based Signatures." In: *CANS 18: 17th International Conference on Cryptology and Network Security*. Ed. by Jan Camenisch and Panos Papadimitratos. Vol. 11124. Lecture Notes in Computer Science. Springer, Heidelberg, 2018, pp. 235–255. DOI: 10.1007/978-3-030-00434-7_12 (cit. on pp. iii, 35, 43, 103, 104).

[BEJ18c] 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. Springer, Heidelberg, Apr. 2018, pp. 470–490. DOI: 10.1007/978-3-319-76953-0_25 (cit. on p. 54).

[Bel+09] Mira Belenkiy, Jan Camenisch, Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Hovav Shacham. "Randomizable Proofs and Delegatable Anonymous Credentials." In: *Advances in Cryptology – CRYPTO 2009*. Ed. by Shai Halevi. Vol. 5677. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2009, pp. 108–125. DOI: 10.1007/978-3-642-03356-8_7 (cit. on p. 154).

[Bem17] Pascal Bemmann. "Attribute-based Signatures using Structure Preserving Signatures." https://ris.uni-paderborn.de/record/117. MA thesis. Universität Paderborn, 2017 (cit. on pp. iv, 88, 95).

[Bem+18] Kai Bemmann, Johannes Blömer, Jan Bobolz, Henrik Bröcher, Denis Diemert, Fabian Eidens, Lukas Eilers, Jan Haltermann, Jakob Juhnke, Burhan Otour, Laurens Porzenheim, Simon Pukrop, Erik Schilling, Michael Schlichtig, and Marcel Stienemeier. "Fully-Featured Anonymous Credentials with Reputation System." In: *Proceedings of the 13th International Conference on Availability, Reliability and Security, ARES 2018, Hamburg, Germany, August*

27-30, 2018. Ed. by Sebastian Doerr, Mathias Fischer, Sebastian Schrittwieser, and Dominik Herrmann. ACM, 2018, 42:1–42:10. DOI: 10.1145/3230833.3234517 (cit. on pp. 29, 154, 155).

[Ben+15] Eli Ben-Sasson, Alessandro Chiesa, Matthew Green, Eran Tromer, and Madars Virza. "Secure Sampling of Public Parameters for Succinct Zero Knowledge Proofs." In: *2015 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2015, pp. 287–304. DOI: 10.1109/SP.2015.25 (cit. on p. 20).

[BF11] Dan Boneh and David Mandell Freeman. "Homomorphic Signatures for Polynomial Functions." In: *Advances in Cryptology – EUROCRYPT 2011*. Ed. by Kenneth G. Paterson. Vol. 6632. Lecture Notes in Computer Science. Springer, Heidelberg, May 2011, pp. 149–168. DOI: 10.1007/978-3-642-20465-4_10 (cit. on p. 60).

[BF14] Mihir Bellare and Georg Fuchsbauer. "Policy-Based Signatures." In: *PKC 2014: 17th International Conference on Theory and Practice of Public Key Cryptography*. Ed. by Hugo Krawczyk. Vol. 8383. Lecture Notes in Computer Science. Springer, Heidelberg, Mar. 2014, pp. 520–537. DOI: 10.1007/978-3-642-54631-0_30 (cit. on pp. 40, 42, 46–48, 54, 55, 65, 81).

[BFG13] David Bernhard, Georg Fuchsbauer, and Essam Ghadafi. "Efficient Signatures of Knowledge and DAA in the Standard Model." 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. Springer, Heidelberg, June 2013, pp. 518–533. DOI: 10.1007/978-3-642-38980-1_33 (cit. on p. 88).

[BFGP22] Daniel Bosk, Davide Frey, Mathieu Gestin, and Guillaume Piolle. "Hidden Issuer Anonymous Credential." In: *Proceedings on Privacy Enhancing Technologies* 2022.4 (2022), pp. 571–607. DOI: 10.56553/popets-2022-0123 (cit. on p. 154).

[BFW15] David Bernhard, Marc Fischlin, and Bogdan Warinschi. "Adaptive Proofs of Knowledge in the Random Oracle Model." In: *PKC 2015: 18th International Conference on Theory and Practice of Public Key Cryptography*. Ed. by Jonathan Katz. Vol. 9020. Lecture Notes in Computer Science. Springer, Heidelberg, 2015, pp. 629–649. DOI: 10.1007/978-3-662-46447-2_28 (cit. on pp. 26, 67, 157, 168, 184).

[BGG19] Sean Bowe, Ariel Gabizon, and Matthew D. Green. "A Multi-party Protocol for Constructing the Public Parameters of the Pinocchio zk-SNARK." In: *FC 2018 Workshops*. Ed. by Aviv Zohar, Ittay Eyal, Vanessa Teague, Jeremy Clark, Andrea Bracciali, Federico Pintore, and Massimiliano Sala. Vol. 10958. Lecture Notes in Computer Sci-

ence. Springer, Heidelberg, Mar. 2019, pp. 64–77. DOI: 10.1007/978-3-662-58820-8_5 (cit. on p. 20).

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. "Functional Signatures and Pseudorandom Functions." In: *PKC 2014: 17th International Conference on Theory and Practice of Public Key Cryptography*. Ed. by Hugo Krawczyk. Vol. 8383. Lecture Notes in Computer Science. Springer, Heidelberg, Mar. 2014, pp. 501–519. DOI: 10.1007/978-3-642-54631-0_29 (cit. on pp. 40, 60, 65).

[BGPR20] Karim Baghery, Alonso González, Zaira Pindado, and Carla Ràfols. "Signatures of Knowledge for Boolean Circuits Under Standard Assumptions." In: *AFRICACRYPT 20: 12th International Conference on Cryptology in Africa*. Ed. by Abderrahmane Nitaj and Amr M. Youssef. Vol. 12174. Lecture Notes in Computer Science. Springer, Heidelberg, July 2020, pp. 24–44. DOI: 10.1007/978-3-030-51938-4_2 (cit. on pp. 19, 67).

[BHSB19] Michael Backes, Lucjan Hanzlik, and Jonas Schneider-Bensch. "Membership Privacy for Fully Dynamic Group Signatures." In: *ACM CCS 2019: 26th Conference on Computer and Communications Security*. Ed. by Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz. ACM Press, Nov. 2019, pp. 2181–2198. DOI: 10.1145/3319535.3354257 (cit. on p. 72).

[BKSV21] Karim Baghery, Markulf Kohlweiss, Janno Siim, and Mikhail Volkhov. "Another Look at Extraction and Randomization of Groth's zk-SNARK." In: *Financial Cryptography and Data Security - 25th International Conference, FC 2021, Virtual Event, March 1-5, 2021, Revised Selected Papers, Part I*. Ed. by Nikita Borisov and Claudia Díaz. Vol. 12674. Lecture Notes in Computer Science. Springer, 2021, pp. 457–475. DOI: 10.1007/978-3-662-64322-8_22 (cit. on pp. 17, 68).

[BL13] Foteini Baldimtsi and Anna Lysyanskaya. "Anonymous credentials light." In: *ACM CCS 2013: 20th Conference on Computer and Communications Security*. Ed. by Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung. ACM Press, Nov. 2013, pp. 1087–1098. DOI: 10.1145/2508859.2516687 (cit. on pp. 154, 156, 161).

[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. DOI: 10.1007/s00145-004-0314-9 (cit. on p. 7).

[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. Springer, Heidelberg,

May 2003, pp. 614–629. DOI: 10.1007/3-540-39200-9_38 (cit. on pp. 46, 54, 72).

[Bob+06] Rakeshbabu Bobba, Omid Fatemieh, Fariba Khan, Carl A. Gunter, and Himanshu Khurana. "Using Attribute-Based Access Control to Enable Attribute-Based Messaging." In: *22nd Annual Computer Security Applications Conference (ACSAC 2006), 11-15 December 2006, Miami Beach, Florida, USA*. IEEE Computer Society, 2006, pp. 403–413. DOI: 10.1109/ACSAC.2006.53 (cit. on pp. 37, 107).

