AUDITABILITY IN BLOCKCHAIN SYSTEMS
USING CRYPTOGRAPHIC PROTOCOLS

by

Panagiotis Chatzigiannis
A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
In Partial fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Computer Science

Committee:

Dr. Foteini Baldimtsi, Dissertation Director

Dr. Giuseppe Ateniese, Committee Member

Dr. S. Dov Gordon, Committee Member

Dr. Jiasun Li, Committee Member

Dr. Konstantinos Chalkias, Committee Member

Dr. David S. Rosenblum, Department Chair

Date: 08/09/2022

Summer Semester 2022
George Mason University
Fairfax, VA

Auditability in Blockchain Systems Using Cryptographic Protocols

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

By

Panagiotis Chatzigiannis
Bachelor of Science
Hellenic Naval Academy, 2001
Master of Science
Naval Postgraduate School, 2012

Director: Dr. Foteini Baldimtsi, Professor
Department of Computer Science

Summer Semester 2022
George Mason University
Fairfax, VA

Dedication

I dedicate this dissertation to my beloved wife Stella and to my wonderful kids, Sophia and Vasiliki, for their constant support, encouragement, love and patience throughout my PhD.

Acknowledgments

I am very grateful to my PhD advisor Foteini Baldimtsi, first for accepting me as her PhD student, then giving me this great opportunity to develop my skills and knowledge throughout the PhD program. She continuously supported and guided me for five full years, a critical period in my life after making a big career decision when I decided to pursue a doctoral degree in Computer Science. She created a fine balance between supervising my research and leaving room for my personal freedom to make progress with my ideas, she provided ways and inspiration to build necessary background in the field, she helped me succeed with publishing my research work to the best conferences and journals, and (perhaps most importantly) she helped me prepare with my transition to the industry by providing invaluable advice, even from the first years in the program. It was a great privilege to work with her and to be one of her first doctoral students.

I would like to especially thank Kostas Chalkias, who had a role of my co-mentor, starting from my first summer internship with Facebook as my manager, up to serving as external committee member in my thesis. I consider myself very fortunate working with him, as apart from providing me with inspiring ideas (which eventually transformed into great research works!), he also exposed me to the industry needs in the field, which proved critical for me making the best possible choices in my new career.

I would also like to thank my doctoral committee members, Dov Gordon, Giuseppe Ateniese and Jiasun Li, not only for taking the time to be part of my committee and for their valuable feedback which had a great impact on the development of my dissertation, but also for directly helping me improve my research work during my PhD studies.

My PhD experience was also unique and delightful as I was fortunate to work with brilliant lab-mates in the security lab, exchange ideas, co-author papers and have fun discussions. In my first years in the PhD program, I found myself with supportive and friendly lab-mates such as Mohammad Rezaeirad, Sahar Mazloom and Phi Hung Le, and then I continued having excellent collaboration and fun discussions with Ioanna Karantaidou and Daniel McVicker, who were co-authors and co-workers in my classes and in my research papers and projects. Special thanks to Georgios Georgakis who helped me navigate the challenges of the PhD program and provided lots of useful advice. The great interaction I had all these years with my fellow PhD students is something I will always remember.

Finally, I would like to thank my beloved wife and kids for believing in my goals and for standing by me all these years, often by making difficult sacrifices, such as being apart for extended time periods. Despite having to go through difficult circumstances, they always followed me into pursuing my goals. Without the continuous support from my wonderful family, the successful completion of my long PhD journey would not have been made possible!

Table of Contents

List of Tables

List of Figures

Abstract

AUDITABILITY IN BLOCKCHAIN SYSTEMS USING CRYPTOGRAPHIC PROTOCOLS

Panagiotis Chatzigiannis, PhD

George Mason University, 2022

Dissertation Director: Dr. Foteini Baldimtsi

Enforcement of policy regulations and availability of auditing mechanisms are crucial building blocks for the adoption of distributed payment systems. In this thesis we provide a series of proposals towards implementing such systems (usually based on a blockchain), augmented with such functionalities.

We first propose MiniLedger, a new standalone system that provides both auditability and privacy for its users, while achieving near-constant storage requirements by all participants who maintain it. We then propose add-on protocols for the Diem blockchain, a recently developed permissioned payment system, which enables proving assets held by custodial wallets in that system. Then we propose gOTzilla, which enables proving assets owned by organizations or exchanges in permissionless blockchain systems (such as Bitcoin), while remaining efficient even in the case such systems scale to millions of unspent transaction outputs.

Finally, we describe BBox-IoT, which provides a way for resource-constrained devices (typically encountered in an Industrial IoT setting) to efficiently sign data, therefore enabling auditing data provenance in a blockchain-based Industrial IoT system.

Chapter 1: Introduction

During the last decade, we witnessed a surge of proposals for several different distributed payment systems, commonly referred to as *cryptocurrencies*, as an alternative to the traditional, centralized banking system. This term is derived from the fact that these systems are based on a series of cryptographic primitives. For instance, Bitcoin [1], which was the first cryptocurrency to emerge, uses relatively simple cryptographic tools, which include public key cryptography, hash functions and distributed consensus. These tools are used to secure transactions, as well as guarantee agreement on the state of a common append-only public ledger, known as a blockchain. User participation in these systems can be controlled or unrestricted, categorizing such systems in *permissioned* and *permissionless* respectively.

The distributed nature of the ledger, typically accessible by the open public or by a wide base of participants (even in permissioned systems), enables external observers to access transaction information, for example sender/receiver addresses or transaction amounts. These addresses are essentially random-looking strings and provide their owners a sense of anonymity, especially in permissionless payment systems where anyone can easily create multiple such addresses on demand. However, it has been shown it is possible to associate these "pseudo-anonymous" addresses with real identities (for instance using clustering techniques [2–4]). These concerns led to a number of privacy-enhancing proposals, with most of them using more advanced cryptographic tools (e.g ring signatures, zero-knowledge proofs etc.) Some were stand-alone cryptocurrencies offering strong privacy guarantees such as Zcash [5] and Monero [6], as well as academic works such as Quisquis [7] and Solidus [8], while others were add-on functionalities to existing systems, such as CoinJoin [9] or TumbleBit [10]. These approaches obfuscate (or completely hide) both the transaction graph which can be generated from connecting senders and receivers of funds (as well as those

associated amounts) from public view, thereby providing a level of privacy equivalent to cash. But such systems in turn raised concerns for regulatory and law-enforcement authorities, since the abuse of such strong privacy guarantees provides users the potential to circumvent regulatory controls (e.g. tax evasion or unauthorized money transmission) or even engage in fraudulent/illegal activities (e.g. money laundering, extortion or drug trafficking [11]). In this setting, state authorities or audit firms (e.g. Deloitte or KPMG [12,13]) will need to be convinced that the auditee "follows the rules" by meeting certain regulatory requirements. For example, all participants in a payment system should be compliant with Anti-Money Laundering and Counter-Terrorism Financing (AML/KYC) per Financial Action Task Force (FATF) Travel Rule [14], while an auditor should be able to verify compliance with regulations such as the European General Data Protection Regulation [15] or industry-specific requirements such as the Health Insurance Portability and Accountability Act (HIPAA).

In the above setting, enforcing regulation becomes challenging, as the notions of privacy and regulation are contradictory. In such distributed payment systems, regulation implies auditability and accountability at user level (e.g. disclosing the user's assets or past transactions) or transaction level (e.g. disclosing the participants or value associated with some transaction), while the public ledger entries continues to hide such information from parties not directly associated with those regulatory functions. A limited type of accountability already existed in traditional, private electronic cash [16] where when a coin was double-spent, the identity of its owner could be revealed. Recently, a handful of academic works attempted to provide some basic accountability or auditability functionalities, either on top of existing privacy-preserving payment systems [17,18], or as new, stand-alone ones [19,20].

At the same time, as cryptocurrency popularity grew, various organizations were established to bridge the gap between existing financial systems and the new cryptocurrency world, such as cryptocurrency exchanges, which provide services for exchanging cryptocurrency with US Dollars, Euros etc., "stablecoins", i.e. cryptocurrencies having their value tied to standard currencies [21], or other types of centralized organizations which hold users'

coins. However, even with "pseudoanonymous" distributed payment systems which do not hide transaction information on the public ledger, such information is not enough to provide regulatory control and additional regulation functionalities are still needed, as these organizations are typically opaque to their internal operations. In fact, several infamous examples exist where users lost their funds without holding these organizations accountable [22] or organizations investing users' funds instead of focusing on their solvency [23]. Also, the Conference of State Bank Supervisors proposed a model regulatory framework including cryptographic solvency proofs as a means of demonstrating solvency [24]. In that end, some works [17, 25] focused on making these services more transparent to earn users trust, and provide auditability functionalities to authorities (i.e., proving that they are solvent) without disclosing additional information or exposing their users' privacy[1]. However, preserving privacy does not come for free, as this makes proof manipulation or collusion possible to falsely convince an auditor of the organization's claims.

Problem statement. From the above discussion, it is evident that there is a need for incorporating regulation in distributed payment systems from a technical standpoint. Specifically, in this thesis we consider the problem of designing appropriate protocols that can facilitate the needed regulation functionalities, both at a microscopic (i.e. single user or transaction) and at a macroscopic (i.e. large organization) level. We also consider solutions to this problem either by proposing new systems, or by augmenting existing ones, without deviating from the core blockchain characteristics, such as its decentralized nature and the privacy offered to its participants.

Results summary. In this thesis, we begin by considering the first aspect of the problem above, namely combining privacy and auditability functionalities at a microscopic level into a distributed payment system. To that end, we construct MiniLedger as a proposed solution [26], a system with strong privacy guarantees while providing a wide range of

[1] In some scenarios, non-private auditing might suffice. However, such a protocol would be trivial from a security standpoint, and to our knowledge no related proposal exists.

auditing functionalities, while also being efficient in terms of the needed storage costs. We provide formal security definitions, an extension serving an arbitrarily large user base, as well as evaluation results of our implemented prototype to showcase its efficiency.

Then we consider the problem of proving the solvency of organizations in the cryptocurrency world holding users' coins. As [17] observed, such a solvency proof consists of two independent proofs: a Proof of Assets and a Proof of Liabilities, which combined together form a proof of solvency. While Proof of Liabilities has been extensively studied [25,27], we observed a research gap for efficient and practical Proof of Assets since the seminal work of [17]. Therefore, we first provide a Proof of Assets framework [28] tailored for Diem cryptocurrency [29] (as well as other such systems with a similar hierarchical structure). In our framework, we consider several cases encountered in practice, such as offline wallets, locked assets, delegated spending capabilities and account pruning, while also proposing practical optimizations towards supporting light clients.

Then we focus on large, popular permissioned payment systems such as Bitcoin. In this setting, we propose gOTzilla[30], which provides an efficient protocol for such cryptocurrency exchanges and organizations to prove their assets, even when those cryptocurrencies scale to millions of accounts. While our protocol is designed with the above use case scenario in mind, we also construct similar protocols for a wider range of applications that are of independent interest in the cryptography community.

Finally, we consider the use of blockchains in a resource-constrained device setting such as Industrial Internet of Things (IIoT). Towards auditing data collected from sensors in such environments, we construct BBox-IoT [31], a system that provides a way for operators of an IIoT enclave to audit sensing data. BBox-IoT is based on a novel signature scheme relying on simple cryptographic primitives and is tailored for the most resource constrained class of IIoT devices where we evaluate its efficiency.

Thesis outline. This thesis is organized as follows. Chapter 2 contains the necessary background for the primitives used throughout this thesis. We then propose MiniLedger

in Chapter 3 which efficiently implements privacy and auditability functionalities into a distributed payment system. Towards Proof of Assets, we propose our framework for the Diem cryptocurrency in Chapter 4 and our protocol for popular permissioned blockchains such as Bitcoin in Chapter 5. In Chapter 6 we present BBox-IoT as a proposed efficient solution for auditing resource-constrained devices in a blockchain-based IIoT setting. We conclude in Chapter 7 and provide insights for future work.

Chapter 2: Preliminaries

We first define the notation we will be using throughout this thesis. By λ we denote the security parameter, by pp the public parameters and by $z \leftarrow \mathcal{Z}$ the uniformly random selection of an element z from space $\mathcal{Z}$. A probabilistic polynomial-time (PPT) algorithm B with input a and output b is written as $b \leftarrow B(a)$. By $:=$ we denote deterministic computation and by $a \rightarrow b$ we denote assignment of value a to value b. We denote a protocol between two parties A and B with inputs x and y respectively as $\{\mathsf{A}(x) \leftrightarrow \mathsf{B}(y)\}$. By $(\mathsf{pk}, \mathsf{sk})$ we denote a public-private key pair and by $[x_i]_{i=1}^{y}$ a list of elements $(x_1, x_2, ..., x_y)$. We denote an n-dimensional vector $\mathbf{v} = \{v_1, \ldots, v_n\}$. By $x \parallel y$ we denote concatenation of bit strings x and y. We denote a matrix M with m rows and n columns as M_{mn} and a $i-$th row and j-th column cell in the matrix as (i, j). Finally for simplicity, we omit λ and pp and implied as inputs to the respective algorithms and protocols as needed, unless denoted explicitly.

2.1 Cryptographic primitives

Public key encryption. A public key encryption scheme consists of the following algorithms [32]:

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}()$: Outputs a public key pk and a secret key sk, .
- $C \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$: On input of a message m and a public key pk, outputs a ciphertext C.
- $m \leftarrow \mathsf{Dec}(\mathsf{sk}, C)$: On input of a ciphertext C and a secret key sk, outputs a plaintext message m.

To formalize security of a public key encryption scheme, we consider the following experiment $\mathsf{IND}-\mathsf{CPA}(\lambda)$:

- An adversary $\mathcal{A}$ knowing a public key pk performs a polynomial number of queries to a challenger for messages m, and the challenger outputs encryptions of the respective messages to $\mathcal{A}$.
- $\mathcal{A}$ outputs a pair of messages m_0, m_1 to a challenger.
- Challenger chooses a random bit b and outputs ciphertext $C \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)$ to $\mathcal{A}$.
- $\mathcal{A}$ outputs b' and $\mathsf{IND}-\mathsf{CPA}$ outputs "1" if $b == b'$, else it outputs "0".

Definition 1. *A public key encryption scheme is indistinguishable under chosen plaintext attack (IND-CPA), if for all PPT* $\mathcal{A}$, $\Pr[\mathsf{IND}-\mathsf{CPA}(\lambda) = 1] = 1/2 + negl(\lambda)$.

Homomorphic Encryption. An (additive) homomorphic encryption scheme is a public-key encryption scheme equipped with an operation $\boxplus$ over the ciphertext space such that for any two plaintexts a, b, $\mathsf{Dec}(\mathsf{Enc}(a) \boxplus \mathsf{Enc}(b)) = a + b$.

ElGamal encryption. ElGamal encryption [33] consists of the following algorithms:

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$: On input of security parameter λ, outputs public parameters $\mathsf{pp} = (\mathbb{G}, g, p)$ where g is generator of cyclic group $\mathbb{G}$ of prime order p. We consider these parameters as a default input to all following algorithms and we omit them for simplicity.
- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}()$: Outputs a secret-public key pair as $\mathsf{sk} \leftarrow \mathbb{Z}_p$, $\mathsf{pk} = g^{\mathsf{sk}}$.
- $(c_1, c_2) \leftarrow \mathsf{Enc}(\mathsf{pk}, x)$: Samples $r \leftarrow \mathbb{Z}_p$, computes $c_1 = g^r, c_2 = x \cdot \mathsf{pk}^r$ and outputs ciphertext $C = (c_1, c_2)$.
- $x \leftarrow \mathsf{Dec}(\mathsf{sk}, (c_1, c_2))$: Compute $x = c_2 / c_1^{\mathsf{sk}}$.

ElGamal encryption is IND-CPA secure under the Decisional Diffie-Hellman assumption [32]. Also, ElGamal encryption is multiplicatively homomorphic:

$\mathsf{Enc}(\mathsf{pk}, m_1) \cdot \mathsf{Enc}(\mathsf{pk}, m_2) = \mathsf{Enc}(\mathsf{pk}, m_1 \cdot m_2)$.

Additively homomorphic ElGamal encryption. ElGamal encryption can be modified to be additively homomorphic [34] and consists of the following algorithms:

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$: On input of security parameter λ, outputs public parameters $\mathsf{pp} = (\mathbb{G}, g, p)$ where g is generator of cyclic group $\mathbb{G}$ of prime order p.
- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}()$: Outputs a secret-public key pair as $\mathsf{sk} \leftarrow \mathbb{Z}_p$, $\mathsf{pk} = g^{\mathsf{sk}}$.
- $(c_1, c_2) \leftarrow \mathsf{Enc}(\mathsf{pk}, x)$: Samples $r \leftarrow \mathbb{Z}_p$, computes $c_1 = g^r, c_2 = g^x \cdot \mathsf{pk}^r$ and outputs ciphertext $C = (c_1, c_2)$.
- $x \leftarrow \mathsf{Dec}(\mathsf{sk}, (c_1, c_2))$: Compute $g^x = c_2 / c_1^{\mathsf{sk}}$. While x cannot be directly computed from g^x, it can be recovered through a pre-computed lookup table, assuming that the message space is relatively small (e.g. 2^{32}).

ElGamal Encryption Variant. We now review another variant of ElGamal encryption, called "twisted ElGamal" (TEG) [35]. Compared to the standard additive version of ElGamal encryption, it requires an additional group generator (denoted by h below) in the public parameters pp, which makes it possible to homomorphically add ciphertexts c_2 and c_2' generated for *different* public keys pk and pk' and intentionally leak information on the relation of encrypted messages m and m' as we discuss below. This variant only works when message space is small due to the need for a lookup table on decryption. TEG is secure against chosen plaintext attacks and works as follows:

- $\mathsf{pp} \leftarrow \mathsf{SetupTEG}(1^\lambda)$: Outputs $\mathsf{pp} = (\mathbb{G}, g, h, p)$ where g, h are generators of cyclic group $\mathbb{G}$ of prime order p.
- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{GenTEG}(\mathsf{pp})$: Outputs $\mathsf{sk} \leftarrow \mathbb{Z}_p$, $\mathsf{pk} = h^{\mathsf{sk}}$.

- $(c_1, c_2) \leftarrow \mathsf{EncTEG}(\mathsf{pk}, m)$: Sample $r \leftarrow \mathbb{Z}_p$, compute $c_1 = \mathsf{pk}^r, c_2 = g^m h^r$ and output $C = (c_1, c_2)$
- $m \leftarrow \mathsf{DecTEG}(\mathsf{sk}, (c_1, c_2))$: Compute $g^m = c_2 / c_1^{(1/\mathsf{sk})}$ and recover m from a look-up table (assuming that the message space is relatively small).

Both of the two above ElGamal scheme versions are additively homomorphic, e.g.:

$$\mathsf{EncTEG}(\mathsf{pk}, m_1) \cdot \mathsf{EncTEG}(\mathsf{pk}, m_2) = \mathsf{EncTEG}(\mathsf{pk}, m_1 + m_2).$$

Specifically for TEG encryption, if $(c_1, c_2) \leftarrow \mathsf{EncTEG}(\mathsf{pk}, m)$ and $(c_1', c_2') \leftarrow \mathsf{EncTEG}(\mathsf{pk}', m')$, then $c_2 \cdot c_2'$ contains an encryption of $m + m'$. This implies if $c_2 \cdot c_2' = 1$, then any external observer can deduce that $m = -m'$ (for properly chosen r, r').

Digital Signatures. A digital signature scheme consists of the following algorithms [32, 36]:

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{SignGen}(1^\lambda)$: Outputs a pair of keys $(\mathsf{pk}, \mathsf{sk})$.
- $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$: Takes as input a private key sk and a message m and outputs a signature σ.
- $\mathsf{SVrfy}(\mathsf{pk}, m, \sigma) := b$: Takes as input a public key pk, a message m and a signature σ, and outputs a bit b where $b = 1$ indicates successful verification.

A digital signature is considered secure if an adversary $\mathcal{A}$ cannot forge a signature on a message even after adaptively receiving signatures on messages of its choice. To formalize the security definition we first describe the following experiment $\mathsf{SigForge}(\lambda)$:

1. $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{SignGen}(1^\lambda)$
2. $\mathcal{A}$ on input (pk) queries the signing oracle polynomial number of times q. Let $Q : [m_i, \sigma_i]_{i=1}^q$ be the set of all such queries.
3. $\mathcal{A}$ outputs (m^*, σ^*).

4. $\mathcal{A}$ wins if $\mathsf{SVrfy}(m^*, \sigma^*) = 1$ where $m^* \notin [m_i]_{i=1}^q$ and $\mathsf{SigForge}$ outputs "1", else it outputs "0".

Definition 2. *A digital signature scheme is existentially unforgeable under an adaptive chosen-message attack, if for all PPT $\mathcal{A}$,* $\Pr[\mathsf{SigForge}(\lambda) = 1]$ *is negligible in λ.*

One-time signatures. A digital signature scheme that can be used to sign only one message per key pair is called a one-time signature (OTS) scheme.

Definition 3. *A one-time digital signature scheme is existentially unforgeable under an adaptive chosen-message attack, if for all PPT $\mathcal{A}$ and for $q \leq 1$,* $\Pr[\mathsf{SigForge}(\lambda) = 1]$ *is negligible in λ.*

Hash functions. An (unkeyed) hash function $y := h(m)$ on input of a message m outputs a digest y. A cryptographic hash function is considered secure if the probability to find collisions is negligible (i.e. it is *collision resistant*). More formally, we consider the following experiment $\mathsf{HashColl}$ [32]:

1. $\mathcal{A}$ picks values $x, x' \in \{0,1\}^*$ s.t. $x \neq x'$.
2. $\mathcal{A}$ wins if $h(x) = h(x')$ and $\mathsf{HashColl}$ outputs "1".

Definition 4. *A hash function $h() : \{0,1\}^* \to \{0,1\}^\lambda$ is collision resistant if for all PPT $\mathcal{A}$,* $\Pr[\mathsf{HashColl} = 1]$ *is negligible.*

A weaker notion for security of a hash function is preimage-resistance. We consider the following experiment $\mathsf{PreIm}(\lambda, y)$:

1. $\mathcal{A}$ is given $y \in \{0,1\}^\lambda$
2. $\mathcal{A}$ outputs x.
3. $\mathcal{A}$ wins if $h(x) = y$. If $\mathcal{A}$ wins PreIm outputs "1", else it outputs "0".

Definition 5. *A hash function* $h() : \{0,1\}^* \to \{0,1\}^\lambda$ *is preimage resistant if* $\forall$ *ppt* $\mathcal{A}$ *and* $\forall y \in \{0,1\}^\lambda$, $\Pr[\mathsf{PreIm}(\lambda, y) = 1]$ *is negligible in* λ.

Corollary 1. *A collision resistant hash function is also preimage resistant.*

Commitment schemes. A non-interactive commitment takes as input public parameters pp, message m and randomness r and outputs value $\mathsf{cm} \leftarrow \mathsf{Com}(\mathsf{pp}, m, r)$ such that, on one hand, reveals no information about the message (*hiding* property) but, on the other hand, it is hard to find (m', r') such that $\mathsf{Com}(\mathsf{pp}, m, r) = \mathsf{Com}(\mathsf{pp}, m', r')$, when $m' \neq m$ (*binding* property).

Pedersen commitments. Pedersen commitments [37] are constructed as follows:

- $\mathsf{pp} \leftarrow \mathsf{ComGen}(1^\lambda)$. Outputs $\mathsf{pp} = (\mathbb{G}, g, h, p)$ where g, h are generators of cyclic group $\mathbb{G}$ of prime order p.
- $\mathsf{cm} \leftarrow \mathsf{Com}(\mathsf{pp}, m, r)$. On pp, a message $m \in [1...p]$ and randomness $r \in [1...p]$, outputs a commitment $\mathsf{cm} = g^m h^r$.
- $b \leftarrow \mathsf{Open}(\mathsf{pp}, \mathsf{cm}, m, r)$. A verifier given a commitment cm and an opening (m, r) returns a verification bit b.

Pedersen commitments are perfectly hiding and computationally binding under the discrete logarithm assumption [37]. Also, Pedersen commitments are also additively homomorphic and are zero-knowledge proof-friendly (ZK proofs described below). Note that ciphertext c_2 from the above "twisted" ElGamal variant is equivalent to a Pedersen commitment.

