



DS-06-2017: Cybersecurity PPP: Cryptography

PRIViLEDGE
Privacy-Enhancing Cryptography in Distributed Ledgers


D2.2 – Definitions and Notions of Privacy-Enhancing Cryptographic Primitives for Ledgers

Due date of deliverable: June 30th, 2019

Actual submission date: June 30th, 2019

Grant agreement number: 780477
Start date of project: 1 January 2018
Revision 1.0

Lead contractor: Guardtime AS
Duration: 36 months

	Project funded by the European Commission within the EU Framework Programme for Research and Innovation HORIZON 2020
Dissemination Level	
PU = Public, fully open	X
CO = Confidential, restricted under conditions set out in the Grant Agreement	
CI = Classified, information as referred to in Commission Decision 2001/844/EC	

D2.2

Definitions and Notions of Privacy-Enhancing Cryptographic Primitives for Ledgers

Editor

Luisa Siniscalchi, Ivan Visconti (UNISA)

Contributors

Björn Tackmann (IBM)
Karim Baghery, Toomas Krips (UT)
Berry Schoenmakers, Toon Segers (TUE)
Michele Ciampi, Aggelos Kiayias (UEDIN)

Reviewers

Michele Ciampi (UEDIN), Panos Louridas (GRNET)

June 30th, 2019

Revision 1.0

The work described in this document has been conducted within the project PRIViLEDGE, started in January 2018. This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 780477.

The opinions expressed and arguments employed herein do not necessarily reflect the official views of the European Commission.

©Copyright by the PRIViLEDGE Consortium

Executive Summary

This document focuses on new definitions and notions of cryptographic primitives that are relevant to distributed ledger technologies (DLTs). The document consists of several technical contributions of partners of PRIViLEDGE that in the first 18 months of the project have identified some completely new notions and some refinements of previous notions that can have an impact on constructing and using DLTs, and therefore are worthy to study.

The first chapter summarizes the notions considered in this document while the last five chapters present the formal definitions along with useful discussions. The natural next step consists of studying constructions for such notions and the results of this work will be reported in deliverable D2.3.

Contents

1	Introduction	1
2	Notation	3
3	Publicly Verifiable Proofs	4
3.1	Introduction	4
3.2	Related Work	4
3.3	Blockchain Protocols	5
3.4	New Definitions of Publicly Verifiable Arguments	6
4	Token Management Platforms with Proof-of-Transfer	9
4.1	Introduction	9
4.2	Preliminaries	10
4.3	Model and Definitions	10
4.4	Realizing the Functionality with Existing Schemes	12
4.5	Planned Extensions	13
5	Timed Signatures and Zero-Knowledge Proofs	14
5.1	Introduction	14
5.2	The Model	17
5.3	The (Weak) Beacon functionality	17
5.4	Timed Signatures (TSign)	21
5.5	Timed Zero-Knowledge Proof of Knowledge (TPoK)	23
5.5.1	The NIZK functionality	23
5.6	Timed Signatures of Knowledge	25
5.7	A Survey of Related Works on Timed ZK and Signature	26
6	Bulletin Boards and BPK Model	31
6.1	Introduction	31
6.2	Previous Work and Improvements	31
6.3	E-Voting Defined via Bulletin Boards	32
6.3.1	Syntax of a Bulletin Board System	32
6.3.2	Introducing our Security Framework	33
6.3.3	(Confirmable) Persistence Definition	34
6.3.4	θ -Confirmable Liveness Definition	34
6.4	No-Auxiliary String Bare Public Key Model	35

7	Ring Signatures	37
7.1	Introduction	37
7.2	Definition	38
7.3	Discussion	38
8	Conclusion	40

Chapter 1

Introduction

The rise of distributed ledger technologies (DLTs) introduces new challenges for preserving data privacy. Previous definitions of privacy-preserving cryptographic tools are not always satisfying when data are uploaded on distributed ledgers or when there is no centralized setup.

This document consists of technical contributions of partners of PRIViLEDGE that have identified new notions and refinements of previous notions of privacy-enhancing cryptographic tools that are relevant to construct and use distributed ledgers. The notions are illustrated in five chapters each representing a technical contribution of a PRIViLEDGE partner including both known and new definitions. Moreover, each chapter discusses the importance of the presented definitions for DLTs. We briefly summarize the contents of the five chapters in the next five paragraphs.

Publicly verifiable proofs. A proof system allows a party called prover to convince another party called verifier about the veridicity of a statement $x \in L$, where L is an \mathcal{NP} language. Chapter 3 introduces the notion of privacy-preserving publicly verifiable proofs over a blockchain (i.e., a specific distributed ledger). The main feature of privacy-preserving proofs is that they are publicly verifiable which means that they can be verified by anyone, not just the participants in the protocol.

Publicly verifiability is useful in many settings where a prover would like to reuse the same proof with many verifiers, or in general when we want the proof to be transferable. In the proposed definition the verification is non-interactive, and the verifier will just access a blockchain for reading the messages that constitute the proof. The prover, however, is a blockchain player. Indeed the crucial idea is to allow the prover to interact with the blockchain, therefore producing a valid interactive proof that later on can be non-interactively verified by everyone.

Token management platforms with proof-of-transfer. Chapter 4 provides a unifying definitional framework for token systems (e.g., payment instruments) that abstracts both the representation of the tokens on the ledger and the use of the tokens. The proposed definitional framework allows a flexible parameterization and this can be very useful to model different types of tokens. Finally, this chapter highlights that for a practical use of a token system one needs a proof of transfer that allows a party to prove that she acted as the sender of a token.

Timed signatures and zero-knowledge proofs. Chapter 5 gives a formal treatment of timestamping cryptographic primitives in the Universal Composition (UC) framework [Can01] with respect to a global clock. It proposes timed versions of primitives commonly used for authenticating information, such as digital signatures, non-interactive zero-knowledge proofs, and signatures of knowledge. The definitions introduce a fine-grained treatment of the different timestamping guarantees, namely security against *postdating* and *backdating* attacks. It also defines an ideal beacon functionality with the goal of implementing it using DLTs.

Bulletin boards and BPK model. Chapter 6 first provides a formal definition of quasi-adaptive non-interactive zero-knowledge (QA-NIZK) arguments. They are NIZK arguments where the setup is based on the language associated to the NIZK. QA-NIZK is presented in a new model that is inspired by the BPK model [CGGM00] where the verifier sends her public key to a key authority that uses a distributed ledger to publish all public keys.

The second part of Chapter 6 focuses on bulletin boards noticing that previous definitions of bulletin boards for distributed ledgers were not suitable for e-voting. Therefore Chapter 6 includes a new definition of a bulletin board that takes into account the requirements of e-voting systems.

Ring signatures. Chapter 7 gives a formal definition of ad-hoc traceable ring signature schemes, that is an adaptation of [FS07]. A ring signature scheme is a cryptographic signature scheme that allows a signer to create a signature on behalf of a group of signers, without revealing their identities any further. The group of potential signers is called a ring [RST01]. An ad-hoc ring signature scheme furthermore allows a signer to create an arbitrary ring starting only from the public keys of other users, without any special setup. This notion can help to enhance user privacy in a distributed ledger. However, the ad-hoc ring signature in order to be useful in DLTs should prevent coins from being spent multiple times. In other words, using a ring signature scheme it should be possible to efficiently determine whether two (or more) signatures were created using the same secret key. Ring signature schemes which allow signatures to be matched are called traceable ring signatures [FS07].

Chapter 2

Notation

Here we introduce some common notation used in the next chapters. We denote the security parameter by λ and use “ $||$ ” as concatenation operator (i.e., if a and b are two strings then by $a||b$ we denote the concatenation of a and b). For a finite set Q , $x \leftarrow_s Q$ means sampling of x from Q with uniform distribution. We use the abbreviation PPT that stands for probabilistic polynomial time. We use $\text{poly}(\cdot)$ to indicate a generic polynomial function. A *polynomial-time relation* \mathcal{R} (or *polynomial relation*, in short) is a subset of $\{0, 1\}^* \times \{0, 1\}^*$ such that membership of (x, w) in \mathcal{R} can be decided in time polynomial $|x|$. For $(x, w) \in \mathcal{R}$, we call x the *instance* and w a *witness* for x . For a polynomial-time relation \mathcal{R} , we define the \mathcal{NP} -language $L_{\mathcal{R}}$ as $L_{\mathcal{R}} = \{x \mid \exists w : (x, w) \in \mathcal{R}\}$.

Definition 1 (Computational indistinguishability). Let $X = \{X_{\lambda}\}_{\lambda \in \mathbb{N}}$ and $Y = \{Y_{\lambda}\}_{\lambda \in \mathbb{N}}$ be ensembles, where X_{λ} 's and Y_{λ} 's are probability distributions over $\{0, 1\}^q$, for some $q = \text{poly}(\lambda)$. We say that $X = \{X_{\lambda}\}_{\lambda \in \mathbb{N}}$ and $Y = \{Y_{\lambda}\}_{\lambda \in \mathbb{N}}$ are *computationally indistinguishable*, denoted $X \approx Y$, if for every PPT distinguisher \mathcal{D} there exists a negligible function ν such that for sufficiently large $\lambda \in \mathbb{N}$,

$$\left| \Pr \left[t \leftarrow_s X_{\lambda} : \mathcal{D}(1^{\lambda}, t) = 1 \right] - \Pr \left[t \leftarrow_s Y_{\lambda} : \mathcal{D}(1^{\lambda}, t) = 1 \right] \right| < \nu(\lambda).$$

We note that in the usual case where $|X_{\lambda}| = \Omega(\lambda)$ and λ can be derived from a sample of X_{λ} , it is possible to omit the auxiliary input 1^{λ} . In this deliverable we also use the definition of *Statistical Indistinguishability* where the distinguisher \mathcal{D} is unbounded. In this case we use $X \equiv_s Y$ to denote that two ensembles are statistically indistinguishable.

Chapter 3

Publicly Verifiable Proofs

3.1 Introduction

In a proof system¹ a honest prover \mathcal{P} convinces a honest verifier \mathcal{V} about the veridicity of a statement $x \in L$, where L is an \mathcal{NP} language. At the same time, a proof system prevents a malicious prover to convince \mathcal{V} about the veridicity of a false statement. Obviously a prover could just send a witness proving membership in L to convince the verifier. However in many cases this might correspond to an excessive leakage of information and thus the concept of a privacy-preserving proof system was put forth under several flavors (e.g., zero knowledge, witness indistinguishability). For instance, the zero-knowledge property guarantees that a proof does not reveal any information beyond the mere membership of x in L .

A proof system is *publicly verifiable*, if everyone, by looking at the transcript of the proof, can be convinced that the theorem is true. The verification procedure is therefore non-interactive. Public verifiability is useful in many settings where a prover would like to reuse the same proof with many verifiers, or in general when we want the proof to be transferable.

The goal of this chapter is to define publicly verifiable proof systems that can be run leveraging currently available blockchains. Starting from the formal definition of a blockchain given in [PSS17,GG17] we will define a public verifiable proof system where \mathcal{P} makes use of a blockchain to provide a publicly-verifiable proof to any \mathcal{V} . In our definition the prover is a blockchain party (i.e., she is allowed to write/read data to/from the blockchain). This captures the idea of allowing the prover to play with the blockchain, therefore producing a valid transcript that later on can be verified by everyone. Instead, the verification phase is non-interactive, the verifier is not necessarily a blockchain party, and she just needs to run on input the same blockchain used by the prover. We will define publicly-verifiable proof systems from blockchains considering both the zero knowledge (ZK) [GMR85] and the witness indistinguishability (WI) properties [FLS90].

3.2 Related Work

Previous definitions of publicly-verifiable non-interactive zero-knowledge proofs have considered trusted setups [FLS90,GO07,BFM88,GOS06] or the random oracle model. The notion of non-interactive WI in the plain model was defined in [BOV07]. Goyal and Goyal in [GG17] defined WI and ZK from a blockchain, requiring that prover and verifier receive a blockchain as input.

Our definitions compared to [GG17]. Our definitions differ from the ones of [BOV07,GG17] since in our case the prover is also a party of the blockchain protocol, while the verifier remains non-interactive.

¹When discussing informally, we will often say proof system even when the security is only about polynomial-time adversarial provers. In the formal definitions however we will explicitly use the standard term “argument”.

3.3 Blockchain Protocols

The next text follows in large part [PSS17, GG17]. A blockchain protocol Γ consists of 3 polynomial-time algorithms (UpdateState, GetRecords, Broadcast) with the following syntax.

- UpdateState($1^\lambda, st$): It takes as input the security parameter λ , and a state st and outputs the updated state st' ².
- GetRecords($1^\lambda, st$): It takes as input the security parameter λ and a state st . It outputs the longest ordered sequence of valid blocks \mathbf{B} (or simply blockchain) contained in the state, where each block in the chain itself contains an unordered sequence of records messages³.
- Broadcast($1^\lambda, m$): It takes as input the security parameter λ and a message m , and broadcasts the message over the network to all nodes executing the blockchain protocol. It does not give any output.

As in [GKL15, PSS17], the blockchain protocol is also parameterized by a validity predicate V that captures semantics of any particular blockchain application. The validity predicate takes as input a sequence of blocks \mathbf{B} and outputs a bit, where 1 certifies the validity of the blockchain \mathbf{B} and 0 its invalidity⁴. We will indicate with Γ^V a blockchain protocol Γ that has with validity predicate V .

Execution of Γ^V . The execution of the protocol Γ^V proceeds in rounds that model time steps. Each participant in the protocol runs algorithm UpdateState to keep track of the current (latest) blockchain state. This corresponds to listening on the broadcast network for messages from other nodes. The algorithm GetRecords is used to extract an ordered sequence of blocks encoded in the blockchain state, which is considered as the common public ledger among all nodes. The Broadcast algorithm is used by a party when she wants to post a new message m on the blockchain. Note that the message m is accepted by the blockchain protocol only if it satisfies the validity predicate V given the current state, (i.e., the current sequence of blocks).

Following prior work [GKL15, PSS17], we define the protocol execution in the Universal Composability framework of [Can01] since it elegantly allows to model the activation of parties and the execution of a protocol while other protocols are being executed. For any blockchain protocol $\Gamma^V = (\text{UpdateState}, \text{GetRecords}, \text{Broadcast})$, the protocol execution is directed by the environment $\mathcal{Z}(1^\lambda)$ where λ is the security parameter. The environment \mathcal{Z} activates the parties as either honest or corrupt, and is also responsible for providing inputs/records to all parties in each round. All the corrupt parties are controlled by the adversary \mathcal{A} and \mathcal{A} can corrupt them adaptively after that the execution of Γ^V is started. The adversary is also responsible for the delivery of all network messages. Honest parties start by executing UpdateState on input 1^λ with an empty local state $st = \epsilon$.

- In round r , each honest party P_i potentially receives a message(s) m from \mathcal{Z} and potentially receives incoming network messages (delivered by \mathcal{A}). It may then perform any computation, broadcast a message (using the Broadcast algorithm) to all other parties (which will be delivered by the adversary; see below) and update its local state st_i . It could also attempt to “add” a new block to its chain.
- \mathcal{A} is responsible for delivering all messages sent by parties (honest or corrupted) to all other parties. \mathcal{A} cannot modify the content of messages broadcast by honest parties, but it may delay or reorder the delivery of a message as long as it eventually delivers all messages within a certain time limit. The identity of the sender is not known to the recipient.
- At any point \mathcal{Z} can communicate with adversary \mathcal{A} .

²The local state should be considered as the entire blockchain (i.e., a sequence of mined blocks along with metadata) in Bitcoin and other cryptocurrencies.

³The sequence \mathbf{B} should be considered as the entire transaction history in Bitcoin and other cryptocurrencies, where the blocks are ordered in the sequence they were mined.

⁴The validity predicate could be used to capture various fundamental properties (e.g., in Bitcoin and other cryptocurrencies, it could be used to check for double spending).

Blockchain notation.

Let P be a party playing in Γ^V protocol, the view of P consists of the messages received during the execution of Γ^V , along with its randomness and its inputs. Let $\text{view} \leftarrow \text{Exec}^{\Gamma^V}(\mathcal{A}, \mathcal{H}, \mathcal{Z}, 1^\lambda)$ be the random variable denoting the joint view of all parties in the execution of protocol Γ^V with adversary \mathcal{A} and set of honest parties \mathcal{H} in environment \mathcal{Z} . This joint view view fully determines the execution. Let $\Gamma_{\text{view}}^V(\mathcal{A}, \mathcal{H}, \mathcal{Z}, 1^\lambda)$ denote an execution of $\Gamma^V(\mathcal{A}, \mathcal{H}, \mathcal{Z}, 1^\lambda)$ producing view as joint view. We will denote by \mathcal{P}^{st} a stateful algorithm \mathcal{P} with state st .

3.4 New Definitions of Publicly Verifiable Arguments

We now present our new definitions for public verifiability argument system from blockchains.

Definition 2. A pair of stateful PPT algorithms $\Pi = (\mathcal{P}, \mathcal{V})$ over a blockchain protocol Γ^V is a publicly verifiable argument system for the \mathcal{NP} -language \mathcal{L} with witness relation \mathcal{R} if it satisfies the following properties:

Completeness. $\forall x, w$ s.t. $\mathcal{R}(x, w) = 1$, \forall PPT adversary \mathcal{A} and set of honest parties \mathcal{H} and environment \mathcal{Z} , assuming that $\mathcal{P} \in \mathcal{H}$, there exist negligible functions $\nu_1(\cdot), \nu_2(\cdot)$ such that:

$$\Pr \left[\begin{array}{l} \text{view} \leftarrow \text{Exec}^{\Gamma^V}(\mathcal{A}, \mathcal{H}, \mathcal{Z}, 1^\lambda) \\ \mathcal{V}(x, \pi, \mathbf{B}) = 1 \quad \bullet \quad \pi \leftarrow \mathcal{P}^{\text{st}\mathcal{P}}(x, w) \\ \mathbf{B} = \text{GetRecords}(1^\lambda, \text{st}_j) \end{array} \right] \geq 1 - \nu_1(|x|) - \nu_2(\lambda)$$

where $\text{st}_{\mathcal{P}}$ denotes the state of \mathcal{P} ⁵ during the execution $\Gamma_{\text{view}}^V(\mathcal{A}, \mathcal{H}, \mathcal{Z}, 1^\lambda)$ ⁶.

Furthermore st_j is the state of an honest party $P_j \in \mathcal{H}$ at the end of the execution $\Gamma_{\text{view}}^V(\mathcal{A}, \mathcal{H}, \mathcal{Z}, 1^\lambda)$.

If all messages of π are in the blockchain then the proof is *on-chain*, and is *off-chain* otherwise.

Soundness. $\forall x \notin \mathcal{L}$, \forall stateful PPT adversary \mathcal{A} and set of honest parties \mathcal{H} and environment \mathcal{Z} , there exist negligible functions $\nu_1(\cdot), \nu_2(\cdot)$ such that:

$$\Pr \left[\begin{array}{l} \text{view} \leftarrow \text{Exec}^{\Gamma^V}(\mathcal{A}, \mathcal{H}, \mathcal{Z}, 1^\lambda) \\ \mathcal{V}(x, \pi, \mathbf{B}) = 1 \quad \bullet \quad \pi, x \leftarrow \mathcal{A}^{\text{st}\mathcal{A}} \\ \mathbf{B} = \text{GetRecords}(1^\lambda, \text{st}_j) \end{array} \right] \leq \nu_1(|x|) + \nu_2(\lambda)$$

where $\text{st}_{\mathcal{A}}$ denotes the state of \mathcal{A} during the execution $\Gamma_{\text{view}}^V(\mathcal{A}, \mathcal{H}, \mathcal{Z}, 1^\lambda)$. Furthermore st_j is the state of an honest party $P_j \in \mathcal{H}$ at the end of the execution $\Gamma_{\text{view}}^V(\mathcal{A}, \mathcal{H}, \mathcal{Z}, 1^\lambda)$.

Comment on Definition 2. Notice that the prover \mathcal{P} is a blockchain party (i.e., she is allowed to write/read data to/from the blockchain). Also, notice that the statement to be proved can be chosen adaptively by the adversary.

Definition 3. A public verifiable argument system $\Pi = (\mathcal{P}, \mathcal{V})$ over a blockchain protocol Γ^V for the \mathcal{NP} -language \mathcal{L} with witness relation \mathcal{R} is an Argument of Knowledge (AoK) if it satisfies the following property.

Argument of Knowledge (AoK). There is a stateful PPT algorithm \mathcal{E} such that for all x , any stateful PPT adversary \mathcal{A} and any set of honest parties \mathcal{H} and environment \mathcal{Z} , there exist negligible functions $\nu_1(\cdot), \nu_2(\cdot)$ such that:

$$\left\{ (\text{view}_{\mathcal{A}}) : \text{view} \leftarrow \text{Exec}^{\Gamma^V}(\mathcal{A}, \mathcal{H}, \mathcal{Z}, 1^\lambda) \right\} \equiv_s \left\{ (\text{view}_{\mathcal{A}}) : \text{view} \leftarrow \text{Exec}^{\Gamma^V}(\mathcal{A}, \mathcal{E}, \mathcal{Z}, 1^\lambda) \right\}$$

⁵The running time of \mathcal{P} is polynomial in the size of the blockchain $\mathbf{B} = \text{GetRecords}(1^\lambda, \text{st}_j)$ where st_j is the state of $P_j \in \mathcal{H}$ at the end of the execution $\Gamma_{\text{view}}^V(\mathcal{A}, \mathcal{H}, \mathcal{Z}, 1^\lambda)$.