[Bob+21] Jan Bobolz, Fabian Eidens, Stephan Krenn, Sebastian Ramacher, and Kai Samelin. "Issuer-Hiding Attribute-Based Credentials." In: *Cryptology and Network Security - 20th International Conference, CANS 2021, Vienna, Austria, December 13-15, 2021, Proceedings*. Ed. by Mauro Conti, Marc Stevens, and Stephan Krenn. Vol. 13099. Lecture Notes in Computer Science. Springer, 2021, pp. 158–178. DOI: 10.1007/978-3-030-92548-2_9 (cit. on pp. 154, 190).

[BPR20] Karim Baghery, Zaira Pindado, and Carla Ràfols. "Simulation Extractable Versions of Groth's zk-SNARK Revisited." In: *CANS 20: 19th International Conference on Cryptology and Network Security*. Ed. by Stephan Krenn, Haya Shulman, and Serge Vaudenay. Vol. 12579. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 2020, pp. 453–461. DOI: 10.1007/978-3-030-65411-5_22 (cit. on p. 108).

[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 Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby. ACM Press, Nov. 1993, pp. 62–73. DOI: 10.1145/168588.168596 (cit. on pp. 25, 26).

[Bra99] Stefan Brands. "Rethinking Public Key Infrastructure and Digital Certificates – Building in Privacy." PhD thesis. Eindhoven Institute of Technology, 1999 (cit. on p. 155).

[BS04] Dan Boneh and Hovav Shacham. "Group Signatures With Verifier-Local Revocation." In: *ACM CCS 2004: 11th Conference on Computer and Communications Security*. Ed. by Vijayalakshmi Atluri, Birgit Pfitzmann, and Patrick McDaniel. ACM Press, Oct. 2004, pp. 168–177. DOI: 10.1145/1030083.1030106 (cit. on p. 47).

[BS20] Dan Boneh and Victor Shoup. *A Graduate Course in Applied Cryptography*. 2020. URL: https://crypto.stanford.edu/~dabo/cryptobook/BonehShoup_0_5.pdf (cit. on p. 8).

[Bsk06] Aslak Bakke Buan, Kristian Gjø steen, and Lillian Krå kmo. *Universally Composable Blind Signatures in the Plain Model*. Cryptology ePrint Archive, Report 2006/405. https://eprint.iacr.org/2006/405. 2006 (cit. on p. 130).

[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. Springer, Heidelberg, Feb. 2005, pp. 136–153. DOI: 10.1007/978-3-540-30574-3_11 (cit. on p. 54).

[Cam+16a] Jan Camenisch, Robert R. Enderlein, Stephan Krenn, Ralf Küsters, and Daniel Rausch. "Universal Composition with Responsive Environments." In: *Advances in Cryptology – ASIACRYPT 2016, Part II*. Ed. by Jung Hee Cheon and Tsuyoshi Takagi. Vol. 10032. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 2016, pp. 807–840. DOI: 10.1007/978-3-662-53890-6_27 (cit. on pp. 112, 118).

[Cam+16b] Jan Camenisch, Stephan Krenn, Anja Lehmann, Gert L. Mikkelsen, Gregory Neven, and Michael Ø. Pedersen. "Formal Treatment of Privacy-Enhancing Credential Systems." In: *SAC 2015: 22nd Annual International Workshop on Selected Areas in Cryptography*. Ed. by Orr Dunkelman and Liam Keliher. Vol. 9566. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2016, pp. 3–24. DOI: 10.1007/978-3-319-31301-6_1 (cit. on pp. 29, 153, 155).

[Can01] Ran Canetti. "Universally Composable Security: A New Paradigm for Cryptographic Protocols." In: *42nd Annual Symposium on Foundations of Computer Science*. IEEE Computer Society Press, Oct. 2001, pp. 136–145. DOI: 10.1109/SFCS.2001.959888 (cit. on pp. iii, 5, 30, 31, 104, 107, 113).

[Can03] Ran Canetti. *Universally Composable Signatures, Certification and Authentication*. Cryptology ePrint Archive, Report 2003/239. https://eprint.iacr.org/2003/239. 2003 (cit. on pp. 31, 107, 108, 118, 125).

[Can20] Ran Canetti. *Universally Composable Security: A New Paradigm for Cryptographic Protocols*. Cryptology ePrint Archive, Report 2000/067. Version 11th Feb. 2020, https://eprint.iacr.org/2000/067. 2020 (cit. on pp. 30, 31, 104–106, 109–117, 119).

[CCL15] Ran Canetti, Asaf Cohen, and Yehuda Lindell. "A Simpler Variant of Universally Composable Security for Standard Multiparty Computation." In: *Advances in Cryptology – CRYPTO 2015, Part II*. Ed. by Rosario Gennaro and Matthew J. B. Robshaw. Vol. 9216. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2015, pp. 3–22. DOI: 10.1007/978-3-662-48000-7_1 (cit. on p. 104).

[CCs08] Jan Camenisch, Rafik Chaabouni, and abhi shelat. "Efficient Protocols for Set Membership and Range Proofs." In: *Advances in Cryptology – ASIACRYPT 2008*. Ed. by Josef Pieprzyk. Vol. 5350. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 2008, pp. 234–252. DOI: 10.1007/978-3-540-89255-7_15 (cit. on pp. 29, 30, 185).

[CDD17] Jan Camenisch, Manu Drijvers, and Maria Dubovitskaya. "Practical UC-Secure Delegatable Credentials with Attributes and Their Application to Blockchain." In: *ACM CCS 2017: 24th Conference on Computer and Communications Security*. Ed. by Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu. ACM Press, 2017, pp. 683–699. DOI: 10.1145/3133956.3134025 (cit. on p. 154).

[CDHK15] Jan Camenisch, Maria Dubovitskaya, Kristiyan Haralambiev, and Markulf Kohlweiss. "Composable and Modular Anonymous Credentials: Definitions and Practical Constructions." In: *Advances in Cryptology – ASIACRYPT 2015, Part II*. Ed. by Tetsu Iwata and Jung Hee Cheon. Vol. 9453. Lecture Notes in Computer Science. Springer, Heidelberg, 2015, pp. 262–288. DOI: 10.1007/978-3-662-48800-3_11 (cit. on pp. 54, 72, 106, 107, 154).

[CDL16] Jan Camenisch, Manu Drijvers, and Anja Lehmann. "Universally Composable Direct Anonymous Attestation." In: *PKC 2016: 19th International Conference on Theory and Practice of Public Key Cryptography, Part II*. Ed. by Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang. Vol. 9615. Lecture Notes in Computer Science. Springer, Heidelberg, Mar. 2016, pp. 234–264. DOI: 10.1007/978-3-662-49387-8_10 (cit. on p. 106).

[CDPW07] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. "Universally Composable Security with Global Setup." In: *TCC 2007: 4th Theory of Cryptography Conference*. Ed. by Salil P. Vadhan. Vol. 4392. Lecture Notes in Computer Science. Springer, Heidelberg, Feb. 2007, pp. 61–85. DOI: 10.1007/978-3-540-70936-7_4 (cit. on p. 104).

[CDR16] Jan Camenisch, Maria Dubovitskaya, and Alfredo Rial. "UC Commitments for Modular Protocol Design and Applications to Revocation and Attribute Tokens." In: *Advances in Cryptology – CRYPTO 2016, Part III*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9816. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2016, pp. 208–239. DOI: 10.1007/978-3-662-53015-3_8 (cit. on p. 31).

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. "Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols." In: *Advances in Cryptology – CRYPTO'94*. Ed. by Yvo Desmedt. Vol. 839. Lecture Notes in Computer Science. Springer,

Heidelberg, Aug. 1994, pp. 174–187. DOI: 10.1007/3-540-48658-5_19 (cit. on pp. 29, 67, 88, 185).

[CFQ19] Matteo Campanelli, Dario Fiore, and Anaïs Querol. "LegoSNARK: Modular Design and Composition of Succinct Zero-Knowledge Proofs." In: *ACM CCS 2019: 26th Conference on Computer and Communications Security*. Ed. by Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz. ACM Press, Nov. 2019, pp. 2075–2092. DOI: 10.1145/3319535.3339820 (cit. on p. 8).

[CG08] Jan Camenisch and Thomas Groß. "Efficient attributes for anonymous credentials." In: *ACM CCS 2008: 15th Conference on Computer and Communications Security*. Ed. by Peng Ning, Paul F. Syverson, and Somesh Jha. ACM Press, Oct. 2008, pp. 345–356. DOI: 10.1145/1455770.1455814 (cit. on p. 154).