Zero-knowledge proofs. A zero-knowledge (ZK) proof is a two-party protocol between a prover $\mathcal{P}$, holding some private data (or else *witness*) w for a public instance x, and a verifier $\mathcal{V}$. The goal of $\mathcal{P}$ is to convince $\mathcal{V}$ that some property of w is true i.e. $R(x, w) = 1$,

for an NP-relation R, without $\mathcal{V}$ learning anything more. Some common types of ZK proofs we consider are:

1. ZK proof on the opening of a commitment: Using Camenisch-Stadler notation [38] (used throughout this thesis): $\{(w, r) : X = g^w h^r \bmod n\}(X, g, h, n)$ where (X, g, h, n) are the public statements given as common input to both parties, and (w, r) is the secret witness.
2. ZK proof of knowledge of discrete log: $\{(x) : X = g^x \bmod n\}(X, g, n)$.
3. ZK proof of equality of discrete logs: $\{(x, r, r') : X = g^x h^r \bmod n, Y = g^x h^{r'} \bmod n\}(X, Y, g, h, n)$.
4. ZK range proof that a committed value v lies within a specific interval (a, b): $\{(v, r) : X = g^v h^r \bmod n \wedge v \in (a, b)\}(X, g, h, n)$. Such proofs can also be used to show that the value v is positive or does not overflow some modulo operation. Most well-known construction families for range proofs include square decomposition, multi-base decomposition (used by zkLedger [39, 40]) and the recent Bulletproofs [41].

ZK proofs can be composed as follows: (1) AND composition $\pi_1 \wedge \pi_2$ which can be easily constructed by sequential or parallel proving of underlying assertions, and (2) OR composition $\pi_1 \vee \pi_2$ (also referred to as *disjunctive* ZK proofs), which can be constructed by proving knowledge for the one and simulating knowledge for the other, without revealing which of the two is actually proved and which is simulated. We also note the ZK proofs we use are public coin and can be converted to non-interactive using the FS heuristic [42]. The needed properties of a zero-knowledge proof are Completeness, Soundness and Zero-Knowledge, defined formally as follows.

- *Complenetess:* If $R(x, w) = 1$ and both players are honest $\mathcal{V}$ always accepts.
- *Soundness:* For every malicious and computationally unbounded $\mathcal{P}^*$, there is a negligible function $\epsilon(\cdot)$ s.t. if x is a false statement (i.e. $R(x, w) = 0$ for all w), after $\mathcal{P}^*$ interacts with $\mathcal{V}$, $\Pr[\mathcal{V} \text{ accepts}] \leq \epsilon(|x|)$.

- *Zero Knowledge:* For every malicious PPT $\mathcal{V}^*$, there exists a PPT simulator $\mathcal{S}$ and negligible function $\epsilon(\cdot)$ s.t. for every distinguisher D and $(x, w) \in R$ we have $|\Pr[D(\text{View}_{\mathcal{V}^*}(x, w)) = 1] - \Pr[D(\mathcal{S}) = 1]| \leq \epsilon(|x|)$.

ZK-SNARKs. ZK-SNARKs (Succinct Non-Interactive Arguments of Knowledge) [43–45] is another well studied type of ZK proofs that received a lot of attention the last few years due to their use in private cryptocurrencies [5]. Their goal is to offer constant, succinct proof sizes and short verification times. In particular, ZK-SNARKs can be verified in time that is linear in the length of the input x, rather than the length of the circuit C. However, they suffer from large prover overhead, since they require the prover to perform a large number of public-key operations that is proportional to the size of the circuit representing the statement. Finally, many ZK-SNARKS constructions require an additional trust assumption. Namely, to guarantee soundness, they need a common reference string (CRS) that is generated ahead of time by a trusted party (or a distributed protocol).

Mixed statements. Let f and g be non-algebraic and algebraic relations with public instances y and z respectively. A ZK proof on a mixed statement has the generic form $\{(w) : f(y, w) = 1 \wedge g(z, w) = 1\}(y, z)$.

Cryptographic Accumulators. Accumulators allow the succinct and binding representation of a set of elements S and support constant-size proofs of (non) membership on S. We focus on *additive* accumulators where elements can be added over time to S and to *positive* accumulators which allow for efficient proofs of membership. We consider *trapdoorless* accumulators in order to prevent the need for a trusted party that holds a trapdoor and could potentially create fake (non)membership proofs. Finally we require the accumulator to be *deterministic*, i.e. always produce the same representation given a specific set. An accumulator typically consists of the following algorithms [46]:

- $(\mathsf{pp}, D_0) \leftarrow \mathsf{AccSetup}(\lambda_{acc})$ generates the public parameters and instantiates the accumulator initial state D_0.

- $\mathsf{Add}(D_t, x) := (D_{t+1}, \mathsf{upmsg})$ adds x to accumulator D_t, which outputs D_{t+1} and upmsg which enables witness holders to update their witnesses.
- $\mathsf{MemWitCreate}(D_t, x, S_t) := w_x^t$ Creates a membership proof w_x^t for x where S_t is the set of elements accumulated in D_t. $\mathsf{NonMemWitCreate}$ creates the equivalent non-membership proof u_x^t.
- $\mathsf{MemWitUp}(D_t, w_x^t, x, \mathsf{upmsg}) := w_x^{t+1}$ Updates membership proof w_x^t for x after an element is added to the accumulator. $\mathsf{NonMemWitUp}$ is the equivalent algorithm for non-membership.
- $\mathsf{VerMem}(D_t, x, w_x^t) := \{0, 1\}$ Verifies membership proof w_x^t of x in D_t.

The basic security property of accumulators is *soundness* (or else *collision-freeness*) which states that for every element *not* in the accumulator it is infeasible to prove membership.

Some well-known types of accumulators are the RSA accumulator [47] (which are additive and universal and Merkle Trees [48] (which are additive and positive). We note that RSA accumulator can become trapdoorless if a trusted party (or an MPC protocol) is used to compute the primes for the modulo n, or a public RSA challenge number (i.e. from RSA Labs) is adopted. In addition, "batching" techniques can be applied in element additions and membership proofs [47]. In Section 3.5 we discuss comparisons and trade-offs between these two accumulator types considering different implementation scenarios.

MACs A circuit-based one-time Message Authentication Code (MAC) on x is defined as $t = ax + b$ where a and b are randomly sampled by the verifier, and can be opened after the prover has committed to t [49].

Oblivious Transfer *1-out-of-2 oblivious transfer (OT)* is a fundamental functionality in secure computation between a sender S that holds two values v_0, v_1 and a receiver R. At the end of the protocol, the receiver learns exactly one of the sender values while the sender learns nothing. *1-out-of-n OT* is a generalized version of 1-out-of-2 OT where the

sender has n values, and the receiver learns one of them. In Figure 2.1 we describe the ideal functionality for 1-out-of-n OT.

Functionality $\mathcal{F}_{\text{OT}}^{1:n}$

Functionality $\mathcal{F}_{\text{OT}}^{1:n}$ communicates with sender S and receiver R, and adversary $\mathcal{A}$.

1. Upon receiving input $(sid, v_1, \cdots, v_n)$ from S where $v_i \in \{0,1\}^\kappa$, record $(sid, v_0, \cdots, v_n)$.
2. Upon receiving (sid, i) from R where $i \in \{1, \cdots, n\}$, send v_i to R. Otherwise, abort.

Figure 2.1: The 1-out-of-n OT functionality.

MPC in the Head An MPC protocol $\Pi_{\mathcal{F}}$ is an interactive protocol between m parties $P_1, \ldots, P_m$ to securely compute some function $\mathcal{F}$ on the joint input of all parties. The MPC-in-the-Head paradigm (introduced by Ishai et al. [50]) considers a single party which simulates the execution of all m parties locally and records transcripts of the interaction between the simulated parties. These simulated views can later be selectively opened to prove statements about the inputs of the simulated parties.

Formally, we require the following properties for $\Pi_{\mathcal{F}}$ to be an admissible protocol for MPCitH:

Definition 6. *Let $\Pi_{\mathcal{F}}$ be an MPC protocol for a functionality $\mathcal{F}(x_1, \ldots, x_m)$.*

- *We say $\Pi_{\mathcal{F}}$ realizes $\mathcal{F}$ with* correctness *if for all possible inputs the probability that the output of any party P_j running (semi-honest) $\Pi_{\mathcal{F}}$ is different from $\mathcal{F}(x_1, \ldots, x_m)$ is negligible in λ.*
- *We say $\Pi_{\mathcal{F}}$ realizes $\mathcal{F}$ with* t-privacy *if for all sets of (semi-honest) corrupt parties $I \subset \{P_1, \ldots, P_m\}$ s.t. $|I| \leq t$ there exists a PPT simulator $\mathcal{S}$ s.t. for all inputs the set of views $\{view_j\}_{j \in I}$ is statistically indistinguishable from $\mathcal{S}(I, \{x_j\}_{j \in I}, \mathcal{F}(x_1, \ldots, x_m))$.*

We model the local simulation of the MPCitH protocol with the black-box functionality Π_{MPCitH} in Figure 2.2.

Π_{MPCitH}

Input: An m-party MPC protocol $\Pi_{\mathcal{F}}$ implementing the functionality $\mathcal{F}$ which takes as input (X_j, Y_j) from each party P_j and outputs to all parties $\mathcal{F}(\oplus_{j=1}^{m} X_j, \oplus_{j=1}^{m} Y_j)$ with completeness and $(m-1)$-privacy, (X_j, Y_j) for $j = 1, \ldots, m$

Protocol: Run a simulation of $\Pi_{\mathcal{F}}$ as follows:

- Set each party P_j's input as (X_j, Y_j)
- Sample P_j's initial randomness r_k
- Set $view_j \leftarrow \{X_j, Y_j, r_j\}$
- Execute the steps $\Pi_{\mathcal{F}}$, adding each message received by P_j to $view_j$
- Add the output to each $view_j$

Output: $(view_1, \ldots, view_m)$

Figure 2.2: MPC-in-the-Head subroutine.

Garbled Circuits A Garbled Circuit (GC) scheme is defined by the following algorithms [49]:

- $e, d, GC \leftarrow \mathsf{Gb}(\mathsf{pp}, \lambda, f, r)$ which on input of a boolean circuit f outputs a garbled circuit GC and encoding and decoding information e and d.
- $X \leftarrow \mathsf{Enc}(e, x)$ which on input of encoding information e and an input x corresponding to f, outputs a garbled input X.
- $Y \leftarrow \mathsf{Eval}(GC, X)$ on input of garbled circuit GC and garbled input X outputs encoded output Y.
- $y \leftarrow \mathsf{Dec}(Y, d)$ which on input of an encoded output Y and decoding information d outputs y.
- $\{d, \emptyset\} \leftarrow \mathsf{Ver}(GC, e, f)$ which on input of garbled circuit GC, encoding information e and boolean function f outputs either decoding information d or $\emptyset$.

Definition 7. *A GC scheme* $(\mathsf{Gb}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec}, \mathsf{Ver})$ *must satisfy the following properties:*

- Correctness: $\forall f, x, GC, e, d$: *(a) For* $Y \leftarrow \mathsf{Eval}(GC, X)$, $f(x) \leftarrow \mathsf{Dec}(Y, d)$ *and (b) for* $d \leftarrow \mathsf{Ver}(GC, e, f)$ *with* $d \neq \emptyset$ *and* $Y \leftarrow \mathsf{Eval}(GC, X)$, $\mathsf{Dec}(Y, d) = f(x)$.

- Authenticity: $\forall f, x$ *and PPT algorithm* $\mathcal{A}$, *there is a negligible function* $\epsilon(\cdot)$ *s.t.* $\Pr[\exists y \neq f(x), y = \mathsf{Dec}(d, d') : (e, d, GC) \leftarrow \mathsf{Gb}(\mathsf{pp}, \lambda, f, r), d' \leftarrow \mathcal{A}(GC, \mathsf{Enc}(e, x))]| \leq \epsilon(\lambda)$.

- Privacy: *There exists a PPT simulator* $\mathcal{S}$ *s.t. the distributions* $(\mathsf{Gb}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec}, \mathsf{Ver})$, $X \leftarrow \mathsf{Enc}(e, x)$ *and* $\mathcal{S}(f, f(x))$ *are indistinguishable.*

2.2 Blockchain primitives

Byzantine Generals Problem. Leslie Lamport [51] was among the first to study the problem of achieving agreement on a distributed (synchronous) system. His work first considered a set of n inter-connected parties wishing to agree on a common output, while possibly having conflicting initial inputs and under the assumption that up to f "Byzantine" parties might behave maliciously. In this "Byzantine Agreement" (BA) problem, binary consensus (i.e. for an output of 0 or 1) can be achieved with $n = 3f + 1$ nodes using unauthenticated messages in rounds, where a leader sends a proposed binary value in a fully connected network, then the honest nodes would propagate that value using the same protocol acting as leaders themselves. Authenticated messages improves the resilience of the protocol to $n = 2f + 1$, as Byzantine faults can be tolerated under the assumption that dishonest parties cannot forge the leader's signature. In both cases however, having a communication complexity $O(n^2)$, the protocol does not scale to a large number of parties.

Blockchain. A blockchain is ledger, consisting of a series of blocks linked through some cryptographic primitive. In the vast majority of cases, the linking between blocks is implemented through a hash function, and the blockchain is append-only, making it immutable. Also in most applications, the blockchain is publicly accessible, i.e. a public ledger.

Consensus protocols. A consensus protocol (denoted by CN) allows a set, S_{CN}, of distributed parties to reach agreement in the presence of Byzantine faults. For the purpose of this thesis, we consider agreement in the data posted on a ledger L. Participation in the consensus protocol can be either *permissioned* (i.e. only authenticated parties have write access in the ledger) or *permissionless* (i.e. any party can write in the ledger as in most blockchain-based systems like Bitcoin or Ethereum). More formally, a ledger consensus protocol $\mathsf{Consensus}(x, L) := L'$ allows system participants given some input value x and ledger state L, to agree on a new ledger L'.

The two fundamental properties of ledger consensus are [52, 53]: (a) Consistency: An honest node's view of the ledger on some round j is a prefix of an honest node's view of the ledger on some round $j + \ell, \ell > 0$. (b) Liveness: An honest party on input of a value x, after a certain number of rounds will output a view of the ledger that includes x.

Definition 8. *Let parties* $[P_i]_{i=1}^{n}$*, each having a view of the blockchain* $\mathsf{BC}(i)$*, and receive a common sequence of messages in rounds* $[1..j..]$ *A protocol solves the ledger consensus problem if the following properties hold:*

i. Consistency: An honest node's view of the blockchain on some round j is a prefix of an honest node's view of the blockchain on some round $j+\ell, \ell > 0$, or $\mathsf{BC}(i)_j||\mathsf{B_{j+1}}||...||\mathsf{B_{j+\ell}} = \mathsf{BC}(i')_{j+\ell}, \forall P_i, P_{i'}, j, \ell$.

ii. Liveness: An honest party P_i on input of an operation (or transaction) tx*, after a certain number of rounds will output a view of the blockchain* $\mathsf{BC}(i)$ *that includes* tx.

The protocol can be augmented with the existence of a TP assigning membership credentials (which requires an additional trusted setup phase) resulting in an *Authenticated* ledger consensus protocol [51][54]. Such a protocol consists of the following algorithms:

1. $(\mathsf{pp}) \leftarrow \mathsf{TPSetup}(1^{\lambda})$: A trusted party TP generates the system paremeters pp.

2. $(\mathsf{pk_i}, \mathsf{sk_i}) \leftarrow \mathsf{PartyGen}(\mathsf{pp})$: Each party i generates a public-private key pair.

Table 2.1: Consensus algorithm comparison

Algorithm	Adversarial model	Byzantine tolerant	Dynamic membership	Scalable	DoS resistant
PBFT	$3f+1$	✔	×	×	Semi
Kafka	$2f+1$	×	✔	✔	×
BFT-SMaRt	$3f+1$	✔	✔	Semi	✔
Nakamoto consensus	$2f+1$	✔	Permissionless	✔	✔

3. $\mathsf{TPMembers}(\widehat{[\mathsf{pk}]}) := [\mathsf{pk}]$: The TP chooses the protocol participants from a list of public keys $\widehat{[\mathsf{pk}]}$, and outputs a list of authenticated participant public keys $[\mathsf{pk}]$.

4. $\mathsf{Consensus}([[\mathsf{pk_j}]_{j=1}^{n}, \mathsf{sk_i}, \mathsf{st_i}, \mathsf{BC}(i)]_{i=1}^{n}) := \mathsf{BC}'$: All system participants with state $\mathsf{st_i}$ and a view of the blockchain $\mathsf{BC}(i)]_{i=1}^{n}$), agree on a new blockchain BC'.

Practical Byzantine Fault Tolerance. Practical Byzantine Fault Tolerance [55] was an exemplary consensus protocol for many others to follow, including ledger-based ones. It assumes a partially asynchronous setting (i.e. offering *eventual synchrony*), where a message is assumed to arrive at the destination node in the network after some unknown but bounded time t. In each round, a leader orders messages and propagates them to all nodes in the network in three phases. The leader is defined by a sequence of "views", and the backup nodes can propose a leader (or view) change if he has faulty behavior (timeout exceeded). The protocol assumes $n = 3f + 1$, and has communication complexity $O(n^2)$, which again is in practice not scalable for more than 20 nodes [56]. It also assumes static membership of participating nodes.

Consensus algorithm categorization. Consensus algorithms can be categorized through the following properties and assumptions:

1. Byzantine vs. Crash Fault Tolerant: The crash tolerant model of consensus does not take Byzantine (i.e. malicious) behavior of nodes into account [57].

2. Synchronicity assumptions.

3. Incentive-compatible, i.e. motivations to participate in the system and follow its rules [58].

4. Setup assumptions.

Nakamoto consensus. First introduced with Bitcoin [59], Nakamoto consensus [60] was the first to enable agreement in a permissionless setting, with parties joining and leaving the system. Its most prominent instantiation is Proof of Work, a computationally-intensive puzzle where participants' voting power is reflected by the ability to find a hash output with specific properties in a brute-force fashion. Nakamoto consensus properties [52] include: a) *common prefix*, i.e. blockchains maintained by the honest parties will possess a large common prefix assuming suitably bounded adversarial power, and b) *chain quality*, i.e. a bounded ratio of ratio of adversarial blocks in the chain of any honest player.

Blockchain layers. Blockchain based systems (e.g. cryptocurrencies) can be studied in a layered approach [61]. The most common layers considered are the following:

- Network layer, which considers the sharing of the blockchain state, transactions and consensus-related communication in a peer-to-peer fashion.

- Consensus layer, which considers the consensus algorithm and its elements, needed to reach agreement on the blockchain state.

- Application (or transaction) layer, which considers the application-specific elements such as account addresses, signatures and smart contracts.

Using this layered approach we can then consider potential blockchain security vulnerabilities and mitigations, for instance eclipse attacks in the network layer [62], adversarial voting power in consensus algorithms [52] or deanonymization techniques [3].

Chapter 3: A new private and auditable distributed payment scheme

3.1 Introduction

As discussed in Chapter 1, privacy preserving distributed payment schemes focus on improving user privacy, but typically at the cost of generating new regulatory concerns, as in a "private" ledger, the transactions cannot be subject to any level of auditing (e.g. tracing illegal behaviors).

In this chapter we present MiniLedger as a solution for a distributed privacy-preserving and auditable payment system. MiniLedger not only guarantees the privacy of transactions, but also offers built-in functionalities for various types of audits by any external authority. It is the *first* private and auditable payment system with storage costs independent of the number of transactions. To achieve such a storage improvement, we introduce pruning functionalities for the transaction history while maintaining integrity and auditing. We provide formal security definitions and a number of extensions for various auditing levels. Our evaluation results show that MiniLedger is practical in terms of storage, requiring as low as 70KB per participant for 128 bits of security, and depending on the implementation choices, can prune 1 million transactions in less than a second.

The work presented in this chapter has been published in [26].

Our contributions. MiniLedger is the *first space efficient*, distributed private payment system that allows an authorized set of participants to transact with each other, while also allowing for a wide set of auditing by consent operations by *any* third party auditor. We provide formal, game based definitions and a construction that relies upon a number of cryptographic primitives: a consensus protocol, semi-homomorphic encryption, compact set

representation techniques (cryptographic accumulators) and non-interactive zero-knowledge proofs (NIZKs).

At a high level, MiniLedger consists of n Banks transacting with each other through a common transaction history, or else a ledger L which is maintained by a consensus mechanism (orthogonal to our work). The ledger is modeled as a two-dimensional table with n columns, one for each participating Bank, and rows representing transactions. Whenever Bank B_j wishes to send funds of value v to another B_k, it creates a n-sized vector containing (semi)homomorphic encryptions and NIZK proofs which is appended in L. B_j encrypts the value that is sent to each participating Bank in the system using each receiving Bank's public key, i.e. the encrypted values would be v for B_k, $-v$ for B_j and 0 for any other Bank. These encryptions provide privacy in MiniLedger since they hide values as well as the sender and recipient of each transaction, while still allowing all participating Banks to decrypt the value that corresponds to them and to compute their total assets at any point. This overcomes the need for any out-of-band communication between Banks which created security issues in previous works (ref. Section 3.3.2). Finally, the included ZK proofs guarantee that transactions are valid without revealing any information.

MiniLedger provides auditability *by consent*. Any third party auditor with access to L can ask audit queries to a Bank and verify the responses based on the public information on L. The simplest audit is to learn the value of a cell in L, i.e. the exact amount of funds a Bank received/sent at any point. This basic audit can be used to derive more complex audit types as we discuss in Section 3.4.2, such as transaction history, account balance, spending limit etc., without disclosing more information to the auditor than needed.

Space Efficiency. The main innovation of MiniLedger lies in the maintenance and storage of L. In previous auditable schemes (such as zkLedger [19]) the *full* L needs to be stored at *any time* and by *all* participants. The challenge in MiniLedger design was compacting the ledger while maintaining security and a wide set of auditing functionalities. As noted above, completely erasing transaction information would make auditing impossible (since an auditor cannot possibly audit something that no longer exists). MiniLedger employs

a smart type of transaction pruning: participating Banks can prune their own transaction history by computing a provable, *compact representation* of their previously posted history and broadcast the resulting digest to the consensus layer. Once consensus participants agree to a pruning operation (i.e. verify the digest as a valid representation of the Bank's history), that history can be erased from L and thus by all system participants (except the pruning Bank itself which always need to store its own transaction history locally). Auditing is still possible since a compact digest of transaction information is always stored in L and the Bank under audit can prove that the revealed values correctly correspond to the digest. As a result, the size of L in MiniLedger can be nearly constant (i.e. independent of the number of transactions that ever happened).

Our compact transaction history representation can lead to multiple additional benefits (besides obviously reduced storage requirements). First, a compact L makes addition of new system participants (i.e. Banks) much more efficient (typically, new parties need to download the whole L requiring large bandwidth and waiting time). Then, although the structure of L does not allow for a very large number of participating Banks n (as the computation cost of a single transaction is linear in n), the compactness of L allows augmenting MiniLedger with more fine-grained types of auditing and enabling audits in a client level (instead of a Bank level). We present MiniLedger+, an extension that serves a much larger user base in Section 3.4.1.

Finally, we implement a prototype of the transaction layer of MiniLedger and evaluate its performance in terms of transaction costs, pruning costs and size of L which we estimate to be as low as 70KB of storage for each Bank. We show that transaction computation cost, for a system with 100 Banks, takes about 4 sec, while transaction auditing is less than 5 ms, independent of the number of Banks. Transaction computation costs increase linearly to the number of Banks (as in zkLedger) but by optimizing the underlying ZK proofs we achieve some small constant improvement. Although the linear transaction computation cost might still seem high, we note that using our MiniLedger+ extension, a small number of Banks can serve a very large user base. We perform experiments on two different types of pruning

instantiations, one using Merkle trees [48] and one using batch RSA accumulators [47]. Both result in pruning measurements that are independent of the number of participating Banks. Our experiments show that we can prune 1 million transactions in less than a second using Merkle trees and in about 2 hours using the RSA accumulator, and can perform audits in milliseconds in the same transaction set. We also show that we can audit multiple transactions at a time more efficiently with the RSA accumulator, and can create audit openings for all transactions in less than a millisecond with a single exponentiation, assuming pre-computation of the necessary witness. As we show in Section 3.5, the above trade-offs between the two instantiations suggest that the eventual choice is up to the deployment use-case of MiniLedger.

Related Work. We present an non-exhaustive overview of related works, and point the reader to our Systematization of Knowledge work [63] for a more extensive review.

Anonymous distributed payment systems. Zcash [5], is a permissionless protocol hiding both transacting parties and transaction amounts using zero knowledge proofs. Other notable systems are CryptoNote and the Monero cryptocurrency [6], based on decoy transaction inputs and ring signatures to provide privacy of transactions within small anonymity sets, and Quisquis [7] which provides a similar anonymity level to Monero but allows for a more compact sized ledger. Zether [64] is a smart contract based payment system which only hides transaction amounts. Mimblewimble [65] uses Confidential Transactions [66] to hide transaction values in homomorphic commitments, and prunes intermediate values from the blockchain after being spent (which might be insecure in other UTXO systems such as Bitcoin), improving its scalability. In the permissioned setting, Solidus [8] allows for confidential transactions in public ledgers, employing Oblivious RAM techniques to hide access patterns in publicly verifiable encrypted Bank memory blocks. This approach enables users to transact in the system anonymously using Banks as intermediaries.