⁶Note that after that \mathcal{P} outputs π the execution of $\text{Exec}^V(\mathcal{A}, \mathcal{H}, \mathcal{Z}, 1^\lambda)$ could still continue giving eventually view as output.

and

$$\Pr \left[\begin{array}{l} \text{view} \leftarrow \text{Exec}^{\Gamma^V}(\mathcal{A}, \mathcal{E}, \mathcal{Z}, 1^\lambda) \\ \mathcal{V}(x, \pi, \mathbf{B}) = 0 \vee \mathcal{R}(x, w) = 1 \quad \bullet \quad \mathbf{B} = \text{GetRecords}(1^\lambda, \text{st}_j) \\ (\pi, x) \leftarrow \mathcal{A}^{\text{st}_\mathcal{A}}, w \leftarrow \mathcal{E}(\pi, x) \end{array} \right] \geq 1 - \nu_1(|x|) - \nu_2(\lambda)$$

where $\text{st}_\mathcal{P}$, $\text{st}_\mathcal{A}$ denote the state of \mathcal{P} and of \mathcal{A} during the execution $\Gamma_{\text{view}}^V(\mathcal{A}, \mathcal{E}, \mathcal{Z}, 1^\lambda)$ and $\text{view}_\mathcal{A}$ is the view of \mathcal{A} in view. Furthermore st_j is the state of an honest party $P_j \in \mathcal{H}$ at the end of the execution $\Gamma_{\text{view}}^V(\mathcal{A}, \mathcal{H}, \mathcal{Z}, 1^\lambda)$.

Comment on Definition 3. In this definition we consider two experiments. In the first experiment, the adversary interacts with the other honest blockchain parties. In the second experiment we require the existence of an expected polynomial-time extractor \mathcal{E} that interacts with \mathcal{A} and emulates all the blockchain honest parties. We require that the views of the adversarial prover in the two experiments are statistically indistinguishable. If π is accepting, then with overwhelming probability \mathcal{E} outputs the witness w used by \mathcal{A} to compute π w.r.t. x . The statement x is adaptively chosen by \mathcal{A} (this is not true for the definition of AoK in [GG17]). The probability is expressed on the length of x and on the length of the blockchain.

Definition 4. A publicly verifiable argument system $\Pi = (\mathcal{P}, \mathcal{V})$ over a blockchain protocol Γ^V for the \mathcal{NP} -language \mathcal{L} with witness relation \mathcal{R} is witness indistinguishable (PVWI) if it satisfies the following property:

$\forall x, w_0, w_1$ s.t. $\mathcal{R}(x, w_0) = 1$ and $\mathcal{R}(x, w_1) = 1$, \forall stateful PPT adversary \mathcal{A} and set of honest parties \mathcal{H} and environment \mathcal{Z} , assuming that $\mathcal{P} \in \mathcal{H}$ it holds that:

$$\left\{ (\text{view}_\mathcal{A}, \pi) : \text{view} \leftarrow \text{Exec}^{\Gamma^V}(\mathcal{A}, \mathcal{H}, \mathcal{Z}, 1^\lambda), \pi \leftarrow \mathcal{P}^{\text{st}_\mathcal{P}}(x, w_0) \right\} \\ \approx \\ \left\{ (\text{view}_\mathcal{A}, \pi) : \text{view} \leftarrow \text{Exec}^{\Gamma^V}(\mathcal{A}, \mathcal{H}, \mathcal{Z}, 1^\lambda), \pi \leftarrow \mathcal{P}^{\text{st}_\mathcal{P}}(x, w_1) \right\}$$

where $\text{st}_\mathcal{P}$ denotes the state of \mathcal{P} during the execution $\Gamma_{\text{view}}^V(\mathcal{A}, \mathcal{H}, \mathcal{Z}, 1^\lambda)$ and $\text{view}_\mathcal{A}$ ⁷ is the view of \mathcal{A} in view.

Comment on Definition 4. In the first experiment, the honest prover uses w_0 as witness while in the second experiment the honest prover computes the proof π using the witness w_1 . We require that the views of the adversarial verifier in the two experiments be computationally indistinguishable.

We require that the execution of the blockchain protocol still continues (in both experiments) after the proof π is seen by \mathcal{A} . Therefore \mathcal{A} can potentially distinguish the experiment where w_0 is used as witness from the one where w_1 is used as witness based also on data available on the blockchain (this is not explicitly mentioned in the WI definition of [GG17]).

Definition 5. A public verifiable argument system $\Pi = (\mathcal{P}, \mathcal{V})$ over a blockchain protocol Γ^V for the \mathcal{NP} -language \mathcal{L} with witness relation \mathcal{R} is zero knowledge if it satisfies the following property:

There is a stateful PPT algorithm \mathcal{S} such that $\forall x, w$ s.t. $\mathcal{R}(x, w) = 1$, \forall PPT adversary \mathcal{A} and set of honest parties \mathcal{H} and environment \mathcal{Z} , where $\mathcal{P} \in \mathcal{H}$ it holds that

$$\left\{ (\text{view}_\mathcal{A}, \pi) : \text{view} \leftarrow \text{Exec}^{\Gamma^V}(\mathcal{A}, \mathcal{S}, \mathcal{Z}, 1^\lambda), \pi \leftarrow \mathcal{S}^{\text{st}_\mathcal{S}}(x) \right\} \\ \approx \\ \left\{ (\text{view}_\mathcal{A}, \pi) : \text{view} \leftarrow \text{Exec}^{\Gamma^V}(\mathcal{A}, \mathcal{H}, \mathcal{Z}, 1^\lambda), \pi \leftarrow \mathcal{P}^{\text{st}_\mathcal{P}}(x, w) \right\}$$

⁷Note that $\text{view}_\mathcal{A}$ can contain auxiliary inputs from the execution of $\Gamma^V(\mathcal{A}, \mathcal{H}, \mathcal{Z}, 1^\lambda)$ that could continue after that π is computed.

where $st_{\mathcal{P}}$, $st_{\mathcal{S}}$ denote respectively the state of \mathcal{P} and the state of \mathcal{S} during the execution of $\Gamma^{\mathcal{V}}(\mathcal{A}, \mathcal{H}, \mathcal{Z}, 1^\lambda)$ that defines view and where $view_{\mathcal{A}}$ is the view of \mathcal{A} in view.

Note that since \mathcal{S} is emulating all the honest parties, the state $st_{\mathcal{S}}$ can be seen as the set of states of the simulated honest parties.

Comment on Definition 5. In this definition we are considering two experiments called, respectively, the real experiment and the simulated experiment. In the real experiment, the adversary interacts with the honest parties: the honest prover and the other honest blockchain parties. For the simulated experiment we require the existence of a PPT simulator \mathcal{S} which is able to simulate the view of the adversary without having access to the witness. In the simulated experiment the adversary interacts with \mathcal{S} alone. The simulator in turn emulates all the honest parties including the honest prover, and the others blockchain parties. We require that the views of the adversarial verifier in the two experiments be computationally indistinguishable. Intuitively, the fact that the simulator \mathcal{S} emulates all the honest parties, it will help \mathcal{S} in producing a view that is indistinguishable from the one of the real experiment.

As for the Definition 4 we require that the execution of the blockchain protocol still continues (in both experiments) after the proof π is seen by \mathcal{A} . Therefore \mathcal{A} can potentially distinguish the simulated experiment from the real experiment based on the additional data from the blockchain. In the ZK definition of [GG17] the adversary does not have this additional power.

Chapter 4

Token Management Platforms with Proof-of-Transfer

4.1 Introduction

The first and still arguably most prominent application of Distributed Ledger Technologies (DLTs) is that of a cryptocurrency, or, more technically, a *token management platform*. (A cryptocurrency can be seen as a token platform whose tokens are traded freely and not backed by other assets or currencies at a fixed rate.) The DLT serves as a distributed database storing the information which users own which tokens, and allowing users to transfer tokens from or to other users. Various token schemes exist, and despite the fact that the intended high-level functionality is similar in all cases, they differ in how the tokens are represented as well as in the security guarantees they provide. For instance, most cryptocurrencies following Bitcoin [Nak09] use a so-called *unspent transaction-output (UTXO)* model, in which a transaction spends tokens by explicitly addressing the previous transaction in which they were received. Ethereum [Woo14] uses a *balance* model, in which the ledger stores the balance of all tokens the user controls. Furthermore, platforms such as Monero [vS13] or Zerocash [BCG⁺14] build on a similar transaction concept as Bitcoin, but additionally include mechanisms to protect privacy. Token schemes are also used by platforms in a variety of ways, such as in a permissionless cryptocurrency where new tokens are generated during *mining* associated with the consensus protocol, or in a permissioned platform where tokens are *minted* by dedicated parties. Nevertheless, the core underlying technologies are always the same.

The role of this chapter is two-fold. First, we aim to provide a unifying definitional framework for token systems that abstracts from the representation of the tokens on the ledger as well as the use of the token system, and allows for a flexible parameterisation to model different types of, e.g., privacy guarantees or token issuance policies. Second, we point out that for a practical use of a (especially privacy-preserving) token system as, e.g., a payment instrument in a different protocol, one needs a *proof of transfer* that allows a party to prove that it acted as the sender of a token and is therefore eligible to, e.g., access a certain good or service. The formal framework we describe allows to model different types of token systems and includes operations for a proof of transfer.

Related work. Several other definitions have appeared in the literature. Many of them are for UTXO-based systems and describe the algorithms accordingly. Security is defined in terms of properties of these algorithms [BCG⁺14, ACCK18]. Of course, these definitions are very specific to the individual schemes and make it difficult to compare the security achieved by different schemes. Overall, it is fair to say that the definitions better capture the security of the considered schemes, and we therefore view them as complementary to our approach, which targets a more abstract perspective with the goal of making different schemes comparable and interchangeable.

4.2 Preliminaries

Our formalization follows the approach put forth in the frameworks of Canetti [Can01] or Pfitzmann and Waidner [PW01], in which the goal of a protocol or scheme is specified in an ideal functionality that describes both the required operations and the permissible imperfections, and a protocol is then proved to realize this ideal functionality. Yet, both of these early models lack the flexibility required for modeling different types of using token schemes in a unified and compact way.

Let us elaborate, using the example of token creation. In many permissionless cryptocurrencies, token issuance is incorporated as *mining* in the consensus process: the party eligible for generating a block may include a *coinbase transaction* [Nak09] that creates new tokens and assigns them to some party, usually the miner itself. In permissioned systems such as central-bank issued tokens, the creation of new tokens is implemented as *minting* by a specific authoritative entity. In Ethereum ERC20 token contracts [Eth18], token issuance is freely programmable at the discretion of the creator of the contract. While the *mechanism* of token creation is the same in the above cases, and depends only on the technical implementation of the token system, the *policy* that describes the eligibility for creating tokens is different. Our formalization aims at describing the mechanism of token creation as an ideal functionality, while allowing to specify the policy in a way that is flexible and allows for using the same functionality in different application contexts.

More technically, we follow an approach that (to the best of our knowledge) was first used in the recent work of Badertscher, Maurer, and Tackmann [BMT18]. The separation between functionality and policy is technically achieved by means of an additional interface that controls, in this case, the token creation. In security proofs (and this is in main departure from the framework of [Can01]), this interface can be accessed directly by the distinguisher in both the real-world and the ideal-world models. (This is reminiscent of the environment access to the global setup functionality in the GUC model [CDPW07], but in our model each functionality can have such a free interface.) This *free* interface then allows to prove the functionality of a scheme independently of policies, i.e., the functionality is provided for *any* policy, which can be understood as being absorbed into the distinguisher. When using the token functionality in a proof of a complete DLT system, the interface is connected to a converter (in the framework of Maurer and Renner [MR11]) or additional ideal functionality (in the framework of Küsters and Tuengerthal [KT13]) that implements the policy.

A free interface allows the distinguisher to interact with both the ideal and the hybrid functionalities directly. This results in a stronger and more general condition compared to considering the capabilities at that interface as part of the attacker’s interface and, therefore, in the ideal-world model providing them to the simulator. More intuitively, the free interface can be seen as a way for the distinguisher to enforce that certain aspects in the real and the ideal world are the same. We will use the free interface to let the distinguisher control the transmission semantics; this leaves our statements general and independent of any concrete such semantics.

4.3 Model and Definitions

We describe the general but basic (ideal) functionality of a token system as \mathcal{F}_{Tok} in Figure 4.1. This version of the functionality is, for simplicity, specified without the proof-of-transfer functionality, which we add later in Figure 4.2. In the current version, we only allow for static corruption of parties.

Basic functionality \mathcal{F}_{Tok} . Functionality \mathcal{F}_{Tok} , as specified in Figure 4.1, has the following interfaces for controlling its behavior: A set \mathcal{P} of party interfaces that represent users that control (i.e., own or transfer) tokens between each other, an interface *Issue* that controls token creation, and an interface *Setup* that is needed for initializing the functionality. On a very high level, the functionality manages for each party $P \in \mathcal{P}$ an account balance $acc_P \in \mathbb{N}$ and allows parties to transfer tokens under the condition that their balance remains positive. The choice of set \mathcal{P} allows to model permissioned or permissionless settings.

At the *Setup* interface, the functionality only accepts a single message *setup*, which must be provided before any other input can be given. This allows to model schemes such as Zerocash [BCG⁺14], in which a centralized setup procedure must be performed before the system can be used. The *Issue* interface allows to

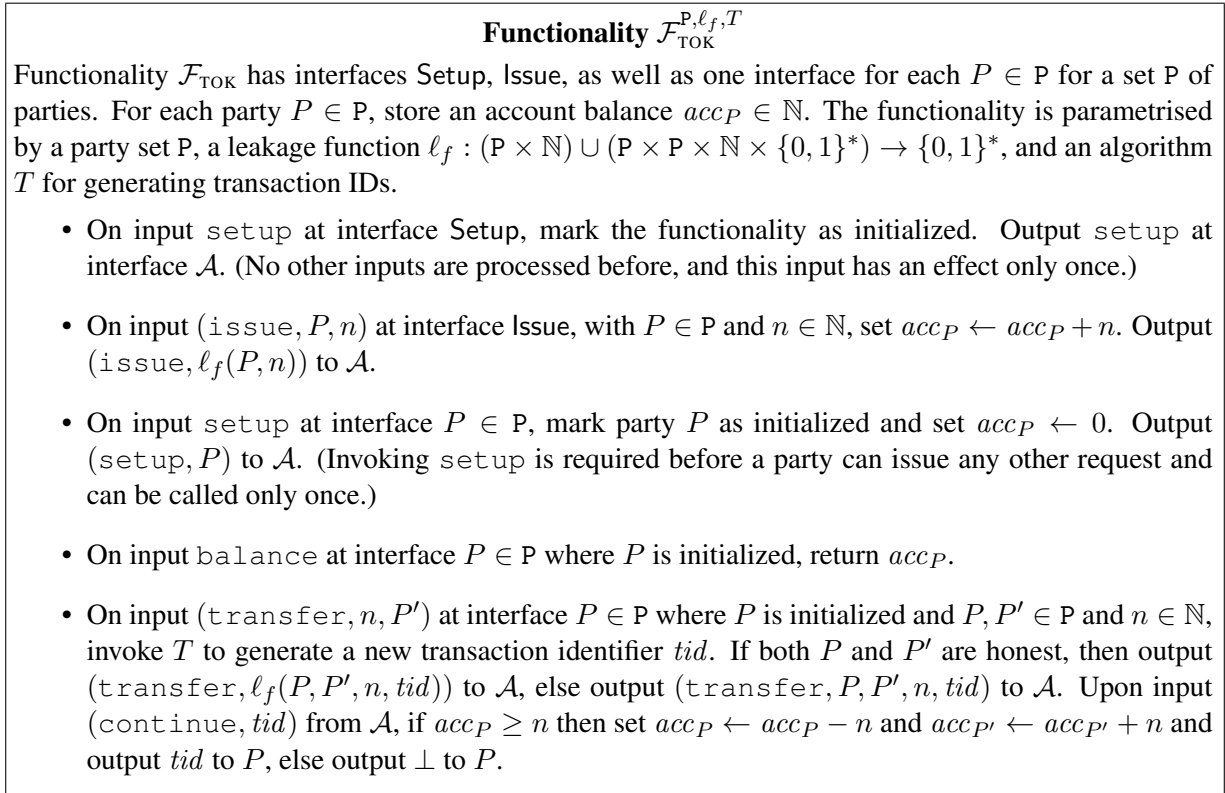


Figure 4.1: The general token functionality \mathcal{F}_{Tok} , parameterised by a party set \mathcal{P} , a leakage function $\ell_f : (\mathcal{P} \times \mathbb{N}) \cup (\mathcal{P} \times \mathcal{P} \times \mathbb{N} \times \{0, 1\}^*) \rightarrow \{0, 1\}^*$, and a stateful algorithm T for generating transaction identifiers.

control the creation of new tokens associated to some party account. This is triggered by the input `(issue, P, n)` with $P \in \mathcal{P}$ and $n \in \mathbb{N}$, and results in n tokens being assigned to party P . As discussed above, the functionality is oblivious of who controls token creation in a given platform, and access to the interface could be fully delegated to a specific party (e.g., in a permissioned DLT), or it could be controlled by another functionality (e.g., in a permissionless DLT).

Each party $P \in \mathcal{P}$ is given access to three functions it can invoke in the functionality. The first, `setup`, must be invoked initially and only once, and models mechanisms such as the creation of a party’s key pair in a token protocol. The second function, `balance`, returns the amount of tokens currently controlled by the party. The implementation of this function depends on the token system, it may be implemented by reading the party’s balance (in a balance-based system) or scanning the ledger for unspent outputs (in a UTXO-based system). Finally, and most importantly, the function `(transfer, tid, n, P')` takes the token amount n that is to be transferred, and a recipient $P' \in \mathcal{P}$. The function generates a transaction identifier tid , whose role will be explained below, via the algorithm T . The value tid is also returned to the invoking party.

The privacy properties are described by means of a leakage function $\ell_f : (\mathcal{P} \times \mathbb{N}) \cup (\mathcal{P} \times \mathcal{P} \times \mathbb{N} \times \{0, 1\}^*) \rightarrow \{0, 1\}^*$. For `issue` requests, ℓ_f describes the leakage on the pair of recipient P and amount n . For requests of the `transfer` type, ℓ_f describes the leakage on sender P and receiver P' , amount n , and transaction identifier tid .

Proof of transfer. We extend the functionality \mathcal{F}_{Tok} according to the description in Figure 4.2. In addition to \mathcal{F}_{Tok} in Figure 4.1, the functionality keeps track of all transactions in a map $TX : \mathcal{P} \times \{0, 1\}^* \rightarrow \mathcal{P} \times \mathbb{N} \times \{0, 1\}$ that is initially empty. The map is indexed by party identifier P of the sender and the transaction identifier tid , and stores the recipient P' and the amount n of the transaction, along with a bit that signifies whether the transaction has been proved.

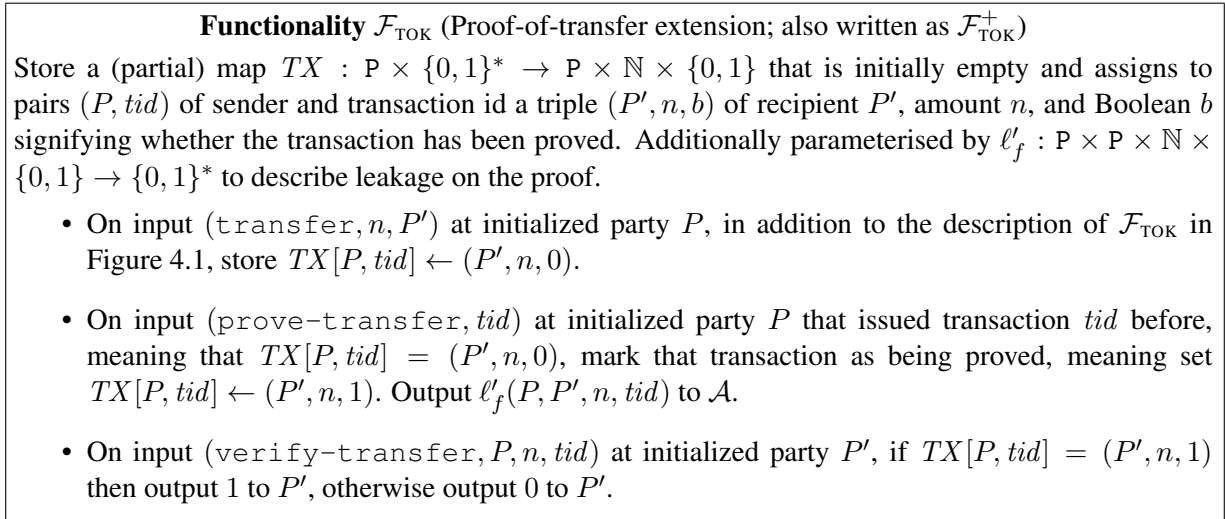


Figure 4.2: Extension of functionality \mathcal{F}_{Tok} that allows a proof-of-transfer between the sender and the receiver of a token transaction. The functionality that includes the descriptions of Figure 4.1 together with the functions described here is denoted as $\mathcal{F}_{\text{Tok}}^+$.