[CGH11] Scott E. Coull, Matthew Green, and Susan Hohenberger. "Access controls for oblivious and anonymous systems." In: *ACM Trans. Inf. Syst. Secur.* 14.1 (2011), 10:1–10:28. DOI: 10.1145/1952982.1952992 (cit. on p. 155).

[Cha81] David Chaum. "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms." In: *Commun. ACM* 24.2 (1981), pp. 84–88. DOI: 10.1145/358549.358563 (cit. on p. 154).

[Cha82] David Chaum. "Blind Signatures for Untraceable Payments." In: *Advances in Cryptology – CRYPTO'82*. Ed. by David Chaum, Ronald L. Rivest, and Alan T. Sherman. Plenum Press, New York, USA, 1982, pp. 199–203 (cit. on p. 12).

[Cha85] David Chaum. "Security Without Identification: Transaction Systems to Make Big Brother Obsolete." In: *Commun. ACM* 28.10 (1985), pp. 1030–1044. DOI: 10.1145/4372.4373 (cit. on p. 154).

[Che+13] Cheng Chen, Jie Chen, Hoon Wei Lim, Zhenfeng Zhang, Dengguo Feng, San Ling, and Huaxiong Wang. "Fully Secure Attribute-Based Systems with Short Ciphertexts/Signatures and Threshold Access Structures." In: *Topics in Cryptology – CT-RSA 2013*. Ed. by Ed Dawson. Vol. 7779. Lecture Notes in Computer Science. Springer, Heidelberg, 2013, pp. 50–67. DOI: 10.1007/978-3-642-36095-4_4 (cit. on pp. 37, 38, 40, 41, 71).

[CKKR19] Jan Camenisch, Stephan Krenn, Ralf Küsters, and Daniel Rausch. "iUC: Flexible Universal Composability Made Simple." In: *Advances in Cryptology – ASIACRYPT 2019, Part III*. Ed. by Steven D. Galbraith and Shiho Moriai. Vol. 11923. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 2019, pp. 191–221. DOI: 10.1007/978-3-030-34618-8_7 (cit. on p. 104).

[CKLM14] Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. "Malleable Signatures: New Definitions and Delegatable Anonymous Credentials." In: *CSF 2014: IEEE 27st Computer Security Foundations Symposium*. Ed. by Anupam Datta and Cedric Fournet. IEEE Computer Society Press, 2014, pp. 199–213. DOI: 10.1109/CSF.2014.22 (cit. on pp. 40, 72).

[CKS10] Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. "Solving Revocation with Efficient Update of Anonymous Credentials." In: *SCN 10: 7th International Conference on Security in Communication Networks*. Ed. by Juan A. Garay and Roberto De Prisco. Vol. 6280. Lecture Notes in Computer Science. Springer, Heidelberg, Sept. 2010, pp. 454–471. DOI: 10.1007/978-3-642-15317-4_28 (cit. on pp. 154, 155).

[CKS11a] Jan Camenisch, Stephan Krenn, and Victor Shoup. *A Framework for Practical Universally Composable Zero-Knowledge Protocols*. Cryptology ePrint Archive, Report 2011/228. https://eprint.iacr.org/2011/228. 2011 (cit. on p. 30).

[CKS11b] Jan Camenisch, Stephan Krenn, and Victor Shoup. "A Framework for Practical Universally Composable Zero-Knowledge Protocols." In: *Advances in Cryptology – ASIACRYPT 2011*. Ed. by Dong Hoon Lee and Xiaoyun Wang. Vol. 7073. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 2011, pp. 449–467. DOI: 10.1007/978-3-642-25385-0_24 (cit. on p. 30).

[CL01] Jan Camenisch and Anna Lysyanskaya. "An Efficient System for Non-transferable Anonymous Credentials with Optional Anonymity Revocation." In: *Advances in Cryptology – EUROCRYPT 2001*. Ed. by Birgit Pfitzmann. Vol. 2045. Lecture Notes in Computer Science. Springer, Heidelberg, May 2001, pp. 93–118. DOI: 10.1007/3-540-44987-6_7 (cit. on pp. 40, 72, 152, 154–156, 161).

[CL02] Jan Camenisch and Anna Lysyanskaya. "Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials." In: *Advances in Cryptology – CRYPTO 2002*. Ed. by Moti Yung. Vol. 2442. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2002, pp. 61–76. DOI: 10.1007/3-540-45708-9_5 (cit. on p. 154).

[CL03] Jan Camenisch and Anna Lysyanskaya. "A Signature Scheme with Efficient Protocols." In: *SCN 02: 3rd International Conference on Security in Communication Networks*. Ed. by Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano. Vol. 2576. Lecture Notes in Computer Science. Springer, Heidelberg, Sept. 2003, pp. 268–289. DOI: 10.1007/3-540-36413-7_20 (cit. on pp. 154, 161).

[CL04] Jan Camenisch and Anna Lysyanskaya. "Signature Schemes and Anonymous Credentials from Bilinear Maps." In: *Advances in Cryptology – CRYPTO 2004*. Ed. by Matthew Franklin. Vol. 3152. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2004, pp. 56–72. DOI: 10.1007/978-3-540-28628-8_4 (cit. on pp. 40, 152, 154, 156, 161, 187).

[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. Springer, Heidelberg, Aug. 2006, pp. 78–96. DOI: 10.1007/11818175_5 (cit. on pp. 15, 19, 54).

[CL19] Elizabeth C. Crites and Anna Lysyanskaya. "Delegatable Anonymous Credentials from Mercurial Signatures." In: *Topics in Cryptology – CT-RSA 2019*. Ed. by Mitsuru Matsui. Vol. 11405. Lecture Notes in Computer Science. Springer, Heidelberg, Mar. 2019, pp. 535–555. DOI: 10.1007/978-3-030-12612-4_27 (cit. on p. 154).

[CLNR14] Jan Camenisch, Anja Lehmann, Gregory Neven, and Alfredo Rial. "Privacy-Preserving Auditing for Attribute-Based Credentials." In: *ESORICS 2014: 19th European Symposium on Research in Computer Security, Part II*. Ed. by Miroslaw Kutylowski and Jaideep Vaidya. Vol. 8713. Lecture Notes in Computer Science. Springer, Heidelberg, Sept. 2014, pp. 109–127. DOI: 10.1007/978-3-319-11212-1_7 (cit. on p. 154).

[CLNS17] Jan Camenisch, Anja Lehmann, Gregory Neven, and Kai Samelin. "UC-Secure Non-interactive Public-Key Encryption." In: *CSF 2017: IEEE 30st Computer Security Foundations Symposium*. Ed. by Boris Köpf and Steve Chong. IEEE Computer Society Press, 2017, pp. 217–233. DOI: 10.1109/CSF.2017.14 (cit. on p. 106).

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. "Universally composable two-party and multi-party secure computation." In: *34th Annual ACM Symposium on Theory of Computing*. ACM Press, May 2002, pp. 494–503. DOI: 10.1145/509907.509980 (cit. on p. 108).

[CLP22] Aisling Connolly, Pascal Lafourcade, and Octavio Perez-Kempner. "Improved Constructions of Anonymous Credentials from Structure-Preserving Signatures on Equivalence Classes." In: *Public-Key Cryptography - PKC 2022 - 25th IACR International Conference on Practice and Theory of Public-Key Cryptography, Virtual Event, March 8-11, 2022, Proceedings, Part I*. Ed. by Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe. Vol. 13177. Lecture Notes in Computer Science. Springer, 2022, pp. 409–438. DOI: 10.1007/978-3-030-97121-2_15 (cit. on p. 154).

[CM99] Jan Camenisch and Markus Michels. "Proving in Zero-Knowledge that a Number Is the Product of Two Safe Primes." In: *Advances in Cryptology – EUROCRYPT'99*. Ed. by Jacques Stern. Vol. 1592. Lecture Notes in Computer Science. Springer, Heidelberg, May 1999, pp. 107–122. DOI: 10.1007/3-540-48910-X_8 (cit. on p. 29).