Adding auditability/accountability. A number of Zcash extensions [18, 68, 69] proposed the addition of auxiliary information to coins to be used exclusively by a designated, trusted authority for accountability purposes. While this allows for a number of accountability

Table 3.1: Confidential payment schemes comparison. By $\checkmark^S$ we denote set anonymity, $\checkmark^T$ auditing through a TP and $\checkmark^K$ through "view keys" (which reveal all private information of an account). By O: permissionless and C: permissioned we refer to the set of parties that participate in the payment scheme and not the underlying consensus.

	Record	Anonymity	Auditing	Permission	Pruning
Zcash [5, 18]	UTXO	✔	$\checkmark^T$	O	×
Monero [6,67]	TXO	$\checkmark^S$	$\checkmark^K$	O	×
Quisquis [7]	Hybrid	$\checkmark^S$	×	O	✔
MW [65]	UTXO	×	×	O	✔
Solidus [8]	Ac cnt	$\checkmark^S$	×	C	×
zkLedger [19]	Accnt	✔	✔	C	×
PGC [35]	Accnt	×	✔	O	×
Zether [64]	Accnt	option	×	O	×
MiniLedger	Accnt	✔	✔	C	✔

functionalities, it suffers from centralization of auditing power. Additionally, all such works inherit the underlying limitations of Zcash such as the need for trusted setup and strong computational assumptions. Traceable Monero [67] attempts to add accountability features on top of Monero. In a similar idea to Zerocash, a designated "tracing" authority can link anonymous transactions with the same spending party and learn the origin or destination of a transaction. The tracing authority's role can again be distributed among several authorities to prevent single point of failure and distribute trust. PRCash [20] aims to achieve accountability for transaction volume over time. A regulation authority (can be distributed using threshold encryption) issues anonymous credentials to the system's transacting users. If transaction volume in a period exceeds a spending limit, the user can voluntarily deanonymize himself to the authority to continue transacting. PRCash however only focuses on this specific audit type. zkLedger presented a unique architecture for implementing various interactive audit types in a permissioned setting, but its linear-growing storage requirements in terms of transactions make it impractical for real deployment. Additionally, it assumes transaction values are communicated out-of-band, creating an attack vector that could prevent participants from answering audits. Fundamentally, the requirement of communicating values out-of-band defeats the whole purpose of its construction.

We discuss details of these shortcomings in the full version of our paper [70]. We also note an extension to zkLedger using private swaps [71] for supporting asynchronous submission of transactions, which however is orthogonal to our work.

Finally, some works attempt to provide auditability and accountability in an organization rather than in an account level. For instance, [17, 72] propose accountability mechanisms for cryptocurrency exchanges, enabling them to prove their solvency based on Merkle Trees and range proofs. A standard has been recently proposed for such functionalities in [73]. Also [74, 75] proposed accountability solutions for general public records using public ledgers or blockchains.

In Table 3.1 we summarize properties of private payment schemes and refer the reader to [63] for a systematization of knowledge on auditable and accountable distributed payment systems.

Prunable and stateless blockchains. Given the append-only immutability property for most ledgers, the concern for ever-growing storage requirements in blockchains was stated even in the original Bitcoin whitepaper [1], which considered *pruning* old transaction information without affecting the core system's properties. Ethereum [76], being an account-based system, supports explicit support of "old state" pruning as a default option, and defers to "archival" nodes for any history queries. Coda (Mina) [77] is a prominent example of a stateless (succinct) blockchain, which only needs to store the most recent state with recursive verifiability using SNARKs. Accumulators and vector commitments have also been proposed to maintain a stateless blockchain [47, 78]. All such approaches however might negatively impact auditability and are therefore not directly applicable in our setting.

3.2 MiniLedger model

We consider the following system participants: a Trusted Party TP, a set of consensus participants S_{CN}, a static set of n Banks with IDs defined by $[\mathsf{B}_j]_{j=1}^{n}$ (known to everyone) and an arbitrary number of Auditors A. Each Bank has a key pair $[(\mathsf{pk_j}, \mathsf{sk_j})]_{j=1}^{n}$ and an

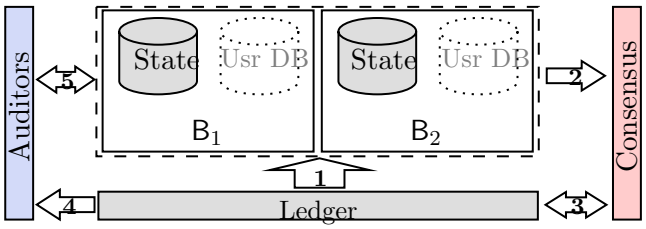


Figure 3.1: MiniLedger overview. "State" is a private database for each Bank and UsrDB is an optional private database for Bank's clients. Banks read from Ledger to create or prune transactions (1) and forward the transaction creation or pruning output to consensus (2). Consensus verifies and updates the Ledger (3). Auditor read Ledger (4) and interact with Banks to audit transactions (5).

initial asset value $[v_j]_{j=1}^n$. Banks maintain an internal state $[st_j]_{j=1}^n$. We denote transactions by tx_i where i represents the transaction's index. We store transactions in a public ledger L maintained by a consensus layer CN and stored by all banks.

We summarize the role of each participant in MiniLedger and provide the architecture overview in Figure 3.1:

- TP is a trusted entity which runs an one-time setup to instantiate the system public parameters and verifies the initial assets of each Bank. TP could be replaced by an MPC protocol (i.e [79]) executed by the Banks.
- Banks generate transactions tx that transfer some of their assets to one or more other Banks, while hiding the value and the transacting parties. Transactions are sent to the consensus layer CN (via an anonymous communications protocol, i.e Tor) and if valid are appended on L. Banks can prune their transaction history on L and "replace" it by a digest D. The pruning Bank needs to send D to CN (incentives for the Bank to perform the pruning operation are orthogonal to our construction) and is responsible[1]to keep a copy of the pruned transactions in its private database "State" . If D is valid, CN participants update L by deleting the pruned transactions and replacing them by D.
- Auditors, by observing the ledger, can audit the Banks at any point for any set of transactions through interactive protocols. Auditors should be able to audit the value

[1]Failure to locally store transaction history can lead to audit failures.

of a single transaction or a subset of transactions, whether these transactions are still in L or have been pruned.

We now state the assumptions required in MiniLedger and then describe our security and privacy goals.

Assumptions. We focus on the transaction layer and consider issues with underlying consensus and network layers and their mitigations orthogonal to this work. Specifically, we assume the fundamental consensus properties, as defined in Sec. 2.2, always hold. On a network level, we assume a malicious Bank cannot block another Bank's view (Eclipse attacks). In addition we assume that transactions are sent to *all* Banks using anonymous communication channels to preserve anonymity of the sending and the receiving Bank(s). We *do not* require out-of-band communication between Banks.

The "race conditions" problem in CreateTx(), where different transactions might be created concurrently, is considered orthogonal to our work. We can either assume all transactions are submitted to consensus for verification in a synchronous manner (i.e. no "mempool" functionality), or can adopt an existing solution that uses an atomic exchange protocol, proposed for zkLedger in [71]. Finally, for the sake of simplicity, we assume the set of participating Banks is static but it is easy to extend our system to dynamically add/remove Banks. We also assume the Random Oracle model to convert our ZK proofs to non-interactive.

Security Goals. MiniLedger should satisfy the following properties (formally defined in a game-based fashion in Sections 3.6.1, 3.6.2):

Theft prevention and balance: When spending, we require that a) transaction is authorized by sending Bank, b) after spending, Bank's balance decreases by the appropriate amount and c) sending Bank cannot spend more than its total assets.

Secure pruning: Ledger pruning, which is executed in a user base, outputs a digest that a) is only created by the respective Bank, b) contains the correct transactions in the correct order, and c) does not contain bogus transactions. Ultimately secure pruning prevents tampering with ledger history.

Ledger correctness: The ledger only accepts valid transactions and pruning operations.

Correct and Sound Auditability: An honest Bank following the protocol can always answer audits correctly and convince an Auditor, while an Auditor always rejects false claims from a malicious Bank.

Privacy: The ledger hides both the identities of transacting parties and values of transactions from any external observer (except auditors who learn specific information during the auditing protocol).

3.3 MiniLedger construction

Overview. We consider n Banks that transact with each other in an anonymous and auditable way by posting data in a common ledger L (a two-dimensional table with n columns, one for each participating Bank, and a number of rows which represent transactions). The ledger is maintained by consensus participants, who verify every submitted transaction, and is stored by all Banks. The Banks could be running consensus themselves, or out source this operation to any external set of consensus parties.

For each tx_i, the sending Bank (i.e. the transaction creator) creates a whole row in L which includes twisted ElGamal encryptions $C_{ij} = (c_1, c_2)$ that hide the transferred value v_{ij} that corresponds to each cell (i, j). For instance, if we assume that there's only one receiving Bank in a transaction, the sending Bank would compute an encryption of $-v$ for its own cell, an encryption of v for the receiver cell, and a number of encryptions of 0 for the rest of the cells. This makes the transmitted values indistinguishable to any external observer due to ElGamal IND-CPA security(assuming the sender uses different randomness values for each encryption). Additionally, the sending Bank accompanies each encryption with a NIZK proof π to prevent dishonest Bank behavior as discussed in details below. This specific ledger structure allows an external auditor to audit for a value sent/received by a Bank at any given point, with the Bank replying with a value v and a ZK proof π^{Aud} for its claim. This basic audit protocol can be extended to more complex queries (such as total

assets held by a Bank or if a transaction exceeds a set limit) as we explain in Section 3.4.2.

A straightforward implementation of such a transaction table, as done in zkLedger, leads to a ledger L that grows linearly to the number of posted transactions. This makes schemes like zkLedger hard to adopt in practice, since every single participant would have to store a table of size n times the total number of transactions that have ever occurred. Besides storage costs, the overall computational performance would also degrade even more over time.

Reducing storage costs. The main idea for MiniLedger, is that each Bank B_j periodically initiates a pruning operation, which prunes (or "compacts") all transactions in its corresponding column on L (this is in contrast to typical consensus pruning, where nodes may be offline and have their transaction history in the public ledger pruned without their consent). When a Bank performs a pruning operation, it has to publish a digest D containing the pruned transactions. The consensus layer will first verify that D is indeed a valid digest, i.e. contains the transactions to be pruned, and then, come to an agreement about the pruning operation. Note that B_j is still responsible for maintaining a private copy of *all* its pruned transactions, however, there are great storage savings for the public version of the ledger L which *everyone* in the system has to maintain. In other words, with each Bank pruning its own transaction history, the whole ledger is effectively "sharded" to all Banks, where each Bank is responsible for maintaining a correct copy of its own history, while the public ledger only contains the compact representation D_j of each B_j's transaction history (as well as a few recent transactions that might have not been pruned yet). We note that the cost of a pruning operation depends on the number of transactions to be pruned but is independent of the number of participating Banks n and can be amortized based on the pruning frequency.

When B_j is audited for a pruned (i.e. not publicly visible) transaction value v_{ij}, it would have to present the needed data to the auditor by recovering it from its private copy of its transaction history. In addition, it would have to prove to the auditor not only that this data is contained in D, but also that it had been posted on the specific row i (i.e. maintain

ordering).

We implement this pruning operation using cryptographic accumulators since they achieve a short, constant size representation of D. We require schemes which are (a) *additive*, i.e. have an update functionality that enables a Bank to prune additional transactions and update an already published D by adding the newly pruned ones, (b) *positive*, i.e. allow for proofs of membership but also capable of providing a "position"/"ordering" proof, and (c) *trapdoorless*, i.e. nobody has a trapdoor to create fake proofs of membership.

3.3.1 Our construction

For our construction we assume the following building blocks: the "twisted" variant of additive ElGamal encryption ($\mathsf{SetupTEG}, \mathsf{GenTEG}, \mathsf{EncTEG}, \mathsf{DecTEG}$), an EU-CMA signature scheme ($\mathsf{SignGen}, \mathsf{Sign}, \mathsf{SVrfy}$), an additive, positive cryptographic accumulator ($\mathsf{AccSetup}$, Add, $\mathsf{MemWitCreate}$, $\mathsf{MemWitUp}$,VerMem) with additional properties as discussed above, the Pedersen commitment scheme (ComGen, $\mathsf{Com}, \mathsf{Open}$), a consensus protocol $\mathsf{Consensus}$ and a NIZK proof system. The phases of MiniLedger work as follows:

Setup: Setup can be executed with the help of a trusted third party or via an MPC protocol amongst Banks.

1. $\mathsf{SysSetup}\{\mathsf{TP}(1^\lambda) \leftrightarrow [B_j(v_j)]_{j=1}^n\}$. This interactive protocol is executed between TP and a set of n Banks. TP sets as κ the number of bits that can represent a value and verifies the initial asset value v_j for each Bank . TP generates the public parameters for the accumulator by running $\mathsf{AccSetup}()$, the key parameters of the ElGamal variant encryption scheme by executing $\mathsf{SetupTEG}()$ (which are also used for the Pedersen commitment scheme), the consensus protocol parameters by running $\mathsf{TPCSetup}$, and the joined set of parameters denoted as pp is sent to all Banks. Each Bank generates an ElGamal key pair $(\mathsf{pk}_{\mathsf{B_j}}, \mathsf{sk}_{\mathsf{B_j}})$ through $\mathsf{GenTEG}()$ and sends $\mathsf{pk}_{\mathsf{B_j}}$ to TP. Finally, TP encrypts the initial values of each Bank by running $C_{0j} = (c_1^{(0j)}, c_2^{(0j)}) \leftarrow \mathsf{EncTEG}(\mathsf{pk}_{\mathsf{B_j}}, v_j)$[2]. Then, it initializes a "running

[2]To simplify notation, from now on we will drop the superscripts from the two parts of Elgamal ciphertext, i.e., we will simply write $C_{0j} = (c_1, c_2)$.

value total" which starts as $Q_{0j} = C_{0j}$ and will hold the encryption of the total assets of each Bank at any point. The vector $[Q_{0j}, C_{0j}]_{j=1}^{n}$ consists of the "genesis" state of the ledger L along with the system parameters pp containing the key parameters and all Bank public keys. At any point, the ledger L is agreed by the consensus participants and we assume that all Banks (whether participating in consensus layer or not) store $L.\mathsf{pp}$ and L are default inputs everywhere below.

Transaction creation:

2. $\mathsf{tx}_i \leftarrow \mathsf{CreateTx}(\mathsf{sk}_{\mathsf{B_k}}, [v_{ij},]_{j=1}^{n})$. This algorithm is run by Bank B_k wishing to transmit some (or all) of its assets to other Banks in L. For each B_j in L (including itself), B_k executes $C_{ij} \leftarrow \mathsf{EncTEG}(\mathsf{pk}_{\mathsf{B_j}}, v_{ij})$ and computes $Q_{ij} \rightarrow Q_{(i-1)j} \cdot C_{ij}$. In order to prove *balance*, similarly to [19], B_k should pick randomness values for the ElGamal variant encryptions such that $\sum_{j=1}^{n} r_{ij} = 0$. Then, the sending Bank B_k generates a NIZK π_{ij} $\forall j \in (1, ..n)$ which proves the following (the exact description of π_{ij} can be found in Section 3.7):

Proof of Assets: Shows that *either* a) B_j is receiving some value ($v_{ij} \geq 0$), *or* b) B_j is spending no more than its total assets ($\sum_{k=1}^{i} v_{kj} \geq 0$) and within the valid range after transaction execution, while proving knowledge of its secret key $\mathsf{sk_j}$ showing it authorized the transfer. In both cases, an auxiliary commitment $\mathsf{cm_{ij}}$ is used which commits to either v_{ij} or $\sum_{k=1}^{i} v_{kj}$, so the proof includes a single range proof for the commitment value to reduce computational costs, as the range proof is the most computationally expensive part of π.

Proof of Consistency: Ensures consistency for the encryption randomness r in c_1 and c_2 in both cases of the previous sub-proof, which guarantees correct decryption by Bank k.

The transaction $\mathsf{tx}_i = [C_{ik}, \mathsf{cm_{ik}}, \pi_{ik}, Q_{ik}]_{k=1}^{n}$ is sent to consensus layer CN.

3. $\mathsf{VerifyTx}(\mathsf{tx}_i) := b_i$. Verify all ZK proofs $[\pi_j]_{j=1}^{n}$, check that $\prod_{j=1}^{n} c_2^{(ij)} = 1$ (proof of balance) and that $Q_{ij} = Q_{(i-1)j} \cdot C_{ij}$. On successful verification output 1, else output $\perp$.

Transaction pruning:

4. $(D_{\beta j}, st'_j, \sigma_j) \leftarrow \mathsf{Prune}(st_j)$ This algorithm is executed by B_j when it wishes to prune

its transaction history of depth $q = \beta - \alpha$ and "compact" it to an accumulator digest $D_{\beta j}$, where α is the latest digest and β is a currently posted row number (usually a Bank will prune everything between its last pruning and the latest transaction that appeared in L). It parses C_{ij} from each tx_{ij}. It fetches its previous digest $D_{\alpha j}$ (if $\alpha = 1$, sets $D \rightarrow D_{\alpha j}$ as the initial accumulator value where A is defined from pp). Then $\forall C_{ij}, i \in [\alpha, \beta]$ it consecutively runs accumulator addition $\mathsf{Add}(D_{(i-1)j}, (i \parallel C_{ij}))$ (note the inclusion of index i which preserves ordering of pruned transactions in D_j). Finally it stores all transaction encryptions $[i, C_{ij}]_{i=\alpha}^{\beta}$ to its local memory, updates st_j to st'_j, computes $\sigma_j \leftarrow \mathsf{Sign}(D_{\beta j})$ and sends $D_{\beta j}, \sigma_j$ to CN. Note that $D_{\beta j}$ does not include proofs π, and pruning breaks proofs of balance in rows for all Banks. Still "breaking" these old proofs is not an issue, as they have already been verified.

5. $\mathsf{PruneVrfy}(D_{\beta j}, \sigma_j) := b_j$ On receipt of $D_{\beta j}$, locally executes $\mathsf{Prune}()$ for the same transaction set to compute $D'_{\beta j}$. If $D'_{\beta j} = D_{\beta j}$ (given the accumulator is deterministic) outputs 1, else outputs $\perp$.

We note that after a *successful* pruning operation (i.e. one that is agreed upon in the consensus layer), all system participants that store L can delete all existing data in cells $(i, j) \forall i < \beta$ and just store $D_{\beta j}$ along with the latest $Q_{\beta j}$.

Consensus protocol: This is handled in the consensus layer CN with its details orthogonal to our scheme. Similar to typical blockchain consensus, participants will only update L with a new tx or D if this is valid according to the corresponding verification algorithms (i.e. in Bitcoin, consensus participants validate transactions before posting them in L).

6. $\mathsf{Consensus}(\mathsf{tx} \text{ or } D) := L'$. Runs the consensus protocol among S_{CN} to update the ledger with a new tx or pruning digest D after checking their validity. If consensus participants come to an agreement, L is updated to a new state L'.

Auditing: Our auditing protocols below include a basic audit for a value v (that has either been pruned or not) and a set's sum of such values (which might be all past transactions, thus auditing Bank's total assets). These audits are interactive and require the Bank's

consent. MiniLedger can support additional audit types and/or non-interactive audits as we discuss in Section 3.4.2.

7. $\mathsf{Audit}\{\mathsf{A}(C_{ij}) \leftrightarrow \mathsf{B}_j(\mathsf{sk_j})\}$ is an interactive protocol between an auditor A and a Bank B_j. In this basic audit, A audits B_j for the value v_{ij} of a specific transaction tx_{ij} (that has not been pruned from L so far), encrypted as C_{ij} on the ledger L. B_j first decrypts the encrypted transaction through $\mathsf{DecTEG}()$ and sends v_{ij} to A, as well as a NIZK π^{Aud} : $\{(\mathsf{sk_j}) : c_2/g^{v_{ij}} = (c_1)^{1/\mathsf{sk_j}}\}(c_1, c_2, v_{ij}, \mathsf{pk_j}, g, h)$. Then A accepts the audit for v_{ij} if π^{Aud} successfully verifies.

8. $\mathsf{AuditSum}\{\mathsf{A}([C_{ij}]_{i=\alpha}^{\beta}) \leftrightarrow \mathsf{B}_j(\mathsf{sk_j})\}$ is an interactive protocol between an auditor A and a Bank B_j. Here A audits B_j for the sum of the values $\sum_{k=\alpha}^{\beta} v_{kj}$ for transactions $\mathsf{tx}_{\alpha j}$ up to $\mathsf{tx}_{\beta j}$ (that have not been pruned from L so far). This protocol is a generalization of the $\mathsf{Audit}\{\}$ protocol outlined above, (with $\mathsf{Audit}\{\}$ having as inputs $(\prod_{i=\alpha}^{\beta} C_{ij})$ and $\prod$ denoting direct product for ciphertexts c_1, c_2), because of ElGamal variant additive homomorphism. Note that although in this protocol the transactions are assumed to be consecutive for simplicity, its functionality is identical if the transactions are "isolated". Also if indices $\alpha = 1$ and β equals to the most recent transaction index (and no pruning has happened in the system), the audit is performed on the Bank's total assets.

9. $\mathsf{AudPruned}\{\mathsf{A}([(i,j)]_{i=\alpha}^{\gamma}, [C_{ij}]_{i=\gamma}^{\beta}) \leftrightarrow \mathsf{B}_j(\mathsf{sk_j})\}$ is an interactive protocol between an auditor A and a Bank B_j, where transactions $[\mathsf{tx}_{ij}]_{i=\alpha}^{\gamma}$ have been pruned from L (and thus the auditor only knows their indices and nothing else), and transactions $[\mathsf{tx}_{ij}]_{i=\gamma}^{\beta}$ which are still public in L (i.e. not pruned) and thus the auditor still sees their encryptions. This protocol generalizes $\mathsf{AuditSum}\{\}$. It allows the auditor to audit B_j for: (a) specific transactions (pruned or not) and, (b) sums of assets (pruned or not). For case (a), besides auditing a transaction with index in $[\gamma, \ldots, \beta]$ which is still in L, the auditor can also audit B_j for a specific transaction that has been pruned from L (i.e. ask: "Which was the value of the i-th transaction?"). The Bank would respond with the corresponding C_{ij} and depending

Table 3.2: MiniLedger Architecture and Pruning.

(a) Ledger state before pruning, assuming B_1 had pruned before at tx_{10}.

	B_1	...	B_n
tx_1 ... tx_9	$D_{9,1}, Q_{9,1}$		...
tx_{10}	$C_{10,1} = (c_1 = \mathsf{pk_1}^{r_{10,1}},\ c_2 = g^{v_{10,1}} h^{r_{10,1}})$ $\pi_{10,1}, \mathsf{cm}_{10,1}, Q_{10,1}$		...
tx_{11}	$C_{11,1} = (c_1 = \mathsf{pk_1}^{r_{11,1}},\ c_2 = g^{v_{11,1}} h^{r_{11,1}})$ $\pi_{11,1}, \mathsf{cm}_{11,1}, Q_{11,1}$		...

(b) Ledger state after B_1 prunes at tx_{12}. Digest $D_{11,1}$ represents $C_{10,1}, C_{11,1}$ and ciphertexts that were represented in $D_{9,1}$.

	B_1	...	B_n
tx_1 ... tx_{11}	$D_{11,1}, Q_{11,1}$		...
tx_{12}	$C_{12,1} = (c_1 = \mathsf{pk_1}^{r_{12,1}},\ c_2 = g^{v_{12,1}} h^{r_{12,1}})$ $\pi_{12,1}, \mathsf{cm}_{12,1}, Q_{12,1}$		...

on the underlying accumulator used, B_j would also provide a proof that C_{ij} is a member of its pruned history D_j with index i. For case (b), an auditor can audit the total (or a range of) assets of B_j no matter what transaction information of B_j remains on L. Auditing total assets works as follows: B_j fetches the stored transaction encryptions $[C_{ij}]_{i=\alpha}^{\gamma}$ from its local memory st_j, computes $[w_{ij}]_{i=\alpha}^{\gamma} \leftarrow \mathsf{MemWitCreate}(D_j, [C_{ij}]_{i=\alpha}^{\gamma}, st_j)$ [3] . Then A reads D_j from L and executes $\mathsf{VerMem}(D_j, (i \parallel C_{ij}), w_{ij}) \forall i \in (\alpha, \gamma)$, outputting $[b_{ij}]_{i=\alpha}^{\gamma}$. For every i, if $b_{ij} == 1$ it executes the $\mathsf{Audit}\{\}$ protocol with C_{ij} as input.