The functionality $\mathcal{F}_{\text{Tok}}^+$ that consists of both the basic functions of \mathcal{F}_{Tok} and the extended ones described in Figure 4.2 models a token system with proof-of-transfer. In the `transfer` function, in addition to the behavior specified in Figure 4.1, stores both the recipient P' and the amount n of the transaction and marks it as unproved. The sender can later decide to call $(\text{prove-transfer}, tid)$ to prove that it indeed sent transaction tid to the recipient. This is recorded in the functionality by updating the state of the transaction; the leakage to the adversary \mathcal{A} is described by the additional leakage function ℓ'_f as in the case of a transfer. Later on, the recipient can verify the transaction details by invoking $(\text{verify-transfer}, P, n, tid)$ with presumed sender P , amount n , and transaction identifier tid .

4.4 Realizing the Functionality with Existing Schemes

In this section we describe, on a high level, how existing token systems realize the described functionality, and with which parameters.

Plain Bitcoin transactions. Bitcoin uses the UTXO transaction model and supports multiple-input-multiple-output (MIMO) transactions [Nak09]. Every transaction output is controlled by a public key. A transaction then contains the *inputs* of the transaction, where each input is a reference to an unused output of some previous transaction together with a signature over the current transaction, which can be validated using the public key controlling that output. The transaction additionally contains the *outputs*, which are essentially described as a list of pairs of amounts and public keys.

The creation of tokens in Bitcoin occurs through the *coinbase* transaction included in each block, which is essentially a transaction without input and with an output whose amount depends on the current block height and whose receiver can be freely chosen by the block creator. Such a transaction is directly modeled through the `issue` function of \mathcal{F}_{Tok} .

Each party is represented by one address (hash of a public key), which corresponds to a party identity $P \in \mathcal{P}$ in our model. Each party locally tracks the *pool* of unspent transaction outputs it can use, and updates it by scanning the ledger. The `balance` function is therefore implemented locally as an algorithm on that pool. The `transfer` function uses unspent outputs from that pool (via a fixed strategy, such as in order of receiving) and builds a MIMO transaction using the selected outputs as inputs. The outputs of the transfer are (1) the intended

amount for the intended receiver, and (2) a residual transaction to the sender itself.

The proof-of-transfer in Bitcoin can be implemented by having the sender prove knowledge of the secret key that belongs to its address. This can be achieved by, e.g., having the sender sign a nonce provided by the receiver, where one needs to ensure that the nonce is encoded in a way that is not a valid Bitcoin transaction.

The leakage functions ℓ_f and ℓ'_f simply encode all the inputs, that is, Bitcoin leaks the entire information of each transaction. As further discussed below in Section 4.5, Bitcoin’s MIMO transactions allow for transaction types that are not supported in $\mathcal{F}_{\text{Tok}}^+$, such as, for instance, atomic-swap transactions or more generally transactions in which the signatures of multiple parties are needed.

Confidential transactions. Confidential transactions as described by Poelstra et al. [PBF⁺17] operate similarly to Bitcoin transactions, with the main difference that the amounts are hidden in cryptographic commitments. The main difference with the above description of Bitcoin is that the leakage functions hide the transaction amounts. As there is no privacy with respect to the identities, meaning that party addresses appear in the clear, the proof-of-transfer can also be performed as in the case of Bitcoin.

Ethereum account-based model and ERC20 tokens. The Ethereum model of accounts that are controlled by secret keys corresponding to addresses (as in Bitcoin) matches more directly with the formalization described here. The mapping between party keys and identities is as in the description of Bitcoin above, and proof-of-transfer as well as the leakage functions are analogous. ERC20 tokens operate, from the perspective of our definitions, analogously to plain Ethereum accounts.

Zerocash transactions. Zerocash transactions, as Bitcoin, are built on a UTXO model. Also as in Bitcoin, each transaction output belongs to an address that is controlled by a secret key. Unlike Bitcoin, or even Confidential Assets [PBF⁺18], the transaction information is hidden completely, that is, neither the party addresses nor the amount is leaked. Consequently, the proof-of-transfer must be performed through an explicit privacy-preserving proof from the spending party to the receiving party.

In a recent draft, Androuraki et al. [ACDE18] describe a security definition that is similar in spirit to the one discussed here. Yet, that definition is specific to the case of permissioned blockchain systems and fixes various parameters according to the proposed scheme, such as the achieved privacy as well as the availability of auditing.

4.5 Planned Extensions

Despite the already discussed “open ends” such as proving that existing schemes realize the described functionality for an appropriate set of parameters, and exemplifying the use of the functionality in a protocol context, several functions achieved by one or more of the token systems. Tokens adhering to the ERC20 standard [Eth18] support delegation of some specified amount of tokens to another party (which can even be a smart contract in Ethereum), which is not represented in the current formalism. More complex transactions, such as Bitcoin’s MIMO transactions in which multiple senders jointly sign the same transaction are also not supported in the current functionality. Finally, *transaction fees* which inherently transfer some part of the amount to the miner creating the current block are not supported by the current version of our model; this modification will be needed to faithfully model existing mining-based cryptocurrencies.

Chapter 5

Timed Signatures and Zero-Knowledge Proofs

Timestamping is an important cryptographic primitive with numerous applications. The availability of a decentralized blockchain such as that offered by the Bitcoin protocol offers new possibilities to realise timestamping services. Nevertheless, to our knowledge, neither the classical timestamping results, nor recent blockchain-based proposals are formally defined and proved in a composable setting.

In this chapter, we put forth a formal treatment of timestamping cryptographic primitives in the UC framework with respect to a global clock—we refer to the corresponding primitives as *timed* to indicate this association. We propose timed versions of primitives commonly used for authenticating information, such as digital signatures, non-interactive zero-knowledge proofs, and signatures of knowledge. Our definitions introduce a fine-grained treatment of the different timestamping guarantees, namely security against *postdating* and *backdating* attacks; our results treat each of these cases separately and in combination. We also define an ideal beacon functionality.

5.1 Introduction

Timestamping allows for a (digital) object—typically a document—to be associated with a creation time (interval), such that anyone seeing the timestamp can verify that the document was not created before or after this time. It has numerous applications from synchronizing asynchronous distributed systems to establishing originality of scientific discoveries and patents. In fact, the idea of timestamping has been implicit in science for centuries, with anagram-based instantiations being traced back to Galileo and Newton. However, the first cryptographic instantiation of timestamping was proposed by Haber and Stornetta [HS91].

A cryptographic timestamping scheme involves a document creator (or client) and a verifier, where the document creator wishes to convince the verifier that a document was at his possession at time T . In typical settings, the aim is to achieve universal verification, where any party can verify the timestamp but one can also consider the simpler designated verifier-set version. Ideally, the protocol aims to protect against both *backdating* and *postdating* of a digital document. To define these two properties, let A be a digital document which was generated at time T . In backdating, an adversary attempts to claim that A was generated at time $T' < T$. In postdating, an adversary is trying to claim that A was generated at time $T' > T$. We note in passing that no existing solution achieves the above perfect form of timestamping. Indeed, this would be feasible only by means of perfect synchrony and zero-delay channels. Instead, timestamping protocols, including those presented in this work, allow to prove backdating and postdating security for a sufficiently small time interval around T .

Haber and Stornetta [HS91] achieve timestamping using a *hash-chain of documents*. In the plain, centralized version of their scheme the parties have access to a semi-trusted (see below) third party, called a *timestamping server* (TS). Whenever a client wishes to sign a document, he sends his ID and (hash of) his document to TS who produces a signed certificate, given the client's request. The certificate includes the current time (according

to TS), the client’s request, a counter, and a hash of the the previous certification which links it to that certificate. The idea is that assuming the TS processes the documents in the time and order they were received, if a document A appears in the hash chain before the hash of a document B, then B must have been generated after A. If someone wants to check the order in which the two documents were generated, he can check the certificate, and assuming that he trusts TS’s credentials, he can derive the order.

The above solution suffers from the TS being a single point of failure. Concretely, the timestamping protocol is only effective if the TS is constantly online and responsive. This opens the possibility of denial-of-service attacks. Furthermore, when used in the context of patents, in order to avoid the need to trust the TS from claiming the patent as its own, one needs to combine it with blinded primitives, such as blind signatures [Cha83]. To circumvent such issues, [HS91] proposed a decentralized version of their scheme, where the clients interactively cooperate with each other to timestamp their documents. The efficiency and participation requirements of that original construction were later improved by Benaloh and de Mare [BdM91].

More recently, [BLT14] proposes a protocol that requires multiple non-colluding servers who interactively time-stamp a document. Although such a level of decentralization eliminates the single-failure point issue, it brings additional complications. Concretely, first, it can only work if the servers are properly synchronized and their communication network is synchronous. Indeed, [BdM91, BLT14] have an implicit round structure where every server/client is always in the same round as all other servers/clients. Second, to avoid attacks by malicious servers that attempt to backdate or postdate a document (e.g., by creating a fork in the hash-chain) it seems necessary to assume that a majority of them are honest and will therefore keep extending the honest chain. Third, the identities and signature certificates of the servers and clients need to be public knowledge, leading to the so called *permissioned* model that often requires mechanisms for registering and deregistering (revoking) parties’ certificates.

We note in passing that the above issues are implicit in the treatment of [HS91, BdM91], and there is no known technique to mitigate them. Moreover, to our knowledge, neither these works nor follow-up works include a formal treatment—security model and definition, and proof of security in this model—that accounts for these issues. We further remark that these issues are similar to the core problem treated by blockchains and their associated cryptocurrencies [Nak, Woo14, KRDO17]. Thus, one could use techniques from such primitives, e.g. relying on proofs of work or space, to develop a timestamping blockchain. In fact, there are existing commercial solutions, e.g., Guardtime¹, that use this idea to offer a blockchain-based timestamping system. However, even recent more rigorous treatments of the problem, e.g. [BS04], fall short in formally casting such timestamping as a service that protocols can connect to in a composable manner. Note that since timestamping is intended as a service, such a composable analysis is necessary to allow black-box use of the services within higher level protocols/applications and enable modular security proofs.

This chapter contributions. We put forth a formal composable treatment of timestamping of cryptographic primitives. Concretely, we devise a formal model of protocol execution for timestamping cryptographic primitives with respect to a global clock that parties have access to. We use the term *timed*, as in *timed (digital) signatures* to distinguish timestamping with respect to such a global clock from the guarantee offered by existing timestamping schemes [HS91, BdM91], which only establishes causality of events—i.e., which of the hash-chained document was processed first—but does not necessarily link it to a global clock. We stress that although, for simplicity our treatment assumes ideal access to a global clock—which is captured as in [BMTZ17] by a global clock functionality, it trivially extends to allow for parties having bounded-drift view of the clock [KMTZ13]—i.e. the adversary is allowed at time t to make a party think that the time is t' which might lie within a distance d from t for a known drift parameter d .

We then define *timed* versions of primitives commonly used for authenticating information, such as digital signatures, non-interactive zero-knowledge proofs [SMP87, BFM88], and signatures of knowledge [CL06] in Canetti’s Universal Composition (UC) framework [Can01]. Our treatment explicitly captures security against *backdating* and *postdating* separately, and investigates the associated assumptions required to achieve each of these security notions. Concretely, the functionalities capturing the timed versions of the above primitives are

¹<https://guardtime.com>

parameterized by a flag $\tau \in \{-, +, \pm\}$ that indicates whether the corresponding primitive achieves security against backdating attacks ($\tau = “-”$), postdating attacks ($\tau = “+”$), or both ($\tau = “\pm”$).

The beacon functionality. In a cryptographic setting, and in the real world, it is sometimes necessary to assume that there exists an unpredictable randomness beacon that generates a new value in every round, with the property that anyone can query it with a round index and receive the value that the beacon output in that round.

Here we investigate how an ideal beacon as above can be weakened so that it is implementable by a protocol which uses the ledger functionality (and a random oracle). In particular, we specify a *weak beacon* functionality, denoted as \mathcal{B}^w . In a nutshell, the beacon functionality is relaxed in the following way: First, the weak beacon is slower, and is only guaranteed to generate a new value every `MaxRound` many rounds, where `MaxRound` is a parameter that depends to the ledger’s liveness parameter².

Second, although the sequence of outputs of the beacon cannot be changed once set, instead of every party being able to learn this sequence at any time, the adversary is allowed to make different parties witness different prefixes of this sequence in any round; this can, however, happen only under the following two restrictions, which are derived from the properties of the ledger specified in [GKL15]³: (1) the lengths of the prefixes seen by different parties do not differ by more than `WindowSize`—again a parameter which depends on the ledger—and (2) the prefixes increase monotonically as the rounds advance (albeit not necessarily at the same rate). Third, and most importantly, the adversary has a limited capability of predicting the beacon’s output. In a nutshell, this predictability will allow the adversary to be able to predict several future outputs, under the restriction that in every t outputs at least one of them could not have been predicted more than k rounds before it was generated by the beacon, where k is a parameter that will depend on the ledger’s transaction liveness parameter.

The relation with cryptographic ledgers. In this document we provide new notions that will be implemented relying on existing distributed ledgers. For this reason we try to keep the description of our primitives as generic as possible in order to use most of the existent cryptographic ledgers in a black-box way. We believe that these notions (and their instantiation) might be useful in the context of the project given that the use cases and part of the toolkits already require the existence of a cryptographic ledger. Therefore, once that a ledger is part of the requirement of a use case or a toolkit, one could use our construction for timed signature and zero-knowledge without any further assumption.

Related work. We have already reviewed the milestones in the timestamping literature and discussed its relation with the notions proposed in this paper. We have also discussed solutions using blockchain technologies, e.g. proofs of work and stake. For completeness we include a more detailed survey of that literature in the last part of this chapter (Sec 5.7); in the same section, we discuss basic results in zero knowledge (including some recent attempts that use time [DNS04, LTCL07, EO94]). To our knowledge none of these works includes a formal treatment among existing timestamping schemes, only the server-aided schemes support a fine-grained and accurate time-stamping, at the cost of fully trusting the servers, as there is only one clock that belongs to the server who is supposed to accurately time-stamp all data it receives. Furthermore, none of the existing blockchain-based solutions obtains timestamping with only ideal (blackbox) access to the ledger nor includes a formal composable proof of the claimed security. There is also literature on so called time-lock encryption and commitments, and time released signatures [RSW96, BN00, GJ02, LJKW18, LGR15]. Despite the similarity in the name, these works do not (aim to) achieve timestamping guarantees.

Organization of chapter. The remainder of this chapter is structured as follows. In Section 5.2 we put forth our execution modeling reviewing relevant aspects of the (G)UC framework. In Section 5.3 we describe our (weak) beacon functionality. Timed signatures are introduced in Section 5.4 and in Section 5.5 we present timed proof of knowledge while in Section 5.6 we generalize this notion to signatures of work. In the last section we provide a more detailed survey about existent works that put in relation time with the notions of ZK and Signature.

²The ledger’s liveness property from [BMTZ17] corresponds to the *chain growth* property from [GKL15].

³Our aim is to implement the weak beacon using the blockchain in a black-box way. This is the reason why we also need to restrict the power of the beacon.

5.2 The Model

Following the recent line of works proving composable security of blockchain ledgers [BMTZ17, BGK⁺18] we provide our protocols and security proofs in Canetti’s universal composition (UC) framework [Can01]. In this section we discuss the main components of our real-world model. We assume that the reader is familiar with simulation-based security and has basic knowledge of the UC framework. We review all the aspects of the execution model that are needed for our protocols and proof, but omit some of the low-level details and refer the more interested reader to these works wherever appropriate. We note that for obtaining a better abstraction of reality, some of our hybrids are described as global (GUC) setups [CDPW07]. The main difference of such setups from standard UC functionalities is that the former are accessible by arbitrary protocols and, therefore, allow the protocols to share their (the setups’) state. The low-level details of the GUC framework—and the extra points which differentiate it from UC—are not necessary for understanding this chapter; we refer the interested reader to [CDPW07] for these details.

Protocol participants are represented as parties—formally Interactive Turing Machine instances (ITIs)—in a multi-party computation. We assume a central adversary \mathcal{A} who corrupts miners and uses them to attack the protocol. The adversary is *adaptive*, i.e., can corrupt (additional) parties at any point and depending on his current view of the protocol execution. Our protocols are synchronous (G)UC protocols [BMTZ17, KMTZ13]: parties have access to a (global) clock setup, denoted by $\mathcal{G}_{\text{clock}}$, and can communicate over a network of authenticated multicast channels. We assume instant and *fetch-based* delivery channels [KMTZ13, CGHZ16]. Such channels, whenever they receive a message from their sender, they record it and deliver it to the receiver upon his request with a “fetch” command. In fact, all functionalities we design in this work will have such fetch-based delivery of their outputs. We remark that the instant-delivery assumption is without loss of generality as the channels are only used for communicating the timestamped object to the verifier which can anyway happen at any point after its creation. However, our treatment trivially applies also to the setting where parties communicate over bounded-delay channels as in [BMTZ17].

We adopt the *dynamic availability* model implicit in [BMTZ17] which was fleshed out in [BGK⁺18]. We next sketch its main components: All functionalities and global setups have a dynamic party set. I.e., they all include special instructions allowing parties to register, deregister, and allowing the adversary to learn the current set of registered parties. Additionally, global setups allow any other setup (or functionality) to register and deregister with them, and they also allow other setups to learn their set of registered parties.

We conclude this section by elaborating Global Clock functionality which is formally showed in Fig. 5.1. The *clock functionality* was initially proposed in [KMTZ13] to enable synchronous execution of UC protocols. Here we adopt its global-setup version, denoted by $\mathcal{G}_{\text{clock}}$, which was proposed by [BMTZ17] and was used in the (G)UC proofs of the ledger’s security.⁴ $\mathcal{G}_{\text{clock}}$ allows parties to ensure that the protocol they are running proceeds in synchronized rounds; it keeps track of a round variable whose value can be retrieved by parties via sending to it the pair: CLOCK-READ. This value is increased when every honest party has sent to the clock a command CLOCK-UPDATE. The parties use the clock as follows. Each party starts every operation by reading the current round from $\mathcal{G}_{\text{clock}}$ via the command CLOCK-READ. Once any party has executed all its instructions for that round it instructs the clock to advance by sending a CLOCK-UPDATE command, and gets in an idle mode where it simply reads the clock time in every activation until the round advances.

5.3 The (Weak) Beacon functionality

Before discussing the timed versions of the cryptographic primitives proposed in this chapter, we describe a functionality that provides a source of sufficiently unpredictable randomness, which we refer to as a *weak (randomness) beacon*. We present this definition taking with the aim of implement it using a transaction ledger. Note that any implementation of an ideal randomness beacon would be expected to satisfy (at least) the following properties:

⁴As a global setup, $\mathcal{G}_{\text{clock}}$ also exists in the ideal world and the ledger connects to it to keep track of rounds.

The functionality is available to all participants. It is parameterized with variable τ , a set of parties \mathcal{P} , and a set \mathcal{F} of functionalities. For each party $p \in \mathcal{P}$ it manages variable d_p . For each $f \in \mathcal{F}$ it manages variable d_f . Initially, $\tau := 0$, $\mathcal{P} := \emptyset$, $\mathcal{F} = \emptyset$

Synchronization

- Upon receiving (CLOCK-UPDATE, sid_C) from some party $p \in \mathcal{P}$, set $d_p := 1$; execute *Round-Update()* and forward (CLOCK-UPDATE, sid_C, p) to \mathcal{A} .
- Upon receiving (CLOCK-UPDATE, sid_C) from some functionalities $f \in \mathcal{F}$ set $d_f := 1$; execute *Round-Update()* and forward (CLOCK-UPDATE, sid_C, f) to \mathcal{A} .
- Upon receiving (CLOCK-READ, sid_C) from any participant (including the environment, the adversary, or any ideal-shared or local-functionality) return (CLOCK-READ, sid_C, τ) to the requester.

Procedure: Round-Update() if $d_f = 1$ for all $f \in \mathcal{F}$ and $d_p = 1$ for all honest $p \in \mathcal{P}$, then set $\tau = \tau + 1$ and reset $d_f := 0$ and $d_p := 0$ for all parties in \mathcal{P} .

Figure 5.1: The Global Clock Setup $\mathcal{G}_{\text{clock}}$

Agreement on the Output: The output of the beacon can be verified by any party who has access to the beacon.

Liveness: The beacon generates new values as time advances. The output of the beacon can be verified (albeit at some point in the future) by any party who has access to the beacon.

Perfect Unpredictability: No one should be able to bias or even predict (any better than guessing) the outcome of the beacon before it has been generated.

However, since our aim is to have an implementation of a weak beacon that relies on the existence of a cryptographic ledger, we cannot obtain such a perfect beacon from the ledgers that are implemented by common cryptocurrencies (cf. also [BGZ16] for an impossibility). Nonetheless, as it turns out, even under a worst-case analysis as in [GKL15, BMTZ17], the contents of the ledger are periodically updated with fresh unpredictable randomness. In the following, we provide a formal definition of a beacon satisfying a weaker notion of liveness and unpredictability.