[CMZ14] Melissa Chase, Sarah Meiklejohn, and Greg Zaverucha. "Algebraic MACs and Keyed-Verification Anonymous Credentials." In: *ACM CCS 2014: 21st Conference on Computer and Communications Security*. Ed. by Gail-Joon Ahn, Moti Yung, and Ninghui Li. ACM Press, Nov. 2014, pp. 1205–1216. DOI: 10.1145/2660267.2660328 (cit. on p. 154).

[COM22] ELECTRIC COIN COMPANY. *Zcash Parameter Generation*. 2022. URL: https://z.cash/technology/paramgen/ (visited on 09/19/2022) (cit. on p. 20).

[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. Springer, Heidelberg, Aug. 1997, pp. 410–424. DOI: 10.1007/BFb0052252 (cit. on p. 29).

[CV02] Jan Camenisch and Els Van Herreweghen. "Design and Implementation of The Idemix Anonymous Credential System." In: *ACM CCS 2002: 9th Conference on Computer and Communications Security*. Ed. by Vijayalakshmi Atluri. ACM Press, Nov. 2002, pp. 21–30. DOI: 10.1145/586110.586114 (cit. on p. 155).

[Dam10] Ivan Damgard. "On Σ-protocols." In: (2010). https://www.cs.au.dk/~ivan/Sigma.pdf (cit. on pp. 15, 29, 184).

[DDD05] Liesje Demuynck and Bart De Decker. *Anonymous updating of credentials*. CW Reports, Department of Computer Science, K.U.Leuven, Leuven, Belgium. http://www.cs.kuleuven.be/publicaties/rapporten/cw/CW430.abs.html. 2005 (cit. on p. 155).

[DDM17] Pratish Datta, Ratna Dutta, and Sourav Mukhopadhyay. *Attribute-Based Signatures for Turing Machines from Standard Assumptions*. Cryptology ePrint Archive, Report 2017/801. https://eprint.iacr.org/2017/801. 2017 (cit. on pp. 38–40).

[Deu+18] Dominic Deuber, Matteo Maffei, Giulio Malavolta, Max Rabkin, Dominique Schröder, and Mark Simkin. "Functional Credentials." In: *Proceedings on Privacy Enhancing Technologies* 2018.2 (Apr. 2018), pp. 64–84. DOI: 10.1515/popets-2018-0013 (cit. on p. 154).

[DGJL21] Denis Diemert, Kai Gellert, Tibor Jager, and Lin Lyu. "More Efficient Digital Signatures with Tight Multi-user Security." In: *PKC 2021: 24th International Conference on Theory and Practice of Public Key Cryptography, Part II*. Ed. by Juan Garay. Vol. 12711. Lecture Notes in Computer Science. Springer, Heidelberg, May 2021, pp. 1–31. DOI: 10.1007/978-3-030-75248-4_1 (cit. on p. 88).

[DGM18] Constantin Catalin Dragan, Daniel Gardham, and Mark Manulis. "Hierarchical Attribute-Based Signatures." In: *CANS 18: 17th International Conference on Cryptology and Network Security*. Ed. by Jan Camenisch and Panos Papadimitratos. Vol. 11124. Lecture Notes in Computer Science. Springer, Heidelberg, 2018, pp. 213–234. DOI: 10.1007/978-3-030-00434-7_11 (cit. on pp. 37–39, 45, 54, 87).

[DOT19] Pratish Datta, Tatsuaki Okamoto, and Katsuyuki Takashima. "Efficient Attribute-Based Signatures for Unbounded Arithmetic Branching Programs." In: *PKC 2019: 22nd International Conference on Theory and Practice of Public Key Cryptography, Part I*. Ed. by Dongdai Lin and Kazue Sako. Vol. 11442. Lecture Notes in Computer Science. Springer, Heidelberg, Apr. 2019, pp. 127–158. DOI: 10.1007/978-3-030-17253-4_5 (cit. on pp. 37–40, 88).

[DP06] Cécile Delerablée and David Pointcheval. "Dynamic Fully Anonymous Short Group Signatures." In: *Progress in Cryptology - VIETCRYPT 06: 1st International Conference on Cryptology in Vietnam*. Ed. by Phong Q. Nguyen. Vol. 4341. Lecture Notes in Computer Science. Springer, Heidelberg, Sept. 2006, pp. 193–210 (cit. on p. 54).

[DS18] David Derler and Daniel Slamanig. "Highly-Efficient Fully-Anonymous Dynamic Group Signatures." In: *ASIACCS 18: 13th ACM Symposium on Information, Computer and Communications Security*. Ed. by Jong Kim, Gail-Joon Ahn, Seungjoo Kim, Yongdae Kim, Javier López, and Taesoo Kim. ACM Press, Apr. 2018, pp. 551–565 (cit. on pp. 54, 72).

[DS19] David Derler and Daniel Slamanig. "Key-homomorphic signatures: definitions and applications to multiparty signatures and non-interactive zero-knowledge." In: *Des. Codes Cryptogr.* 87.6 (2019), pp. 1373–1413. DOI: 10.1007/s10623-018-0535-9 (cit. on p. 88).

[ECGD14] Ali El Kaafarani, Liqun Chen, Essam Ghadafi, and James H. Davenport. "Attribute-Based Signatures with User-Controlled Linkability." In: *CANS 14: 13th International Conference on Cryptology and Network Security*. Ed. by Dimitris Gritzalis, Aggelos Kiayias, and Ioannis G. Askoxylakis. Vol. 8813. Lecture Notes in Computer Science. Springer, Heidelberg, Oct. 2014, pp. 256–269. DOI: 10.1007/978-3-319-12280-9_17 (cit. on pp. 38, 39, 49, 72, 87).

[EE16] Rachid El Bansarkhani and Ali El Kaafarani. *Post-Quantum Attribute-Based Signatures from Lattice Assumptions*. Cryptology ePrint Archive, Report 2016/823. `https://eprint.iacr.org/2016/823`. 2016 (cit. on pp. 38, 40, 54, 67, 72, 87).

[EG17] Ali El Kaafarani and Essam Ghadafi. "Attribute-Based Signatures with User-Controlled Linkability Without Random Oracles." In: *16th IMA International Conference on Cryptography and Coding*. Ed. by Máire O'Neill. Vol. 10655. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 2017, pp. 161–184 (cit. on pp. 38–41, 49, 71, 72, 87).

[EGK14] Ali El Kaafarani, Essam Ghadafi, and Dalia Khader. "Decentralized Traceable Attribute-Based Signatures." In: *Topics in Cryptology – CT-RSA 2014*. Ed. by Josh Benaloh. Vol. 8366. Lecture Notes in Computer Science. Springer, Heidelberg, Feb. 2014, pp. 327–348. DOI: `10.1007/978-3-319-04852-9_17` (cit. on pp. 37–39, 49, 72, 87).

[EHM11] Alex Escala, Javier Herranz, and Paz Morillo. "Revocable Attribute-Based Signatures with Adaptive Security in the Standard Model." In: *AFRICACRYPT 11: 4th International Conference on Cryptology in Africa*. Ed. by Abderrahmane Nitaj and David Pointcheval. Vol. 6737. Lecture Notes in Computer Science. Springer, Heidelberg, July 2011, pp. 224–241 (cit. on pp. 37–39, 45, 130).

[EK18] Ali El Kaafarani and Shuichi Katsumata. "Attribute-Based Signatures for Unbounded Circuits in the ROM and Efficient Instantiations from Lattices." In: *PKC 2018: 21st International Conference on Theory and Practice of Public Key Cryptography, Part II*. Ed. by Michel Abdalla and Ricardo Dahab. Vol. 10770. Lecture Notes in Computer Science. Springer, Heidelberg, Mar. 2018, pp. 89–119. DOI: `10.1007/978-3-319-76581-5_4` (cit. on pp. 37–40, 54, 67, 87).

[FHH14] Eduarda S. V. Freire, Julia Hesse, and Dennis Hofheinz. "Universally Composable Non-Interactive Key Exchange." In: *SCN 14: 9th International Conference on Security in Communication Networks*. Ed. by Michel Abdalla and Roberto De Prisco. Vol. 8642. Lecture Notes in Computer Science. Springer, Heidelberg, Sept. 2014, pp. 1–20. DOI: `10.1007/978-3-319-10879-7_1` (cit. on p. 106).