10. $\mathsf{AudTotal}\{\mathsf{A}(Q_{ij}) \leftrightarrow \mathsf{B}_j(\mathsf{sk_j})\}$ is equivalent to $\mathsf{Audit}\{\}$ for auditing B_j's total assets $\sum_{i=1}^{m} v_{ij}$ instead of a single v_{ij}.

We informally state the following theorem for the security of our scheme and provide proof sketches in Section 3.6.3.

Theorem 1. *Assuming the security of the ElGamal variant, Pedersen commitments, NIZK, Accumulators and Consensus properties,* MiniLedger *construction satisfies our security definitions as given in Section 3.6.2.*

[3] When using batch RSA accumulator as we discuss later, we don't send a set of witnesses but a single witness for all encryptions.

3.3.2 Discussion and comparisons

Although MiniLedger architecture resembles zkLedger [19], there exist crucial differences that make MiniLedger superior both in terms of efficiency and security. We give an overview below, and a thorough analysis in the full version of our paper [70].

Storage. As already discussed, MiniLedger by leveraging consensus properties applies a pruning strategy which achieves $O(n)$ storage requirements for L, compared to $O(mn)$ for zkLedger (where m is the total number of transactions ever happened, and is a monotonically increasing value).

Security. MiniLedger does not require any out-of-band communication, as all needed information is communicated through the ledger using encryptions. On the other hand, zkLedger assumes if a Bank is actually receiving some value in a transaction, it should be notified by the sending Bank and also learn the associated value (which was hidden in the commitment) through an out-of-band channel. zkLedger however, does not require receiving Banks to be directly informed on the randomness (i.e. commitment $\mathsf{cm}_{\mathsf{ij}}$ is never opened), since they can still answer the audits correctly using the audit tokens, provided that it knows its total assets precisely. This assumption is very strong and can potentially lead to attacks, such as the *"unknown value"* attack where a malicious sending Bank informs the receiving Bank on a wrong value (or does not inform it at all), which then prevents the receiving Bank from answering audits or even participating in the system. More importantly, with transaction values communicated out-of-bank, the randomness could be included with them as well. This would make the system trivial and defeat the purpose of most of its architecture, as the ledger would consist of just Pedersen commitments and proofs of assets. In this version the above attack would not work assuming all Banks are *always* online and verify the openings in real time, which is also a very strong assumption.

Computation. MiniLedger optimizes ZK proof computation over zkLedger by combining disjunctive proof of assets and proof of consistency into a single proof, giving an efficiency gain of roughly 10% in space and computation. (We note that this optimization could also benefit zkLedger as discussed in Section 3.7.)Additionally, while zkLedger's computation

Table 3.3: MiniLedger+ public ledger L. The extra information to be stored is denoted in blue color.

	B_1	...	B_j	...
...				
tx_i			$C_{ij} = (c_1 = \mathsf{pk}_1{}^{r_{ij}}, c_2 = g^{v_{ij}} h^{r_{ij}})$ $\pi_{ij}, \mathsf{cm}_{ij}, D_{ij}, Q_{ij}, R_{ij}, H_{ij}$	

performance degrades over time (as the monotonically-increasing ledger requires more operations to construct transactions), MiniLedger through the running totals Q achieves steady optimal performance.

On setup parameters. We argue that even with the use of a TP, the trust level is rather low. The parameters of ElGamal are just random generators (similar to Pedersen commitments in zkLedger) and for certain accumulator instantiations (such as Merkle trees) there is no trapdoor behind the parameter generation. Finally, the consensus setup essentially consists of choosing trapdoorless parameters (i.e. block specifics etc) and the set of participating parties. Thus, the only trust placed in TP is to pick a valid set of participants – something that all participants can check, exactly as in zkLedger. In comparison to zkLedger (given that ElGamal parameters are the same as Pedersen commitment parameters), the only additional setup is that of the accumulator which as discussed, for certain instantiations can be completely trapdoorless.

3.4 MiniLedger security and extensions

3.4.1 Adding clients for fine-grained auditing

The compact nature of MiniLedger allows for fine-grained auditing where Banks represented on the ledger can serve as an intermediary for a set of clients, allowing them to exchange values using their Banks as intermediaries. We overview this system which we call MiniLedger+ below and provide its detailed construction in Section 3.8.

Protocol overview. Each Bank B_j maintains a *private* ledger of clients L_{B_j} (denoted as

"UsrDB" in Figure 3.1), independent of the public ledger L. For each client m, B_j stores its transactions in encrypted format. These clients can be dynamically added or removed from L_{B_j}. Inspired by [8], each client is uniquely and publicly associated with a single Bank as $f(\mathsf{pk_{jm}}) = \mathsf{B}_j$ where f is a well-defined mapping of public keys pk to Banks.

When a client of B_s wishes to transfer value v to another client in the system, she creates a transaction that includes a transaction id, encryptions of the recipient client's pk, the receiver's Bank B_r and v, as well as appropriate NIZKs to prove consistency with the protocol. This information is recorded on the private ledger L_{B_s}. Then, B_s, constructs a transaction on L that transfers v to B_r. In addition to the transaction information required for standard MiniLedger, *each* cell will also contain an encryption of the recipient client's pk under $\mathsf{pk}_{\mathsf{B}_r}$, the transaction id, and NIZKs that prove that the correct value is sent to the correct Bank. All this information will be concatenated and represented by R_{ij} in L (as shown in Table 3.3). B_s also computes a digest H of the constructed transaction using a collision-resistant hash function $H()$ and includes it in all cells of the respective row. Note that information across rows in R, H is redundant but necessary to preserve ledger indistinguishability. Finally, the receiving B_r will perform the reverse steps to allocate the value to its client. B_s might elect to aggregate many transactions of its clients into a single one on L (recall that a transaction can have multiple receiving Banks, details in Section 3.8.4).

In MiniLedger+, monetary values are not owned by the clients themselves, but are co-managed with the client and his respective Bank (as it happens in the actual banking system), meaning that an honest Bank having total assets Σv represented in L, will always have them distributed internally to its clients. As a result, Σv reflected in a column in L should always match the sum of value sums for all clients in L_{B_j}. However since L_{B_j} is private, the mechanism ensuring that this invariant holds will be *reactive*, in contrast with the main ledger L where the mechanism is proactive. In other words, an auditor would have to perform regular audits of the Banks to ensure they allocate the funds to their clients correctly and follow the protocol. In Section 3.8 we describe this audit in detail, as well as

the extended threat model of MiniLedger+.

Auditors in MiniLedger+ can also perform audits at a client level as follows. Auditor first asks B_j to present its private L_{B_j} and fully validates its correctness and consistency with L as before. Then, the auditor performs audits on clients in a similar fashion as with Banks in standard MiniLedger (i.e. audit single transactions, sums etc.). Note that although the client is responsible for answering its audits correctly, B_j would be responsible for any inconsistencies between L_{B_j} and L. We provide more details in Section 3.8.3.

Comparisons. MiniLedger+ has minimal storage overhead in L compared to MiniLedger. Note that the additional entries R and H in each column can still be pruned as in MiniLedger, thus maintaining a ledger of constant size. The additional computation costs associated with this extension are linear in the number of clients a Bank has, but can be made more efficient by aggregating techniques. We provide an overview of these costs in Table 3.6 under Section 3.8.

Solidus [8] has a similar Bank-Client architecture but does not hide the identity of transacting parties and does not offer auditability functionalities while employing expensive ORAM operations. RSCoin [80] also has a similar paradigm of "mintettes" associated with clients but without any privacy guarantees. Finally, [81] proposes a privacy-preserving payment UTXO-based system (as opposed to account-based in our work), which also supports fine-grained auditing at a user level. In this work, there exists a powerful designated auditor for each user, who can decrypt all user's transactions, making system-wide deanonymization possible in case auditors collude. The auditing costs are much greater compared to MiniLedger+ and the protocol is overall much more complex while it does not support different audit types such as total assets or value thresholds.

3.4.2 Additional types of audits

As shown in Section 3.3.1, MiniLedger basic audit functionality $\mathsf{Audit}\{\}$ is on the value v_{ij} of specific transaction tx_{ij}. Several more audit types can be constructed which reduce to that basic audit. We discuss some of those below, and provide more details for audit

extensions in Section 3.9.2.

Full transaction audit: For an auditor to learn the full details of a transaction (sender, receiver and values), they would have to audit the entire row (i.e. perform n audits on v_{ij} $\forall j$).

Statistical audits: Audits such as average or standard deviation are supported by utilizing "bit flags" to disregard zero-value transactions, proved for correctness in zero knowledge.

Value or transactions exceeding limit: Utilizing appropriate range proofs, an auditor can learn if a sent or received value exceeds some limit t. Multiple range proofs can show a Bank has not exceeded the limit over a time period.

Transaction recipient: The goal of this audit type is for a sending Bank to prove the recipients for one of its transactions. While a Bank doesn't know (and therefore cannot prove) where a *received* value came from (unless learning it out-of-band as in zkLedger), for *outbound* transactions the Bank can keep an additional record of its transaction recipients in its local memory. As an example, for proving in tx_i that the Bank really sent v_{ij} to B_j, it could send this claim to the auditor who in turn would simply then audit B_j to verify this claim.

Client audits: Audits in a client level (e.g. statistical audits or transaction limits) can be performed similar to the respective audits in a Bank level, however the auditor needs first to learn and verify the Bank's private ledger L_B as discussed in Section 3.4.1.

3.5 Evaluation

We implement a prototype of the transaction layer of MiniLedger in Python using the *petlib*[4] library to support cryptographic operations over the secp256k1 elliptic curve. We use the *zksk* library [82] for the ZK and implement range proofs using the Schoenmakers' Multi-Base Decomposition method [83][5]. The measurements were performed on Ubuntu 18.04 -

[4] `https://github.com/gdanezis/petlib`

[5] By using twisted ElGamal [35], MiniLedger is fully-compatible with Bulletproofs [41] which can further reduce its concrete storage requirements.

i5-8500 3.0 GHz CPU - 16GB RAM using a single thread[6]. As we focus on the transaction layer, we do not include measurements of the underlying consensus and network layers as they are orthogonal to our work. We do however discuss the potential additional costs in Section 3.5.

Accumulator Instantiation. A critical implementation choice is how to instantiate the accumulator needed for the pruning operations. For efficiency reasons, we require schemes with constant size public parameters and no upper bound on the number of accumulated (i.e pruned) elements. We only consider schemes that have at most sublinear computation and communication complexity (in the number of pruned elements) for opening/proving a single transaction to the auditor and where the auditor's verification cost is also at most sublinear.

We first consider Merkle trees [48]. Assuming a Bank prunes q transactions, the Merkle root provides $O(1)$ representation in terms of storage with $O(1)$ public parameters. Opening and verification complexity of Merkle proofs for a single transaction audit involves $O(\log q)$ hashing operations. However although hashes are relatively cheap operations, the over-linear verification complexity might be a concern when auditing a series of transactions. Finally, it should be noted that Merkle trees only support membership proofs.

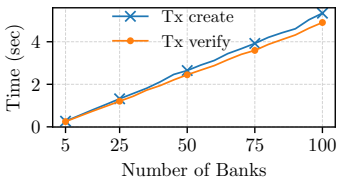
We then consider the batch-RSA accumulator [47]. Given that all RSA typeaccumulators can only accumulate primes p, we use a deterministic prime mapping hash function (as in [47]) to enable accumulation of arbitrary inputs. The batch-RSA accumulator has $O(1)$ storage for its digest with $O(1)$ public parameters as well. Proving membership for a single element p in the standard RSA accumulator, requires the prover computing a witness w equal to the primes' product in the accumulator without p (an $O(q)$ operation as shown in Figure 3.4), and the verifier checking that $(g^w)^p = A$ where A is the current state of the accumulator. However, in batch-RSA, the prover can reduce computation costs by "batching" these operations for a set of elements $(p_1, p_2, ...)$ which are in the accumulator and provide a Proof of Knowledge of Exponent $\pi^{PoKE} : \{(x) : (g^w)^x = A\}(w, g, A)$ on the

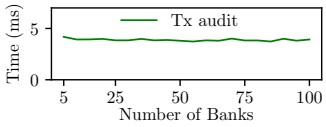
[6]A basic implementation of MiniLedger is available at `https://github.com/PanosChtz/Miniledger`

product $x = p_1 p_2 ...$ convincing a verifier without sending x (which is typically large) and thus reducing communication costs. In our setting we use batch-RSA in auditing in order to improve computational efficiency of membership proofs (when being audited for multiple transactions) and we note that the Bank under audit does not need to include a PoKE, as the auditor needs to recompute the prime mapping of the ciphertexts he is given by the Bank anyway (this is a trade-off between PoKE computation cost for the Bank and higher communication cost between Bank and Auditor). Through these techniques, batch-RSA achieves the same complexity $O(q)$ as when proving membership of a single element (while Merkle Trees have $O(\ell \log q)$ complexity for ℓ elements). Consequently, the basic pruning operations Prune() and PruneVrfy() are about two orders of magnitude more expensive compared to Merkle trees as we show in Figure 3.3. However they are efficient when auditing large amounts of transactions especially if auditing the total sum. Then, batching allows for negligible computation costs for the proving Bank, and negligible audit verification cost for a single transaction (which can enable audit extensions as discussed in Section 3.8). Thus, choice between Merkle trees and batch-RSA accumulators ultimately relies on the use-case requirements.

Finally, an alternative approach is the use of Vector Commitments [47, 84, 85]. However these constructions although offering additional properties, either have linear or quadratic public parameter costs [84,85] or are more expensive when proving and verifying membership compared to Merkle trees or RSA accumulators, as we show on Table 3.7 in Section 3.10.

For our batch-RSA implementation, we use the SHA-256 hash function and the Miller - Rabin primality test for hashing to prime numbers, and we use an RSA-3072 modulo to maintain the same level of security [86]. We decouple the witness computation cost from the proof of membership cost for the Bank, as the Bank might elect to pre-compute the witnesses before its audit (assuming however that it does not prune again until the audit, since that would require the witnesses to be recomputed again). Note the auditor needs to run the hashing to prime mapping function again for all audited values (i.e. the auditor cannot rely on the "honesty" of a Bank presenting pre-computed prime numbers for its





Create	Verify	Storage	Params
53ms	49ms	68KB	2KB

Costs per Bank

Figure 3.2: Transaction creation, verification and auditing costs.

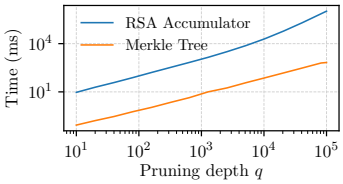


Figure 3.3: Pruning computation cost

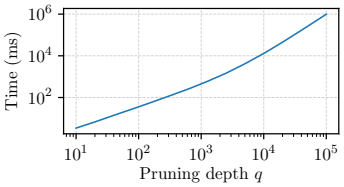


Figure 3.4: RSA witness Generation cost

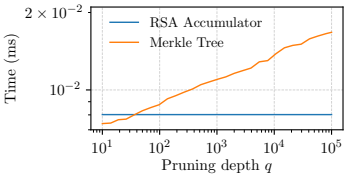


Figure 3.5: Audit open cost for one tx

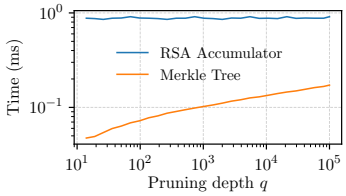


Figure 3.6: Audit verify cost

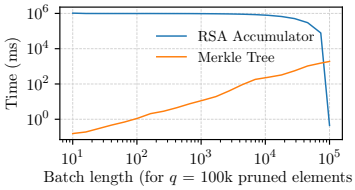


Figure 3.7: Batch audit open costs

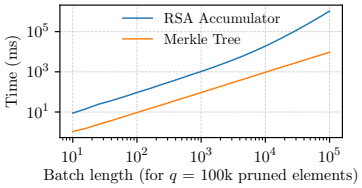


Figure 3.8: Batch audit verify costs

pruned transactions). For our Merkle tree implementation, we adopt a relevant python library[7], and we use SHA-256 as the underlying hash function.

Transaction Creation, Verification and Auditing. Every MiniLedger transaction includes an ElGamal ciphertext C, a commitment cm, a NIZK π and a running total Q for *each* Bank. Naturally, this results in linearly-increasing computation costs in terms of number of Banks as shown in Figure 3.2 for both transaction creation and verification. Note that storing the running total Q leads to constant transaction creation and verification computational costs (for a fixed number of Banks), making total assets auditing much more efficient. In contrast, zkLedger's growing ledger size also implies linearly-increasing NIZK proof verification costs, as the verifier would need to compute the product of all

[7] `https://github.com/vpaliy/merklelib`

transaction elements for each Bank(applying our running total technique to zkLedger could have analogous benefits). The transaction creation and verification costs are 53ms and 49ms respectively (for a single cell in L)and are roughly comparable with [19]. Note that although we cannot directly compare different implementations, in Table 3.5 we show that our more efficient NIZK requires less expensive operations.

Auditing any single value on the ledger takes about 4 ms as shown in Figure 3.2. This is the cost for the complete auditing protocol, namely the decryption and proving cost for the Bank and the verification cost for the Auditor. In contrast to [19], the auditing cost is constant without being impacted either by the number of Banks or the number of past transactions.

Transaction Pruning. We evaluate the computation requirements of the pruning operation which involves executing Prune() and PruneVrfy() to create the digest D_j. Our results in Figure 3.3 show it is possible to prune and verify about 1 million transactions in less than a second using Merkle trees and in about 2 hours using RSA accumulator. Note prime number multiplication costs dominate the total costs (which also include hashing to primes and an exponentiation) when the pruning depth becomes large. We also stress that these computation requirements are *independent* of the number of Banks n in the system.

For transaction auditing in AudPruned{} interactive protocol, auditing opening and verification costs are shown in Figures 3.5 and 3.6 respectively. As previously discussed, we do not include the RSA accumulator's witness creation costs (which can be pre-computed) and are shown in Figure 3.4.

For auditing sums of values (i.e. "batch" auditing), the associated costs for opening and verifying a 100K transaction digest are shown in Figures 3.7 and 3.8 respectively, with x-axis representing the number of audited transactions. Note that for auditing 10^5 transactions (i.e. the whole sum), RSA accumulator opening is significantly cheaper compared to Merkle trees, as the audited Bank would only need to retrieve the respective transactions from its local memory (which implies nearly O(1) cost) and send them to the auditor (who would in turn need to recompute all primes and perform the exponentiation of their product).

Based on our evaluation results and the discussion above, the choice between Merkle tree and batch RSA accumulator depends on use-case. Merkle trees fit a system expected to incorporate sparse audits on individual transactions, while RSA accumulator is preferred on deployments with frequent auditing on many transactions at a time (e.g. sums of assets or value thresholds over a time period).

Storage Costs. The storage cost for L has a $64n$-byte lower bound for the ElGamal variant encryptions (which represent the running total Q), plus the needed storage for each digest D and the system's pp, assuming all n Banks have pruned their transaction history and the ledger is made of a single row. Although during the system's operation where transactions are continuously appended on the ledger, its actual size will be more than that lower bound, enforcing frequent pruning operations through appropriate incentives (or penalties) will keep the size of the ledger close to its minimum. These savings in storage costs are huge compared to [19] where all Banks would have to store a ledger of size $O(nm)$ where m is the total number of transactions that have happened since the system's genesis.

Concretely, in our implementation each transaction's communication and storage cost is 68KB per Bank, which includes the ElGamal variant encryption, the auxiliary commitment, the NIZK and the running total. Note that we provide the actual memory footprint of our implementation (which relies on the underlying libraries' efficiencies) and not the theoretical lower bounds. A MiniLedger instantiation including the necessary public parameters, one transaction and a digest requires only 70KB of storage per Bank.

Network and Consensus Costs. As discussed in Section 3.2, MiniLedger focuses on the transaction layer, thus consensus layer costs depend on the exact instantiation choice. *Any* consensus protocol can be plugged to MiniLedger as long as it satisfies the basic properties of consistency and liveness. Although we consider consensus orthogonal to our implementation, we do recognize that its choice (along with network latency) ultimately affects transaction throughput. All benchmarks performed so far focus on metrics independent from consensus. Providing a full implementation of MiniLedger including a

Table 3.4: Consensus costs

Banks	Peers	Tx/s	Network
10	80	21	LAN
100	4	2	WAN

consensus layer is out of scope, we note previous works [7, 19, 20, 35] also take a similar approach on evaluation and do not include consensus measurements. For instance, zkLedger [19] evaluation only takes network latency into account which is not useful without considering consensus costs (consensus is needed to guarantee agreement on L at any time). To showcase an implementation scenario, we discuss below how MiniLedger could be implemented using existing systems in the consensus layer and also provide some rough cost estimations.

We chose Hyperledger, one of the most prominent distributed operating systems for permissioned blockchains, to provide some estimated consensus measurements. Using Hyperledger Fabric with Kafka [87] as a permissioned consensus layer requires at least 0.5 seconds to complete a full consensus operation with 4 peers and 256-bit ECDSA [88]. We previously discussed that Banks can store the ledger themselves and/or also have a "consensus verifier" role in our system (recall that while they could run consensus themselves, they do not have to, as we decouple consensus from Banks allowing any consortium of parties for ledger maintenance). Thus, Banks could act as Hyperledger "clients", "peers" and "orderers" simultaneously, which would impact performance especially with a PBFT-family consensus algorithm. For simplicity and efficiency, we consider Banks only acting as "clients", outsourcing ledger storage and consensus operations to an arbitrary number of "peer" and "orderer" nodes respectively. These numbers are entirely dependent on the use-case and do not affect MiniLedger performance or scalability. This separation between Banks and consensus participants is quite natural. As another example, Diem [29], uses similar architecture with MiniLedger (decoupled permissioned consensus and provider-intermediated transactions [89]) but has different goals in terms of privacy and auditability.

Based on the Hyperledger evaluation, we derive conservative estimations of the expected transaction throughput, shown in Table 3.4. These estimations are more than sufficient for intra-Bank transactions in a deployed system (recall that any number of client-to-client transactions in MiniLedger+ can be aggregated in a single MiniLedger transaction). MiniLedger could also be imported in Diem, however the expected transaction throughput is lower as it uses a Byzantine-tolerant consensus [90].

Although permissioned consensus generally seems more fitting to MiniLedger, permissionless consensus could also be utilized. For instance, MiniLedger could be implemented on top of an Ethereum smart contract, where Banks would be responsible to pay the respective gas fees. While technically any "Proof-of-X" consensus could be used, the underlying game-theoretic aspects should also be considered.

Fine-grained Audit Extension. For MiniLedger+ we need to store extra information on L, namely R_{ij} and H_{ij} for each entry. For consistency we use the same 256-bit hash function as in our RSA accumulator. R_{ij} has a total size of 512 bits, (excluding the size of id), still pruning makes a ledger with the same lower bound possible as before. In Appendix 3.8 we derive the computation overhead for the sending Bank to create R_{ij} roughly equivalent to $\mathsf{CreateTx}()$ (in terms of number of clients and Banks respectively), showing computation costs are doubled compared to basic MiniLedger. As the Bank only needs to include additional information in D for each transaction, the effect on pruning costs for L is negligible.

3.6 MiniLedger security

3.6.1 Scheme definitions

We define MiniLedger for a static set of n Banks with IDs defined by $[\mathsf{B}_j]_{j=1}^n$. Each Bank has a key pair $[(\mathsf{pk_j}, \mathsf{sk_j})]_{j=1}^n$ and an initial asset value $[v_j]_{j=1}^n$. Banks maintain an internal state $[st_j]_{j=1}^n$. We assume that the set of participating Banks IDs, $[\mathsf{B}_j]_{j=1}^n$, is known to all system participants.

MiniLedger is composed of the following protocols:

- $\mathsf{SysSetup}\{\mathsf{TP} \leftrightarrow [\mathsf{B}_j]_{j=1}^n\}$: executed between TP and a set of Banks $[\mathsf{B}_j]_{j=1}^n$ (or by Banks through an MPC protocol). It verifies the initial values of Banks and outputs the system parameters pp, and an initial ledger L with total system value $v_T = \sum_{j=1}^n v_j$. (We assume that pp and L are default inputs everywhere below.)
- $\mathsf{CreateTx}(B_j, B_k, v)$: run by a Bank B_j, outputs transaction tx which transfers assets v to Bank B_k[8].
- $\mathsf{VerifyTx}(\mathsf{tx})$: run by any party, verifies the validity of tx and outputs a verification bit b.
- $\mathsf{Prune}(st_j)$: run by a Bank B_j, outputs a digest D containing a "compact" representation of its transaction history and updates the Bank's state to st'_j.
- $\mathsf{PruneVrfy}(D)$: run by any party, verifies the validity of digest D and outputs a verification bit b.
- $\mathsf{Consensus}(\mathsf{tx}$ or $D)$: is run by all consensus participants in S_{CN}. On input a transaction tx or a pruning digest D, the consensus participants will verify the transaction/digest using the corresponding algorithms and update the ledger to L'.
- $\mathsf{Audit}\{\mathsf{A} \leftrightarrow \mathsf{B}_j\}$: executed between a Bank B_j and an auditor A, where the auditor audits a specific transaction (or a set of them) the Bank reveals the value(s) v of that transaction (or set) to the auditor including a proof π^{Aud} of correct presentation of the value(s) v.
- $\mathsf{AudVrfy}(v, \pi^{\mathsf{Aud}})$: executed by an auditor A to verify the validity of the proof π^{Aud} based on the provided value v.