In a nutshell, our weak beacon generates an unpredictable value η every δ outputs, but we allow the adversary to bias η (in a limited manner) before it is output by the beacon. Concretely, we define our weak beacon as a UC-functionality \mathcal{B}^w in the $\mathcal{G}_{\text{clock}}$ -hybrid model. Note that an ideal beacon functionality is straightforward to define in this model as follows. It maintains a vector H of random values available to anyone upon request, and in each round it appends to this string a new uniformly random value. Before we formally define our weak beacon \mathcal{B}^w , we review the ways in which our weak beacon relaxed the ideal-beacon properties, and the additional capabilities the it offers to the adversary. \mathcal{B}^w is parameterize by a set of parameters $w = ((\mu, \ell), \text{MaxRound}, \text{WindowSize}, \text{bias})$ whose role will become clear as we go over the adversary’s capabilities:

Eventual Agreement on the Output: Similar to the ideal beacon, the functionality maintains an output sequence vector H . However, instead of the parties guaranteed a consistent view of H , the adversary might chose a prefix of H that each party sees, with the restriction that length difference of the prefixes seen by any two parties in any round is upper bounded by a parameter `WindowSize`.

More precisely, each party p_i can see only the first pt_i elements of H , where pt_i is adversarially chosen in each round, with the restriction that the adversary is that $|H| - \text{pt}_i \leq \text{WindowSize}$ for all p_i registered to \mathcal{B}^w . In our weak beacon functionality this restriction will be enforced by means of a checking procedure, denotes as `check_time_table`, which will be executed whenever the adversary attempts to rewrite indexes; if the check fails then another procedure, `force_time_table`, is invoked which overwrites the adversary’s choices

with values of pt_i that adhere to the above policy.

Slow Liveness: \mathcal{B}^w does not necessarily generate a new value in every round. Instead, the adversary can delay the generation of a new value but only by at most `MaxRound` rounds.

Weak Unpredictability: The adversary has the following influence on the output of the beacon:

- The adversary can bias some of the outputs of the beacon. More precisely, assume that \mathcal{B}^w is about to choose its i -th value to be appended to its output vector H . The adversary is given a set \mathcal{S}_i of random values (where $|\mathcal{S}_i| = \text{bias} \in \mathbb{N}$) and a choice: he can either allow the beacon to randomly choose the i -th output (in this case this output is considered honest), or he can decide on a value $\eta_i \in \mathcal{S}_i$ to append to the output vector. However, the restriction is that within every window of ℓ outputs, at least μ of them will be honest (looking ahead, this restriction will be enforced by means of a procedure `check_validity` described in the following.)
- The adversary can predict, in the worst case, the next $\ell - \mu$ outputs of the beacon. More precisely, let n be the size of H ; the adversary can ask \mathcal{B}^w to see $\ell - \mu$ sets $\mathcal{S}_{n+1}, \dots, \mathcal{S}_{n+\ell-\mu}$ from which the next $\ell - \mu$ outputs will be chosen. In terms of rounds, this means that at any point the adversary might *predict* the output of a beacon for up to the next $\Delta = (\ell - \mu + \text{WindowSize}) \cdot \text{MaxRound}$ rounds.

In the following we elaborate on the exact power that each of the above properties yields to the adversary. For capturing eventual agreement on the output and slow liveness, we introduce the notion of a *time table* T . It is a table with one column for each party that has ever been seen or registered with the beacon, indexed by the ID of the corresponding party (recall that we allow parties to register and deregister), and one row for each (clock) round. The table is extended in both dimensions as new parties register and as the time advances. For a party p_i and (clock-)round τ , the entry $T[\tau, p_i]$ is an integer `tsl` that we call *time-slot index*. This value `tsl` defines the size of the prefix of the beacon’s output H that p_i can see at round τ . That is, p_i at round τ can request any of the first `tsl` outputs of \mathcal{B}^w , denoted by $H[1], \dots, H[\text{tsl}]$.

The adversary is allowed to instruct \mathcal{B}^w as to how T should be populated under the following restrictions: (1) for any party the values of its column, i.e., its time-slot indices, are monotonically non-decreasing and they are increasing by at least once in every `MaxRound` rounds (this will enforce slow liveness), and (2) in any given round/row, no two time-slot indices (of two different parties) can be more than `WindowSize` far apart (this together will enforce the eventual agreement property). These properties are formally enforced by two procedures, called `force_time_table` and `check_time_table` that check if the adversary complies with the above policy as follows: The procedure `check_time_table` takes as input the current time table T , a new table T' proposed by the adversary, the set of parties n registered to \mathcal{B}^w , the current round R , $\max_{\text{tsl}} = |H|$; it outputs 0 if T' is invalid, and 1 otherwise. The procedure `force_time_table` is invoked to enforce the policy mandated by `check_time_table` in case the adversary is caught trying to violate it. In a nutshell, it generates a valid and randomly generated time table T' to be adopted instead of the adversary’s proposal. More concretely, `force_time_table` is invoked in the following two cases: 1) If H has not been extended in the last `MaxRound` rounds. In this case \mathcal{B}^w generates a random output, appends it to H and extends T using `force_time_table`. 2) If the adversary has not updated T in the last round, then a new T (that extends the previous one) is generated via `force_time_table`.

The above two procedures are formally described in Fig. 5.2 and are, in fact, useful also for the definition of our other (timed) UC-functionalities described in the following section. The trickiest of the above properties to capture (and enforce in the functionality) is weak unpredictability.

The idea is the following. Assume that the beacon has already generated outputs $\eta_1, \dots, \eta_{i-1}$, where η_{i-1} was generated in round τ . Recall that, per the slow liveness property, the beacon does not generate outputs in every round. In every round after τ , the adversary is given a sequence of $\ell - \mu$ *output candidate sets* $\mathcal{S}_i, \dots, \mathcal{S}_{i+\ell-\mu}$ sampled by \mathcal{B}^w and can do one of the following:

- (1) decide to set the i -th beacon’s output to a value from \mathcal{S}_i of his choice. In this case, η_i is set to this value and flagged as dishonest (this is formally done by setting a flag $\text{hflag}_i \leftarrow 0$ and storing the pair (η_i, hflag_i)); the adversary is also given a next set $\mathcal{S}_{i+\ell-\mu+1}$ of size `bias` sampled by the beacon by choosing `bias`-many random values from $\{0, 1\}^\lambda$ (cf. procedure `define_new_set(bias)` in Figure 5.2). Looking ahead in our beacon protocol, $\{0, 1\}^\lambda$ will correspond to the bits of entropy that are guaranteed to be included in an


```

check_time_table( $T, T', n, R, \max_{\text{tsl}}$ )
  If all the following conditions are satisfied
    1.  $T$  is a prefix of  $T'$ ;
    2. For each corrupted party  $p_i$   $T'[R, p_i] = \max_{\text{tsl}}$ ;
    3. For all  $j \in \{0, \dots, R-1\}$ ,  $T'[j+1, p_i] \geq T'[j, p_i]$ ;
    4. For all  $l \in \{0, \dots, R\}$ , for all  $p_i, p_j \in \mathcal{P}$ ,  $|T'[l, p_j] - T'[l, p_i]| \leq \text{WindowSize}$ 
  then return 1, else return 0.

force_time_table( $T, n, R, \max_{\text{tsl}}$ )
  Generate a random  $T'$  such that  $\text{check\_time\_table}(T, T', n, R, \max_{\text{tsl}}) = 1$ 
  Return  $T'$ .

define_new_set(bias)
  Define  $\mathcal{S} = \emptyset$ . For  $i = 1$  to bias pick  $\eta_i \leftarrow_{\$} \{0, 1\}^\lambda$  and add  $\eta_i$  to  $\mathcal{S}$ . Return  $\mathcal{S}$ .

check_validity( $j$ )
   $c \leftarrow 0$ 
  For  $i = j - \ell, \dots, j$ 
    Parses  $H[i]$  as  $(\eta_i, \text{hflag}_i)$ 
    If  $\text{hflag} = 1$  then set  $c \leftarrow c + 1$ 
  if  $c \geq \mu$  then return 1, else return 0.

force_liveness( $\max_{\text{tsl}}, T, H$ )
   $\max_{\text{tsl}} \leftarrow \max_{\text{tsl}} + 1$ ,
   $T \leftarrow \text{force\_time\_table}(T, n, R, \max_{\text{tsl}})$ ,
   $\eta \leftarrow_{\$} \{0, 1\}^\lambda$ ,
   $H[\max_{\text{tsl}}] \leftarrow (\eta, 1)$ .
  return  $(\max_{\text{tsl}}, T, H)$ 

check_liveness( $\tau_{\text{last}}, R$ )
  if  $\tau_{\text{last}} + \text{MaxRound} = R$  then return 0 else return 1

```

Figure 5.2: Auxiliary Procedures.

honestly generated ledger block (e.g., the size of the random nonces that miners use for their hash puzzles). $\mathcal{S}_{i+\ell-\mu+1}$ will be the output candidate set for the $(i + \ell - \mu + 1)$ -th beacon output.

- (2) instruct the beacon to ignore \mathcal{S}_i and instead choose a uniformly random value for η_i . In this case, the beacon marks the i -th output as honest, i.e., sets $\text{hflag}_i := 1$, informs the adversary about η_i , disposes of all existing output candidates sets, samples $\ell - \mu$ fresh candidates sets $\mathcal{S}_{i+1}, \dots, \mathcal{S}_{i+\ell-\mu+1}$ and hands them to the adversary.
- (3) instruct the beacon to not include any new output in the current round.

The choice (1) above captures the fact that the adversary can predict the next $\ell - \mu$ outputs of the beacon. However, to ensure that the above weakened unpredictability is meaningful, does not mess with liveness, and also achieves a guarantee similar to the chain quality property—i.e. that a truly random (honest) output is generated in sufficiently small intervals—the beacon enforces a policy on the adversary which ensures that the adversary’s choices abide to the following restrictions: (A) any sequence of ℓ outputs of the beacon contains (at least) μ honest outputs, generated (randomly) by \mathcal{B}^w , and (B) the adversary can leave the beacon without an output for at most MaxRound sequential rounds. Condition A is checked by the procedure `check_validity` whenever the adversary attempts to propose a new output from the corresponding candidate set, by taking choice (1) above; if the check fails the proposal of the adversary is ignored. Condition B is checked by procedure `force_liveness(\max_{tsl}, T, H)`; if it fails, i.e., the adversary tries to delay the beacon’s update by more than MaxRound rounds, then procedure `force_liveness(\max_{tsl}, T, H)` is invoked which forces the above policy in a default manner. The helper procedures and the formal description of our weak beacon functionality are included in Figures 5.2 and 5.3, respectively.

The functionality is parameterized by the algorithms `check_time_table`($T, T', n, R, \max_{\text{tsl}}$), `define_new_set`(`bias`), `force_time_table`($T, n, R, \max_{\text{tsl}}$), `check_validity`(`tsl`) along with the parameters `bias`, `MaxRound`, (ℓ, μ) , $\tau_{\text{last}} \leftarrow 0$, $\max_{\text{tsl}} \leftarrow 0$, the time table T , a set of parties \mathcal{P} and adversary \mathcal{A} . $T[0, p_i] = 0$ for all $p_i \in \mathcal{P}$. We assume the functionality to be registered to $\mathcal{G}_{\text{clock}}$. The functionality manages the output vector H that records the random values issued by the functionality. H is initially empty. The functionality is also initialized with the sets $\mathcal{S}_1 \leftarrow \text{define_new_set}(\text{bias}), \dots, \mathcal{S}_{\ell-\mu} \leftarrow \text{define_new_set}(\text{bias})$ and $p \leftarrow 0$. Let R be the response obtained by querying $\mathcal{G}_{\text{clock}}$, upon receiving any input I from any party or from the adversary act as follows:

- If `check_liveness`(τ_{last}, R) = 0 then $(\max_{\text{tsl}}, T, H) = \text{force_liveness}(\max_{\text{tsl}}, T, H)$
- If `check_time_table`($T, T', n, R, \max_{\text{tsl}}$) = 0 then $T \leftarrow \text{force_time_table}(T, n, R, \max_{\text{tsl}})$.

Fetch

- If $I = (\text{FETCH}, \tau_{\text{req}}, \text{sid})$ is received from party p_i or from \mathcal{A} (on behalf of a corrupted party p_i) check if $\tau_{\text{req}} \leq R$. If it is not then ignore I ; otherwise, do the following:
 - $\text{tsl} \leftarrow T[\tau_{\text{req}}, p_i]$;
 - parses $H[\text{tsl}]$ as (η, hflag) ;
 - returns $(\text{FETCH}, \text{sid}, \text{tsl}, \eta)$ to p_i .

Sampling

- If $I = (\text{READ_SETS}, \text{sid})$ is received from \mathcal{A} , then send $(\text{READ_SETS}, \text{sid}, \mathcal{S}_1, \dots, \mathcal{S}_{\ell-\mu})$ to \mathcal{A} .
- If $I = (\text{SET}, \text{sid}, \eta)$ is received from \mathcal{A} , then check if $H[\max_{\text{tsl}} + 1]$ has not been set yet, `check_validity`(\max_{tsl}) = 1 and $\eta \in \mathcal{S}_p$. If it is not, then ignore I ; otherwise, do the following: $\tau_{\text{last}} \leftarrow R$, $\max_{\text{tsl}} \leftarrow \max_{\text{tsl}} + 1$, $H[\max_{\text{tsl}}] \leftarrow (\eta, 0)$, $p \leftarrow p + 1$ and send $(\text{ok}, \text{sid}, \eta)$ to \mathcal{A} .
- If $I = (\text{SET_RANDOM}, \text{sid})$ is received from \mathcal{A} , then check if $H[\max_{\text{tsl}} + 1]$ has not been set yet. If it is not, then ignore I ; otherwise, do the following:
 - $\tau_{\text{last}} \leftarrow R$, $\max_{\text{tsl}} \leftarrow \max_{\text{tsl}} + 1$, $\eta \leftarrow_s \{0, 1\}^\lambda$, $H[\max_{\text{tsl}}] \leftarrow (\eta, 1)$, $p \leftarrow 0$.
 - For $i = 1, \dots, \ell - \mu$
 - $\mathcal{S}_i \leftarrow \text{define_new_set}(\text{bias})$
 - send $(\text{ok}, \text{sid}, \eta)$ to \mathcal{A} .

Set delays

If $I = (\text{SET-DELAYS}, \text{sid}, T')$ is received from \mathcal{A} , then set $b \leftarrow \text{check_time_table}(T, T', n, R, \max_{\text{tsl}})$. If $b = 1$, then set $T = T'$; ignore I otherwise.

Figure 5.3: The \mathcal{B}^w functionality.

5.4 Timed Signatures (TSign)

In this section, we extend the standard notion of digital signature by different levels of timing guarantees. For self-containment we have included the standard signatures functionality proposed by Canetti [Can03] in Fig. 5.4.

In our model, a timestamped signature σ for a message m is equipped with a time mark τ that contains information about when σ was computed by the signer. We refer to this special notion of signature for a time mark τ that is associated with the global clock $\mathcal{G}_{\text{clock}}$ as *Timed Signature (TSign)*. We define three categories of security for TSign: *backdate*, *postdate* security, and their combination which we refer to just as *timed security*. Intuitively, backdate security guarantees that the signature σ time-marked with τ has been computed some time *before* τ ; postdate security guarantees that the signature σ was computed some time *after* τ ; and timed security provides to the party that verifies the signature σ a time interval around τ in which σ was computed. We formally define these three new security notions by means of a single UC-functionality $\mathcal{F}_\sigma^{w, \tau}$ (see Figures 5.5 and 5.6 for a

Key Generation.

Upon receiving a value $(\text{keygen}, \text{sid})$ from some party $S \in n$, verify that $\text{sid} = (S, \text{sid}')$ for some sid' . If not, then ignore the request. Else, hand $(\text{keygen}, \text{sid})$ to the \mathcal{A} . Upon receiving $(\text{VERIFICATION_KEY}, \text{sid}, v)$ from \mathcal{A} , output $(\text{VERIFICATION_KEY}, \text{sid}, v)$ to S , and record the pair (S, v) .

Signature. If $I = (\text{sign}, \text{sid}, m)$ is received from party S , verify that $\text{sid} = (S, \text{sid}')$ for some sid' . If not, then ignore the request, else send $(\text{sign}, \text{sid}, m, \sigma)$ to \mathcal{A} . Upon receiving $I = (\text{SIGNATURE}, \text{sid}, m, \sigma)$ from \mathcal{A} , verify that no entry $(m, \sigma, v, 0)$ is stored. If it is, then output an error message to S and halt. Else, send $(\text{SIGNATURE}, \text{sid}, m, \sigma)$ to S , and store the entry $(m, \sigma, v, 1)$.

Verification Upon receiving a value $(\text{verify}, \text{sid}, m, \sigma, v')$ from some party P_i , hand $(\text{verify}, \text{sid}, m, \sigma, v')$ to the adversary. Upon receiving $(\text{VERIFIED}, \text{sid}, m, \phi)$ from the adversary do:

1. If $v' = v$ and the entry $(m, \sigma, v, 1)$ is recorded, then set $f = 1$. (This condition guarantees completeness: If the verification key v' is the registered one and σ is a legitimately generated signature for m , then the verification succeeds.)
2. Else, if $v' = v$, the signer is not corrupted, and no entry $(m, \sigma, v, 1)$ for any σ' is recorded, then set $f = 0$ and record the entry $(m, \sigma, v, 0)$. (This condition guarantees unforgeability: If v' is the registered one, the signer is not corrupted, and never signed m , then the verification fails.)
3. Else, if there is an entry (m, σ, v', f') stored, then let $f = f'$. (This condition guarantees consistency: All verification requests with identical parameters will result in the same answer.)
4. Else, let $f = \phi$ and record the entry (m, σ, v', ϕ)

Send $(\text{VERIFIED}, \text{sid}, m, f)$ to P_i .

Figure 5.4: The $\mathcal{F}^{\text{SIGN}}$ functionality of [Can03]

detailed description.) $\mathcal{F}_\sigma^{w, \tau}$ is parameterized by a flag $\tau \in \{+, -, \pm\}$ where $\tau = "-"$ indicates that the functionality guarantees backdate security, $\tau = "+"$ indicates postdate security, and $\tau = "\pm"$ indicates timed security. Analogously to the weak beacon, $\mathcal{F}_\sigma^{w, \tau}$ and all parties that have access to this functionality, are registered to $\mathcal{G}_{\text{clock}}$ which provides the notion of time inherently required by our model. For most generality, we parameterize $\mathcal{F}_\sigma^{w, \tau}$ with $w = (\delta, \text{MaxRound}, \text{WindowSize}, \text{waitingTime})$, where the meaning of these parameters is discussed below.

In a nutshell, the functionality $\mathcal{F}_\sigma^{w, \tau}$ provides to its registered parties a new time-slot $\text{tsl} \in \mathbb{N}$ every MaxRound rounds (in the worst case). The exact moment in which each such time slot is issued is decided by the adversary \mathcal{A} via the input $(\text{NEW_SLOT}, \text{sid})$. Once a time slot tsl is issued, it can be used to time(stamp) a signature σ . The meaning of tsl depends on the notion of security that we are considering. For backdate security (i.e., $\tau = "-"$), a signature σ marked with tsl denotes that σ was computed during a time slot $\text{tsl}' \leq \text{tsl}$. For postdate security ($\tau = "+"$) tsl denotes that σ was computed during a time slot $\text{tsl}' \geq \text{tsl}$. For timed security, the signature σ is equipped with two time-marks tsl_{back} and tsl_{post} that denote that σ was computed in a time-slot tsl' such that $\text{tsl}_{\text{post}} \leq \text{tsl}' \leq \text{tsl}_{\text{back}}$.

A new time-slot issued by $\mathcal{F}_\sigma^{w, \tau}$ can be immediately seen and used by \mathcal{A} . However, \mathcal{A} can delay honest parties from seeing new time-slots—i.e., truncate the view that each honest party has of the available time-slots.

That is, for each party P_i , \mathcal{A} can decide to *hide* the most recent `WindowSize`-many available time-slots. This means that, for example, in any round R the party p_1 could see (and use) the most recent time-slot `tsl`, whereas p_2 's view might have `tsl - WindowSize` as the most recent time-slot.

To keep track of the association between rounds and time-slots, $\mathcal{F}_\sigma^{w,t}$ manages a time table T in the same way as \mathcal{B}^w . That is, an entry $T[\tau, P_i]$ is an integer `tslPi`, where P_i represents a party registered to $\mathcal{F}_\sigma^{w,t}$ and τ represents a round number. The value `tslPi` defines the view that the party P_i has of the available time-slots in round τ . In particular, at round τ party P_i can access and use the time slots $1, \dots, \text{tsl}_{P_i}$. The time table T is controlled by \mathcal{A} but it is limited to change the content of T according to the parameter `WindowSize` as we discussed above. More formally, $\mathcal{F}_\sigma^{w,t}$ checks that the changes made by \mathcal{A} to T are valid using the procedures `check_time_table` and `force_time_table` (the same ones that were used by our weak beacon, see Figure 5.2.)

Note that the way to obtain postdate security is by relying on the unpredictability of the beacon. However, this creates the following subtlety. As the adversary is able to predict future values of our (weak) beacon he can attempt to postdate signatures as far in the future as his prediction reaches. To capture this behaviour, our functionality is parameterized by a value $\delta \in \mathbb{N}$, which we call the *prediction parameter*. This parameter is only relevant when $t \in \{+, \pm\}$. With this parameter we allow the adversary to use, before of any honest party, δ new time-slots. This means that, for the case of postdate and timed security, an adversary can compute a signature σ marked with a time slot `maxtsl + δ` , where `maxtsl` denotes the most recent time-slot. However this creates a new issue, this time with the security proof: when the simulator receives from its adversary a signature timed with a presumably predicted beacon value, he (the simulator) cannot be sure whether or not the adversary will indeed instruct the beacon to output this value when its time comes. To resolve that, the functionality allows its simulator/adversary to withdraw signatures which refer to a *future* time slot `tsl > maxtsl` via the command `(DELETE, sid, ·)`.