[FHJ20] Marc Fischlin, Patrick Harasser, and Christian Janson. "Signatures from Sequential-OR Proofs." In: *Advances in Cryptology – EUROCRYPT 2020, Part III*. Ed. by Anne Canteaut and Yuval Ishai. Vol. 12107. Lecture Notes in Computer Science. Springer, Heidelberg, May 2020, pp. 212–244. DOI: `10.1007/978-3-030-45727-3_8` (cit. on p. 88).

[FHKS16] Georg Fuchsbauer, Christian Hanser, Chethan Kamath, and Daniel Slamanig. "Practical Round-Optimal Blind Signatures in the Standard Model from Weaker Assumptions." In: *SCN 16: 10th International Conference on Security in Communication Networks*. Ed. by Vassilis Zikas and Roberto De Prisco. Vol. 9841. Lecture Notes in Computer Science. Springer, Heidelberg, 2016, pp. 391–408. DOI: 10.1007/978-3-319-44618-9_21 (cit. on pp. 12, 187).

[FHS15] Georg Fuchsbauer, Christian Hanser, and Daniel Slamanig. "Practical Round-Optimal Blind Signatures in the Standard Model." In: *Advances in Cryptology – CRYPTO 2015, Part II*. Ed. by Rosario Gennaro and Matthew J. B. Robshaw. Vol. 9216. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2015, pp. 233–253. DOI: 10.1007/978-3-662-48000-7_12 (cit. on pp. 185, 187).

[FHS19] Georg Fuchsbauer, Christian Hanser, and Daniel Slamanig. "Structure-Preserving Signatures on Equivalence Classes and Constant-Size Anonymous Credentials." In: *Journal of Cryptology* 32.2 (Apr. 2019), pp. 498–546. DOI: 10.1007/s00145-018-9281-4 (cit. on pp. 154, 161).

[Fis05] Marc Fischlin. "Communication-Efficient Non-interactive Proofs of Knowledge with Online Extractors." In: *Advances in Cryptology – CRYPTO 2005*. Ed. by Victor Shoup. Vol. 3621. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2005, pp. 152–168. DOI: 10.1007/11535218_10 (cit. on pp. 26, 27, 29, 40, 67, 109, 157, 168, 184, 185).

[FKMV12] Sebastian Faust, Markulf Kohlweiss, Giorgia Azzurra Marson, and Daniele Venturi. "On the Non-malleability of the Fiat-Shamir Transform." In: *Progress in Cryptology - INDOCRYPT 2012: 13th International Conference in Cryptology in India*. Ed. by Steven D. Galbraith and Mridul Nandi. Vol. 7668. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 2012, pp. 60–79. DOI: 10.1007/978-3-642-34931-7_5 (cit. on pp. 15, 26, 29, 40, 67, 184).

[FLM11] Marc Fischlin, Benoît Libert, and Mark Manulis. "Non-interactive and Re-usable Universally Composable String Commitments with Adaptive Security." In: *Advances in Cryptology – ASIACRYPT 2011*. Ed. by Dong Hoon Lee and Xiaoyun Wang. Vol. 7073. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 2011, pp. 468–485. DOI: 10.1007/978-3-642-25385-0_25 (cit. on p. 106).

[FN16a] Dario Fiore and Anca Nitulescu. *On the (In)security of SNARKs in the Presence of Oracles*. Cryptology ePrint Archive, Report 2016/112. https://eprint.iacr.org/2016/112. 2016 (cit. on pp. 59, 60, 64, 65, 189).

[FN16b] Dario Fiore and Anca Nitulescu. "On the (In)Security of SNARKs in the Presence of Oracles." In: *TCC 2016-B: 14th Theory of Cryptography Conference, Part I*. Ed. by Martin Hirt and Adam D. Smith. Vol. 9985. Lecture Notes in Computer Science. Springer, Heidelberg, 2016, pp. 108–138. DOI: 10.1007/978-3-662-53641-4_5 (cit. on pp. 59, 60, 64, 65, 189).

[FO11] Marc Fischlin and Cristina Onete. "Relaxed Security Notions for Signatures of Knowledge." In: *ACNS 11: 9th International Conference on Applied Cryptography and Network Security*. Ed. by Javier Lopez and Gene Tsudik. Vol. 6715. Lecture Notes in Computer Science. Springer, Heidelberg, June 2011, pp. 309–326. DOI: 10.1007/978-3-642-21554-4_18 (cit. on p. 19).

[FPS20] Georg Fuchsbauer, Antoine Plouviez, and Yannick Seurin. "Blind Schnorr Signatures and Signed ElGamal Encryption in the Algebraic Group Model." In: *Advances in Cryptology – EUROCRYPT 2020, Part II*. Ed. by Anne Canteaut and Yuval Ishai. Vol. 12106. Lecture Notes in Computer Science. Springer, Heidelberg, May 2020, pp. 63–95. DOI: 10.1007/978-3-030-45724-2_3 (cit. on p. 12).

[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. Springer, Heidelberg, Aug. 1987, pp. 186–194. DOI: 10.1007/3-540-47721-7_12 (cit. on pp. 26, 40, 54, 67, 184).

[FS90] Uriel Feige and Adi Shamir. "Witness Indistinguishable and Witness Hiding Protocols." In: *22nd Annual ACM Symposium on Theory of Computing*. ACM Press, May 1990, pp. 416–426. DOI: 10.1145/100216.100272 (cit. on p. 86).

[Fuc10] Georg Fuchsbauer. "Automorphic signatures and applications." PhD thesis. ENS, Paris, 2010 (cit. on p. 90).

[GGM14] Christina Garman, Matthew Green, and Ian Miers. "Decentralized Anonymous Credentials." In: *ISOC Network and Distributed System Security Symposium – NDSS 2014*. The Internet Society, Feb. 2014 (cit. on p. 155).

[Gha15] Essam Ghadafi. "Stronger Security Notions for Decentralized Traceable Attribute-Based Signatures and More Efficient Constructions." In: *Topics in Cryptology – CT-RSA 2015*. Ed. by Kaisa Nyberg. Vol. 9048. Lecture Notes in Computer Science. Springer, Heidelberg, Apr. 2015, pp. 391–409. DOI: 10.1007/978-3-319-16715-2_21 (cit. on pp. 37–40, 67, 87).

[GJ18] Kristian Gjøsteen and Tibor Jager. "Practical and Tightly-Secure Digital Signatures and Authenticated Key Exchange." In: *Advances in Cryptology – CRYPTO 2018, Part II*. Ed. by Hovav Shacham and Alexandra Boldyreva. Vol. 10992. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2018, pp. 95–125. DOI: 10.1007/978-3-319-96881-0_4 (cit. on p. 88).

[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. Springer, Heidelberg, Aug. 2017, pp. 581–612. DOI: 10.1007/978-3-319-63715-0_20 (cit. on pp. 8, 10, 16–18, 20, 25, 54, 59, 61, 67–71).

[GM19] Daniel Gardham and Mark Manulis. "Hierarchical Attribute-Based Signatures: Short Keys and Optimal Signature Length." In: *ACNS 19: 17th International Conference on Applied Cryptography and Network Security*. Ed. by Robert H. Deng, Valérie Gauthier-Umaña, Martín Ochoa, and Moti Yung. Vol. 11464. Lecture Notes in Computer Science. Springer, Heidelberg, June 2019, pp. 89–109. DOI: 10.1007/978-3-030-21568-2_5 (cit. on pp. 37, 38, 40, 54, 88).

[GM22] Daniel Gardham and Mark Manulis. "Revocable Hierarchical Attribute-Based Signatures from Lattices." In: *Applied Cryptography and Network Security - 20th International Conference, ACNS 2022, Rome, Italy, June 20-23, 2022, Proceedings*. Ed. by Giuseppe Ateniese and Daniele Venturi. Vol. 13269. Lecture Notes in Computer Science. Springer, 2022, pp. 459–479. DOI: 10.1007/978-3-031-09234-3_23 (cit. on pp. 37, 38, 40, 88).

[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 (Apr. 1988), pp. 281–308 (cit. on pp. 11, 107).

[GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. "Proofs That Yield Nothing But Their Validity Or All Languages in NP Have Zero-Knowledge Proof Systems." In: *Journal of the ACM* 38.3 (1991), pp. 691–729 (cit. on p. 184).