3.6.2 Security definitions

To define security we first describe the oracles provided to an adversary $\mathcal{A}$. We assume a challenger CH maintains a corrupted Bank list T_c and performs some "bookkeeping" for

[8]Here we assume a single receiving Bank for notational simplicity, but this algorithm can be easily extended to support multiple receivers.

honest Banks $\notin T_c$ on oracle queries, where honest Banks have total assets v^{CH}. This bookkeeping includes the following: a) for $\mathsf{CreateTx}()$, it appends the output transaction tx to the ledger L, and b) for $\mathsf{Prune}()$, it replaces the honest Bank's transaction history with digest D on L and updates the Bank's state to st'_j. Note that for security definitions below that involve auditability, we only consider the basic audit on a transaction value - for brevity we omit security definitions that involve more complex auditing.

- $\mathcal{O}_c(\mathsf{B}_j)$: $\mathcal{A}$ corrupts Bank B_j and takes full control of it. $\mathcal{O}_{corr}$ records B_j to T_c. This oracle captures that $\mathcal{A}$ can corrupt honest Banks.
- $\mathcal{O}_{tx}((\mathsf{B}_j, -v_j), [\mathsf{B}_k, v_k])$: $\mathcal{A}$ queries $\mathcal{O}_{tx}$ to create a transaction which transfers v_j from Bank j (where $j \notin T_c$) to recipient Bank(s) B_k. If Bank j has at least assets v_j, $\mathcal{O}_{tx}$ executes $\mathsf{CreateTx}()$ outputting tx_i and runs bookkeeping, else it outputs $\perp$. This oracle captures that $\mathcal{A}$ can direct honest Banks to make specific (but valid) transactions of its choosing.
- $\mathcal{O}_{prn}(\mathsf{B}_j)$: $\mathcal{A}$ queries $\mathcal{O}_{prn}$ to prune the transaction history of $\mathsf{B}_j \notin T_c$. $\mathcal{O}_{prn}$ generates digest D and runs bookkeeping. This oracle captures that $\mathcal{A}$ can prune the transaction history of an honest Bank.

Definition 9 (Theft prevention and balance)**.** *For all security parameters λ, for all probabilistic polynomial time adversaries $\mathcal{A}$ with oracle access to $\mathcal{O}_c, \mathcal{O}_{tx}$:*

$$\Pr\left[\begin{array}{l} \mathsf{pp} \leftarrow \mathsf{SysSetup}\{\lambda, [\mathsf{B}_j]_{j=1}^n\}, v_t = \sum_{j=1}^n v_i; \\ \mathsf{tx}^* \leftarrow \mathcal{A}^{\mathcal{O}_c, \mathcal{O}_{tx}}(\mathsf{pp}) : \\ \left\{ \; (\mathsf{B}_j \notin T_c) \vee (v_t^* \neq v_t) \vee (\mathsf{B}_j \in T_c, \mathsf{B}_k \notin T_c, v^* > v^{\mathcal{A}}) \; \right\} \wedge \\ \mathsf{VerifyTx}(\mathsf{tx}^*) = 1 \end{array}\right] \leq negl(\lambda)$$

where tx^* is a transaction spending v^* from B_j to B_k, $v^{\mathcal{A}}$ the adversary's total assets before tx^* and v_t, v_t^* the system's total value before and after tx^* respectively.

This property captures the requirements that: only the owner of the assets can spend them, a transaction results in a decrease of sender's assets by the value represented in the transaction and that the sender cannot spend more than its total assets.

Definition 10 (Secure pruning)**.** *For all security parameters λ, for all probabilistic polynomial time adversaries $\mathcal{A}$ with oracle access to $\mathcal{O}_c, \mathcal{O}_{tx}, \mathcal{O}_{prn}$:*

$$\Pr\left[\begin{array}{l} \mathsf{pp} \leftarrow \mathsf{SysSetup}\{\lambda, [\mathsf{B}_j]_{j=1}^{n}\}; \\ D^* \leftarrow \mathcal{A}^{\mathcal{O}_c, \mathcal{O}_{tx}, \mathcal{O}_{prn}}(\mathsf{pp}) : \\ \left\{\begin{array}{l} \mathsf{B}_j \notin T_c \vee \\ (\exists \mathsf{tx}_{\mathsf{B}_j} \in L \wedge \mathsf{tx}_{\mathsf{B}_j} \notin D^*) \vee \\ (\exists \mathsf{tx}^*_{\mathsf{B}_j} \notin L \wedge \mathsf{tx}^*_{\mathsf{B}_j} \in D^*) \end{array}\right\} \wedge \\ \mathsf{PruneVrfy}(D^*) = 1 \end{array}\right] \leq negl(\lambda)$$

where $\mathsf{tx}_{\mathsf{B}_j}$ denotes a transaction where B_j is involved in, and $\mathsf{tx} \in D^*$ denotes representation of a transaction in the digest.

In the above experiment, $\mathcal{A}$ wins if he creates a digest D^* on behalf of an honest Bank B_j, or if D^* *either* does not contain a transaction $\mathsf{tx}_{\mathsf{B}_j}$ that exists in L [9] *or* contains a transaction $\mathsf{tx}^*_{\mathsf{B}_j}$ that does not exist in L. This property captures the requirement that digest outputs are only created by the respective Banks, that pruning operations contain the correct transactions in the correct order, and do not contain bogus transactions.

Definition 11 (Ledger Correctness)**.** *For all security parameters λ, for all probabilistic*

[9] $\mathcal{A}$ also wins if the transaction is indeed in D^* but out of order.

polynomial time adversaries $\mathcal{A}$ with oracle access to $\mathcal{O}_c, \mathcal{O}_{tx}, \mathcal{O}_{prn}$:

$$\Pr\left[\begin{array}{l} \mathsf{pp} \leftarrow \mathsf{SysSetup}\{\lambda, [\mathsf{B}_j]_{j=1}^n\}; \\ \mathsf{tx}^* \backslash D^* \leftarrow \mathcal{A}^{\mathcal{O}_c, \mathcal{O}_{tx}, \mathcal{O}_{prn}}(\mathsf{pp}) : \\ \left\{\begin{array}{l} \mathsf{VerifyTx}(\mathsf{tx}^*) = 0 \wedge \mathsf{tx}^* \in L \\ \mathsf{PruneVrfy}(D^*) = 0 \wedge D^* \in L \end{array}\right\} \end{array}\right] \leq negl(\lambda)$$

This property captures the requirement that only valid transactions or pruning operations are accepted on the ledger.

Definition 12 (Correct Auditability)**.** *For all security parameters λ, for all probabilistic polynomial time adversaries $\mathcal{A}$ with oracle access to $\mathcal{O}_c, \mathcal{O}_{tx}$ and $\mathcal{O}_{prn}$ and for any honest auditor* A*:*

$$\Pr\left[\begin{array}{l} \mathsf{pp} \leftarrow \mathsf{SysSetup}\{\lambda, [\mathsf{B}_j]_{j=1}^n\}; \\ \mathcal{A}^{\mathcal{O}}(\mathsf{pp}) : \\ \exists \mathsf{B}_j \notin T_c \wedge \exists \mathsf{tx}_{\mathsf{B}_j} \wedge \\ \mathsf{Audit}\{\mathsf{A}(\mathsf{tx}_{\mathsf{B}_j}) \leftrightarrow \mathsf{B}_j(v)\} := \pi^{\mathsf{Aud}} \wedge \\ \mathsf{AudVrfy}(\pi^{\mathsf{Aud}}) := 0 \end{array}\right] \leq negl(\lambda)$$

where $\mathsf{tx}_{\mathsf{B}_j}$ denotes a transaction where B_j is involved in and v is input of $\mathsf{CreateTx}()$ *which outputs $\mathsf{tx}_{\mathsf{B}_j}$.*

In the above experiment, after $\mathcal{A}$ has made a polynomial number of queries to the oracles (which include generating transactions and pruning them), $\mathcal{A}$ wins if any honest Bank is unable to correctly answer some audit. This property captures the requirement that any Bank following the protocol should always be able to answer audits correctly.

Definition 13 (Sound Auditability)**.** *For all security parameters λ, for all probabilistic polynomial time adversaries $\mathcal{A}$ with oracle access to $\mathcal{O}_c, \mathcal{O}_{tx}$ and $\mathcal{O}_{prn}$ and for any honest*

auditor A*:*

$$\Pr\left[\begin{array}{l} \mathsf{pp} \leftarrow \mathsf{SysSetup}\{\lambda, [\mathsf{B}_j]_{j=1}^n\}; \\ \mathcal{A}^{\mathcal{O}}(\mathsf{pp}) : \\ \mathsf{B}_j \in T_c \wedge \exists \mathsf{tx} \wedge \\ \mathsf{Audit}\{\mathsf{A}(\mathsf{tx}) \leftrightarrow \mathsf{B}_j(v^*)\} := \pi^{\mathsf{Aud}} \wedge \\ v^* \neq v \wedge \\ \mathsf{Vrfy}(\pi^{\mathsf{Aud}}) := 1 \end{array}\right] \leq negl(\lambda)$$

where v *is input of* $\mathsf{CreateTx}()$ *which outputs* tx *and* v^* *is any arbitrary value.*

In the above experiment, after $\mathcal{A}$ has made polynomial number of queries to the oracles (which include generating transactions and pruning them), $\mathcal{A}$ wins if it succeeds on cheating an auditor when a corrupted Bank is audited for a transaction. This property captures the requirement that an auditor cannot accept false audit claims.

Definition 14 (Privacy)**.** *For all security parameters* λ*, for all probabilistic polynomial time adversaries* $\mathcal{A}$ *with oracle access to* $\mathcal{O}_{tx}$ *and* $\mathcal{O}_{prn}$*:*

$$\Pr\left[\begin{array}{l} \mathsf{pp} \leftarrow \mathsf{SysSetup}\{\lambda, [\mathsf{B}_j]_{j=1}^n\} \\ b \in \{0,1\}, L_b, L_{1-b} \leftarrow \mathsf{CH}; \\ \text{StartLoop} \\ \mathsf{tx}, \mathsf{tx}' \leftarrow \mathcal{A}^{(\mathcal{O}_{tx}, \mathcal{O}_{prn})_b, (\mathcal{O}_{tx}, \mathcal{O}_{prn})_{1-b}}(\mathsf{pp}) \\ \mathsf{CH}(\mathsf{tx}, \mathsf{tx}') : \mathsf{VerifyTx}(\mathsf{tx}), \mathsf{VerifyTx}(\mathsf{tx}') \\ \mathsf{CH} : \mathsf{tx} \rightarrow L'_b, \mathsf{tx}' \rightarrow L'_{1-b} \\ \text{End Loop :} \\ b' \leftarrow \mathcal{A}, \\ b = b' \end{array}\right] \leq \frac{1}{2} + negl(\lambda)$$

In the above experiment, the challenger CH instantiates two ledgers L and L' in the setup phase and flips a bit. Then $\mathcal{A}$ queries two instances of oracles $\mathcal{O}_{tx}$ and $\mathcal{O}_{prn}$ (each

uniquely associated with a ledger) a polynomial number of times. However $\mathcal{A}$ is only allowed to query the same oracle type (i.e. it cannot query $\mathcal{O}_{tx}$ and $\mathcal{O}_{prn}$ the same time). For each query set, the challenger updates the two ledgers as chosen by b in the setup phase. Finally $\mathcal{A}$ guesses b. The privacy property captures the requirement that the ledger hides both transacting parties and the associated values.

3.6.3 Security proofs

We now informally argue about the security of Theorem 1.

Lemma 1. MiniLedger *satisfies Theft prevention and balance under the assumptions of ZK soundness, and hardness of the discrete logarithm problem.*

Proof. We distinguish the following two cases:

Case 1 (theft prevention):

Case 1a: In case $\mathcal{A}$ can derive a secret key from a public key (i.e. discrete logarithm hardness assumption does not hold), this immediately enables theft of funds.

Case 1b: In the case $\mathcal{A}$ can break ZK soundness, when outputting tx^* can either a) For $\pi^* = \pi^*_L \vee \pi^*_R$ in tx^*, $\mathsf{Ver}(\pi^*_R) = 1, R(x, \mathsf{sk}) = 0$ (i.e. output the π^*_R component of the OR proof π without knowing sk, or b) violate the range proof component of the proof π, i.e. $\mathsf{cm} = g^{v'} h^{r''} \wedge v' \in [0, 2^k$, $\mathsf{Ver}(\pi^*_R) = 1, v < 0$ ($v > 2^k$ is equivalent to $v < 0$), thus permitting committing to a negative value under an honest Bank (which effectively spends from that Bank).

Case 2 (balance):

Case 2a: Winning the game by changing the system's total value (i.e. having instantiated a system with total value v_t, construct a verifiable tx^* which then results into system's total value $v^*_t \neq v_t$): A verifiable tx^* implies $\prod_{j=1}^{n} c_2^{(ij)} = 1 \implies g^{v_1+v_2+..+v_n+e} h^{r_1+r_2+..+r_n} = 1$, where $e \neq 0$, which implies $\mathcal{A}$ can find $r^* = r_1 + r_2 + .. + r_n \neq 0$ such that $h^{r^*} = 1/g^{v^*_t}$, which contradicts the hardness of the discrete logarithm problem.

Case 2b: Winning the game by spending more than the total assets: In the case $\mathcal{A}$ can break ZK soundness, when outputting tx^* he can violate the range proof component π_r of the proof π^*, i.e. for $\pi_r : \mathsf{cm} = g^{v'}h^{r''} \wedge v' \in [0, 2^k, \mathsf{Ver}(\pi_R^*) = 1, v < 0$ ($v > 2^k$ is equivalent to $v < 0$). This enables committing to a negative sum of assets under a corrupted Bank, which implies spending more than its total assets.

□

Lemma 2. MiniLedger *satisfies secure pruning assuming Accumulator soundness, EU-CMA signatures and consistency of the consensus layer.*

Proof. **Case 1:** $\mathcal{A}$ can win the game in Definition 10 by pruning on behalf of an honest Bank. This can only succeed if $\mathcal{A}$ manages to sign on behalf of that Bank, which directly contradicts signature unforgeability.

Case 2: $\mathcal{A}$ wins the game by adding an arbitrary transaction into a digest D on behalf of a corrupted Bank. This implies either breaking accumulator soundness, as the adversary would be able to successfully prove membership for a non-accumulated element, or consensus consistency (recall that the digest is created by the consensus participants as well).

Case 3: $\mathcal{A}$ wins the game by omitting a transaction in the digest D, which would also contradict accumulator soundness as in the previous case.

□

Lemma 3. MiniLedger *satisfies Ledger Correctness assuming consistency of the consensus layer.*

Proof. This follows directly from the assumed consensus properties. □

Lemma 4. MiniLedger *satisfies Correct Auditability assuming ZK soundness.*

Proof. For Correct Auditability we define the following game:

1. Setup: CH runs $\mathsf{pp} \leftarrow \mathsf{SysSetup}\{\lambda, [\mathsf{B}_j]_{j=1}^n\}$ and sends pp to $\mathcal{A}$, which include $[\mathsf{pk}_{\mathsf{B_j}}, C_{0j}, Q_{0j}]_{j=1}^n$.
2. Query: $\mathcal{A}$ queries $\mathcal{O}_c, \mathcal{O}_{tx}, \mathcal{O}_{prn}, \mathcal{O}_{Aud}$ and CH answers the queries as in Section 3.6.2, maintaining the ledger L.

3. Output: For an honest Bank j, $\mathcal{A}$ outputs a transaction $\mathsf{tx}_{\mathsf{B}_j}$ where the Bank fails to convince an honest Auditor A that the value of $\mathsf{tx}_{\mathsf{B}_j}$ is v.

Assume $\mathcal{A}$ wins the above game with non negligible probability. Then $\mathcal{B}$ would break ZK soundness as follows: On input of $\pi^* : \{c_2 = g^v h^r \wedge c_1 = \mathsf{pk}^r\}$, $\mathcal{B}$ runs $\mathsf{SysSetup}$, however it replaces pk_1 with pk and sets as g, h the output of $\mathsf{SetupTEG}$. Then forwards (pp, L) to $\mathcal{A}$, with $\mathcal{B}$ providing oracle access to $\mathcal{A}$ for $\mathcal{O}_c, \mathcal{O}_{tx}, \mathcal{O}_{Aud}$ and $\mathcal{O}_{prn}$. For $\mathcal{O}_{tx}$, $\mathcal{B}$ outputs tx_i to $\mathcal{A}$ that includes $[\pi]_{k=1}^n$ which also ensures consistency for randomness r between each pair of ciphertexts c_1 and c_2. However in each $\mathcal{O}_{tx}$ query, $\mathcal{B}$ adds $c_{2,1} = g^v h^r, c_{1,1} = \mathsf{pk}^r$ to each update of L. When $\mathcal{A}$ queries $\mathcal{O}_c$ for B_1, $\mathcal{B}$ aborts. When $\mathcal{A}$ outputs a $\mathsf{tx}_{\mathsf{B}_j}$, if $j \neq 1$, $\mathcal{B}$ aborts. Else if $j = 1$, with $\mathsf{Audit}\{\mathsf{A}(\mathsf{tx}_{\mathsf{B}_1}) \leftrightarrow \mathsf{B}_1(v)\} := \pi^{\mathsf{Aud}} \wedge \mathsf{AudVrfy}(\pi^{\mathsf{Aud}}) := 0$, implies that B_1 fails to decrypt correctly to answer the audit, thus $c_2 = g^v h^r \wedge c_1 = \mathsf{pk}^{r'}$ breaking ZK soundness.

□

Lemma 5. MiniLedger *satisfies Sound Auditability assuming ZK soundness, Accumulator soundness and consistency of the consensus layer.*

Proof. **Case 1:** tx is still in L. An $\mathcal{A}$ breaking ZK soundness can convince CH for some false v^*.

Case 2: tx is pruned and represented in the digest D in L. An $\mathcal{A}$ breaking Accumulator soundness can prove membership to CH for an arbitrary tx^* and then proceed with auditing on that tx^*.

□

Lemma 6. MiniLedger *satisfies Privacy assuming IND-CPA security of the ElGamal encryption variant, Pedersen commitment hiding and Zero-knowledgeness of NIZKs.*

Proof. We describe a sequence of hybrid experiments as follows:

Hybrid 0:

1. Setup: CH runs $\mathsf{pp} \leftarrow \mathsf{SysSetup}\{\lambda, [\mathsf{B}_j]_{j=1}^n\}$, flips a bit $b \in \{0,1\}$ and makes two identical initialized ledgers L_b, L_{1-b}. CH sends pp to $\mathcal{A}$, which include $[\mathsf{pk}_{\mathsf{B_j}}, C_{0j}, Q_{0j}]_{j=1}^n$ and the two ledgers L_b, L_{1-b}.
2. Query: $\mathcal{A}$ for ledgers L_b, L_{1-b} makes queries $\mathcal{O}_{tx}, \mathcal{O}_{prn}$ where each oracle has two separate instantiations uniquely associated with a ledger. CH answers the queries as in Section 3.6.2, verifying transactions or prune operations and maintaining the ledgers L_b, L_{1-b} accordingly. $\mathcal{A}$ is restricted to make simultaneous oracle queries of the same type on the ledgers.
3. Output: $\mathcal{A}$ outputs a bit b' and wins if $b = b'$.

Hybrid 1: Same as Hybrid 0 but now CH when answering queries to $\mathcal{O}_{tx}$, simulates the ZK proofs π_{ik} in each $\mathsf{tx}_i = [C_{ik}, \mathsf{cm_{ik}}, \pi_{ik}, Q_{ik}]_{k=1}^n$.

Hybrid 2: Same as Hybrid 1 but now CH when answering queries to $\mathcal{O}_{tx}$ replaces ciphertexts C_{ik} and running totals Q_{ik} with random strings.

Hybrid 3: Same as Hybrid 2 but now CH when answering queries to $\mathcal{O}_{tx}$ replaces commitments $\mathsf{cm_{ik}}$ with random strings.

Corollary 1. *Hybrids 0 and 1 are indistinguishable.*

Proof. Immediately follows from the zero-knowledge property of π. □

Corollary 2. *Hybrids 1 and 2 are indistinguishable.*

Proof. Immediately follows from the IND-CPA property of ElGamal encryption variant. □

Corollary 3. *Hybrids 2 and 3 are indistinguishable.*

Proof. Immediately follows from the hiding property of Pedersen commitments. □

□

3.7 MiniLedger zero knowledge proof

In MiniLedger we further improve on zkLedger's proofs by providing a single disjunctive [91] proof π with proving consistency for ElGamal encryptions in both cases as follows (note this optimization is also applicable to zkLedger, we point the reader to the full version of our paper [70] for details).

$$\{(v, v', r, r', r'', \hat{v}, \hat{r}, \mathsf{sk}) : [(\mathsf{cm} = g^v h^{r'} \wedge c_2 = g^v h^r \wedge c_1 = \mathsf{pk}^r) \vee (\mathsf{cm} = g^{\hat{v}} h^{r'} \wedge \hat{c_2} = g^{\hat{v}} \hat{c_1}^{1/\mathsf{sk}} \wedge c_2 = g^v h^r \wedge c_1 = \mathsf{pk}^r \wedge \mathsf{pk} = h^{\mathsf{sk}})] \wedge [\mathsf{cm} = g^{v'} h^{r''} \wedge v' \in [0, 2^k]]\}(\mathsf{cm}, c_1, c_2, \hat{c_1}, \hat{c_2}, g, h, \mathsf{pk})$$

The details for the above proof are shown in Figure 3.9 where we omit the range proof part for notation simplicity as before. Our optimization results in the following total computational and storage costs:

Transaction (prover's) computation cost. The proving costs for π would be $3n$ prime-order exps and $5n$ prime-order multi-exps.

Verifier's computation cost. $11n$ prime-order exps and $5n$ prime-order multi-exps.

Auditing costs for π^{Aud}. Same as in zkLedger.

Storage costs. Each MiniLedger transaction is associated with the following communication/storage costs for the needed ZK proofs: $3n$ exps and $5n$ multi-exps elements as well as $10n$ exponent values.

In Table 3.5 we provide a comparison between the computational and storage costs between zkLedger and MiniLedger for their respective ZK proofs.