For analogous reasons as above, we also introduce a parameter `waitingTime`, which is relevant when $t \in \{-, \pm\}$ and allows for the following adversarial interference: Whenever an honest party wants to time-mark a signature, \mathcal{A} can decide to delay the marking operation until that `waitingTime` time-slots have been issued by $\mathcal{F}_\sigma^{w,t}$. This means that an honest party that requests to time-mark σ in round R has to wait, in the worst case, `waitingTime · MaxRound` rounds in order to see σ time-marked. To guarantee that a new time-slot is available every `MaxRound` (at least) rounds, any time that an input is received the functionality checks that a new time-slot has been issued using the procedure `check_liveness` following exactly the same approach of \mathcal{B}^w (see. Sewc 5.3 for more details on how the liveness is enforced).

Finally, to keep the already complicated description of the functionality as compact as possible, $\mathcal{F}_\sigma^{w,t}$ is also equipped with procedure `standard_verification` (see Figure 5.7) that abstracts the checks for completeness, unforgeability and consistency performed by Canetti's standard signature verification phase in $\mathcal{F}^{\text{SIGN}}$. Figure 5.4).

5.5 Timed Zero-Knowledge Proof of Knowledge (TPoK)

5.5.1 The NIZK functionality

In this section we apply the same methodology used for timed signatures in the previous section to defined analogously timed versions of non-interactive zero-knowledge proofs of knowledge. The basis for our approach is the standard UC Non-Interactive Zero-Knowledge functionality proposed in [GOS12].⁵ In order to use the functionality from [GOS12] as basis for our fetch-based delivery (cf. Section 5.2) timed NIZK functionality, we first turn it into a fetch-based delivery version: instead of waiting for the adversary to deliver the proof to the honest verifier, we allow the verifier to request for the proof, a request which is answered directly if the adversary has allowed the proof to be generated. This is a simple syntactic modification of the functionality from [GOS12]; for self-containment we have included the resulting functionality, denoted by $\mathcal{F}_{\text{NIZK}}$, in Fig. 5.8.

⁵Although [GOS12] focuses on constructing UC perfect NIZK argument of knowledge the corresponding functionality is the same for UC NIZK proof of knowledge.

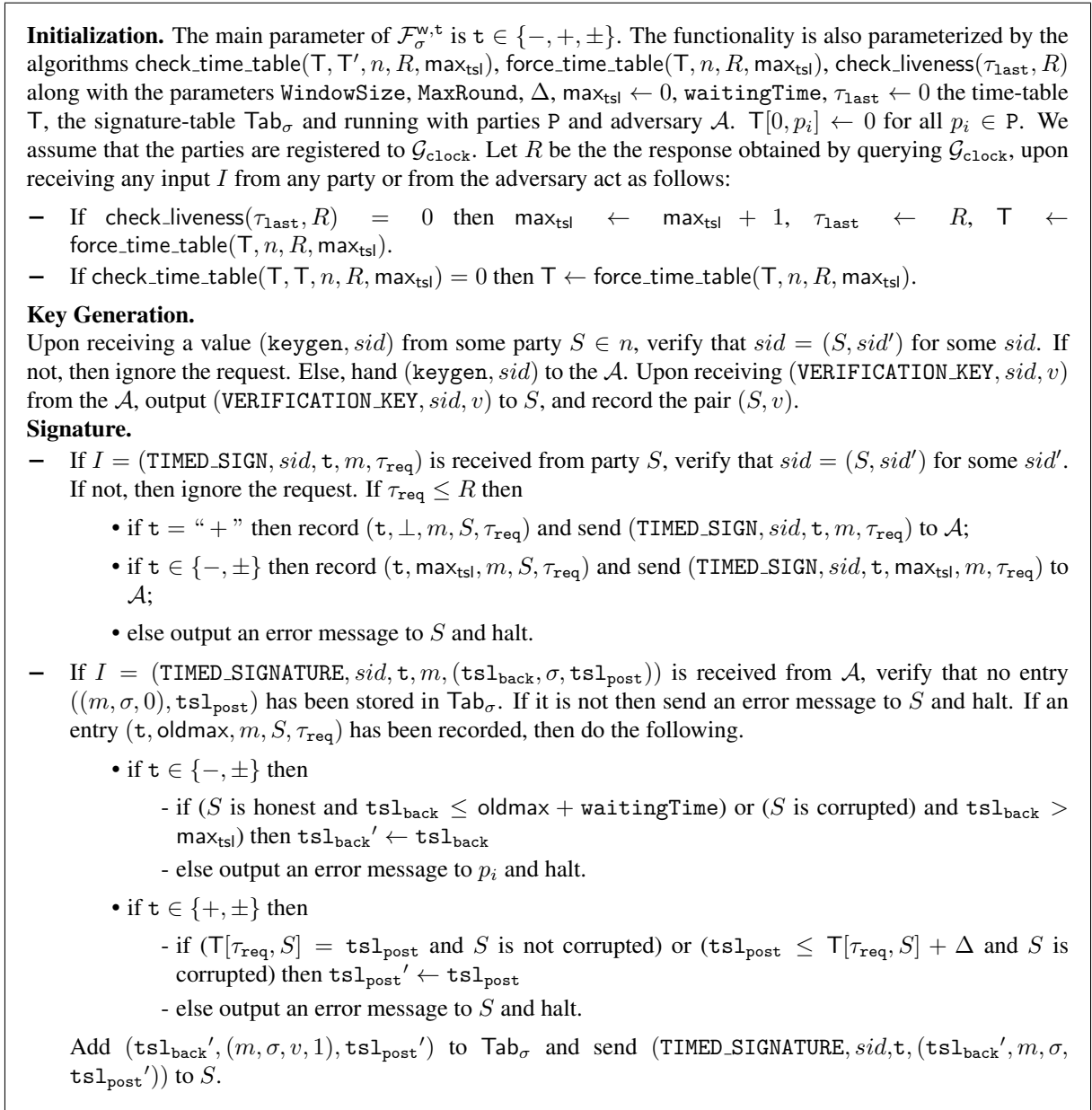
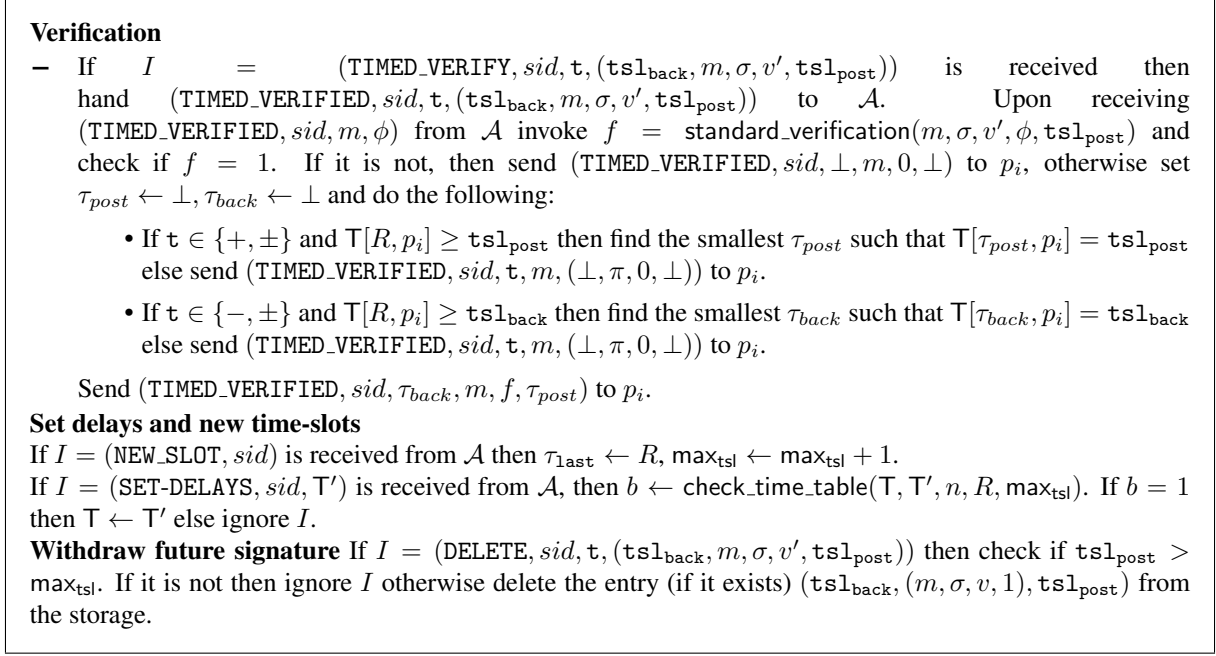
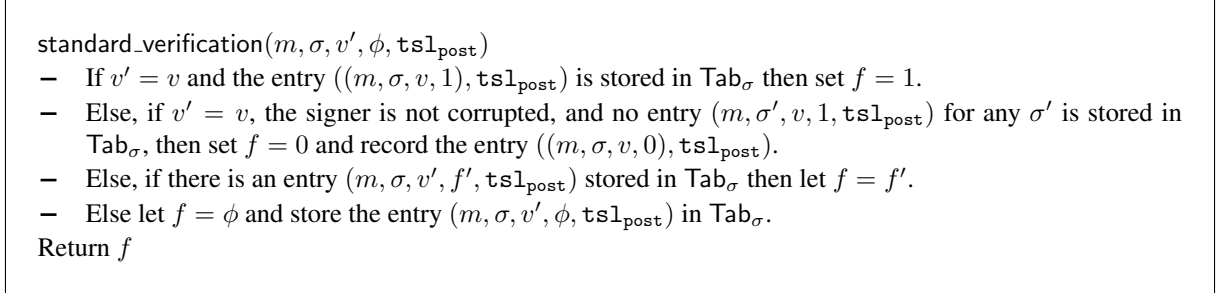


Figure 5.5: The $\mathcal{F}_\sigma^{w,t}$ functionality.

Here is how we extend $\mathcal{F}_{\text{NIZK}}$ to consider different levels of timed-security, in the same way we have done for signatures: A proof π generated with respect to an \mathcal{NP} -statement is equipped with a time-mark tsl that gives some information about when π was computed. We refer to this notion of NIZK as TPoK and consider three categories of security: *backdate*, *postdate* and *timed security*. The formalization of these notions is given by means of the UC-functionalities $\mathcal{F}_{\text{TPoK}}^{w,t}$, with $t = "-", "+, \pm"$. $\mathcal{F}_{\text{TPoK}}^{w,t}$ in the Figures 5.9 and 5.10 (those definitions follow almost exactly the structure of the functionality $\mathcal{F}_\sigma^{w,t}$, but for completeness we report them anyway).

In this, intuitively, a prover P that generates an accepting proof π equipped with a time-mark tsl gives to the verifier V the guarantee that: 1) he knew the witness for the \mathcal{NP} -statement that he is proving; 2) the proof π was generated (using the witness) in some moment specified by tsl . We also provide three instantiations, one for each of the three security notions mentioned above. That is, we show a protocol $\Pi_{\text{TPoK}}^{w,t}$ that UC-realize


Figure 5.6: The $\mathcal{F}_{\sigma}^{\text{w},t}$ functionality (cont'd).

Figure 5.7: The Procedure $\text{standard_verification}$

$\mathcal{F}_{\text{TPoK}}^{\text{w},t}$ for all $t \in \{-, +, \pm\}$.

5.6 Timed Signatures of Knowledge

In Crypto 2006 Chase et al. [CL06] introduced the notion of *signature of knowledge (SoK)*. A signature of knowledge scheme allows to issue signatures on behalf of any \mathcal{NP} -statement. That is, receiving a valid signature for a message m with respect to an \mathcal{NP} -statement x means that the signer of m knew the witness w for the \mathcal{NP} -statement x .

In [CL06] the authors propose a UC-definition of signature of knowledge and provide a construction for it. Moreover, they propose a game-based definition of signature of knowledge and show that the UC and the game-based definitions are indeed equivalent. Chase et al. also provide a construction for a signature of knowledge scheme assuming NIZK and encryption scheme⁶. In this part of the chapter we extend the definition of signature of knowledge by requiring the signatures to be time marked. That is, a signature for a message m with respect to an \mathcal{NP} -statement x has the form (σ, m, tsl) where tsl gives an indication about when the σ was actually

⁶The construction of [CL06] requires the NIZK to be simulation sound and the encryption scheme to be *dense*.

The functionality is parameterized by an \mathcal{NP} -relation \mathcal{R} and running with parties $\mathsf{P} = \{p_1, \dots, p_n\}$ and adversary \mathcal{A} .

Proof

- On input $I = (\text{PROVE}, \text{sid}, (x, w))$ from party $P_i \in \mathsf{P}$, ignore I if $(x, w) \notin \mathcal{R}$, record (x, P_i) and send (PROVE, x) to \mathcal{A} otherwise.
- On input $I = (\text{PROOF}, \pi, (x, P_i))$ from \mathcal{A} , if (x, P_i) is recorded then store (x, π) and send $(\text{PROOF}, \text{sid}, \pi)$ to P_i , ignore I otherwise.

Verification

- On input $I = (\text{verify}, \text{sid}, x, \pi)$ is received from a party $P_i \in \mathsf{P}$ check whether (x, π) is stored. If not then record (x, π, P_i) and send (verify, x, π) to \mathcal{A} , otherwise return $(\text{VERIFICATION}, \text{sid}, 1)$ to P_i .
- On input $I = (\text{WITNESS}, w, (x, \pi, P_i))$ from \mathcal{A} , (x, π, P_i) is recorded and if $(x, w) \in \mathcal{R}$ then store (x, π) and return $(\text{VERIFICATION}, \text{sid}, 1)$ to P_i , return $(\text{VERIFICATION}, \text{sid}, 0)$ to P_i otherwise.

Figure 5.8: The NIZK functionality $\mathcal{F}_{\text{NIZK}}$

computed (using the witness for x). Exactly in the same spirit of signature and NIZK, in this section define the notions of backdate, postdate and timed SoK. That is, we define the UC-functionalities $\mathcal{F}_{\text{TSOK}}^{\mathbf{w}, \mathbf{t}}$ with $\mathbf{t} \in \{-, +, \pm\}$. Those functionalities are analogous to the functionalities for timed signatures and NIZKs proposed in Sections 5.5 (see Figures 5.11 and 5.12).

5.7 A Survey of Related Works on Timed ZK and Signature

Time-stamping digital documents. The idea of time-stamping a digital document was introduced in [HS91]. They provide two schemes, centralised and decentralised based on hash-chain and digital signatures. The former scheme includes a centralised time-stamping server and a set of clients (each of them given a unique ID). Each time a client wants to sign a document, it sends its ID and hash of its document to the time-stamping server who produce a signed certificate, given the clients request, where the certificate includes time, client's request, a counter, as well as information that links it to a certificate issued for the previous client. In this scheme, in order for a party to verify the correctness of the time-stamp included in the certificate of client i^{th} , it has to ask client $i - 1^{\text{th}}$ to provide its certificate. However, this scheme is susceptible to forward-date attack by mounting a Sybil attack. In particular, the server can create a set of IDs and a set of dummy documents. Then, it computes certificates for the dummy documents. This allows it to forward-date the next document given by a legitimate client. Furthermore, for the scheme to support a fine-grained time-stamping mechanism, a sufficiently high number of requests has to be sent frequently to the server. The second scheme, i.e. decentralised one, utilises a pseudorandom generator too. In this scheme, there is no time-stamping server and clients interactively cooperate with each other to time-stamp a document. As explained in [DMD04], an issue with this scheme is that many participants must engage in the process in order to time-stamp an individual document; however, in the absence of enough incentive they may participate that leads to denial of service. A collection of schemes, proposed in [BdM91], improve the efficiency of [HS91]. The schemes allow clients to interactively time-stamp their documents in rounds, such that in every round all documents and previous round time-stamp are combined together and a hash value of the combination is produced. To improve the scheme efficiency, in every round a Merkle tree is built on top

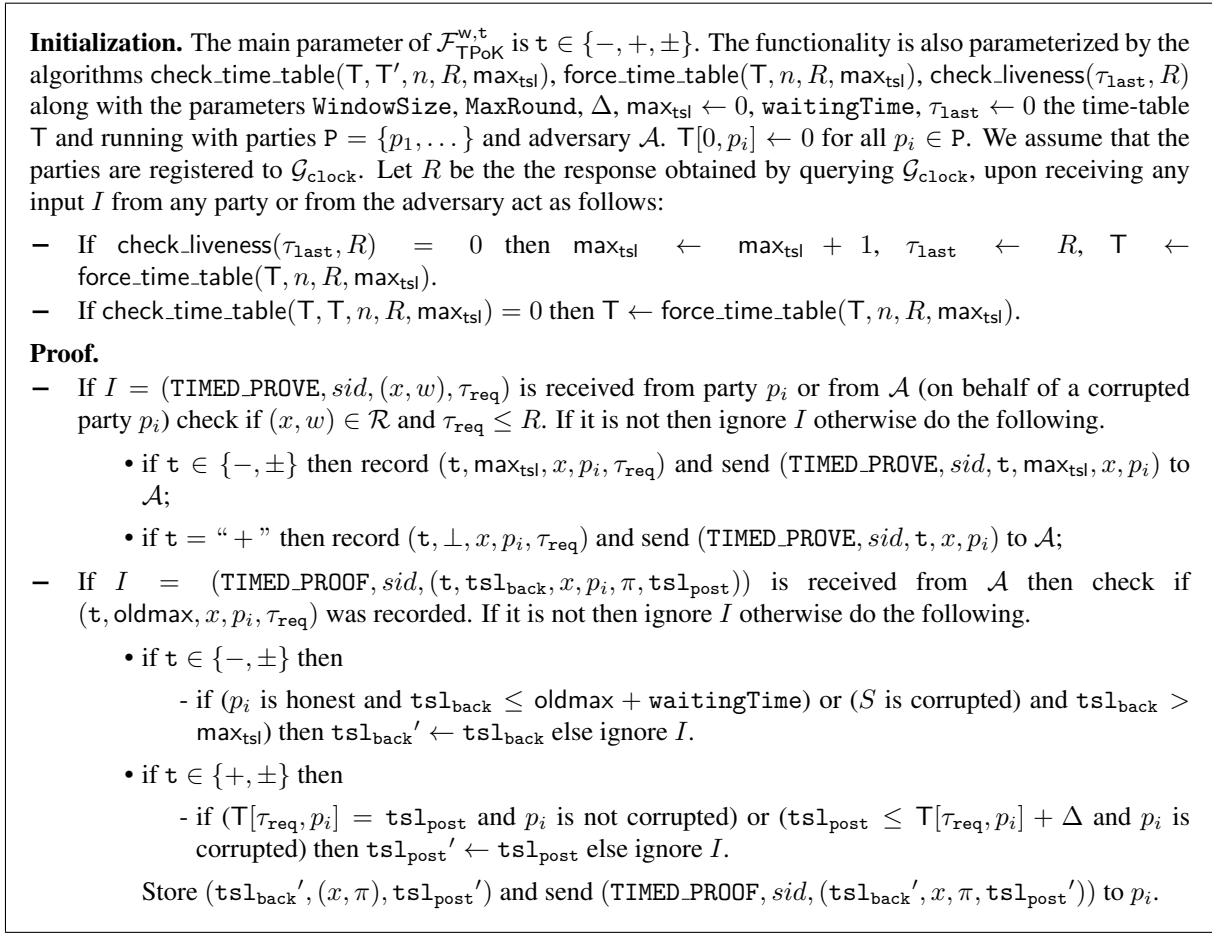


Figure 5.9: The $\mathcal{F}_{\text{TPoK}}^{w, \mathfrak{t}}$ functionality.

of the documents and the previous round time-stamp. Then, the root of the tree is considered as the documents' time-stamp of the round. Moreover, [BLT14] proposes a protocol that requires multiple non-colluding servers who interactively time-stamp a document. Many variants of time-stamping mechanism have been proposed ever since. Recently, Buldas *et al.* [BLT17] proposed an efficient server-aided hash-based time-stamped signature scheme based on a Merkle tree and hash function. The scheme associates a time-slot to each signing key that can be used only for the related time period to sign a message. In particular, the signer first generates a set of signing keys, one for each time-slot, and commits to them. To sign a message at a certain time period, it hashes the related key along with the message and sends the result to the server who time-stamps it, using the tree, and returns the time-stamped value to the signer. When the time-slot has passed, the signer releases the signature and the associated signing key. In the verification phase, the verifier accepts the signature if (a) the signing key and the time-stamp, provided by the server, belong to the same slot, and (b) the server has time stamped the same signature. In this scheme, it is assumed that the time stamping server is fully trusted and does not collude with a signer; otherwise, a signer can back/post-date a signature.