[GMZ12] Aijun-J. Ge, C.-G. Ma, and Zhongwen-F. Zhang. "Attribute-based signature scheme with constant size signature in the standard model." In: *IET Inf. Secur.* 6.2 (2012), pp. 47–54. DOI: 10.1049/iet-ifs.2011.0094 (cit. on p. 41).

[GOS12] Jens Groth, Rafail Ostrovsky, and Amit Sahai. "New Techniques for Noninteractive Zero-Knowledge." In: *J. ACM* 59.3 (2012), 11:1–11:35. DOI: 10.1145/2220357.2220358 (cit. on pp. 29, 40, 89, 91, 108).

[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. Springer, Heidelberg, Dec. 2006, pp. 444–459. DOI: 10.1007/11935230_29 (cit. on pp. 29, 40, 67, 88).

[Gro07] Jens Groth. "Fully Anonymous Group Signatures Without Random Oracles." In: *Advances in Cryptology – ASIACRYPT 2007*. Ed. by Kaoru Kurosawa. Vol. 4833. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 2007, pp. 164–180. DOI: 10.1007/978-3-540-76900-2_10 (cit. on p. 72).

[Gro16] Jens Groth. "On the Size of Pairing-Based Non-interactive Arguments." In: *Advances in Cryptology – EUROCRYPT 2016, Part II*. Ed. by Marc Fischlin and Jean-Sébastien Coron. Vol. 9666. Lecture Notes in Computer Science. Springer, Heidelberg, May 2016, pp. 305–326. DOI: 10.1007/978-3-662-49896-5_11 (cit. on pp. 8, 29).

[GS08] Jens Groth and Amit Sahai. "Efficient Non-interactive Proof Systems for Bilinear Groups." In: *Advances in Cryptology – EUROCRYPT 2008*. Ed. by Nigel P. Smart. Vol. 4965. Lecture Notes in Computer Science. Springer, Heidelberg, Apr. 2008, pp. 415–432. DOI: 10.1007/978-3-540-78967-3_24 (cit. on pp. 29, 30, 40, 65, 89, 90, 96, 98).

[GS12] Jens Groth and Amit Sahai. "Efficient Noninteractive Proof Systems for Bilinear Groups." In: *SIAM J. Comput.* 41.5 (2012), pp. 1193–1232. DOI: 10.1137/080725386 (cit. on pp. 23, 29, 40, 65, 71, 89–91, 96, 98, 185).

[GW11] Craig Gentry and Daniel Wichs. "Separating succinct non-interactive arguments from all falsifiable assumptions." In: *43rd Annual ACM Symposium on Theory of Computing*. Ed. by Lance Fortnow and Salil P. Vadhan. ACM Press, June 2011, pp. 99–108. DOI: 10.1145/1993636.1993651 (cit. on pp. 19, 39, 59, 66).

[Her14] Javier Herranz. "Attribute-based signatures from RSA." In: *Theor. Comput. Sci.* 527 (2014), pp. 73–82. DOI: 10.1016/j.tcs.2014.01.028 (cit. on pp. 37, 38).

[Her16] Javier Herranz. "Attribute-based versions of Schnorr and ElGamal." In: *Appl. Algebra Eng. Commun. Comput.* 27.1 (2016), pp. 17–57. DOI: 10.1007/s00200-015-0270-7 (cit. on pp. 37–39, 46).

[HJ12] Dennis Hofheinz and Tibor Jager. "Tightly Secure Signatures and Public-Key Encryption." In: *Advances in Cryptology – CRYPTO 2012*. Ed. by Reihaneh Safavi-Naini and Ran Canetti. Vol. 7417. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2012, pp. 590–607. DOI: 10.1007/978-3-642-32009-5_35 (cit. on p. 88).

[HK19] Ulrich Haböck and Stephan Krenn. "Breaking and Fixing Anonymous Credentials for the Cloud." In: *CANS 19: 18th International Conference on Cryptology and Network Security*. Ed. by Yi Mu, Robert H. Deng, and Xinyi Huang. Vol. 11829. Lecture Notes in Computer Science. Springer, Heidelberg, Oct. 2019, pp. 249–269. DOI: 10.1007/978-3-030-31578-8_14 (cit. on p. 154).

[HLLR12] Javier Herranz, Fabien Laguillaumie, Benoît Libert, and Carla Ràfols. "Short Attribute-Based Signatures for Threshold Predicates." In: *Topics in Cryptology – CT-RSA 2012*. Ed. by Orr Dunkelman. Vol. 7178. Lecture Notes in Computer Science. Springer, Heidelberg, 2012, pp. 51–67. DOI: 10.1007/978-3-642-27954-6_4 (cit. on pp. 38, 39, 41, 71).

[HP22] Chloé Hébant and David Pointcheval. "Traceable Constant-Size Multi-authority Credentials." In: *Security and Cryptography for Networks - 13th International Conference, SCN 2022, Amalfi, Italy, September 12-14, 2022, Proceedings*. Ed. by Clemente Galdi and Stanislaw Jarecki. Vol. 13409. Lecture Notes in Computer Science. Springer, 2022, pp. 411–434. DOI: 10.1007/978-3-031-14791-3_18 (cit. on p. 154).

[HS15] Dennis Hofheinz and Victor Shoup. "GNUC: A New Universal Composability Framework." In: *Journal of Cryptology* 28.3 (July 2015), pp. 423–508. DOI: 10.1007/s00145-013-9160-y (cit. on p. 104).

[HS21] Lucjan Hanzlik and Daniel Slamanig. "With a Little Help from My Friends: Constructing Practical Anonymous Credentials." In: *ACM CCS 2021: 28th Conference on Computer and Communications Security*. Ed. by Giovanni Vigna and Elaine Shi. ACM Press, Nov. 2021, pp. 2004–2023. DOI: 10.1145/3460120.3484582 (cit. on p. 154).

[ILV11] Malika Izabachène, Benoît Libert, and Damien Vergnaud. "Block-Wise P-Signatures and Non-interactive Anonymous Credentials with Efficient Attributes." In: *Cryptography and Coding - 13th IMA International Conference, IMACC 2011, Oxford, UK, December 12-15, 2011. Proceedings*. Ed. by Liqun Chen. Vol. 7089. Lecture Notes in Computer Science. Springer, 2011, pp. 431–450. DOI: 10.1007/978-3-642-25516-8_26 (cit. on pp. 152, 154, 156, 187).

[IV19] Vincenzo Iovino and Ivan Visconti. "Non-interactive Zero Knowledge Proofs in the Random Oracle Model." In: *Codes, Cryptology and Information Security - Third International Conference, C2SI 2019, Rabat, Morocco, April 22-24, 2019, Proceedings - In Honor of Said El Hajji*. Ed. by Claude Carlet, Sylvain Guilley, Abderrahmane Nitaj, and El Mamoun Souidi. Vol. 11445. Lecture Notes in Computer Science. Springer, 2019, pp. 118–141. DOI: 10.1007/978-3-030-16458-4_9 (cit. on pp. 26, 40).

[JR17] Charanjit S. Jutla and Arnab Roy. "Improved Structure Preserving Signatures Under Standard Bilinear Assumptions." In: *PKC 2017: 20th International Conference on Theory and Practice of Public Key Cryptography, Part II*. Ed. by Serge Fehr. Vol. 10175. Lecture Notes in Computer Science. Springer, Heidelberg, Mar. 2017, pp. 183–209. DOI: 10.1007/978-3-662-54388-7_7 (cit. on p. 95).

[JSI96] Markus Jakobsson, Kazue Sako, and Russell Impagliazzo. "Designated Verifier Proofs and Their Applications." In: *Advances in Cryptology – EUROCRYPT'96*. Ed. by Ueli M. Maurer. Vol. 1070. Lecture Notes in Computer Science. Springer, Heidelberg, May 1996, pp. 143–154. DOI: 10.1007/3-540-68339-9_13 (cit. on p. 88).

[Kaa+17] Nesrine Kaaniche, Maryline Laurent, Pierre-Olivier Rocher, Christophe Kiennert, and Joaquín García-Alfaro. "*PCS* , A Privacy-Preserving Certification Scheme." In: *Data Privacy Management, Cryptocurrencies and Blockchain Technology - ESORICS 2017 International Workshops, DPM 2017 and CBT 2017, Oslo, Norway, September 14-15, 2017, Proceedings*. Ed. by Joaquín García-Alfaro, Guillermo Navarro-Arribas, Hannes Hartenstein, and Jordi Herrera-Joancomartí. Vol. 10436. Lecture Notes in Computer Science. Springer, 2017, pp. 239–256. DOI: 10.1007/978-3-319-67816-0_14 (cit. on pp. 37, 38, 107, 188).