3.8 MiniLedger+ construction and fine-grained audit algorithms

As discussed, we can take advantage of the compactness of MiniLedger to allow for fine-grained auditing in a client level, where Banks are now acting as intermediaries for their

$\{(v, r, r', \hat{v}, \hat{r}, \mathsf{sk}) : (\mathsf{cm} = g^v h^{r'} \wedge c_2 = g^v h^r \wedge c_1 = \mathsf{pk}^r) \vee (\mathsf{cm} = g^{\hat{v}} h^{r'} \wedge \hat{c_2} = g^{\hat{v}} \hat{c_1}^{1/\mathsf{sk}} \wedge c_2 = g^v h^r \wedge c_1 = \mathsf{pk}^r \wedge \mathsf{pk} = h^{\mathsf{sk}})\}(\mathsf{cm}, c_1, c_2, \hat{c_1}, \hat{c_2}, g, h, \mathsf{pk})$

Left part of OR proof is True: (running simulator $\mathcal{S}$ for the right part) - Witnesses: v, r, r'

- P chooses $\chi_2, \chi_3, \psi_1, \psi_3, \psi_4, q, s_1, s_2, t, e_2$ at random
- P computes:

$$R_1 = g^q h^{s_1} \qquad R_1' = g^q h^{s_2} \qquad R_1'' = \mathsf{pk}^{s_2}$$
$$R_2 = \frac{g^{\chi_2} h^{\psi_1}}{g^{e_2 \hat{v}} h^{e_2 r'}} \qquad R_2' = \frac{g^{\chi_2} \hat{c_1}^{\psi_3}}{g^{e_2 \hat{v}} \hat{c_1}^{e_2/\mathsf{sk}}}$$
$$R_2'' = \frac{g^{\chi_3} h^{\psi_4}}{g^{e_2 v} h^{e_2 r}} \qquad R_2''' = \frac{\mathsf{pk}^{\psi_4}}{\mathsf{pk}^{e_2 r}} \qquad R_2^{(4)} = \frac{h^t}{h^{e_2 \mathsf{sk}}}$$

  and sends $R_1, R_1', R_1'', R_2, R_2', R_2'', R_2''', R_2^{(4)}$ to V.
- V picks e at random and sends it to P.
- P computes

$$e_1 = e - e_2 \qquad z_1 = q + v e_1$$
$$z_2 = s_1 + r' e_1 \qquad z_3 = s_2 + r e_1$$
$$z_4 = \chi_2 \qquad z_5 = \psi_1 \qquad z_6 = \psi_3$$
$$z_7 = \chi_3 \qquad z_8 = \psi_4 \qquad z_9 = t$$

  and sends $(z_1, z_2, z_3, z_4, z_5, z_6, z_7, z_8, z_9, e_2)$ to V.
- V computes $e_1 = e - e_2$ and checks if:

$$g^{z_1} h^{z_2} = R_1 \mathsf{cm}^{e_1} \qquad g^{z_1} h^{z_3} = R_1' c_2^{e_1} \qquad \mathsf{pk}^{z_3} = R_1'' c_1^{e_1}$$
$$g^{z_4} h^{z_5} = R_2 \mathsf{cm}^{e_2} \qquad g^{z_4} \hat{c_1}^{z_6} = R_2' \hat{c_2}^{e_2} \qquad g^{z_7} h^{z_8} = R_2'' c_2^{e_2}$$
$$\mathsf{pk}^{z_8} = R_2''' c_1^{e_2} \qquad h^{z_9} = R_2^{(4)} \mathsf{pk}^{e_2}$$

Completeness: Straightforward to verify.
Special Soundness: The extractor completes the protocol with transcript $((R_1, R_1', R_1'', R_2, R_2', R_2'', R_2''', R_2^{(4)}), e, (z_1, z_2, z_3, z_4, z_5, z_6, z_7, z_8, z_9, e_2))$ then rewinds to step 2 and gets transcript $((R_1, R_1', R_1'', R_2, R_2', R_2'', R_2''', R_2^{(4)}), e^*, (z_1^*, z_2^*, z_3^*, z_4^*, z_5^*, z_6^*, z_7^*, z_8^*, z_9^*, e_2^*))$. Now the extractor can compute $v = \frac{z_1 - z_1^*}{e_1 - e_1^*}$, $r' = \frac{z_2 - z_2^*}{e_1 - e_1^*}$ and $r = \frac{z_3 - z_3^*}{e_1 - e_1^*}$. This guarantees the equality of values v and r in cm, c_2 and c_1 because q, s_2 which contain v and r respectively are used by the extractor in all R_1, R_1' and R_1'', and because both relations are hard (based on the hardness of the Discrete Logarithm assumption).

Right part of OR proof is True: (running simulator $\mathcal{S}$ for the left part) - Witnesses: $\hat{v}, v, r', \mathsf{sk}, r$

- P chooses $\chi_1, \psi_1, \psi_2, q_1, q_2, s_1, s_2, t, e_1$ at random
- P computes:

$$R_1 = \frac{g^{\chi_1} h^{\psi_1}}{g^{e_1 v} h^{e_1 r'}} \qquad R_1' = \frac{g^{\chi_1} h^{\psi_2}}{g^{e_1 v} h^{e_1 r}} \qquad R_1'' = \frac{\mathsf{pk}^{\psi_2}}{\mathsf{pk}^{e_1 r}}$$
$$R_2 = g^{q_1} h^{s_1} \qquad R_2' = g^{q_1} \hat{c_1}^{1/t} \qquad R_2'' = g^{q_2} h^{s_2}$$
$$R_2''' = \mathsf{pk}^{s_2} \qquad R_2^{(4)} = h^t$$

  and sends $R_1, R_1', R_1'', R_2, R_2', R_2'', R_2''', R_2^{(4)}$ to V.
- V picks e at random and sends it to P.
- P computes

$$e_2 = e - e_1 \qquad z_1 = \chi_1 \qquad z_2 = \psi_1$$
$$z_3 = \psi_2 \qquad z_4 = q_1 + \hat{v} e_2 \qquad z_5 = s_1 + r' e_2$$
$$z_6 = 1/t + e_2/\mathsf{sk} \qquad z_7 = q_2 + v e_2$$
$$z_8 = s_2 + r e_2 \qquad z_9 = t + \mathsf{sk}\, e_2$$

  and sends $(z_1, z_2, z_3, z_4, z_5, z_6, z_7, z_8, z_9, e_2)$ to V.
- V computes $e_1 = e - e_2$ and checks if:

$$g^{z_1} h^{z_2} = R_1 \mathsf{cm}^{e_1} \qquad g^{z_1} h^{z_3} = R_1' c_2^{e_1} \qquad \mathsf{pk}^{z_3} = R_1'' c_1^{e_1}$$
$$g^{z_4} h^{z_5} = R_2 \mathsf{cm}^{e_2} \qquad g^{z_4} \hat{c_1}^{z_6} = R_2' \hat{c_2}^{e_2} \qquad g^{z_7} h^{z_8} = R_2'' c_2^{e_2}$$
$$\mathsf{pk}^{z_8} = R_2''' c_1^{e_2} \qquad h^{z_9} = R_2^{(4)} \mathsf{pk}^{e_2}$$

Completeness: Straightforward to verify.
Special Soundness: The extractor completes the protocol with transcript $((R_1, R_1', R_1'', R_2, R_2', R_2'', R_2''', R_2^{(4)}), e, (z_1, z_2, z_3, z_4, z_5, z_6, z_7, z_8, z_9, e_2))$ then rewinds to step 2 and gets transcript $((R_1, R_1', R_1'', R_2, R_2', R_2'', R_2''', R_2^{(4)}), e^*, (z_1^*, z_2^*, z_3^*, z_4^*, z_5^*, z_6^*, z_7^*, z_8^*, z_9^*, e_2^*))$. Now the extractor can compute $\hat{v} = \frac{z_4 - z_4^*}{e_2 - e_2^*}$, $v = \frac{z_7 - z_7^*}{e_2 - e_2^*}$, $r' = \frac{z_5 - z_5^*}{e_2 - e_2^*}$, $\mathsf{sk} = \frac{e_2 - e_2^*}{z_6 - z_6^*}$ and $r = \frac{z_8 - z_8^*}{e_2 - e_2^*}$. This guarantees the equality of values $\hat{v}, r$ in $\mathsf{cm}, \hat{c_2}$ and c_2, c_1 respectively because q_1 and s_3 which contain $\hat{v}$ and r respectively are used by the extractor in all R_2, R_2' and R_2'', R_2''', and because both relations are hard (based on the hardness of the Discrete Logarithm assumption).

HVZK: The simulator $\mathcal{S}$ on input of statement $(v, r, r', \hat{v}, \hat{r}, \mathsf{sk})$ randomly chooses $\chi_1, \chi_2, \chi_3, \psi_1, \psi_2, \psi_3, \psi_4, t, e_2, e$ and outputs the transcript $((\frac{R_1}{\mathsf{cm}^{e-e_2}}, \frac{R_1'}{c_2^{e-e_2}}, \frac{R_1''}{c_1^{e-e_2}}, \frac{R_2}{\mathsf{cm}^{e_2}}, \frac{R_2'}{\hat{c_2}^{e_2}}, \frac{R_2''}{c_2^{e_2}}, \frac{R_2'''}{c_1^{e_2}}, \frac{R_2^{(4)}}{\mathsf{pk}^{e_2}}), e, (z_1, z_2, z_3, z_4, z_5, z_6, z_7, z_8, z_9, e_2))$ which is perfectly indistinguishable from a honestly-executed protocol transcript $((R_1, R_1', R_1'', R_2, R_2', R_2'', R_2''', R_2^{(4)}), e, (z_1, z_2, z_3, z_4, z_5, z_6, z_7, z_8, z_9, e_2))$.

Figure 3.9: MiniLedger ZK proof π (without range proof)

Table 3.5: zkLedger and MiniLedger ZK proof costs per transaction

	zkLedger			MiniLedger		
	Prime-order exps	Prime-order multi-exps	Exponents	Prime-order exps	Prime-order multi-exps	Exponents
Transaction cost	$3n$	$6n$	-	$3n$	$5n$	-
Verification cost	$12n$	$6n$	-	$11n$	$5n$	-
Storage cost	$3n$	$6n$	$12n$	$3n$	$5n$	$10n$
Auditee cost	2	-	-	2	-	-
Auditor's cost	4	-	-	4	-	-

client transactions. In this section we provide the detailed construction for MiniLedger+.

Let client pk_{11} associated with Bank B_1 wishes to transfer some of his funds v to a client pk_{21} associated with Bank B_2. The client would then construct a transaction tx as follows:

C1. Construct $id = s \parallel \tau$ where s is a nonce and τ is a timestamp

C2. $C_1 \leftarrow \mathsf{EncTEG}(\mathsf{pk}_{11}, v)$

C3. $C_2 \leftarrow \mathsf{EncTEG}(\mathsf{pk}_{\mathsf{B1}}, v)$

C4. $C_3 \leftarrow \mathsf{EncTEG}(\mathsf{pk}_{\mathsf{B1}}, \mathsf{B}_2)$

C5. $C_4 \leftarrow \mathsf{EncTEG}(\mathsf{pk}_{\mathsf{B2}}, \mathsf{pk}_{21})$

C6. Compute ZKP π_1 for $C_4 \wedge f(\mathsf{pk}_{21}) = \mathsf{B}_2$ that proves consistency in c_4 for $\mathsf{B}_2, \mathsf{pk}_{21}$

C7. Compute ZKP π_2 for $C_3 \wedge C_4$ proving consistency for $\mathsf{B}_2, \mathsf{pk}_{\mathsf{B2}}$

C8. Compute ZKP π_3 for $C_1 \wedge C_2$ proving consistency for v

C9. Sign all the previous outputs as σ

C10. Construct $\mathsf{tx}_c : (id, C_1, C_2, C_3, C_4, \pi_1, \pi_2, \pi_3, \sigma)$ and send it to B_1.

B_1 after receiving tx will perform the following actions:

S1. $\mathsf{Vrfy}(\sigma)$

S2. $\mathsf{Vrfy}(\pi_1 \wedge \pi_2 \wedge \pi_3)$

S3. $\mathsf{DecTEG}(\mathsf{sk}_{\mathsf{B1}}, C_2) := v$

S4. $\mathsf{DecTEG}(\mathsf{sk}_{\mathsf{B1}}, C_3) := \mathsf{B}_2$

S5. Verify that $v \geq \Sigma v$ for client pk_{11} (has the assets to transfer)

S6. Append tx in its private table under pk_{11}

S7. Construct a transaction tx_i in L, where C_{i2} (under B_2's column) transfers v to B_2

S8. Compute ZKP π_4, π_5 for C_{i2}, C_2, C_3, proving consistency for v and B_2 (i.e. proving it sent the correct value to the correct Bank)

S9. Assign $id \parallel C_4 \parallel \pi_4 \parallel \pi_5$ to all entries R_{ij} of row i in L

S10. Assign $id \parallel H(\mathsf{tx}_c)$ to all entries H_{ij} of row i in L

At this point, the sending client pk_{11} by observing the new entries H_{ij} can verify that its transaction has been correctly processed by its Bank B_1. Finally the receiving Bank B_2 would perform the following which completes the transaction:

R1. $\mathsf{DecTEG}(\mathsf{sk}_{\mathsf{B}_2}, C_{i2}) := v$

R2. Parse R_{i2} and perform $\mathsf{DecTEG}(\mathsf{sk}_{\mathsf{B}_2}, C_4) := \mathsf{pk}_{21}$

R3. $C_1' \leftarrow \mathsf{EncTEG}(\mathsf{pk}_{21}, v)$

R4. $C_2' \leftarrow \mathsf{EncTEG}(\mathsf{pk}_{\mathsf{B}_2}, v)$

R5. Compute ZKP π_1' for C_4, C_1' proving consistency for pk_{21}

R6. Compute ZKP π_2' for C_{i2}, C_1', C_2' proving consistency for v

R7. Construct $\mathsf{tx}_{c'} : (id', C_1', C_2', \epsilon, \pi_1', \pi_2', \epsilon, \sigma')$ where ϵ are empty strings and σ' is a signature on $\mathsf{tx}_{c'}$ by B_2

R8. Construct an empty transaction in L, where all entries are empty strings ϵ except $H := id' \parallel H(\mathsf{tx}_{c'})$.

Note that only the receiving Bank B_2 will be able to decrypt C_4 and learn the final recipient pk_{21}, as other Bank's decryptions will fail. This ensures that noone except the original sending client learns both the sender and the end recipient of a transaction. Also an external observer still cannot learn the sending or receiving Bank, as the indistinguishability properties of MiniLedger are maintained (which is not true in a similar system, Solidus [8]).

Table 3.6: Fine-grained audit extension computation costs overview (normalized aggregation costs).

	Without aggregation		With aggregation (aggregation factor θ)	
	Prime-order exps	Prime-order multi-exps	Prime-order exps	Prime-order multi-exps
Client cost	6	$6+q$	6	$6+q$
Sending Bank cost	$9+q+4n$	$5+q+7n$	$10+q+4n/\theta$	$6+q+(2+7n)/\theta$
Receiving Bank cost	6	5	$6+1/\theta$	$4+3/\theta$

3.8.1 Assumptions and threat model

For the above extension we assume the assumptions and threat model described in Section 3.2 regarding the public ledger L. We now assume however that the auditors behave honestly and do not collude with anyone, as they are the entities safeguarding the scheme's security. Similarly as before, we assume an anonymous broadcast functionality between clients and Banks to preserve anonymity.

Otherwise Banks may try to manipulate their private tables and present altered tables to auditors, steal money from their users, make transactions without their permission or make them fail audits. Clients can also be assumed to be malicious - they might try to steal money either from another client or from another Bank in the system, hide their assets, provide false information to audits or try to make a system participant (Bank or client) fail future audits. Banks and/or clients can also freely collude (without however breaking consensus properties).

Finally, while from the Client-Bank relation it is natural for the Bank to know all its clients and their values, the Bank can only learn transactions where their client is a sender or receiver but does not learn the end receiver or the initial sender client respectively.

3.8.2 Auditing banks

Although honest behavior of the Banks can only be verified reactively, it can still be held accountable if any of its clients "complain" about loss of funds. Therefore the auditor, either at random or triggered by such a complaint, will perform the following protocol with the Bank:

AB1. Auditor sends an audit query to Bank

AB2. The Bank presents the encrypted L_{B_j} to the auditor, as well as a ZKP π_Σ that the sum of sums of all its clients is equal to the Bank's sum in L

AB3. The auditor verifies π_Σ and also verifies that each tx_c in L_{B_j} is signed by its respective client and can be matched with an existing digest $H(\mathsf{tx}_c)$ in L.

3.8.3 Auditing clients

An auditor A can also request audits from individual clients, providing that the corresponding Bank presents its encrypted L_{B_j} to A. As in the basic MiniLedger, A can request an audit either for specific client transactions or for the sum of a client's assets in L_{B_j}. The prover can be either the Bank or the client, proving its answers in ZK. The auditing protocol for a client would work as follows:

AC1. For auditing client $\mathsf{pk_{jm}}$, A makes the request from B_j.

AC2. B_j presents L_{B_j} to A.

AC3. Auditor A fully validates L_{B_j}, similarly as in AB3. above.

AC4. On successful verification (i.e. verify π_Σ and that each tx_c signed and matched with a digest $H(\mathsf{tx}_c)$) , A requests value v from the client (or Bank) (the audit could be on a specific tx, or the total assets Σv of the client represented by the product of the respective ciphertexts)

AC5. Client (or Bank) sends v to A, along with a ZKP that the respective ciphertext encrypts v (or the ciphertexts' product if audited for total assets)

An interesting extension family in the client auditing protocol is a client proving that it was *not* involved in tx_c with value v, or has *not* transacted with another client $\mathsf{pk_r}$. At a high level, the first can be achieved with a Bank accumulating $id \parallel v$ in L_{B_j} along with the needed zero-knowledge proofs, and provide the auditor a non-membership proof of $id \parallel v$. The second audit requires the client accumulating the above ciphertexts C_4 (and similarly with a ZK proof for honest accumulation). To prevent tampering with data, hashes of

accumulators and proofs should be regularly posted on the Bank's private ledger (and in turn, hashes to the main ledger). These extensions require a universal accumulator (as the RSA accumulator), since Merkle trees do not provide such functionality.

3.8.4 Aggregating transactions

The above protocol requires a transaction to be posted in L for *each client* transaction, which negatively impacts the overall scalability. However the Bank can defer creating the transaction in L by waiting for more transactions of its clients to be posted on its local "mempool". Then the sending Bank can fulfill multiple transaction requests of its clients with a single transaction in L, since that transaction can accommodate multiple recipient Banks.

While aggregating transactions for recipients belonging to different Banks is simple by including multiple entries in R and H, the protocol needs to be modified to accommodate multiple transactions when two or more recipients belong to the same Bank. Similar to the previous example, let two clients pk_{11} and pk_{12} associated with Bank B_1 wishing to transfer some of their funds v_1 and v_2 to clients pk_{21} and pk_{22} respectively, associated with Bank B_2. First, both clients would construct their transactions $\mathsf{tx}^{(1)}, \mathsf{tx}^{(2)}$ as before. B_1 would then work through steps S1 to S6 as above for each client separately. The rest of its steps would be as follows:

S7. Construct a transaction in L, where C_{i2} under B_2's column transfers $v = v_1 + v_2$ to B_2

S8. Compute ZKP π_4, π_5 for $C_{i2}, (C_2^{(1)} + C_2^{(2)}), (C_3^{(1)} \wedge C_3^{(2)})$, proving consistency for v and B_2 (i.e. proves from homomorphic addition of $C_2^{(1)} + C_2^{(2)}$ it sent the correct value, and separately proves with each $C_3^{(1)}, C_3^{(2)}$ that it sent to the correct Bank)

S9. Compute $C_2^{(1')} = \mathsf{EncTEG}(\mathsf{pk}_{\mathsf{B}_2}, v_1)$ and $C_2^{(2')} = \mathsf{EncTEG}(\mathsf{pk}_{\mathsf{B}_2}, v_2)$

S10. Compute ZKP π_6 that proves consistency between $(C_2^{(1)} \wedge C_2^{(1')})$ and $(C_2^{(2)} \wedge C_2^{(2')})$ for v_1, v_2 respectively

S11. Assign $id^{(1)} \parallel c_4^{(1)} \parallel \pi_4 \parallel \pi_5 \parallel \pi_6$ and $id^{(2)} \parallel c_4^{(2)} \parallel \pi_4 \parallel \pi_5 \parallel \pi_6$ to R. [10] Note that B_1 cannot "mix and match" since it would fail the audit because of tx in its table.

S12. Assign $id^{(1)} \parallel H(\mathsf{tx}^{(1)}) \parallel id^{(2)} \parallel H(\mathsf{tx}^{(2)})$ to H.

Then Bank B_2 would perform the following steps:

R1. Decrypt v from ciphertext C_{i2} in L

R2. $\mathsf{DecTEG}(\mathsf{sk}_{\mathsf{B}_2}, C_4^{(1)}) := \mathsf{pk}_{21}$, $\mathsf{DecTEG}(\mathsf{sk}_{\mathsf{B}_2}, C_4^{(2)}) := \mathsf{pk}_{22}$, $\mathsf{DecTEG}(\mathsf{sk}_{\mathsf{B}_2}, C_2^{(1')}) := v_1$, $\mathsf{DecTEG}(\mathsf{sk}_{\mathsf{B}_2}, C_2^{(2')}) := v_2$

R3. Compute $C_1^{(1')} \leftarrow \mathsf{EncTEG}(\mathsf{pk}_{21}, v_1)$ and $C_1^{(2')} \leftarrow \mathsf{EncTEG}(\mathsf{pk}_{22}, v_2)$

R4. Compute $C_2^{(1')} \leftarrow \mathsf{EncTEG}(\mathsf{pk}_{\mathsf{B}_2}, v_1)$ and $C_2^{(2')} \leftarrow \mathsf{EncTEG}(\mathsf{pk}_{\mathsf{B}_2}, v_2)$

R5. Compute $\pi_1^{(1')}$ for $C_4^{(1)}, C_1^{(1')}$ proving consistency for pk_{21} & $\pi_1^{(2')}$ for $C_4^{(2)}, C_1^{(2')}$ proving consistency for pk_{22}

R6. Compute $\pi_2^{(1')}$ for $C_1^{(1')}, C_2^{(1')}$ proving consistency for v_1, $\pi_2^{(2')}$ for $C_1^{(2')}, C_2^{(2')}$ proving consistency for v_2 and $c_2^{(k2)}, (C_1^{(1')} \cdot C_2^{(1')}), (C_1^{(2')} \cdot C_2^{(2')})$ proving consistency for v

R7. Construct $\mathsf{tx}^{(1')} : (id^{(1')}, C_1^{(1')}, C_2^{(1')}, \epsilon, \epsilon, \pi_1^{(1')}, \pi_2^{(1')}, \epsilon, \sigma^{(1')})$ where ϵ are empty strings and $\sigma^{(1')}$ is a signature on $\mathsf{tx}^{(1')}$ by B_2 (similarly for $\mathsf{tx}^{(2')}$)

R8. Construct an empty tx in L, setting all entries as ϵ except $H := id^{(1')} \parallel H(\mathsf{tx}^{(1')}) \parallel id^{(2')} \parallel H(\mathsf{tx}^{(2')})$.

Note that transaction aggregations do not interfere with audits either at the Bank or at the client level, since all transactions are still "recorded" in the hashtable.

Another interesting case is when a client requests some monetary value to be transferred to another client belonging to the same Bank, which would constitute an "internal" transaction for that Bank. The Bank's assets in L would not change and a separate transaction in L would not be needed, however the Bank would still post the respective digests in L

[10] To prevent leaktzage of information that some Bank receives values for two of its clients, the protocol could enforce posting dummy values equal to the maximum number of clients some Bank has in the system .

to ensure correct auditing. Such transactions can of course be aggregated with external transactions in L as discussed above.

3.8.5 Security analysis

As in basic MiniLedger, malicious Banks cannot steal, hide or manipulate assets in L. In MiniLedger+ however, a malicious Bank could trivially manipulate its private table (e.g. change its clients' values) since that table is not directly observable by an external verifier. However malicious behavior would eventually be detected in an audit as the private table won't be consistent with the added column data in L. In the case the Bank attempts to move funds internally without client authorization (debiting one of its clients and crediting another), even though the balance sum in L_{B_j} would match the balance reflected in L, this unauthorized transaction still wouldn't be matched to an entry in L and would fail the audit, thus exposing the Bank of malicious behavior against its own clients. For the same reason, a Bank cannot "omit" a client transaction in L_{B_j}, as this would also result in a total asset mismatch.

In a client audit, the client or the respective Bank might attempt to provide false answers to the auditor, even by trying to collude. Here the auditor would have verified the validity of L_{B_j} first, so any subsequent audits would be executed on transactions that have been verified to be valid by having matched them with the public ledger L.

3.8.6 Cost analysis for MiniLedger+ without aggregation

Assuming an instantiation with the ElGamal variant encryption scheme, its computation costs are as follows (here we denote $c_1 \to X$ and $c_2 \to Y$):

Client costs. A client needs to construct tx which includes the following computation costs: Compute ciphertexts C_1, C_2, C_3, C_4: 4 prime-order exps and 4 prime-order multi-exps. Compute π_1: Assuming mapping function $f()$ for a client public key $\mathsf{pk_C}$ and Bank's j id B_j is derived from verifying a client's signature $\sigma_{\mathsf{sk_C}}(\mathsf{pk_C}, \mathsf{B}_j)$ and assuming B_j has q

clients in total, π_1 would be an OR zero-knowledge proof of knowledge of randomness r of ciphertext Y for all q client public keys in the ElGamal variant encryption, which would be q prime-order multi-exps. Compute π_2: Since receiving Bank is known to the sending Bank, π_2 would just consist of a PoK of randomness for ciphertext Y of C_3 and for ciphertext X of C_4, which are 2 prime-order exps. Compute π_3: 2 prime-order multi-exps.

Sending bank costs. A Bank on receiving tx from its client has the following computation costs: Verify π_1: q prime-order exps and q prime-order multi-exps. Verify π_2: 4 prime-order exps. Verify π_3: 2 prime-order exps & 2 prime-order multi-exps. Decrypt for C_2, C_3: 2 prime-order exps. MiniLedger $\mathsf{CreateTx}()$ costs. Compute π_4: 2 prime-order multi-exps.Comp. π_5: 1 prime-order exp & 1 prime-order multi-exp.