Time-lock encryptions and timed signatures. Time-lock encryption allows one to encrypt a message such that it cannot be decrypted (even by the sender) until a certain amount of time has passed. The time-lock encryption scheme proposed by [RSW96] is secure against a receiver who may have access to large computational resources. The scheme is based on the Blum-Blum-Shub pseudorandom number generator that relies on modular repeated squaring, believed to be sequential. Later on, [BN00, GJ02] improved the previous work and proposed timed commitment and timed signature schemes. The former scheme allows a party to commit to a value in a way that

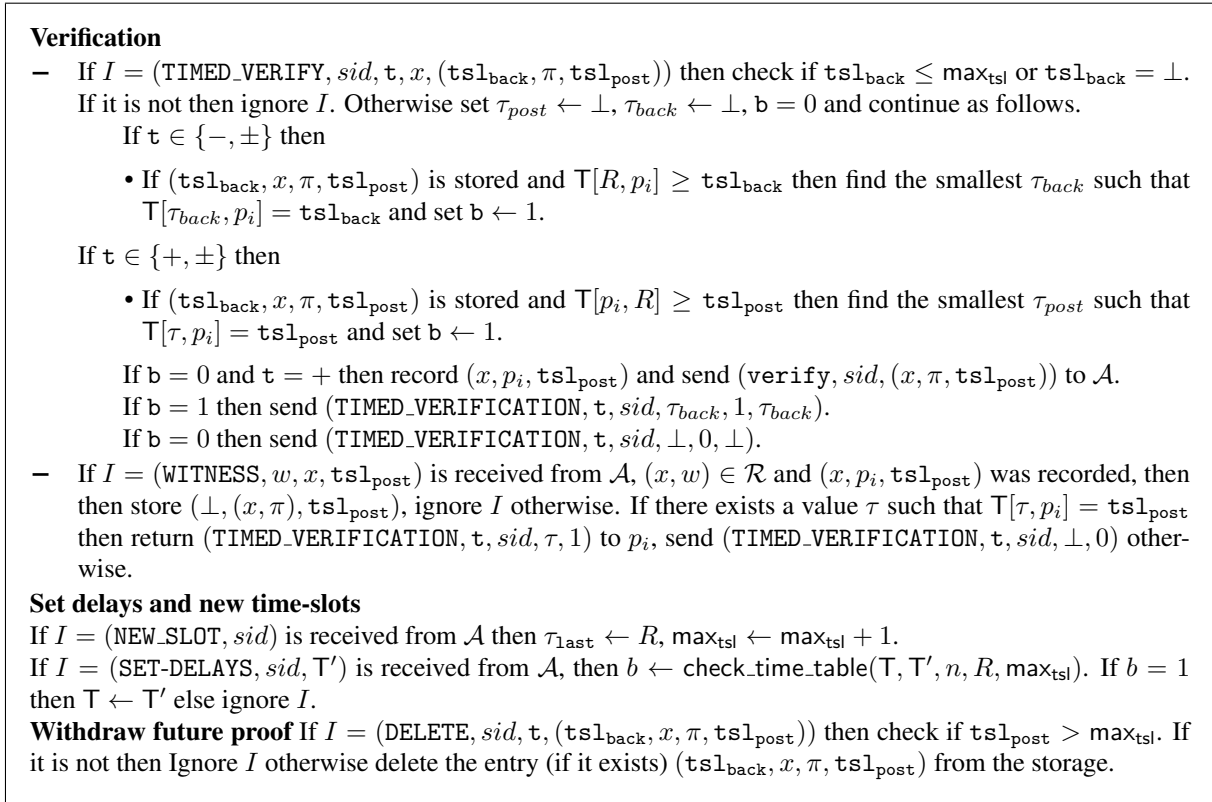


Figure 5.10: The $\mathcal{F}_{\text{TPoK}}^{\mathfrak{w}, \mathfrak{t}}$ functionality (cont'd).

if, later on, it avoids opening the commitment, it can be forced-open by anyone after a certain time period. The proposed timed-signature scheme lets two mutually distrustful parties sign a message, and prove to each other that they have signed the right message, which can be extracted after a certain time period, without disclosing their signature or decrypting it at the proving phase. The signature scheme is based on the repeated modular double-squaring, the timed-commitment and zero-knowledge proof systems. Very recently, after the appearance of Bitcoin and the blockchain technology, a set of blockchain-based time-lock encryptions have been proposed, e.g. [LJKW18, LGR15]. Intuitively, in these protocols, a sender encrypts a message using a witness encryption scheme whose secret key is a hash of a future block, in the blockchain, that will appear after a certain time period. Note that the main difference between the witness encryption-based and the repeated squaring-based (or in general puzzle-based) approaches is that in the former one, when a secret key appears the recipient starts decrypting the ciphertext; whereas, in the latter one, the receiver has to start decrypting it, as soon as the ciphertext is provided, and the decryption process has to be run for the whole life-time of the ciphertext.

Blockchain-based time-stamping service. In the same line of work, there are schemes, such as OriginStamp⁷ or [GMG15, CE12, GN17], in which clients simply store the hash value (or commitment) of their data in a form of a transaction into a blockchain to time-stamp them. These schemes, unlike the previous ones, do not require a centralised trusted party or a collaboration of clients to time-stamp data, at the cost of blockchain transaction fee.

Time and zero-knowledge proofs. Lam *et al.* in [LTCL07] propose a protocol, called timed zero-knowledge proof, that supports secure access of shared computing resources for anonymous clients. The protocol is mainly based on a centralised e-cash scheme proposed in [EO94]. In the former protocol, in order for a client to obtain a

⁷<http://originstamp.org>

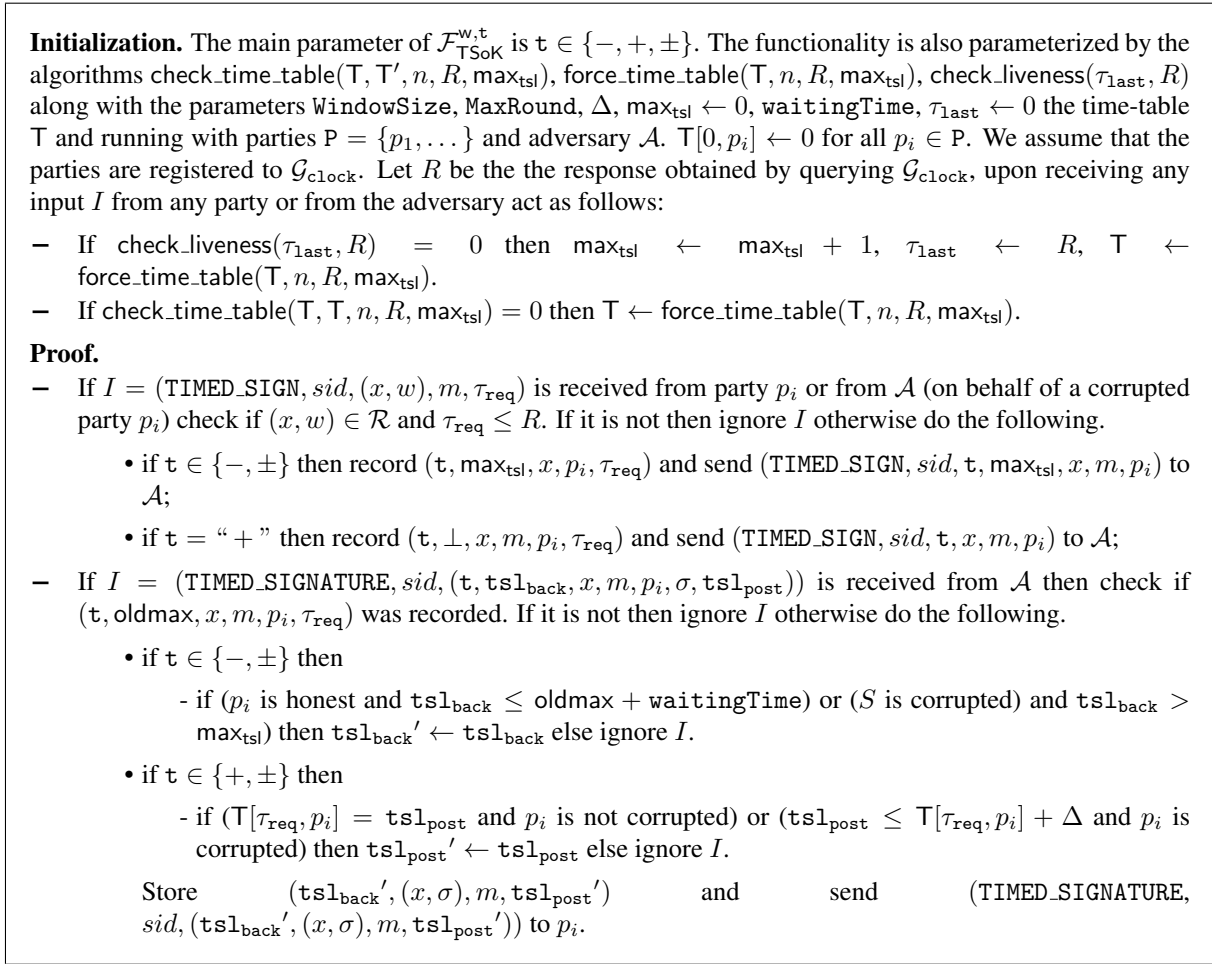


Figure 5.11: The $\mathcal{F}_{\text{TSOK}}^{w,t}$ functionality.

timed zero knowledge proof/token, it first receives a token from a central authority, and sends the token to a server who time-stamps it and returns the result to the client. The client can use the time-stamped token anonymously for a certain session only once and if the client double spends the token, it can be de-anonymised. We highlight that the notion of time has been also considered in concurrent zero-knowledge protocols, e.g. [DNS04], to enable them to be simulated in a concurrent setting. In these protocols, timing constraints are used to impose: (a) delays at the prover-side on sending some messages, and (b) time-outs at the verifier-side on accepting some messages. Thus, in the concurrent zero-knowledge protocols, the time constraints are imposed locally to the protocol's participants and are not embedded in a proof to time-stamp it.

Concluding Remarks. Among all schemes we studied above, only the server-aided schemes support a fine-grained and accurate time-stamping (at the cost of fully trusting the servers), as there is only one clock that belongs to the server who accurately time-stamp all data it receives. However, in the other schemes, the time is approximate, e.g. relative to the growth of a blockchain's length (in the blockchain-based protocols) or the computational power of the recipients (in the puzzle-based approaches).

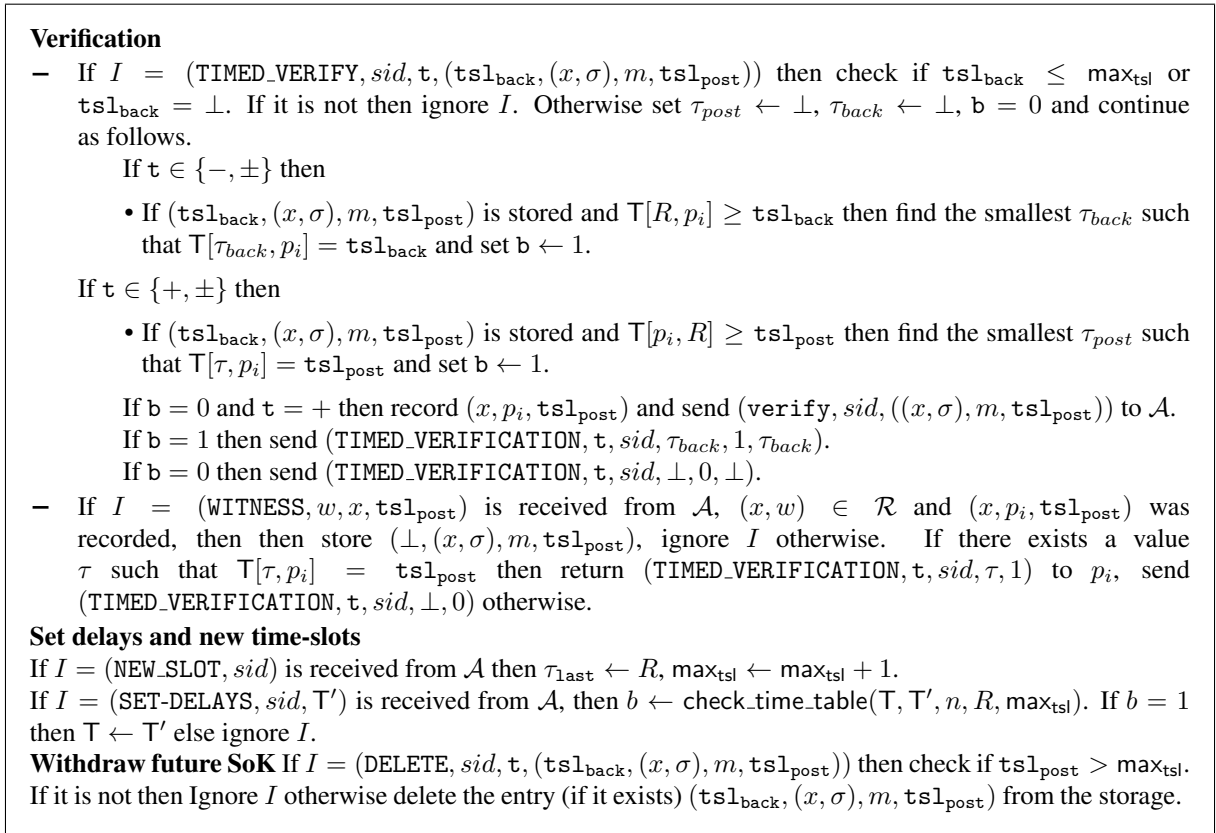


Figure 5.12: The $\mathcal{F}_{\text{TSOK}}^{w, t}$ functionality.

Chapter 6

Bulletin Boards and BPK Model

6.1 Introduction

This section introduces our desideratum on two topics related to the project. First, we note that previous definitions for bulletin boards for distributed ledgers are not suitable for e-voting, as e-voting has its own set of requirements. Thus we will give a new definition that takes into account these requirements.

We also consider the following desideratum with regard to distributed ledgers. As distributed ledgers are a setting where in general parties do not necessarily trust each other, it is therefore useful to have parties that can prove that they have behaved according to a protocol without revealing extra information. This is usually done with zero-knowledge proofs and arguments, as the parties do not want to leak their own private information. A zero-knowledge proof can be done either *interactive* or *non-interactive* (NIZK). For easier use, it is good when the proofs and arguments are non-interactive. Thus we have a natural need for NIZKs and Quasi-Adaptive NIZKs (QA-NIZKs)¹. It is known that the standard model is not sufficient for NIZKs, thus we need some stronger models, known as Random Oracle model or CRS model. In these models we have to either make some nonstandard cryptographic assumptions or trust some third parties. Here we focus on the second approach and improve it by lessening the amount of trust needed. As noted before, often parties do not trust each other. Thus it is commendable if the amount of trust can be reduced. One such avenue for reducing trust is the subversion-resistant ZK (Sub-ZK), meaning that a prover can achieve zero-knowledge without trusting a third party. Instead, a prover needs to run an efficient algorithm to check the well-formedness of CRS elements before using them.

It was shown in [ALSZ18] that Sub-ZK is equivalent to a previously known flavour of ZK, which means that known impossibility and possibility results carry over. In this deliverable, we studied “Sub-ZK” for QA-NIZK protocols that have applications in the design of efficient cryptographic primitives—some of which could be used to design privacy-preserving ledgers. We will introduce a new model for QA-NIZKs that is a new flavour of the Bare Public Key model.

6.2 Previous Work and Improvements

The bulletin board definition is novel as it first presents a cryptographic definition for bulletin board in the context of e-voting. While previous works such as [CS14, GKL15, CZZ⁺16] had definitions for distributed ledgers in general, they did not account for the specific needs of e-voting. Also, many e-voting systems proposed (e.g [FOO92, CGS97, J CJ05, CRS05, Adi08, CEC⁺08]) refer to bulletin boards as a fundamental component of their infrastructure without explicitly realizing it.

¹Quasi-Adaptive NIZKs are NIZKs requiring a common reference string (CRS) and moreover the CRS generation depends on the language parameters.

6.3 E-Voting Defined via Bulletin Boards

Bulletin boards (BB) are a type of ledger where data is inserted during a small timeframe but the transactions should be verifiable for a long time and thus it should provide a consistent view about what has been submitted to it by all the parties. An important part of a BB is the audit board (AB) that any party can read. However, users cannot submit data directly to the AB. A middle step is needed that sanitizes the inputs according to some access policy p_1 . Thus we need to split adding data to the AB into two parts—users *post* data—which means that they make requests that the data should be stored on the AB. Then the data gets checked by the information collectors (IC) that *publishes* the data to the AB provided that it satisfies the necessary requirements. It is also important that it is not possible to remove data. We also want that the user would learn whether his record will be submitted to the AB. For example, in the case of voting, we want the user to know when his submitted vote did not go through for some reason, so he could attempt to vote again. We want that the user would get a receipt from the IC if the data would be published and that all the records for which a receipt was given would be published on the AB. We thus give a formal definition of a BB system.

6.3.1 Syntax of a Bulletin Board System

Entities. A BB system involves the following entities. 1) A *setup authority* SA that generates the setup information and initializes all other entities with their private inputs. 2) The *users* that submit post requests for items of their choice. An item can be any data the user intends to be published, e.g., the voters' ballots, the election results or any necessary audit information. 3) A subsystem of *item collection* (IC) peers P_1, \dots, P_{N_c} that are responsible for (i) interacting with the users for posting all submitted items, and (ii) interacting with the AB (see below) to publish the recorded items. Here N_c denotes the number of IC peers. 4) A subsystem of *audit board* (AB) peers AB_1, \dots, AB_{N_w} where all the posted items are published. Here N_w denotes the number of AB peers.

Setup. During setup, SA specifies a *posting policy* $p_1 = (\text{Accept}, \text{Select}(\cdot))$, where:

1) $\text{Accept} = (U, x)$ is a binary relation over pairs of user IDs and items. For a user U that wants to post item x , $(U, x) \in \text{Accept}$ is a check the IC peers execute to initiate interaction with U for posting x . E.g., a user that is authenticated as a voter may be accepted to post a vote, but nothing else.

2) $\text{Select}(\cdot)$ is a *selection function* over sets of items defined as follows: let X_U be the set of published items associated with posts from user U . Then, $\text{Select}(X_U) \subseteq X_U$ contains all valid published items posted by U , resolving any conflict among clashing items. E.g., in Estonian e-voting [HW14], only the voter's last vote must count. Thus, if the votes for a given user were submitted in time ascending order as x_1, x_2, \dots, x_m , then we set $X_U = (x_1, x_2, \dots, x_m)$ and $\text{Select}(X_U) = x_m$.

The SA initializes other entities with the description of p_1 . Next, all entities engage in a setup interaction such that when finalized, each entity has a private input (e.g., a signing key or an authentication password) and some public parameters params .

BB protocols. The BB functionality comprises the **Posting** and **Publishing** protocols, accompanied by two verification algorithms: (i) VerifyRec , run by the users to verify the successful posting of their items, and (ii) VerifyPub , run by any party for auditing the validity of the data on the AB.

The **Posting** protocol is initiated by a user U that on private input s_U submits a post request for item x . Namely, U uses s_U to generate a credential cr_U ². Then, the user and the IC peers engage in an interaction that results in U obtaining a receipt $\text{rec}[x]$ for the successful posting of x . Upon receiving $\text{rec}[x]$, and using public election parameters params , U may run the algorithm VerifyRec on input $(\text{rec}[x], x, s_U, \text{params})$, that either accepts or rejects.

In the **Publishing** protocol, the IC peers upload their local records of posted items to the AB subsystem. The protocol may encompass a consensus protocol among the AB peers to agree whether a local record is

²E.g., if s_U is a signing key, then cr_U could be a valid signature under s_U ; if s_U is a password, then cr_U can be the pair (U, s_U) .

admissible. In addition, any auditor may run `VerifyPub` on input params and (a subset of the) published data to check consistency of AB.

6.3.2 Introducing our Security Framework

Culnane and Schneider [CS14] propose four properties that a secure BB must satisfy, which are listed below:

- bb.1. *Only items that have been posted may appear on the AB.* This property expresses safety against illegitimate data injection.
- bb.2. *Any item that has a valid receipt must appear on the AB.*
- bb.3. *No clashing items must both appear on the AB.*³
- bb.4. *Once published, no items can be removed from the AB.* According to this property, similar to a ledger the AB subsystem is an *append-only* posting site.

In this section, we integrate the above four properties into a security framework. At a high level, our framework conflates the formal approach in distributed e-voting security of Chondros *et al.* [CZZ⁺16] with the notion of a *robust public transaction ledger (RPTL)* proposed by Garay *et al.* [GKL15]. Namely, we view a secure BB as an RPTL that additionally provides *receipts of successful posting* for honestly submitted items. The security properties of an RPTL stated in [GKL15] are informally expressed as follows:

- *Persistence*: once an honest peer reports an item x as posted, then all honest peers either (i) agree on the position of x on AB, or (ii) not report x .
- θ -*Liveness*: honest peers report honestly submitted items in a delay bound θ .

Persistence and Liveness in the e-voting scenario. In the e-voting setting, honest users should get a valid receipt when engaging at the *Posting* protocol (within some time θ) that confirms the eventual publishing of the respective item. An important observation is that this property that we call θ -*Confirmable Liveness* and property bb.3 can not be satisfied concurrently if we assume that honest users may submit post requests for clashing items (e.g., multiple voting in Estonia [HW14]). To resolve this conflict, we do not require that property bb.3 holds and the subset of valid published items is specified via the selection function `Select(·)`. Given the above, we require that *Persistence* encompasses properties bb.1 and bb.4, and conflict resolution is achieved by applying `Select(·)` on the AB view. Furthermore, we extend *Persistence* by taking into account an AB subsystem that is fully controlled by the adversary. This is formalized by the *Confirmable Persistence* property, where we require that any malicious AB behavior will be detected via the `VerifyPub` algorithm.