[KCD09] Dalia Khader, Liqun Chen, and James H. Davenport. "Certificate-Free Attribute Authentication." In: *12th IMA International Conference on Cryptography and Coding*. Ed. by Matthew G. Parker. Vol. 5921. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 2009, pp. 301–325 (cit. on pp. 37–40).

[KL14] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*. CRC Press, 2014. ISBN: 978-1-466-57026-9. URL: https://www.crcpress.com/Introduction-to-Modern-Cryptography-Second-Edition/Katz-Lindell/p/book/9781466570269 (cit. on pp. 7–9).

[KM16] Franziskus Kiefer and Mark Manulis. "Universally Composable Two-Server PAKE." In: *ISC 2016: 19th International Conference on Information Security*. Ed. by Matt Bishop and Anderson C. A. Nascimento.

Vol. 9866. Lecture Notes in Computer Science. Springer, Heidelberg, Sept. 2016, pp. 147–166. DOI: 10.1007/978-3-319-45871-7_10 (cit. on p. 106).

[Kos+15] Ahmed Kosba, Zhichao Zhao, Andrew Miller, Yi Qian, Hubert Chan, Charalampos Papamanthou, Rafael Pass, abhi shelat, and Elaine Shi. *C∅C∅: A Framework for Building Composable Zero-Knowledge Proofs*. Cryptology ePrint Archive, Report 2015/1093. https://eprint.iacr.org/2015/1093. 2015 (cit. on pp. 17, 108).

[KPW15] Eike Kiltz, Jiaxin Pan, and Hoeteck Wee. "Structure-Preserving Signatures from Standard Assumptions, Revisited." In: *Advances in Cryptology – CRYPTO 2015, Part II*. Ed. by Rosario Gennaro and Matthew J. B. Robshaw. Vol. 9216. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2015, pp. 275–295. DOI: 10.1007/978-3-662-48000-7_14 (cit. on pp. 89, 90, 95).

[Ks22] Yashvanth Kondi and abhi shelat. *Improved Straight-Line Extraction in the Random Oracle Model With Applications to Signature Aggregation*. Cryptology ePrint Archive, Report 2022/393. https://eprint.iacr.org/2022/393. 2022 (cit. on p. 185).

[KTR20] Ralf Küsters, Max Tuengerthal, and Daniel Rausch. "The IITM Model: A Simple and Expressive Model for Universal Composability." In: *J. Cryptol.* 33.4 (2020), pp. 1461–1584. DOI: 10.1007/s00145-020-09352-1 (cit. on p. 112).

[Küs06] Ralf Küsters. "Simulation-Based Security with Inexhaustible Interactive Turing Machines." In: *19th IEEE Computer Security Foundations Workshop, (CSFW-19 2006), 5-7 July 2006, Venice, Italy*. IEEE Computer Society, 2006, pp. 309–320. DOI: 10.1109/CSFW.2006.30 (cit. on p. 104).

[Li+10] Jin Li, Man Ho Au, Willy Susilo, Dongqing Xie, and Kui Ren. "Attribute-based signature and its applications." In: *ASIACCS 10: 5th ACM Symposium on Information, Computer and Communications Security*. Ed. by Dengguo Feng, David A. Basin, and Peng Liu. ACM Press, Apr. 2010, pp. 60–69 (cit. on pp. 37, 38, 72, 87, 107).

[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. Springer, Heidelberg, May 2011, pp. 446–466. DOI: 10.1007/978-3-642-20465-4_25 (cit. on pp. 31, 106).

[LK08] Jin Li and Kwangjo Kim. *Attribute-Based Ring Signatures*. Cryptology ePrint Archive, Report 2008/394. https://eprint.iacr.org/2008/394. 2008 (cit. on p. 38).

[LW10] Allison B. Lewko and Brent Waters. "New Techniques for Dual System Encryption and Fully Secure HIBE with Short Ciphertexts." In: *TCC 2010: 7th Theory of Cryptography Conference*. Ed. by Daniele Micciancio. Vol. 5978. Lecture Notes in Computer Science. Springer, Heidelberg, Feb. 2010, pp. 455–479. DOI: 10.1007/978-3-642-11799-2_27 (cit. on p. 71).

[Mau11] Ueli Maurer. "Constructive Cryptography - A New Paradigm for Security Definitions and Proofs." In: *Theory of Security and Applications - Joint Workshop, TOSCA 2011, Saarbrücken, Germany, March 31 - April 1, 2011, Revised Selected Papers*. Ed. by Sebastian Mödersheim and Catuscia Palamidessi. Vol. 6993. Lecture Notes in Computer Science. Springer, 2011, pp. 33–56. DOI: 10.1007/978-3-642-27375-9_3 (cit. on p. 104).

[MF21] Arno Mittelbach and Marc Fischlin. *The Theory of Hash Functions and Random Oracles - An Approach to Modern Cryptography*. Information Security and Cryptography. Springer, 2021. ISBN: 978-3-030-63286-1. DOI: 10.1007/978-3-030-63287-8 (cit. on p. 26).

[MKWK19] Eduardo Morais, Tommy Koens, Cees van Wijk, and Aleksei Koren. "A Survey on Zero Knowledge Range Proofs and Applications." In: *SN Applied Sciences* 1.8 (July 2019), p. 946. DOI: 10.1007/s42452-019-0989-z (cit. on pp. 29, 30, 185).

[MPR08] Hemanta Maji, Manoj Prabhakaran, and Mike Rosulek. *Attribute-Based Signatures: Achieving Attribute-Privacy and Collusion-Resistance*. Cryptology ePrint Archive, Report 2008/328. https://eprint.iacr.org/2008/328. 2008 (cit. on pp. 37, 39–42, 72, 95, 97, 130).

[MPR10] Hemanta K. Maji, Manoj Prabhakaran, and Mike Rosulek. *Attribute-Based Signatures*. Cryptology ePrint Archive, Report 2010/595. https://eprint.iacr.org/2010/595. 2010 (cit. on pp. 39, 40, 42, 72, 95–98, 100).

[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. Springer, Heidelberg, Feb. 2011, pp. 376–392. DOI: 10.1007/978-3-642-19074-2_24 (cit. on pp. 36–43, 45, 46, 49, 71, 72, 87, 88, 95–101, 107, 130, 187, 189).

[MSBM22] Omid Mir, Daniel Slamanig, Balthazar Bauer, and René Mayrhofer. *Practical Delegatable Anonymous Credentials From Equivalence Class Signatures*. Cryptology ePrint Archive, Report 2022/680. https://eprint.iacr.org/2022/680. 2022 (cit. on p. 154).

[Nao03] Moni Naor. "On Cryptographic Assumptions and Challenges." In: *Advances in Cryptology – CRYPTO 2003*. Ed. by Dan Boneh. Vol. 2729. Lecture Notes in Computer Science. Springer, Heidelberg, Aug. 2003, pp. 96–109. DOI: 10.1007/978-3-540-45146-4_6 (cit. on p. 39).

[NDD06] Vincent Naessens, Liesje Demuynck, and Bart De Decker. "A Fair Anonymous Submission and Review System." In: *Communications and Multimedia Security, 10th IFIP TC-6 TC-11 International Conference, CMS 2006, Heraklion, Crete, Greece, October 19-21, 2006, Proceedings*. Ed. by Herbert Leitold and Evangelos P. Markatos. Vol. 4237. Lecture Notes in Computer Science. Springer, 2006, pp. 43–53. DOI: 10.1007/11909033_5 (cit. on p. 155).

[OT11] Tatsuaki Okamoto and Katsuyuki Takashima. "Efficient Attribute-Based Signatures for Non-monotone Predicates in the Standard Model." In: *PKC 2011: 14th International Conference on Theory and Practice of Public Key Cryptography*. Ed. by Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi. Vol. 6571. Lecture Notes in Computer Science. Springer, Heidelberg, Mar. 2011, pp. 35–52. DOI: 10.1007/978-3-642-19379-8_3 (cit. on pp. 38, 39, 41, 71).