Receiving bank costs. A Bank receiving value from a transaction in L has the following computation costs: Decrypt from transaction in L: 1 prime-order exp. Decrypt c_4: 1 prime-order exp. Encrypt C_1', C_2': 2 prime-order exps and 2 prime-order multi-exps. Compute π_1': Proof would be constructed in a similar way to π_2, 2 prime-order exps. Compute π_2': Is a proof of consistency of v for Y ciphertexts of C_1', C_2' and C_{i2} which costs 3 prime-order multi-exps.

3.8.7 Cost analysis for MiniLedger+ with aggregation

Let us now consider the case a sending Bank aggregates θ transactions where all recipients belong to the same receiving Bank B_j (aggregations for recipients among different Banks have additive costs as in section 3.8.6). For comparison, we outline the *normalized* computation costs per client transaction as follows:

Client costs. Same costs as without aggregation.

Sending bank costs. Verify π_1: q prime-order exps and q prime-order multi-exps. Verify π_2: 4 prime-order exps. Verify π_3: 2 prime-order exps and 2 prime-order multi-exps.

Decrypt C_2, C_3: 2 prime-order exps. MiniLedger $\mathsf{CreateTx}$ cost divided by θ. Compute π_4: $2/\theta$ prime-order multi-exps (due to homomorphic additive property). Comp. π_5: 1 prime-order exp & 1 prime-order multi-exp. Comp. $C_2^{(i')}$: 1 prime-order exp & 1 prime-order multi-exp. Compute π_6: 2 prime-order multi-exps

Receiving bank costs. Decryption costs from zkLedger transaction divided by θ. Decrypt $C_4^{(i)}$ and $C_2^{(i)}$: 2 prime-order exps. Encrypt $C_1^{(i')}, C_2^{(i')}$: 2 prime-order exps and 2 prime-order multi-exps. Compute $\pi_1^{(i')}$: 2 prime-order exps. Compute $\pi_2^{(i')}$: Is a proof of consistency of v for Y ciphertexts of $C_1^{(i')}, C_2^{(i')}$ and $c_2^{(i2)}, \prod C_1^{(i')}, \prod C_2^{(i')}$ which costs $2+3/\theta$ prime-order multi-exps.

3.9 Additional audit types and modifications

3.9.1 Audit without consent

All audit functionalities described in Section 6.4 are interactive and require the Bank's consent. We could enable non-interactive audits by including an encryption of π^{Aud} and its statement for each transaction cell under a predetermined trusted auditor's public key (which preserves privacy). However since $\mathsf{AudPruned}\{\}$ is always interactive (and cannot be converted to a non-interactive protocol because it needs the Bank's input from its memory state), an audit without consent can only take place on non-pruned data. Because of this inherent limitation, the only possible type of audit without consent for pruned transactions is $\mathsf{AudTotal}\{\}$. Then the statement of NIZK π^{Aud} would be $(Q_j, \sum v_j, \mathsf{pk_j}, g, h)$.

With MiniLedger taking this approach, an alternative direction for ledger "compacting" could be pruning each transaction right away (without consent from Banks) in a similar fashion to CODA [77]. Since MiniLedger is account-based, keeping running totals Q after each transaction execution would be sufficient. The ledger would only consist of a single row of running totals representing each Bank's total assets. After a transaction is broadcasted

by a Bank, it would first be checked for validity as before and all running totals would be updated as $Q \cdot C \rightarrow Q'$, ensuring optimal $O(n)$ ledger size (of course without Banks needing to keep local memory state).

Finally, to remove the above trusted auditor requirement we can utilize threshold encryption [92], where a coalition of t out of n Auditors could audit any Bank in the ledger, even without its consent. The protocol would now require a Bank to encrypt the value v_{ij} under the designated threshold encryption public keys. Then t Auditors will be able to audit any Bank's total assets.

3.9.2 Additional audit types

From our "basic" audit on a value v in L, we can derive several more audit types that reduce to a basic audit. We outline some below, however note this list is not exhaustive. We also note again that these audits can still be executed even for pruned data.

Statistical audits. This audit type category is similar to zkLedger's. This requires a sending Bank committing to a bit flag b in each cell in L, which indicates if that Bank participates in that transaction (i.e. $b = 1$ if $v \neq 0$, and $b = 0$ otherwise), accompanied with a NIZK to enforce correctness. "Statistical" audits involve queries that require only non-zero value transaction consideration, as zero-value encryptions would skew the result. For instance, to query the average transacted value for a Bank over a period, the auditor would need to query all the Bank column cells that correspond to that period. The Bank would then reply for all those cells as in the basic audit, with the addition of a reply to the bit flag. Then the auditor after verifying the audit replies, would compute the average value from cells with a bit flag of 1. This category is also applicable to MiniLedger+ in an identical manner.

Value compared to some limit. To query if a Bank sent or received a value less or over an amount t, the audited Bank simply needs to provide a range proof $\pi_r : \{v : (v \geq t)\}$. To preserve correctness, the value v needs to be associated with the hidden value in cm

in π (included in Figure 3.9). As with the basic audit, this proof can be provided either proactively (i.e. posted on L) or reactively (i.e. provided to the auditor during audit).

Limit over time. The auditor might want to learn if a Bank's transactions have exceeded a value over a time period (e.g. if Bank has received over $1M over a week). We can create a conjunction of the two audits previously discussed to create such an audit (i.e. audit on average value over time combined with range proof on that average value). To prevent skewing the result in case the Bank has both sent and received values in that period, additional range proofs are needed to prove if a value is positive or negative and included in the overall limit audit. Also note that the notion of "time" in MiniLedger is equivalent to transaction rows, which can include auxiliary timestamp information.

Transaction recipient. The goal of this audit type is for a sending Bank to prove the recipients for one of its transactions. If that transaction has not pruned parts, the Bank can simply reply with the list of receivers and then the auditor would need to audit each Bank in the transaction row to verify this. However it is likely that at least one Bank has pruned its respective cell in that transaction. In this case, the Bank should keep a record of its transaction recipients in its local memory for each of its *outbound* transactions, and reply to the auditor accordingly. Bit flags used in the Statistical Audits discussed before can also be utilized to make this audit type more efficient.

Client audits in MiniLedger+. To execute audits at a client level, the auditor first needs to fetch the client transactions from the Bank's private ledger, and verify their validity as outlined in Section 3.8.3. From that point, the auditor can perform all audits in a client level in a similar fashion to the respective audits in a Bank level. For instance, to learn if some MiniLedger+ client exceeded a value transaction threshold within a time period or over a number of transactions, this audit can be executed by selecting the client's transactions from the Bank's private table that happened within this period by their *id*'s. The audit would then be on the sum of the values represented by the product of

the respective ciphertexts, and the client would produce a range proof for that ciphertext product as above. and select those with the appropriate timestamp.

A special useful audit would be to learn if a MiniLedger+ client has sent assets to some specific client pk or not. The transactions would need to be augmented with an additive universal accumulator, with each sender adding the end client recipient's pk to the accumulator, while also providing its Bank a ZK proof of adding the correct public key. During an audit, the client would have to prove membership (or non membership) to the auditor. An important note is that the receiving client *does not* directly learn the original sender of a specific transaction in-band, which implies the above approach cannot work for a client to prove if he has *received* (or not) assets from another client.

3.10 Choosing a construction for digest D

In Section 3.5 we discussed our options for instantiating the accumulator used in pruning. Here we provide a summary of comparisons between possible options in Table 3.7.

Table 3.7: Data structure D comparison. q: number of pruned transactions, k: # bits, λ: security parameter, $\mathbb{F}$: group multiplications, $\mathbb{H}$: hash operations, $\mathbb{G}$: group exponentiations. Costs for $\mathsf{Open}(), \pi$ and $\mathsf{Verify}()$ are for a **single** transaction audit, while costs for $\mathsf{BOpen}(), \hat{\pi}$ and $\mathsf{BVerify}()$ are for an ℓ**-batched** transaction audit.

D	$\lvert D\rvert$	$\lvert\mathsf{pp}\rvert$	time (PruneVrfy)	Up-dat-able	Dyn. size	time(Prune)	time(Open) time(BOpen)	$\lvert\pi\rvert$ $\lvert\hat{\pi}\rvert$	time(Verify) time(BVerify)
Merkle Tree	$O(1)$	$O(1)$	$O(q)\mathbb{H}$	✔	✔	$O(q)\mathbb{H}$	$O(\log q)\mathbb{H}$ $O(\ell \log q)\mathbb{H}$	$O(\log q)$ $O(\ell \log q)$	$O(\log q)\mathbb{H}$ $O(\ell \log q)\mathbb{H}$
Catalano-Fiore CDH [84]	$O(1)$	$O(q)$	$O(q)\mathbb{G}$	✔	×	$O(q)\mathbb{G}$	$O(q)\mathbb{G}$ $O(\ell q)\mathbb{G}$	$O(1)$ $O(\ell)$	$O(\lambda)\mathbb{G}$ $O(\ell\lambda)\mathbb{G}$
Lai-Malavolta [85]	$O(1)$	$O(q^2)$	$O(q)\mathbb{G}$	×	×	$O(q)\mathbb{G}$	$O(q)\mathbb{G}$ $O(\ell q)\mathbb{G}$	$O(1)$ $O(1)$	$O(\lambda)\mathbb{G}$ $O(\ell\lambda)\mathbb{G}$
Boneh-Bunz-Fisch VC [47]	$O(1)$	$O(1)$	$O(\lambda)\mathbb{G} +$ $O(kq \log q)\mathbb{F}$	✔		$O(kq \log q)\mathbb{G}$	$O(kq \log q)\mathbb{G}$ $O(kq \log q)\mathbb{G}$	$O(1)$ $O(\lambda)$	$O(\lambda)\mathbb{G} + O(k \log \ell)\mathbb{F}$ $O(\lambda)\mathbb{G} + O(k\ell \log \ell)\mathbb{F}$
Batch-RSA accumulator [47]	$O(1)$	$O(1)$	$O(\lambda)\mathbb{G} +$ $O(q)\mathbb{F} +$ $O(q)\mathbb{H}$	✔	✔	$O(\lambda)\mathbb{G} +$ $O(q)\mathbb{F} +$ $O(q)\mathbb{H}$	$O(q)\mathbb{F}$ $O(q)\mathbb{F}$	$O(1)$ $O(1)$	$O(q)\mathbb{F}$ $O(q)\mathbb{F}$

3.11 Optimizations for decryption operations

As discussed in Section 3.3.1, MiniLedger uses an additively homomorphic ElGamal scheme variant, which requires a precomputed discrete-log lookup table (we note additive ElGamal is used in many recent blockchain-based payment systems [7, 8, 35, 64, 93]). Typically, for most financial applications, the max transaction amount ranges from 2^{32} to 2^{64}, as this is usually enough to encode even the largest reasonable balance. Generating and loading a table for billions and trillions of elements is might be impractical though, especially for constrained devices such as mobile phones, IoT devices and light clients in general. For instance, when using the 256-bit *secp256k1* elliptic curve where a group point is serialized in 33 bytes (in compressed form), 132 GB are required to store 2^{32} elements in binary format.

While there are solutions for trading storage space for computation, such as Shanks algorithm [94] (also known as baby-step giant-step), we propose an extra layer of compression on top of such algorithms, by reducing the lookup table size by a constant factor *without* any additional decryption cost. Our method only require a precomputation phase and produce collision-free tables[11] with size independent of the elliptic curve field size; the larger the size, the better the compression rate.

3.11.1 Methodology

Our goal is to compute a lookup table for $g^x \; \forall x \in [1, 2^n]$. Assuming an elliptic curve (EC) over a finite field $\mathbb{F}_q$ for a security parameter λ, the uncompressed serialized representation of an EC point is $2\log q$ bits. Typically however, a compressed format of $q' = \log q + 1$ bits is selected, where only one coordinate and additional sign bit are enough to reconstruct the EC point. As a first intuition for our approach, we can pick some $\tau < q'$, append $g^x = b_1 b_2 ... b_\tau$ to f where $b_i \in (0, 1)$. However we need careful consideration when picking τ because of the birthday paradox. Picking a τ too small (meaning that the "chopped" portion of g^x is large), many collisions among the whole table will occur, which might result

[11] Similar compression techniques are also discussed in [95], but our work focuses on collision-free tables per group to completely avoid false positives.

in ambiguities during lookup operations (e.g. on ElGamal decryption).

While the above technique is straightforward by trial and error, it results in a significant compression factor, e.g. for secp256k1 elliptic curve, the factor is $1:4$. However we can further improve compression with a variable length truncation algorithm, which works as follows. Assuming we have baby-step giant-step parameters α and β respectively from Shanks algorithm, we choose truncation parameters τ_{start} and τ_{stop} (where $\tau_{start} > \tau_{stop}$), which respectively direct the algorithm on how many bits should start the binary representation for each element with, and how many bits should try to represent the elements in an unambiguous way. The algorithm also needs as input data a set of uncompressed precomputed tables $\mathrm{A}_1, ..., \mathrm{A}_\mathrm{p}$, which are partitioned to lower RAM requirements. Then the algorithm, after initializing a variable truncation index table C, it starts from the "conservative" truncation parameter τ_{start} and checks uniqueness of truncated elements for a baby-step table A2 against all truncated elements of a (precomputed) full table A1. If a collision is found, we update the respective index in C with the previous collision-free truncation parameter τ. This process is repeated by decrementing that parameter (i.e. truncating more bits). Note this process is done in two separate phases, one for checking for collisions of baby-step elements against all values in range $(2^\alpha, 2^{\alpha+\beta}]$ (to ensure that no collisions occur even when doing giant steps) and then for self-collisions between baby step elements.

Therefore from C we can serialize the respective table for 2^α elements by interleaving $(\lceil \log(\tau_{start} - \tau_{stop}) \rceil)$ bits per element. These bits encode the variable length and are necessary to make serialized parsing possible. Alternatively, we can reduce the number of those encoding bits by assigning them into k groups $G_1, G_2, .., G_k$, therefore needing $\lceil \log k \rceil$ bits per element. For instance, we can decrement τ by 2 bits each time instead of 1, and group G_1 would represent lengths of τ_{start} and $\tau_{start} - 1$ bits, group G_2 would represent lengths of $\tau_{start} - 2$ and $\tau_{start} - 3$ bits etc. Also, depending on the results, we might have these groups contain an uneven number of bit representations. For example, truncating with τ_{start} usually turns out to contain relatively very few elements, and therefore devoting a

group for a very small population won't be efficient overall. Still each group G must encode the maximum τ that is included in that group, denoted by $\max \tau_G$. The total size of the serialized lookup table will then be

$$\sum_{i=1}^{k} (\max \tau_{G_i} \cdot |G_i|) + \lceil \log k \rceil \cdot 2^{\alpha}$$

where $\sum_{i=1}^{k} |G_i| = 2^{\alpha}$ and $\max \tau_{G_k} = \tau_{start}$. Note that even though the "grouping" approach reduces the factor $\lceil \log k \rceil \cdot 2^{\alpha}$, the granularity of the algorithm is also reduced, which overall increases the total size. On the other hand, more fine-grained groups will decrease the overall needed storage of the serialized lookup table, but will result in slightly increased computation cost in hashmap lookups[12].

3.11.2 Optimization evaluation, complexity analysis and comparison

We evaluate our compression factor with the secp256k1 curve for $n = 32$, $\alpha = 20$, $\beta = 12$ and $k = 7$. These parameters are consistent with existing implementations in the blockchain domain, and our results are shown in Table 3.8. Note this requires $\lceil \log k \rceil = 3$ bits of length encoding per element, which overall results in about 393KB encoding overhead. We also performed tests for $n = 20, \alpha = 20, \beta = 0$ and $k = 7$ for the secp256k1 and secp521r1 curves, where we achieved a compression factor of 1:10 and 1:20 respectively. For $k = 4$ in the secp256k1 curve shown in Table 3.8, the compression factor slightly reduces to 1:6.85.

For generating a compressed lookup table g^x for $x \in [1, 2^n]$, our algorithm VarTruncate() includes a computationally intensive overhead in addition to just generating the table, similar to [96]. Specifically, [96] has an $O(2^n)$ computational overhead, while our algorithm has $O(k2^n)$ complexity. However this additional cost is paid only once for a specific set of parameters, while our algorithm achieves a) a significant improvement in compression

[12]Note that the cost of a hashmap lookup is insignificant compared to elliptic curve (EC) point addition (about 40 times in our implementation), while a scalar to EC point multiplication is around 32 times more expensive than EC point addition using the *double-and-add* method for small 32-bit scalars.

Table 3.8: Variable length truncation for the secp256k1 curve with $n = 32$, $\alpha = 20$, $\beta = 12$, $k = 4$.

Bits	Bits +encoding	# elements	Total Size
32	35	385479	13491765
36	39	599610	23384790
40	43	59450	2556350
44	47	3786	177942
48	51	238	12138
52	55	12	660
56	59	1	59
Size: 4.72MB		Compress: 1:6.98	

factor and b) collision free encoding which simplifies used data structures.

For recovering the discrete log of g^x, the probabilistic [95] has $O(c2^{n/3})$ computation and storage complexity on average (for a very small $c < 2$), while our decryption computational cost involves $O(2^{\beta})$ multiplications and $O(k2^{\beta})$ map lookups in the worst case, where k is the number of utilized truncation groups. Note that regular Shanks has a lookup multiplication complexity of $O(2^{\beta_2})$, for $\beta_2 > \beta$ (typically, for 256-bit curves $\beta_2 \approx \beta + 3$), when using the same precomputed table size with our scheme.

3.12 Conclusion

We presented MiniLedger, the first *private and auditable payment system with storage independent to the number of transactions*. MiniLedger utilizes existing cryptographic tools and innovates on the meticulous design of optimized ZK proofs to tackle important scalability issues in auditable, private payments. Additionally, we provide the first formal security definitions for auditability and secure pruning in private and auditable payment systems. We achieve huge storage savings compared to previous works that store information for each transaction ever happened. Using our pruning techniques, the overall MiniLedger size can be impressively compacted to 70KB per Bank, no matter how many transactions

have ever occurred. Note that our storage and computation costs could be further improved, e.g. by using Bulletproofs [41] (instead of Schoenmakers multi-base decomposition [83]), more efficient programming languages (e.g. Rust) and libraries, or by utilizing CPU parallelization. However, as in related systems [19, 29] our goal is not to support "thousands" of Banks, but an arbitrary number of clients as discussed in Section 3.4.1, which does not affect the computation/storage costs in the public ledger. MiniLedger can currently serve a small consortium of Banks (e.g. the world's Central Banks) with an *arbitrary* number of clients, or build a hierarchy of a large number of Banks and clients in accordance with MiniLedger+. Evaluating MiniLedger on such a large scale or achieving its properties in a permissionless setting are interesting directions for future work.

Chapter 4: Proving assets in the Diem blockchain

4.1 Introduction

We now shift from the anonymity vs. auditability dilemma at a microscopic level, where we considered MiniLedger as a solution, towards the macroscopic level of the problem this thesis considers, namely the problem of Proof of Assets (PoA), a fundamental part for proving financial solvency on behalf of custodial wallets [17, 25], also known as Virtual Asset Providers (VASPs). As discussed in Chapter 1, it is a cryptographic evidence that the organization possesses sufficient assets which, combined with its proved liabilities, offers the so-called Proof of Solvency (PoSolv). The need of such proofs became even more apparent after infamous cryptocurrency exchange collapses, such as MtGox [22, 97].

This chapter focuses on practical PoA solutions in the Diem blockchain, however parts of our proposal apply to other systems as well. As regulatory compliance, transparency and fund safety are among the top priorities for Diem [29], PoA should be an important feature to achieve a safer wallet ecosystem. Diem's unique hierarchical account model differs from other blockchains and allows for several different PoA types that are not possible in other platforms. Our goal is to formalize and explore many different types of asset proofs in the Diem blockchain. Additionally, as we will show, Diem's PoA, in combination to Know-Your-Customer (KYC) identity verification, can also be useful to mitigate tax evasion, something that is not straight-forward in other blockchains where one can deny ownership of an address. In Diem, wallet addresses are pinned to particular entities, and thus VASPs cannot hide their owned assets on purpose.

In the following, we provide a summary of related work, a detailed analysis of PoA variants handcrafted to Diem's design, and finally practical recommendations for proof compression, aiming to make it more friendly for light (potentially mobile) clients.

The work presented in this chapter has been published in [28].

Our contributions. We first formalize the PoA requirements in account-based blockchains, focusing on the unique hierarchical account structure of the Diem blockchain, formerly known as Libra. In particular, we take into account some unique features of the Diem infrastructure to consider different PoA modes by exploring time-stamping edge cases, cold wallets, locked assets, spending-ability delegation and account pruning, among the others. We also propose practical optimizations to the byte-size of PoA in the presence of light clients who cannot run a full node, including skipping *Validator* updates, while still maintaining the 66.67% Byzantine fault tolerance (BFT) guarantee.

Related work. Bitstamp's Proof of Reserves [98] was one of the first attempts to provide evidence of a custodial wallet's total assets through an interactive protocol with a third party auditor. The process was to prove account-key ownership by signing over a provided random message; briefly, the ability to sign over a challenge string implies control and ownership of the account(s).

Provisions [17] presented a protocol based on zero-knowledge (ZK) proofs to prove assets, as part of a more general scheme to prove solvency. Its focus was to hide which accounts are owned by the audited entity. Briefly, the organization would form an anonymity set by adding random accounts from the public blockchain to those it already controls, and then prove (in ZK) that it knows a set of private keys that add up to or exceed some amount. Unfortunately, Provisions' custom ZK protocol cannot work with *hashed* public keys (which account for the majority of today's on-chain addresses), or with privacy-preserving cryptocurrencies (such as ZCash[99]) and its protocol's efficiency is linear to the size of anonymity set; thus, it cannot practically apply to most of today's blockchains. Agrawal et al. [100] made proving assets with hashed public keys possible as part of a zk-SNARK-based protocol combined with Σ-protocols (in a CRS model based on Pinocchio [43]), tailored for mixing arithmetic and boolean components. However, in addition to the setup assumptions, this approach is not efficient for large disjunctive statements (the size of the UTXO list in Bitcoin

is in the order of hundreds of millions) as as both the computational and space requirements scale linearly with its size, it has expensive concrete costs for the prover because of the underlying SNARKs and range proofs. In Diem, proving costs might be prohibitive in practice for Diem's ZK-unfriendly Pure-Ed25519-with-SHA512 signatures (including multi-sig); even with the latest recursive ZK proof schemes[101].

MProve [102] implemented a PoA algorithm tailored to Monero [6]. Since ring signature obfuscation does not allow for directly applying the Provisions PoA, its approach was to prove that the key images of the addresses controlled by the organization have not previously appeared on the blockchain. As PoA protocols are susceptible to collusion, MProve provides a proof of non-collusion as well by leveraging the one-time nature of key images. Unfortunately, this exposes the sender's identity when these key images are spent, potentially enabling tracing of transactions which breaks Monero's advertised privacy guarantees.

Wang et al. [103] proposed a scheme for a buyer proving assets to a vendor before finalizing a deal, using the transaction's details as a "challenge", which however is limited to a "buyer-vendor" use-case without any privacy characteristics. More importantly, it does not preserve the prover's privacy against the verifier (or regulator) as strongly as Provisions.

Blockstream's proof of reserves [104] consists of signing an "invalid" Bitcoin transaction for each owned Unspent Transaction Output (UTXO). This transaction cannot be published to the blockchain, however it still degrades the organization's privacy against the auditor. A similar approach is followed by Kraken cryptocurrency exchange [105]. The main advantage of this method is that hardware security module (HSM) or cold wallet implementations do not need an extra logic for signing PoA payloads and thus, it is directly backwards compatible with existing custodial wallets.

Ethereum [106] proposed a different payload format when signing a message other than a valid transaction[1]. The purpose of this distinction is to ensure that one should not accidentally sign a transaction masqueraded as a message nonce. In our PoA case, this

[1]Ethereum's message signing uses a flag prefix, to ensure an invalid transaction: *sign(keccak256"\x19Ethereum Signed Message:\n" + len(message) + message)).*

prevents an auditor from maliciously picking a hash of a transaction as an audit-nonce, which if signed, it could be submitted on chain without the user knowing. Also, Iconomi's proof of reserves [107] proved key ownership to Deloitte (auditor) through either signed nonces or predefined transactions from the proving addresses.

Finally, a recent work [63] provided definitions and systematization for several payment systems, including those offering PoA functionalities, and compared them in terms of their properties and efficiency.