System clocks. Like in [CZZ⁺16], we assume that there exists a *global clock* variable $\text{Clock} \in \mathbb{N}$, and that every system entity X is equipped with an *internal clock* variable $\text{Clock}[X] \in \mathbb{N}$. We define the following two events:

The event `Init(X)`: $\text{Clock}[X] \leftarrow \text{Clock}$, that initializes X by synchronizing its internal clock with the global clock.

The event `Inc(Clock[X])`: $\text{Clock}[X] \leftarrow \text{Clock}[X] + 1$, that causes a clock $\text{Clock}[X]$ to advance by one time unit.

Note that the above definition differs from the Global clock functionality in Fig. 5.1 in two ways:

- Firstly, here we are not working in the UC-model.
- Secondly, definition in Fig. 5.1 is meant only for synchronous communication (protocol proceeds in rounds).

Synchronicity and message delay. We parameterize our threat model by (i) an upper bound δ on the delay of message delivery, and (ii) an upper bound Δ on the synchronization loss of the nodes' internal clocks w.r.t. the global clock. By convention, we set $\Delta = \infty$ to denote the fully *asynchronous* setting and $\delta = \infty$, to denote

³For instance, in Estonian e-voting, as a coercion countermeasure, voters may vote multiple times and only the last counts.

that the adversary may drop messages. Values $\delta, \Delta \in [0, \infty)$ refer to *partially synchronous* model, if δ, Δ are unknown.

Notation. We denote by N_c, N_w the number of IC and AB peers, respectively, and by n (an upper bound) on the number of users. In our security analysis, the parameters N_c, N_w, n are assumed polynomial in the security parameter λ . Let $\mathbf{E} := \{\text{SA}\} \cup \{U_i\}_{i \in [n]} \cup \{P_i\}_{i \in [N_c]} \cup \{AB_j\}_{j \in [N_w]}$ be the set of all involved BB system entities. We denote by t_c (resp. t_w) the number of IC (resp. AB) peers that the adversary may statically corrupt out of the total N_c (resp. N_w) peers. We denote the local record of IC peer P_i at global time $\text{Clock} = T$ as the set of accepted and confirmed items $L_{\text{post},i,T} := x_1, \dots, x_{K_{i,T}}$, where $K_{i,T} \in \mathbb{N}$. Similarly, the AB view of peer AB_j at global time $\text{Clock} = T$ is denoted as the set of items $L_{\text{pub},j,T} := x_1, \dots, x_{M_{j,T}}$, where $M_{j,T} \in \mathbb{N}$.

6.3.3 (Confirmable) Persistence Definition

We define Persistence via a security game $\mathcal{G}_{\text{Prst}}^{\mathcal{A}, \delta, \Delta, t_c, t_w}(1^\lambda, \mathbf{E})$ between the challenger \mathcal{C} and an adversary \mathcal{A} . The game is also parameterized by the eventual message delivery and synchronization loss upper bounds δ and Δ . The adversary \mathcal{A} may statically corrupt up to t_c out-of the N_c total IC peers and t_w out-of the N_w total AB peers, and may also choose to corrupt users. \mathcal{C} initializes the BB system on behalf of the SA. Then, \mathcal{C} and \mathcal{A} engage in the **Setup** phase and the **Posting** and **Publishing** protocols, where \mathcal{C} acts on behalf of the honest entities. Intuitively, the goal of \mathcal{A} is to successfully attack the bb.1 property (condition (P.1) in Fig. 6.1) or the bb.4 property (condition (P.2) in Fig. 6.1).

We extend the Persistence notion by defining Confirmable Persistence. Now, the entire AB may be malicious and deviate from the **Publishing** protocol, yet the adversary fails if its attack is detected via the VerifyPub algorithm, on the input view of any AB peer. Formally, Confirmable Persistence is defined via the game $\mathcal{G}_{\text{C.Prst}}^{\mathcal{A}, \delta, \Delta, t_c}(1^\lambda, \mathbf{E})$ that follows the same steps as $\mathcal{G}_{\text{Prst}}^{\mathcal{A}, \delta, \Delta, t_c, t_w}(1^\lambda, \mathbf{E})$, for the special case $t_w = N_w$, except the following differences in the winning conditions for \mathcal{A} : (i) for every $k \in [N_w]$, the published data on AB_k should always verify successfully, and (ii) the inconsistent AB_j referred in the winning conditions may be any (malicious) AB peer. A detailed description of both games is given in Fig. 6.1. We define Persistence and Confirmable Persistence as follows.

Definition 6 ((Confirmable) Persistence). Let λ be the security parameter, $N_c, N_w, t_c, t_w \in \mathbb{N}$, $\delta, \Delta \in [0, +\infty]$, and \mathbf{BB} be a BB system with N_c IC peers and N_w AB peers. We say that \mathbf{BB} achieves Persistence for fault-tolerance thresholds (t_c, t_w) , delay message bound δ and synchronization loss bound Δ , if for every PPT adversary \mathcal{A} it holds that $\Pr[\mathcal{G}_{\text{Prst}}^{\mathcal{A}, \delta, \Delta, t_c, t_w}(1^\lambda, \mathbf{E}) = 1] = \text{negl}(\lambda)$.

We say that \mathbf{BB} achieves Confirmable Persistence for fault tolerance threshold t_c , delay message bound δ and synchronization loss bound Δ , if for every PPT adversary \mathcal{A} , it holds that $\Pr[\mathcal{G}_{\text{C.Prst}}^{\mathcal{A}, \delta, \Delta, t_c}(1^\lambda, \mathbf{E}) = 1] = \text{negl}(\lambda)$.

6.3.4 θ -Confirmable Liveness Definition

We define θ -Confirmable Liveness via a security game $\mathcal{G}_{\theta\text{-C.Live}}^{\mathcal{A}, \delta, \Delta, t_c, t_w}(1^\lambda, \mathbf{E})$ between the challenger \mathcal{C} and an adversary \mathcal{A} , where \mathcal{A} statically corrupts up to t_c (resp. t_w) out-of the N_c (resp. N_w) total IC (resp. AB) peers, while \mathcal{C} plays the role of SA and all peers and users that \mathcal{A} does not corrupt. The adversary wins if it prevents the generation of a valid receipt for an item x or the eventual publishing of x , given that x has been submitted at least θ time prior to the nearest **Publishing** protocol execution. The game is described in detail in Fig. 6.1.

Definition 7 (θ -Confirmable Liveness). Let λ be the security parameter, $N_c, N_w, t_c, t_w, \theta \in \mathbb{N}$, $\delta, \Delta \in [0, +\infty]$ and let \mathbf{BB} be a BB system with N_c IC and N_w AB peers. We say that \mathbf{BB} has θ -Confirmable Liveness for fault-tolerance thresholds (t_c, t_w) , delay message bound δ , and synchronization loss bound Δ , if for every PPT adversary \mathcal{A} , it holds that $\Pr[\mathcal{G}_{\theta\text{-C.Live}}^{\mathcal{A}, \delta, \Delta, t_c, t_w}(1^\lambda, \mathbf{E}) = 1] = \text{negl}(\lambda)$.

6.4 No-Auxiliary String Bare Public Key Model

We introduce a variation of BPK model. Recall that in the BPK model, only the verifier needs to store her public key and the key authority executes the functionality of an immutable bulletin board by storing the received public keys. In particular, one achieves designated-verifier zero knowledge⁴ by using the verifier’s own public key and transferable non-interactive zero knowledge by using the public key of a (trusted-by-many-verifiers) third party. (In the latter case, the public key can be generated by using multi-party computation.)

Since in the BPK model, auxiliary-string non-black-box NIZK for languages not in **BPP** is impossible [Wee07], one can only construct no-auxiliary-string non-black-box (i.e., non-uniform) NIZK. We will carefully define the security of QA-NIZK arguments in the BPK model, following standard QA-NIZK definitions. However, we model the definition of non-uniform NIZK after the Sub-ZK definition of Abdolmaleki *et al.* [ABLZ17]. More precisely, we require that for any efficient malicious public-key creator (either the verifier or a third party) X , there exists an efficient extractor Ext_X , such that if X , by using the language parameter ϱ and any random coins r as an input, generates a public key pk (since there is no auxiliary input, pk has to be generated by X) then Ext_X , given the same input and r , outputs the secret key sk corresponding to pk .

More formally, a tuple of PPT algorithms $\Pi = (\text{Pgen}, \text{K}, \text{PKV}, \mathcal{P}, \mathcal{V}, \text{Sim})$ is a *non-uniform zero knowledge QA-NIZK argument system* in the BPK model for a set of witness-relations $\mathcal{R}_{\text{pp}} = \{\mathcal{R}_{\varrho}\}_{\varrho \in \text{Supp}(\mathcal{D}_{\text{pp}})}$ with ϱ sampled from a distribution \mathcal{D}_{pp} over associated parameter language \mathcal{L}_{pp} , if the following properties (i-iii) hold. Here, Pgen is the parameter generation algorithm, K is the public key generation algorithm, PKV is the public key verification algorithm, \mathcal{P} is the prover, \mathcal{V} is the verifier, and Sim is the simulator.

- (i) **Perfect Completeness:** $\forall \lambda, \text{pp} \in \text{Pgen}(1^\lambda), \varrho \in \mathcal{D}_{\text{pp}}$, and $(x, w) \in \mathcal{R}_{\varrho}$,

$$\Pr \left[\begin{array}{l} (\text{pk}, \text{sk}) \leftarrow \text{K}(\varrho); \pi \leftarrow \mathcal{P}(\varrho, \text{pk}, x, w) \\ \text{PKV}(\varrho, \text{pk}) = 1 \wedge \mathcal{V}(\varrho, \text{pk}, x, \pi) = 1 \end{array} \right] = 1 .$$

- (ii) **Computational Quasi-Adaptive Soundness:** \forall PPT \mathcal{A} , $\text{Adv}_{\mathcal{A}, \Pi}^{\text{qasound}}(\lambda) :=$

$$\Pr \left[\begin{array}{l} \text{pp} \leftarrow \text{Pgen}(1^\lambda); \varrho \leftarrow_{\S} \mathcal{D}_{\text{pp}}; (\text{pk}, \text{sk}) \leftarrow \text{K}(\varrho); \\ (x, \pi) \leftarrow \mathcal{A}(\varrho, \text{pk}) : \mathcal{V}(\varrho, \text{pk}, x, \pi) = 1 \wedge \neg(\exists w : \mathcal{R}_{\varrho}(x, w)) \end{array} \right] \approx 0 .$$

- (iii) **Statistical Non-uniform Zero Knowledge:** For an algorithm \mathcal{A} , let $\text{RND}(\mathcal{A})$ denote the random tape of \mathcal{A} , and let $r \leftarrow \text{RND}(\mathcal{A})$ denote sampling of a randomizer r of sufficient length for \mathcal{A} ’s needs. For any PPT subverter X there exists a PPT Ext_X , such that $\forall \lambda, \forall \text{pp} \in \text{Pgen}(1^\lambda), \varrho \in \mathcal{D}_{\text{pp}}$, and computationally unbounded \mathcal{A} , $\varepsilon_0^{zk} \approx_{\lambda} \varepsilon_1^{zk}$, where

$$\varepsilon_b^{zk} = \Pr \left[\begin{array}{l} r \leftarrow_{\S} \text{RND}(X); (\text{pk}, \text{aux}_X \parallel \text{sk}) \leftarrow (X \parallel \text{Ext}_X)(\varrho; r) \\ \text{PKV}(\varrho, \text{pk}) = 1 \wedge \mathcal{A}^{\text{O}_b(\cdot, \cdot)}(\varrho, \text{pk}, \text{aux}_X) = 1 \end{array} \right] .$$

Here, the oracle $\text{O}_0(x, w)$ returns \perp (reject) if $(x, w) \notin \mathcal{R}_{\varrho}$, and otherwise it returns $\mathcal{P}(\varrho, \text{pk}, x, w)$. Similarly, $\text{O}_1(x, w)$ returns \perp (reject) if $(x, w) \notin \mathcal{R}_{\varrho}$, and otherwise it returns $\text{Sim}(\varrho, \text{pk}, \text{sk}, x)$.

⁴More precisely, a *publicly verifiable* variation of designated-verifier where one does not need to know the secret key to verify as opposed to say [DFN06]. The verifier only needs to be sure that the prover does not know the secret key.

Threat model.

- (I). The adversary \mathcal{A} statically corrupts up to t_c (resp. t_w) out of the N_c (resp. N_w) total peers of the IC (resp. AB) subsystem. Then, \mathcal{A} provides \mathcal{C} with the set $L_{\text{corr}} \subset \mathbf{E}$ of corrupted parties. Throughout the game, \mathcal{C} plays the role of honest entities that include SA.
- (II). When an honest entity X wants to transmit a message \mathbf{M} to an honest entity Y , then it just sends (X, \mathbf{M}, Y) to \mathcal{A} . If the honest entity X sends (X, \mathbf{M}, Y) to \mathcal{A} , when the global time is $\text{Clock} = T$, then \mathcal{A} must write M on the incoming network tape of Y by the time that $\text{Clock} = T + \delta$ (eventual message delivery).
- (III). \mathcal{A} may write on the incoming network tape of any honest entity.
- (IV). \mathcal{A} may invoke the event $\text{Inc}(\text{Clock}[X])$ under the restriction that for any X , $|\text{Clock}[X] - \text{Clock}| \leq \Delta$ (loose clock synchronization).

Protocol execution under the presence of \mathcal{A} .

- The challenger initiates the **Setup** phase playing the role of SA and determines the posting policy $\text{p1} = (\text{Accept}, \text{Select}(\cdot))$. Then, it initializes every system entity $X \in \mathbf{E}$ by running the event $\text{Init}(X)$.
- Upon initialization, \mathcal{C} and \mathcal{A} engage in the **Setup** phase and the **Posting** and **Publishing** protocols, where \mathcal{C} acts on behalf of the honest entities.
- For each user $U_i, i \in [n]$, \mathcal{A} may choose to corrupt, and thus fully control U_i .
- The adversary \mathcal{A} may provide \mathcal{C} with a message (post, U_i, x) for some honest user U_i and an item x of its choice. Upon receiving (post, U_i, x) , \mathcal{C} engages in the **Posting** protocol on behalf of U_i . Upon successful interaction, \mathcal{C} obtains a receipt $\text{rec}[x]$ for x .

Winning conditions for $\mathcal{G}_{\text{Prst}}^{A, \delta, \Delta, t_c, t_w}$.

The game outputs 1 iff there is an AB peer $AB_j \notin L_{\text{corr}}$, at least one of the following holds:

- (P.1). There are $t_c + 1$ honest IC peers $\{P_{i_k}\}_{k \in [t_c + 1]}$, an item x , and moments T, T' , such that (i) $T \leq T'$, (ii) $x \in L_{\text{pub}, j, T}$, and (iii) $x \notin L_{\text{post}, i_k, T'}$, for any $k \in [t_c + 1]$.
- (P.2). There is an item x and moments T, T' such that (i) $T < T'$, (ii) $x \in L_{\text{pub}, j, T}$, and (iii) $x \notin L_{\text{pub}, j, T'}$.

Winning conditions for $\mathcal{G}_{\text{C.Prst}}^{A, \delta, \Delta, t_c}$.

The game outputs 1 iff for every moment T , we have that $\text{VerifyPub}(\langle L_{\text{pub}, j, T} \rangle_{j \in [N_w]}, \text{params}) = \text{accept}$ and there is an AB peer AB_j , such that at least one of the following conditions holds:

- (CP.1). There are $t_c + 1$ honest IC peers $\{P_{i_k}\}_{k \in [t_c + 1]}$, an item x , and moments T', T'' , such that (i) $T' \leq T''$, (ii) $x \in L_{\text{pub}, j, T'}$, and (iii) $x \notin L_{\text{post}, i_k, T''}$, for any $k \in [t_c + 1]$.
- (CP.2). There is an item x and moments T', T'' such that (i) $T' < T''$, (ii) $x \in L_{\text{pub}, j, T'}$, and (iii) $x \notin L_{\text{pub}, j, T''}$.

Winning conditions for $\mathcal{G}_{\theta\text{-C.Live}}^{A, \delta, \Delta, t_c, t_w}$.

The game outputs 1 iff exists an honest user U , an item x and a moment T such that the following hold:

- (CL.1). \mathcal{A} provided \mathcal{C} with the message (post, U, x) at global time $\text{Clock} = T$.
- (CL.2). No honest IC peer engages in the **Publishing** protocol during global time $\text{Clock} \in [T, T + \theta]$.
- (CL.3). Either of the following two is true:
 - (a) By global time $\text{Clock} \leq T + \theta$, \mathcal{C} did not obtain a value z such that $\text{VerifyRec}(z, x, \text{cr}_U, \text{params}) = \text{accept}$
 - (b) There is an AB peer $AB_j \notin L_{\text{corr}}$, such that for any moment T_j , there exists a moment $T'_j \geq T_j$ such that $x \notin L_{\text{pub}, j, T'_j}$.

Chapter 7

Ring Signatures

7.1 Introduction

The original and most prolific application of distributed ledger technology is the cryptocurrency Bitcoin. In this chapter we will refer to Bitcoin explicitly for sake of concreteness. The issues raised in regards to Bitcoin apply to many, though not necessarily all, other forms of cryptocurrency and ledger applications in general as well.

In Bitcoin all transactions that take place are public, and therefore the entire history of a coin can be traced back all the way to its creation. Although Bitcoin allows its users to take on pseudonyms, these are susceptible to analysis due to the way in which transactions are carried out in Bitcoin and it is often possible to link multiple pseudonyms to the same user. The privacy offered by Bitcoin is therefore very limited in scope.

The traceability of coins can also adversely affect the value of the cryptocurrency as a whole in the following way. Suppose part of the community of Bitcoin users disapproves of the purpose for which a certain coin was spent. Since the transaction history is public, that part of the community can collectively decide not to accept transactions involving the controversial coin, or any coins derived from it. In this situation, coins of the same nominal value are not equivalent.

In Bitcoin, a public/private key pair is associated with every coin (or more precisely, every transaction, a coin is merely an unspent transaction output), where the public key is recorded on the ledger and the private key is known only to the holder of the coin. If the holder of a coin wishes to spend it, they compose a transaction statement and provide a cryptographic signature on this transaction using the private key of the coin. The transaction is accepted if the signature is valid and the coin has not been spent in a previous transaction, i.e., the coin does not occur as input in a previous transaction.

In the design of Bitcoin, the ordinary cryptographic signature scheme can be replaced by a signature scheme with some special properties, which can in turn be exploited to improve privacy and mitigate the traceability issue. There exist signature schemes which allow the holder of a coin to create a set of many coins as input to a transaction and only prove that they are spending one of those coins. This, in effect, means that in a transaction a coin cannot be traced with certainty. Moreover, the same coin can occur as input to many transactions in which it is unclear in which, if any, the coin is actually spent.

A *group signature* scheme is a cryptographic signature scheme which allows a signer to create a signature on a message, but without revealing the identity of the signer [CvH91]. Instead, only a group of potential signers is revealed, and the group signature only proves that the message originated from one of the members of this group. Group signatures feature a special role of group manager, who can revoke the anonymity of a signer in case of disputes. Because of the existence of the group manager, a group signature scheme requires special setup before group signatures can be created. This makes group signatures unsuitable for use in Bitcoin.

There also exist signature schemes that, like group signatures, allow the signer to create a signature on behalf of the group, without revealing their identity any further, but without a group manager. Such a scheme is called a *ring signature scheme*, and the group of potential signers is called a ring [RST01]. An *ad hoc* ring signature scheme furthermore allows a signer to create an arbitrary ring given only the public keys of other users, without

any special setup. In the context of cryptocurrencies, ring signatures can be applied to enhance user privacy and reduce the traceability of transactions.

Using a ring signature scheme and providing multiple potential source coins as input to transactions raises the obvious issue that it is no longer possible to check whether a coin has previously been spent. In order to prevent coins from being spent multiple times, an additional property of the signature scheme is required: it should be possible to efficiently determine whether two (or more) signatures were created using the same secret key. A group signature scheme which possesses this property is known as a *list signature scheme* [CSST06]. In addition to the signing and verification algorithm, a list signature scheme features a *matching algorithm*, which can determine whether two signatures were created using the same secret key. There also exists ring signature schemes which allow signatures to be matched (or, linked): so-called *traceable* ring signatures [FS07].

7.2 Definition

We will now define the notion of a ring signature scheme, suitable for use in a cryptocurrency. Specifically, it is an ad hoc ring signature scheme that allows signatures to be *matched* (or *linked*), i.e., that allows determining whether two signatures were created with the same secret key. The definition is an adaptation of [FS07].

Definition 8. A ring signature scheme is a tuple of algorithms (Gen, Sign, Verify, Match) the following holds.