[OT13] Tatsuaki Okamoto and Katsuyuki Takashima. "Decentralized Attribute-Based Signatures." In: *PKC 2013: 16th International Conference on Theory and Practice of Public Key Cryptography*. Ed. by Kaoru Kurosawa and Goichiro Hanaoka. Vol. 7778. Lecture Notes in Computer Science. Springer, Heidelberg, 2013, pp. 125–142. DOI: 10.1007/978-3-642-36362-7_9 (cit. on pp. 37–41, 45, 71, 88).

[OT14] Tatsuaki Okamoto and Katsuyuki Takashima. "Efficient Attribute-Based Signatures for Non-Monotone Predicates in the Standard Model." In: *IEEE Trans. Cloud Comput.* 2.4 (2014), pp. 409–421. DOI: 10.1109/TCC.2014.2353053 (cit. on pp. 37–40, 42, 46, 50, 88, 130).

[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. Springer, Heidelberg, 2016, pp. 111–126. DOI: 10.1007/978-3-319-29485-8_7 (cit. on pp. 70, 152, 156, 157, 161, 168, 184, 185, 187).

[PW01] Birgit Pfitzmann and Michael Waidner. "A Model for Asynchronous Reactive Systems and its Application to Secure Message Transmission." In: *2001 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2001, pp. 184–200. DOI: 10.1109/SECPRI.2001.924298 (cit. on p. 104).

[PZ13] Christian Paquin and Greg Zaverucha. *U-Prove Cryptographic Specification V1.1 (Revision 3)*. Released under the Open Specification Promise `http://www.microsoft.com/openspecifications/en/us/programs/osp/default.aspx`. 2013. URL: `https://www.microsoft.com/en-us/research/publication/u-prove-cryptographic-specification-v1-1-revision-3/` (cit. on p. 155).

[RVH17] Sietse Ringers, Eric R. Verheul, and Jaap-Henk Hoepman. "An Efficient Self-blindable Attribute-Based Credential Scheme." In: *FC 2017: 21st International Conference on Financial Cryptography and Data Security*. Ed. by Aggelos Kiayias. Vol. 10322. Lecture Notes in Computer Science. Springer, Heidelberg, Apr. 2017, pp. 3–20 (cit. on pp. 154, 156).

[SAH16] Yusuke Sakai, Nuttapong Attrapadung, and Goichiro Hanaoka. "Attribute-Based Signatures for Circuits from Bilinear Map." In: *PKC 2016: 19th International Conference on Theory and Practice of Public Key Cryptography, Part I*. Ed. by Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang. Vol. 9614. Lecture Notes in Computer Science. Springer, Heidelberg, Mar. 2016, pp. 283–300. DOI: `10.1007/978-3-662-49384-7_11` (cit. on pp. 37–41, 43, 45, 46, 49, 71, 87–93, 95, 101, 130, 187, 189).

[SAH18] Yusuke Sakai, Nuttapong Attrapadung, and Goichiro Hanaoka. "Practical attribute-based signature schemes for circuits from bilinear map." In: *IET Inf. Secur.* 12.3 (2018), pp. 184–193. DOI: `10.1049/iet-ifs.2017.0029` (cit. on p. 89).

[Sch91] Claus-Peter Schnorr. "Efficient Signature Generation by Smart Cards." In: *Journal of Cryptology* 4.3 (Jan. 1991), pp. 161–174. DOI: `10.1007/BF00196725` (cit. on pp. 29, 184).

[SCI20] SCIPR Lab. *libsnark: a C++ library for zk-SNARK proofs*. 2020. URL: `https://github.com/scipr-lab/libsnark` (visited on 04/04/2022) (cit. on p. 70).

[Sho04] Victor Shoup. *Sequences of games: a tool for taming complexity in security proofs*. Cryptology ePrint Archive, Report 2004/332. `https://eprint.iacr.org/2004/332`. 2004 (cit. on p. 73).

[SKAH18] Yusuke Sakai, Shuichi Katsumata, Nuttapong Attrapadung, and Goichiro Hanaoka. "Attribute-Based Signatures for Unbounded Languages from Standard Assumptions." In: *Advances in Cryptology – ASIACRYPT 2018, Part II*. Ed. by Thomas Peyrin and Steven Galbraith. Vol. 11273. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 2018, pp. 493–522. DOI: `10.1007/978-3-030-03329-3_17` (cit. on pp. 37–41, 49, 72, 87).

[SS09] Siamak Fayyaz Shahandashti and Reihaneh Safavi-Naini. "Threshold Attribute-Based Signatures and Their Application to Anonymous Credential Systems." In: *AFRICACRYPT 09: 2nd International Conference on Cryptology in Africa*. Ed. by Bart Preneel. Vol. 5580. Lecture Notes in Computer Science. Springer, Heidelberg, June 2009, pp. 198–216 (cit. on pp. 38, 156, 157).

[Tsa17] Rotem Tsabary. "An Equivalence Between Attribute-Based Signatures and Homomorphic Signatures, and New Constructions for Both." In: *TCC 2017: 15th Theory of Cryptography Conference, Part II*. Ed. by Yael Kalai and Leonid Reyzin. Vol. 10678. Lecture Notes in Computer Science. Springer, Heidelberg, Nov. 2017, pp. 489–518. DOI: 10.1007/978-3-319-70503-3_16 (cit. on p. 40).

[UKLC15] Miguel Urquidi, Dalia Khader, Jean Lancrenon, and Liqun Chen. "Attribute-Based Signatures with Controllable Linkability." In: *Trusted Systems - 7th International Conference, INTRUST 2015, Beijing, China, December 7-8, 2015, Revised Selected Papers*. Ed. by Moti Yung, Jian-biao Zhang, and Zhen Yang. Vol. 9565. Lecture Notes in Computer Science. Springer, 2015, pp. 114–129. DOI: 10.1007/978-3-319-31550-8_8 (cit. on pp. 38, 39, 49, 72, 87).

[Wat05] Brent R. Waters. "Efficient Identity-Based Encryption Without Random Oracles." In: *Advances in Cryptology – EUROCRYPT 2005*. Ed. by Ronald Cramer. Vol. 3494. Lecture Notes in Computer Science. Springer, Heidelberg, May 2005, pp. 114–127. DOI: 10.1007/11426639_7 (cit. on p. 96).

[Wee09] Hoeteck Wee. "Zero Knowledge in the Random Oracle Model, Revisited." In: *Advances in Cryptology – ASIACRYPT 2009*. Ed. by Mitsuru Matsui. Vol. 5912. Lecture Notes in Computer Science. Springer, Heidelberg, Dec. 2009, pp. 417–434. DOI: 10.1007/978-3-642-10366-7_25 (cit. on p. 26).

[WHHL15] Jianghong Wei, Xinyi Huang, Xuexian Hu, and Wenfen Liu. "Revocable Threshold Attribute-Based Signature against Signing Key Exposure." In: *Information Security Practice and Experience - 11th International Conference, ISPEC 2015, Beijing, China, May 5-8, 2015. Proceedings*. Ed. by Javier López and Yongdong Wu. Vol. 9065. Lecture Notes in Computer Science. Springer, 2015, pp. 316–330. DOI: 10.1007/978-3-319-17533-1_22 (cit. on p. 41).

[Zha+19] Yanhua Zhang, Ximeng Liu, Yupu Hu, Qikun Zhang, and Huiwen Jia. "Attribute-Based Signatures for Inner-Product Predicate from Lattices." In: *Cyberspace Safety and Security - 11th International Symposium, CSS 2019, Guangzhou, China, December 1-3, 2019, Proceedings, Part I*. Ed. by Jaideep Vaidya, Xiao Zhang, and Jin Li. Vol. 11982.

Lecture Notes in Computer Science. Springer, 2019, pp. 173–185. DOI: 10.1007/978-3-030-37337-5_14 (cit. on pp. 37, 38, 40).

[ZXLZ12] Fugeng Zeng, Chunxiang Xu, Qinyi Li, and Xiujie Zhang. "Attribute-based signature scheme with constant size signature." In: *Journal of Computational Information Systems* 8.7 (2012), pp. 2875–2882 (cit. on p. 41).