- $(pk, sk) \leftarrow \text{Gen}(1^\lambda)$ is a probabilistic polynomial time algorithm that takes security parameter λ and outputs a public/private key pair (pk, sk) .
- $\sigma \leftarrow \text{Sign}(m, N, (pk_1, \dots, pk_N), j, sk_j)$ is a probabilistic polynomial time algorithm that takes a message, $m \in \{0, 1\}^*$, the size of the ring, N , a list of public keys, pk_i for $1 \leq i \leq N$, an index, $1 \leq j \leq N$, and a secret key, sk_j , corresponding to public key pk_j , and outputs a signature σ .
- $v \leftarrow \text{Verify}(m, \sigma, N, (pk_1, \dots, pk_N))$ is a deterministic polynomial time algorithm that takes a message, $m \in \{0, 1\}^*$, a signature σ , the size of the ring, N , and a list of public keys, pk_i for $1 \leq i \leq N$, and outputs a bit, v .
- $\ell \leftarrow \text{Match}(\sigma_1, \sigma_2)$ is a deterministic polynomial time algorithms that takes a pair of signatures, σ_1 and σ_2 , and outputs a bit, ℓ .

[FS07] define three security requirements for a traceable ring signature scheme: *tag-linkability*, *anonymity* and *exculpability*. From tag-linkable and exculpable follows that the scheme is *unforgeable*. *Tag-linkability* implies that two signatures are generated by different signers if the Match step outputs that two signatures are independent. We refer to [FS07] and [CSST06] for detailed security definitions.

7.3 Discussion

A prerequisite for the use of ring signatures is that a cryptocurrency supports the use of an arbitrary signature scheme to authenticate transactions. Moreover, to make the use of (ad hoc) ring signatures more effective in hiding the payer's identity, it is assumed that one can switch easily from one address to another, preferably using each address just once.

Intuitively, we desire similar security properties for a ring signature scheme as in ordinary signature schemes, i.e., that honestly generated signatures verify and that it is infeasible to find a signature which verifies given only a set of public keys. For use in a cryptocurrency, we require some additional properties:

- Given a set of n secret keys, it is infeasible to produce more than n signatures that are all mutually independent, as determined by the Match algorithm.

D2.2 – Definitions and Notions of Privacy-Enhancing Cryptographic Primitives for Ledgers

- Given a set of secret keys, it is infeasible to produce a signature that matches a signature produced with a secret key that is not in the set.
- Given a signature and the corresponding list of public keys, it is not possible to determine of which public key the corresponding secret key was used to produce the signature with probability better than guessing.

The first property prevents coins from being spent multiple times. The second property ensures coins cannot be spent without knowledge of the corresponding secret key. The final property provides anonymity within the ring.

Ideally, to spend a coin, a ring consisting of all existing coins would be used. In this case coins could be spent completely anonymously and transactions would be completely untraceable. This would however require that the ring size grow with the total number of transactions conducted. This is, however, not practical and rings of suitable size would have to be used to provide some degree of anonymity while remaining efficient in practice.

Using only a small amount of coins to create a ring raises the further issue of how these coins should be selected. Intuitively, the older a coin is, the more likely that it has already been spent. This suggests that not every coin in a transaction is equally likely. Using too small rings and an unsuitable distribution may leave the transaction graph open for analysis. Indeed, the Monero cryptocurrency, which features ring signatures, but was often used with rings of size 1 (no anonymity at all) or 3 and used a uniform distribution for selecting the ring was found vulnerable to analysis [KFTS17].

When using small rings, compared to the ring of all coins, where users are free to choose the composition of the ring it is possible for a spender to construct a set of transactions which reveals information about which coins are being spent. Any such coins lose part of their ability to hide which coin is being spent when used in other transactions. Complete anonymity is therefore not possible in this setting. Either, a method should be used to select the rings for transactions to limit the leakage on the payer's identity as much as possible. Or, one may consider using approaches such as in ZeroCoin which completely hide the payer's identity, however, using substantially more expensive primitives such as zk-SNARKs.

Chapter 8

Conclusion

For many years, cryptographic notions and protocols have been studied in trust-oriented settings like the common reference string model or the public-key infrastructures. While it is nowadays well understood what is possible and not possible to do in those models, very little is known with respect to achieving security when relying on decentralized distributed ledgers. Moreover, the need of cryptographic tools useful to construct ledgers is increasing and requires proper research to formalize their functional and security requirements.

The provable security framework is a de facto standard for assessing the security of cryptographic constructions. It crucially requires to start with formally defined notions in the specific setting in which they must be used. This is a fundamental step before one can construct a provably-secure cryptographic tool.

This deliverable has presented several new formal notions of cryptographic tools that are relevant for constructing and leveraging distributed ledgers. Starting from the work presented in this deliverable, the natural next step is to study the existence of constructions for such notions both theoretically (i.e., minimal assumptions) and in practice (i.e., efficient enough to be used in real-world applications). PRIViLEDGE partners are indeed working in this direction with the goal to obtain some positive results and to illustrate them in deliverable D2.3, that will focus indeed on new constructions of cryptographic primitives for ledger technology.

Bibliography

- [ABLZ17] Behzad Abdolmaleki, Karim Baghery, Helger Lipmaa, and Michal Zajac. A subversion-resistant SNARK. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part III*, volume 10626 of *Lecture Notes in Computer Science*, pages 3–33. Springer, 2017.
- [ACCK18] Elli Androulaki, Christian Cachin, Angelo De Caro, and Eleftherios Kokoris-Kogias. Channels: Horizontal scaling and confidentiality on permissioned blockchains. In Javier López, Jianying Zhou, and Miguel Soriano, editors, *Computer Security - 23rd European Symposium on Research in Computer Security, ESORICS 2018, Barcelona, Spain, September 3-7, 2018, Proceedings, Part I*, volume 11098 of *Lecture Notes in Computer Science*, pages 111–131. Springer, 2018.
- [ACDE18] Elli Androulaki, Jan Camenisch, Angelo De Caro, and Kaoutar Elkhiaoui. Privacy-preserving asset management for permissioned blockchain systems. Manuscript, 2018.
- [Adi08] Ben Adida. Helios: Web-based open-audit voting. In Paul C. van Oorschot, editor, *Proceedings of the 17th USENIX Security Symposium, July 28-August 1, 2008, San Jose, CA, USA*, pages 335–348. USENIX Association, 2008.
- [ALSZ18] Behzad Abdolmaleki, Helger Lipmaa, Janno Siim, and Michal Zajac. On QA-NIZK in the BPK model. *IACR Cryptology ePrint Archive*, 2018:877, 2018.
- [BCG⁺14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*, pages 459–474. IEEE Computer Society, 2014.
- [BdM91] Josh Benaloh and Michael de Mare. Efficient broadcast time-stamping. Technical report, 1991.
- [BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *STOC*, pages 103–112. ACM, 1988.
- [BGK⁺18] Christian Badertscher, Peter Gazi, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. *IACR Cryptology ePrint Archive*, 2018:378, 2018 (Proceedings version to appear at ACM CCS 2018).
- [BGZ16] Iddo Bentov, Ariel Gabizon, and David Zuckerman. Bitcoin beacon. *CoRR*, abs/1605.04559, 2016.
- [BLT14] Ahto Buldas, Risto Laanoja, and Ahto Truu. Efficient quantum-immune keyless signatures with identity. *IACR Cryptology ePrint Archive*, 2014:321, 2014.
- [BLT17] Ahto Buldas, Risto Laanoja, and Ahto Truu. A server-assisted hash-based signature scheme. In *NordSec*, volume 10674 of *Lecture Notes in Computer Science*, pages 3–17. Springer, 2017.

- [BMT18] Christian Badertscher, Ueli Maurer, and Björn Tackmann. On composable security for digital signatures. In Michel Abdalla and Ricardo Dahab, editors, *Public-Key Cryptography - PKC 2018 - 21st IACR International Conference on Practice and Theory of Public-Key Cryptography, Rio de Janeiro, Brazil, March 25-29, 2018, Proceedings, Part I*, volume 10769 of *Lecture Notes in Computer Science*, pages 494–523. Springer, 2018.
- [BMTZ17] Christian Badertscher, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. Bitcoin as a transaction ledger: A composable treatment. In *CRYPTO (1)*, volume 10401 of *Lecture Notes in Computer Science*, pages 324–356. Springer, 2017.
- [BN00] Dan Boneh and Moni Naor. Timed commitments. In *CRYPTO*, volume 1880 of *Lecture Notes in Computer Science*, pages 236–254. Springer, 2000.
- [BOV07] Boaz Barak, Shien Jin Ong, and Salil P. Vadhan. Derandomization in cryptography. *SIAM J. Comput.*, 37(2):380–400, 2007.
- [BS04] Ahto Buldas and Märt Saarepera. On provably secure time-stamping schemes. In Pil Joong Lee, editor, *Advances in Cryptology - ASIACRYPT 2004, 10th International Conference on the Theory and Application of Cryptology and Information Security, Jeju Island, Korea, December 5-9, 2004, Proceedings*, volume 3329 of *Lecture Notes in Computer Science*, pages 500–514. Springer, 2004.
- [Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 136–145. IEEE Computer Society, 2001.
- [Can03] Ran Canetti. Universally composable signatures, certification and authentication. *IACR Cryptology ePrint Archive*, 2003:239, 2003.
- [CDPW07] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In Salil P. Vadhan, editor, *Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007, Proceedings*, volume 4392 of *Lecture Notes in Computer Science*, pages 61–85. Springer, 2007.
- [CE12] Jeremy Clark and Aleksander Essex. Commitcoin: Carbon dating commitments with bitcoin - (short paper). In *Financial Cryptography*, volume 7397 of *Lecture Notes in Computer Science*, pages 390–398. Springer, 2012.
- [CEC⁺08] David Chaum, Aleksander Essex, Richard Carback, Jeremy Clark, Stefan Popoveniuc, Alan T. Sherman, and Poorvi L. Vora. Scantegrity: End-to-end voter-verifiable optical-scan voting. *IEEE Security & Privacy*, 6(3):40–46, 2008.
- [CGGM00] Ran Canetti, Oded Goldreich, Shafi Goldwasser, and Silvio Micali. Resettable zero-knowledge (extended abstract). In F. Frances Yao and Eugene M. Luks, editors, *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 235–244. ACM, 2000.
- [CGHZ16] Sandro Coretti, Juan A. Garay, Martin Hirt, and Vassilis Zikas. Constant-round asynchronous multi-party computation based on one-way functions. In *ASIACRYPT (2)*, volume 10032 of *Lecture Notes in Computer Science*, pages 998–1021, 2016.
- [CGS97] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. In Walter Fumy, editor, *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*, volume 1233 of *Lecture Notes in Computer Science*, pages 103–118. Springer, 1997.

- [Cha83] David Chaum. Blind signature system. In David Chaum, editor, *Advances in Cryptology, Proceedings of CRYPTO '83, Santa Barbara, California, USA, August 21-24, 1983.*, page 153. Plenum Press, New York, 1983.
- [CL06] Melissa Chase and Anna Lysyanskaya. On signatures of knowledge. In *CRYPTO*, volume 4117 of *Lecture Notes in Computer Science*, pages 78–96. Springer, 2006.
- [CRS05] David Chaum, Peter Y. A. Ryan, and Steve A. Schneider. A practical voter-verifiable election scheme. In Sabrina De Capitani di Vimercati, Paul F. Syverson, and Dieter Gollmann, editors, *Computer Security - ESORICS 2005, 10th European Symposium on Research in Computer Security, Milan, Italy, September 12-14, 2005, Proceedings*, volume 3679 of *Lecture Notes in Computer Science*, pages 118–139. Springer, 2005.
- [CS14] Chris Culnane and Steve Schneider. A peered bulletin board for robust use in verifiable voting systems. *CoRR*, abs/1401.4151, 2014.
- [CSST06] Sébastien Canard, Berry Schoenmakers, Martijn Stam, and Jacques Traoré. List signature schemes. *Discrete Applied Mathematics*, 154(2):189–201, 2006.
- [CvH91] David Chaum and Eugène van Heyst. Group signatures. In Donald W. Davies, editor, *Advances in Cryptology - EUROCRYPT '91, Workshop on the Theory and Application of Cryptographic Techniques, Brighton, UK, April 8-11, 1991, Proceedings*, volume 547 of *Lecture Notes in Computer Science*, pages 257–265. Springer, 1991.
- [CZZ⁺16] Nikos Chondros, Bingsheng Zhang, Thomas Zacharias, Panos Diamantopoulos, Stathis Maneas, Christos Patsonakis, Alex Delis, Aggelos Kiayias, and Mema Roussopoulos. D-DEMOS: A distributed, end-to-end verifiable, internet voting system. In *36th IEEE International Conference on Distributed Computing Systems, ICDCS 2016, Nara, Japan, June 27-30, 2016*, pages 711–720. IEEE Computer Society, 2016.
- [DFN06] Ivan Damgård, Nelly Fazio, and Antonio Nicolosi. Non-interactive zero-knowledge from homomorphic encryption. In Shai Halevi and Tal Rabin, editors, *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*, volume 3876 of *Lecture Notes in Computer Science*, pages 41–59. Springer, 2006.
- [DMD04] Michael De Mare and Lincoln DeCoursey. A survey of the timestamping problem. Technical report, Citeseer, 2004.
- [DNS04] Cynthia Dwork, Moni Naor, and Amit Sahai. Concurrent zero-knowledge. *J. ACM*, 51(6):851–898, 2004.
- [EO94] Tony L. Eng and Tatsuaki Okamoto. Single-term divisible electronic coins. In *EUROCRYPT*, volume 950 of *Lecture Notes in Computer Science*, pages 306–319. Springer, 1994.
- [Eth18] Ethereum Foundation. Erc20 token standard. https://theethereum.wiki/w/index.php/ERC20_Token_Standard, June 2018.
- [FLS90] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple non-interactive zero knowledge proofs based on a single random string (extended abstract). In *31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22-24, 1990, Volume I*, pages 308–317. IEEE Computer Society, 1990.
- [FOO92] Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. A practical secret voting scheme for large scale elections. In Jennifer Seberry and Yuliang Zheng, editors, *Advances in Cryptology*

- *AUSCRYPT '92, Workshop on the Theory and Application of Cryptographic Techniques, Gold Coast, Queensland, Australia, December 13-16, 1992, Proceedings*, volume 718 of *Lecture Notes in Computer Science*, pages 244–251. Springer, 1992.

- [FS07] Eiichiro Fujisaki and Koutarou Suzuki. Traceable ring signature. In Tatsuaki Okamoto and Xiaoyun Wang, editors, *Public Key Cryptography - PKC 2007, 10th International Conference on Practice and Theory in Public-Key Cryptography, Beijing, China, April 16-20, 2007, Proceedings*, volume 4450 of *Lecture Notes in Computer Science*, pages 181–200. Springer, 2007.
- [GG17] Rishab Goyal and Vipul Goyal. Overcoming cryptographic impossibility results using blockchains. In Yael Kalai and Leonid Reyzin, editors, *Theory of Cryptography - 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15, 2017, Proceedings, Part I*, volume 10677 of *Lecture Notes in Computer Science*, pages 529–561. Springer, 2017.
- [GJ02] Juan A. Garay and Markus Jakobsson. Timed release of standard digital signatures. In *Financial Cryptography*, volume 2357 of *Lecture Notes in Computer Science*, pages 168–182. Springer, 2002.
- [GKL15] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *EUROCRYPT (2)*, volume 9057 of *Lecture Notes in Computer Science*, pages 281–310. Springer, 2015.
- [GMG15] Bela Gipp, Norman Meuschke, and André Gernandt. Decentralized trusted timestamping using the crypto currency bitcoin. *CoRR*, abs/1502.04015, 2015.
- [GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In Robert Sedgewick, editor, *Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May 6-8, 1985, Providence, Rhode Island, USA*, pages 291–304. ACM, 1985.
- [GN17] Yuefei Gao and Hajime Nobuhara. A decentralized trusted timestamping based on blockchains. *IEEJ Journal of Industry Applications*, 6(4):252–257, 2017.
- [GO07] Jens Groth and Rafail Ostrovsky. Cryptography in the multi-string model. In Alfred Menezes, editor, *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings*, volume 4622 of *Lecture Notes in Computer Science*, pages 323–341. Springer, 2007.
- [GOS06] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Non-interactive zaps and new techniques for NIZK. In Cynthia Dwork, editor, *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*, volume 4117 of *Lecture Notes in Computer Science*, pages 97–111. Springer, 2006.
- [GOS12] Jens Groth, Rafail Ostrovsky, and Amit Sahai. New techniques for noninteractive zero-knowledge. *J. ACM*, 59(3):11:1–11:35, 2012.
- [HS91] Stuart Haber and W. Scott Stornetta. How to time-stamp a digital document. *J. Cryptology*, 3(2):99–111, 1991.
- [HW14] Sven Heiberg and Jan Willemsen. Verifiable internet voting in estonia. In Robert Krimmer and Melanie Volkamer, editors, *6th International Conference on Electronic Voting: Verifying the Vote, EVOTE 2014, Lochau / Bregenz, Austria, October 29-31, 2014*, pages 1–8. IEEE, 2014.
- [JCJ05] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In Vijay Atluri, Sabrina De Capitani di Vimercati, and Roger Dingledine, editors, *Proceedings of the 2005*

ACM Workshop on Privacy in the Electronic Society, WPES 2005, Alexandria, VA, USA, November 7, 2005, pages 61–70. ACM, 2005.

- [KFTS17] Amrit Kumar, Clément Fischer, Shruti Tople, and Prateek Saxena. A traceability analysis of monero’s blockchain. In Simon N. Foley, Dieter Gollmann, and Einar Snekkenes, editors, *Computer Security - ESORICS 2017 - 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part II*, volume 10493 of *Lecture Notes in Computer Science*, pages 153–173. Springer, 2017.
- [KMTZ13] Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Universally composable synchronous computation. In *TCC*, volume 7785 of *Lecture Notes in Computer Science*, pages 477–498. Springer, 2013.
- [KRDO17] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 357–388. Springer, 2017.
- [KT13] Ralf Küsters and Max Tuengerthal. The IITM model: a simple and expressive model for universal composability. *IACR Cryptology ePrint Archive*, 2013:25, 2013.
- [LGR15] Jia Liu, Flavio Garcia, and Mark Ryan. Time-release protocol from bitcoin and witness encryption for sat. *IACR Cryptology ePrint Archive*, 2015:482, 2015.
- [LJKW18] Jia Liu, Tibor Jager, Saqib A. Kakvi, and Bogdan Warinschi. How to build time-lock encryption. *Des. Codes Cryptography*, 86(11):2549–2586, 2018.
- [LTCL07] TC Lam, Cheng-Chung Tan, Yu-Jen Chang, and Jyh-Charn Liu. Timed zero-knowledge proof (tzkp) protocol. In *submitted to IEEE Real-Time and Embedded Technology and Application Symposium*, 2007.
- [MR11] Ueli Maurer and Renato Renner. Abstract cryptography. In Bernard Chazelle, editor, *Innovations in Computer Science - ICS 2010, Tsinghua University, Beijing, China, January 7-9, 2011. Proceedings*, pages 1–21. Tsinghua University Press, 2011.
- [Nak] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system,” <http://bitcoin.org/bitcoin.pdf>.
- [Nak09] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Whitepaper, 2009. <http://bitcoin.org/bitcoin.pdf>.
- [PBF⁺17] Andrew Poelstra, Adam Back, Mark Friedenbach, Gregory Maxwell, and Pieter Wuille. Confidential assets. <https://blockstream.com/bitcoin17-final41.pdf>, 2017.
- [PBF⁺18] Andrew Poelstra, Adam Back, Mark Friedenbach, Gregory Maxwell, and Pieter Wuille. Confidential assets. In Aviv Zohar, Ittay Eyal, Vanessa Teague, Jeremy Clark, Andrea Bracciali, Federico Pintore, and Massimiliano Sala, editors, *Financial Cryptography and Data Security - FC 2018 International Workshops, BITCOIN, VOTING, and WTSC, Nieuwpoort, Curaçao, March 2, 2018, Revised Selected Papers*, volume 10958 of *Lecture Notes in Computer Science*, pages 43–63. Springer, 2018.
- [PSS17] Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of*

Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part II, volume 10211 of *Lecture Notes in Computer Science*, pages 643–673, 2017.

- [PW01] Birgit Pfizmann and Michael Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *2001 IEEE Symposium on Security and Privacy, Oakland, California, USA May 14-16, 2001*, pages 184–200. IEEE Computer Society, 2001.
- [RST01] Ronald L. Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. In Colin Boyd, editor, *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*, volume 2248 of *Lecture Notes in Computer Science*, pages 552–565. Springer, 2001.
- [RSW96] Ronald L Rivest, Adi Shamir, and David A Wagner. Time-lock puzzles and timed-release crypto. 1996.
- [SMP87] Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. Non-interactive zero-knowledge proof systems. In *CRYPTO*, volume 293 of *Lecture Notes in Computer Science*, pages 52–72. Springer, 1987.
- [vS13] Nicolas van Saberhagen. Cryptonote v 2.0. <https://cryptonote.org/whitepaper.pdf>, October 2013.
- [Wee07] Hoeteck Wee. Lower bounds for non-interactive zero-knowledge. In Salil P. Vadhan, editor, *Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007, Proceedings*, volume 4392 of *Lecture Notes in Computer Science*, pages 103–117. Springer, 2007.
- [Woo14] Dr. Gavin Wood. Ethereum: A secure decentralized generalized transaction ledger. <https://gavwood.com/paper.pdf>, 2014.