4.2 Diem architecture

4.2.1 Keys and accounts

Diem [29] is an account-based blockchain payment system, currently maintained by a permissioned set of *Validators* which participate in its BFT-based consensus protocol[108]. Although there are no built-in privacy preserving protocols for its account states and transactions, due to its permissioned nature, all public queries (including blockchain correctness verifications) are proxied through *full nodes*, which have the same view of the blockchain as *Validators*, but without participating in consensus. Compared to traditional cryptocurrencies, Diem provides the following features:

- *Authentication* keys, known as *auth_keys*, are hashed versions of account public keys, however they can be rotated independently as a proactive or reactive measure to defend against possible key loss. This means that unlike Ethereum, a key rotation does not imply change of address.

- Diem natively supports single Pure Ed25519 [109] or threshold multi-sig (*k-out-of-n* up to $n = 32$) *auth_keys*.

- There exists the concept of *withdrawal capability*, where the permission to spend can be *delegated* to a different account. This implies that the spending key does not necessarily reside in the state of each address.

- It also supports the *key-rotation capability* where users can give permission to other accounts to update their *auth_keys*. This is useful for wallets where one can still refer to another *cold* address to gain access back to their account in case of accidental *hot* *auth_key* key loss.

- Account roles define the account owner's authority in the system. A unique characteristic of Diem is its hierarchical role-based access control [89]. Unlike Bitcoin and Ethereum, especially for VASPs, there exist a KYC-ed parent and child accounts as shown in Fig 4.1.

4.2.2 Hierarchical model

For the purposes of this work, we focus on Diem roles most commonly related to PoA: ParentVASP and ChildVASPs. A ParentVASP represents the primary account of a regulated custodial wallet, while multiple ChildVASPs can be created by ParentVASP accounts[2]. In Diem, a PoA will be requested from the ParentVASP , and these proofs should include all of their children's assets as well. This architecture is not privacy-preserving, due to the well-defined linkability of the accounts belonging to the same entity, and hiding owned addresses is not possible for KYC-ed VASPs. In Section 4.4.1 we provide details about the PoA related Diem data structures.

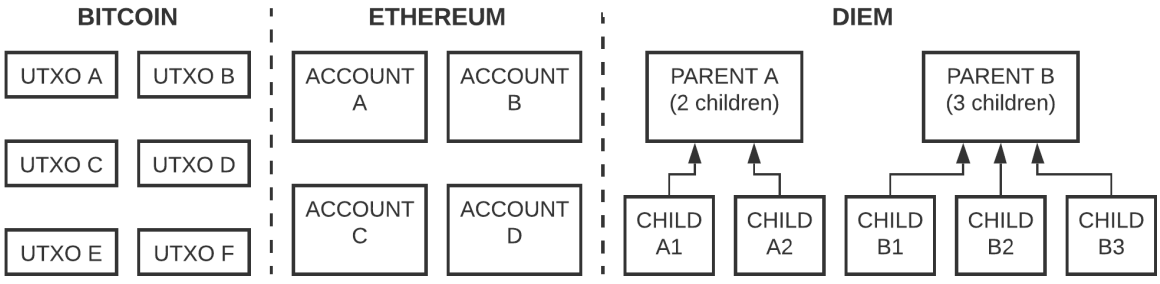


Figure 4.1: Address structure in different blockchains.

As an account-based system, Diem associates each account A with a value v_j at each

[2]Note that in Diem a ChildVASP is not allowed to have any other children itself.

block j. We denote by A^P and A^C accounts with ParentVASP and ChildVASP roles, respectively. An A^P can be linked to n accounts $A^C_1, A^C_2, ...A^C_n$. There is a relation F which maps each child to its parent account, i.e. $F(A^{C_i}) = A^P$. Note however that no inverse relation exists in Diem, i.e. the parent's state does not include a relation $F^{-1}(A^P) = [A^{C_1}, A^{C_2}, ..., A^{C_n}]$. This was probably a design decision to not allow parent account states growing indefinitely when more children are added, because for a large n that map would require significant storage space.

However, the data structure for A^P *does* include the cardinality n, a very important property to later ensure no child is missing in the proofs. Note that although a ParentVASP can create ChildVASPs, this does not necessarily mean that it controls the keys of its children, and ChildVASPs can transact independently. Of course, nobody prevents wallets from reusing the same key in multiple accounts or applying a BIP32 deterministic key derivation [110]. That said, the hierarchy is mainly enforced for KYC-ed account linking and splitting the risk of a key compromise attack; it also allows for different key and asset management policies, such as cold, warm and hot wallets or transaction sharding and parallelization[3].

4.2.3 Diem proof of assets

Generally, a PoA in Diem implies showing that a ParentVASP account is in possession of assets of some specific currency(ies) value. However, there is a subtle distinction on how to actually show this. One could merely use existing blockchain data structures, and sum the values of a ParentVASP and all of its ChildVASP accounts, based on account *ownership*. While straightforward, this proof does not provide key possession guarantees at the time of the auditing taking place. For instance, account holders might have lost access to their keys, which would make them unable to spend their assets. Therefore, we distinguish between the following two PoA types for a query on account A^P for a block j:

[3]While typical account-based systems require a sequence-id to prevent replay attacks, Diem's hierarchical model enables parallelization at the entity level, due to each child maintaining its own sequence-id.

Soft PoA: This proof is non-interactive, and a user (a third party auditor or even a light client) can obtain it at any time and for any block j via a series of blockchain requests to potentially untrusted nodes. It's simple goal is to provably present the total balance for all accounts belonging to the audited entity. No proof-of-knowledge of the spending key is required (and thus the name *soft*), however the parent account is linked with the KYC-ed entity; no other entity can claim this address's balance, and thus some applications might tolerate *soft* proofs. We highlight that this is only possible in Diem due to its hierarchical identity-address binding which makes collusion more difficult and traceable; a *WalletA* cannot just temporarily borrow its private key to a *WalletB* (an on-chain transaction should happen posing the risk of being censored). Such a proof is constructed by showing the following:

1. Given a genesis or any known checkpoint state with Merkle root r_G, prove that the Merkle root r_j is valid (see Section 4.4.1 for details on these data structures). In practice, the auditor will pick the block j for which the PoA is needed. Note that in Diem, this can be shown using a series of epoch change proofs to get the validator-set at block j.

2. For r_j, provide Merkle inclusion proofs for both parent A^P and its children $A^C_1, .., A^C_n$ account states.

3. All related account state balances (i.e. $A^P, A^C_1, .., A^C_n$) sum up to a value V. In PoSolv, this V is typically compared against proofs of liabilities [111].

4. $F(A^C_i) = A^P, \forall i \in (1, ..n)$, where n is the cardinality in account state A^P. This ensures that no child is accidentally or purposely omitted from the list.

Hard PoA: A *hard* PoA is requiring a key-ownership proof on top of *soft* proofs, usually via signing. To prevent replay attacks, the protocol should require each account to sign over some random challenge. Note that it is acceptable to sign with the *auth_key* (or a delegated key via withdrawal capability), valid at a requested block in the past or the most

recent one. We refer to these two types as *dated*-hard PoA and *live*-hard PoA, respectively (further discussed in Section 4.4).

4.3 Implementation considerations

4.3.1 What message to sign?

As previously discussed, *hard* PoA simulates a proof of key-possession by signing over a challenge r to prevent replays. This can either be a "special" hard PoA transaction, included as metadata, or even run off-chain. Options for r include any combination of the following:

- a random string interactively provided by the auditor.
- the hash of the block (or state snapshot) at height $(j-1)$. Note that Diem has the concept of transaction *version*, which is a monotonically increasing integer for all of the on-chain transactions. The latter means that one can even take a snapshot at the middle of the block, but typically, when we refer to height we imply the *version* number of the last transaction in a block.
- the latest Bitcoin block or from other robust proof-of-work blockchains (thus, use an external reference for randomness). However, that would require running a mini light client as a smart contract or trusting an Oracle service that could verify correctness of the external seed input.
- the output of a distributed randomness generation protocol (such as RandHound [112]), which can even be run by Diem *Validators* at each block.
- other publicly verifiable sources of randomness which embed timestamp information [113], such as the closing stock prices in the stock market, weather conditions in major cities etc, ideally with the use of verifiable delay functions (VDFs) [114].

However, some of the above randomness sources are susceptible to collusion attacks. For instance, the auditor and the ParentVASP might collude on the provided randomness in advance, or consensus *Validators* might agree to form a predictable block in Diem (this might be tolerated by the BFT assumption). Therefore we prefer a combination of external verifiable randomness and the RoundHound protocol run from *Validators* which can offer better transparency guarantees. In short, we need a verifiable random and fresh challenge to ensure that the prover could not have predicted and pre-signed it long ago.

While hard PoA could also be automated to be executed at some predetermined times, the above randomness or challenges need to be unpredictable to prevent misbehavior. Note however that unpredictability is weaker than being "bias-proof", a property required by other use-cases (e.g. lottery protocols). For instance, in a lottery protocol an attacker's goal could be to increase the probability of outputting a string that ends in 0. However in the case of hard PoA, biasing the result in this way would have no benefit for the attacker as we're only interested in signing over a fresh unpredictable challenge. More information on what data to-be-signed offers the above freshness and unpredictability properties is provided in Section 4.4.3.

4.3.2 Various PoA considerations

Account state pruning: Many blockchain systems (including Diem) conserve space by pruning old account states, but still keeping the state's hash to preserve the system's security. Therefore, if the latest blockchain height is m and a PoA is requested for some height $j < m$, the full account state containing a balance v_j might not be available on-chain. In this case, the account's state would have to be recovered by a full-node who maintains the full history. Validating the provided pruned state is easy; we just check if the state's hash-output equals the blockchain-maintained hash value for this account.

Cold wallets and valet keys: Hard PoA might be cumbersome when air-gapped wallets are involved, as performing such an operation would require bringing keys out of cold storage. The process sometimes requires expensive ceremonies, i.e. when the key resides

in HSM modules or physical vaults, or when it is split between several parties. A possible approach to improve usability could be a) embedding PoA operations in HSM or b) using valet keys as defined in [115].

Incentives: When proving solvency, malicious auditees might collude to temporarily prove assets *greater* than some value that represents their off-chain liabilities. On the other hand, other auditees might try to *hide* assets on purpose (e.g. for tax evasion purposes). This would be a problem in any system other than Diem, where the auditee could simply claim loss or non-knowledge of some key, and complex blockchain analysis techniques (e.g. clustering) would have to be deployed to prevent such behavior. However Diem's hierarchical, KYC-ed account model mitigates this.

Locked assets: In our model we do not consider locked on-chain assets, i.e., for future atomic swaps or side-chain smart contracts (locked assets are not supported by Diem yet). In fact, proving solvency by taking locked assets into account is an open research challenge in every blockchain, as discussed in the recent ZKProof 2020 workshop [111].

4.4 Diem-specific implementation considerations

4.4.1 Primitives and soft PoA implementation in Diem

Sparse Merkle Trees. Recall a Merkle tree [48] is a binary tree constructed by a collision-resistant hash function h, providing logarithmic proofs with logarithmic complexity. Sparse Merkle trees share the same philosophy, however tree-leaves do not contain the accumulated elements themselves but serve to form an "index" of the element along with its path to the root. This enables them to provide proofs of non-membership, where non-accumulated elements can simply end in a placeholder value to maintain tree balance. However, as these classes of Merkle trees are intractably large, we can also represent them by omitting sub-trees that only contain placeholder values. Diem uses a variant of Sparse Merkle trees (Jellyfish Merkle trees [116]) which enables shorter inclusion/exclusion proof sizes while still providing collision resistance.

In Diem, transactions are accumulated in a Merkle tree, which in turn contains roots of sparse Merkle trees that represent the state of all accounts as the transaction gets executed [29, 117]. The top Merkle tree root defines the block hash and is signed by the *Validators* participating in the consensus (at least 66.67% of them should sign) as transactions are processed and account states are modified accordingly. We describe a specific data structure format in Diem below.

Diem Data Structures [118, 119]. In Diem, account states are represented as an `AccountStateBlob` which includes, among others, the address, balance for each currency and account role (i.e., ParentVASP or ChildVASP). These account states are stored in a sparse Merkle tree called `TransactionInfo`. In turn, this sparse Merkle tree's root hash `state_root_hash` represents all of the accounts' global state at the end of a specific transaction.

In turn, the *most recent* `TransactionInfo` root in an blockchain version, along with the epoch number corresponding to the current *Validator* set and a timestamp, are encapsulated in a `BlockInfo` data structure. This data structure along with a hash value of the consensus Quorum Certificate is encapsulated in a `LedgerInfo` Merkle tree. Note that a version's most recent Transaction (e.g. transaction T4 in Figure 4.2), effectively defines the global state of all accounts for that version.

Finally, `LedgerInfo` along with consensus signatures by the current *Validator* set is encapsulated in a `LedgerInfoWithSignatures` data structure, making it acceptable by anyone trusting Diem's BFT assumptions.

Proofs. A core object for implementing Diem soft PoA is the `AccountStateProof` data structure. This contains a sparse Merkle tree proof (`SparseMerkleProof`) for a `TransactionInfo` object, which in turn is verified by a `TransactionInfoWithProof` proof for the Merkle tree. A second crucial element is `EpochChangeProof`, which includes the list of signatures involved in *Validator* set updates. Through these built-in proof functionalities in Diem, we implemented a CLI Soft PoA functionality [120] which returns total on-chain assets owned by

Diem ParentVASPs.

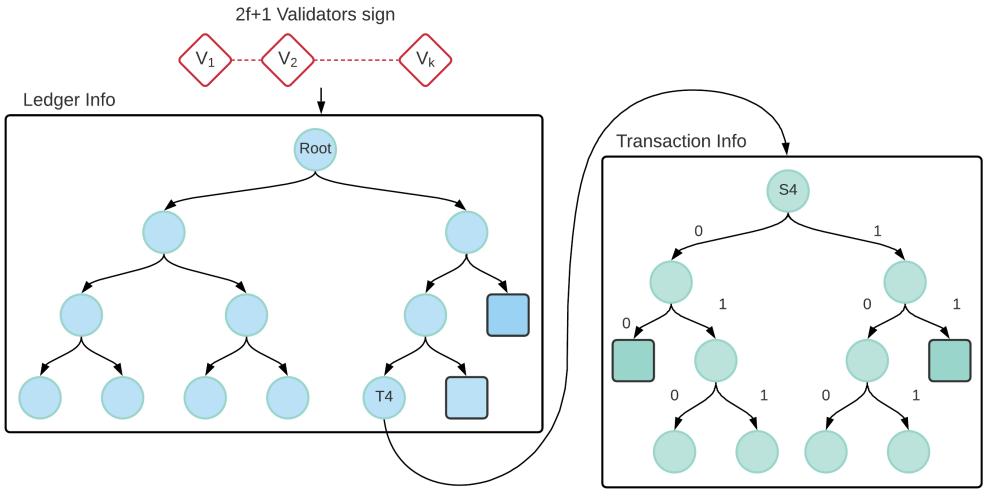


Figure 4.2: Diem data structure overview.

4.4.2 Random challenge consistency

It is recommended, especially when BIP32 [110] is applied or the same key is used between accounts, that r should be the same across all signed messages to minimize proof size. However, as keys in Diem are rotated regularly (discussed in Section 4.2.1), there are two options for signing a hard PoA for time[4] t: a) use an authentication key that was valid at a past instance t, and b) use the most recent authentication key (which will be linked to the key at t using a chain of rotations). While both PoA types are acceptable for proving asset control at t, the latter version is stronger as it also shows key control for a more recent time $t + \Delta$. A reason for picking a slightly older t might be to reduce the probability of wallet collusion; if one does not know for which t they will be audited, temporarily borrowing private keys from other wallets is riskier. It is highlighted though that wallet providers might have deleted old account keys, and thus a t closer to the current time/block should be preferred, unless there is a reason not to, e.g. for proving assets

[4]We assume t is in the past. While it could be possible to make a PoA request for some time in the future in advance, this enables several collusion attack vectors not addressed in this thesis.

exactly at the end of a calendar year.

In any case, we refer to the above two hard PoA types as *dated*-hard PoA and *live*-hard PoA respectively. We mention that especially for cold wallets, it is advised the auditee signs and rotates the keys simultaneously to ensure some additional (although not complete [121]) post-quantum security, due to publishing hashed keys.

4.4.3 Signed block hashes as randomness

In the previous section we discussed that block hashes can be used as a randomness source to sign a hard PoA message, preferably in combination with other randomness sources. Specifically in Diem, to prevent an attacker from manipulating this source, we would pick the root of the `LedgerInfo` tree that includes $2f + 1$ Validator signatures (thus a `LedgerInfoWithSignatures` object), where f denotes the upper bound of Byzantine Validators. Therefore, to manipulate this information, an attacker would need to also subvert more than f Validators which in turn would break the assumption of Byzantine Fault Tolerance. Note that a dishonest leader could in theory selectively pick any $2f + 1$ signature combination when all Validators sign, but fortunately this does not give any advantage as we are interested in a fresh and unpredictable, but not necessarily unbiased, challenge.

4.4.4 Accurate timestamping

Diem's blocks use monotonically increasing timestamps. This implies that (unlike other blockchains) one could use a time reference t instead of a block-height j. In addition, all PoA elements should be consistent for a specific block, with the proof showing the total assets for a *snapshot* of the *same* blockchain height (or timestamp) across all ParentVASP and ChildVASPs. If a variation in height was allowed, malicious provers could move assets among their accounts in neighboring blocks and falsely claim assets greater than those actually owned.

Also, as mentioned before, Diem uses "versions" rather than "blocks-heights", with each transaction resulting in a unique, incremental version. Therefore, as each block has

subsequently a *range* of versions, the account states in the latest version in a block need to be retrieved [116]. This can be implemented through appropriate GetVersionByTimestamp() and GetStateByVersion() functionalities, which would return the blockchain version for some specific timestamp and the blockchain state for some version respectively. Note that as shown in Figure 4.2, the latest transaction in an epoch should be considered (transaction T4) for all account state proofs, and prove that the immediate next transaction belongs to the next epoch. This ensures that account proofs are provided after all transactions in the block have been considered.

4.4.5 Compression

Signature compression: Signatures and public keys account for the largest part of a PoA payload. Actually, there exist three types of signatures:

1. *Validator* signatures over the block data.
2. account signatures for every transaction in a block.
3. key-possession-proof signatures for each auditee key (potentially delegated).

In PoA we are interested in the first and third signature types. Compression can be achieved through various techniques, but some of them require a Diem protocol update and thus, they cannot be applied directly. Examples include having the *Validators* running interactive multi-sig protocols, such as Musig2 [122] and FROST [123], or supporting BLS signatures [124], which allows for aggregation to a single signature (although we still need the public keys). Solutions not requiring a protocol update include the SNARKs [44], STARKs [101] or the recent non-interactive EdDSA half-aggregation[125]. However, for auditee signatures over a challenge, the prover, who controls all of the keys, can simulate a Musig2 in-the-head or apply the Schnorr batching technique of [126].

Epoch proof compression: At the moment, Diem's epoch-change proofs are sent in raw format, without tackling duplication between epochs. We present an easy to implement

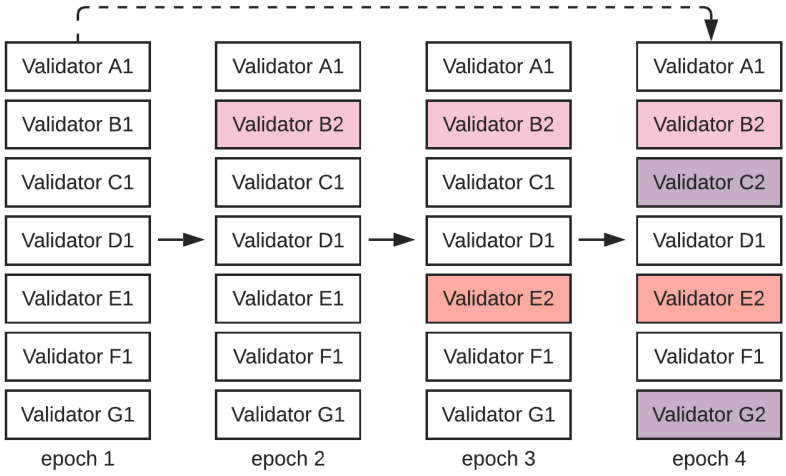


Figure 4.3: Epoch skipping optimization.

partial compression method without advanced ZK protocols.

Normally, to verify epoch changes, where at least one *Validator* rotates its key, we have to verify all intermediate epochs from the last known checkpoint. However we can skip epoch verifications if less than 1/3 *Validators* have been updated, and only require to "jump" to an epoch where a sufficient number of *Validators* have changed (concept shown in Fig 4.3). Note however that this optimization is incompatible with long range attack prevention [127]; still, this might be tolerable in some threat models. We can further optimize epoch proofs by only considering the required $2f + 1$ signatures (omit the rest) along with their key rotation operations, even when all *Validators* have signed.

4.4.6 Multiple currencies

Note that Diem might support several currencies, and asset proofs might be required across all of them. Our recommendation is that PoA should run per currency (but again for the same height). Converting all currencies to a single one using the current exchange rate is not advised for PoSolv purposes [111], as there are examples of extreme volatility (i.e., the case of Swiss franc cap removal on Jan 15, 2015 [128]).

4.4.7 PoA transaction type

In general, as an alternative to a carefully signed message that is distinguishable from a regular transaction by design [129], transferring some amount (or even a zero amount) to a (designated) address would also work for PoA purposes, especially if one wants on-chain PoA recording. In Diem, a hard PoA could also be executed through a special transaction type, with the sole purpose of signing a message, however such "NO-OP" transactions are not yet implemented. Fortunately, Diem allows sending funds to self, which is one way to implement hard PoA. Another option would be to send some amount to a predetermined "sink" account. Such approach has the advantage of consolidating all associated PoA events and making them easier to track.

4.4.8 Withdrawal capability

As discussed in Section 4.2.1, Diem has the unique functionality of granting the capability of spending to other accounts and smart contracts[130]. This delegation mechanism introduces additional complexity when proving assets. Because of withdrawal capability, the PoA message signing should happen on-chain, which would make sure the smart-contract logic that involves withdrawal capabilities would execute. Otherwise, off-chain verifiers would need a copy of the current blockchain's state and be able to simulate transaction execution in this copy, which would make the whole process expensive or cumbersome. Another issue is related to potentially incompatible implementations of custom withdrawal capability logic (smart-contract), because currently there is no enforcement of requiring additional metadata (which in our case is required to attach the random challenge).

4.5 Conclusion

We presented several considerations for implementing proof of assets in the Diem blockchain. By taking advantage of Diem's native hierarchical account structure, two major policies of

asset proofs have been analyzed: *soft* PoA, which can be executed at any time without interaction with account holders, and *hard* PoA which provide extra assurance that account holders are in control of their keys. However, the latter requires a more carefully planned, coordinated interaction.

All of our proofs rely on widely-used cryptographic primitives with standard assumptions (i.e. signatures and Merkle proofs). We discuss several edge-cases that should be taken into account when designing PoA protocols in a hierarchical, KYC-ed, account-based blockchain system (e.g. timestamping and consistency), and propose practical solutions, including options for fresh and unpredictable proof of key-possession challenges. Finally, we propose easy to implement optimizations (e.g. signature and epoch change proof compression), while still remaining compatible with the underlying Diem blockchain.

Chapter 5: gOTzilla: Efficient Disjunctive Zero-Knowledge Proofs from MPC in the Head, with Application to Proofs of Assets in Cryptocurrencies

5.1 Introduction

We continue to consider the Proof of Assets problem at a macroscopic level, and in this chapter we discuss the aspect of regulation in a large scale, permissionless distributed payment system such as Bitcoin [1]. The work presented in this chapter has been published in [30], and is also of independent interest for the cryptography community as discussed below.

As discussed in Chapter 2, a zero-knowledge (ZK) proof [131] allows a prover $\mathcal{P}$ to convince a verifier $\mathcal{V}$ that a statement x is true without revealing any further information. ZK proofs have numerous applications: they are used as a building block in various cryptographic constructions such as secure multiparty computation [132], signatures [133] and anonymous credentials [134] just to name a few, and more recently they have been used as a core component in privacy-preserving cryptocurrencies [5].

ZK proofs can be constructed generically for any NP language [132], however, such generic constructions are usually not efficient. In order to achieve practical constructions, customized ZK proofs have been designed for specific languages (e.g. particular algebraic statements), or with specific optimization goals (e.g. proof size succinctness or non-interactiveness). Many different approaches have been proposed, each with different trade-offs on the types of supported languages, efficiency goals and underlying assumptions. In terms of efficiency, the tradeoffs appear in the prover complexity, the verifier complexity, and the communication costs.

Here we focus on zero-knowledge proofs that can efficiently support very large *disjunctive* statements, and the Proof of Assets (PoA) problem in UTXO-based cryptocurrencies, where a prover (usually some exchange or other organization) wishes to convince a verifier that it knows the respective private keys of *at least* a certain number of coins on the blockchain (without revealing which those coins are), is our application scenario. In Bitcoin, and other cryptocurrencies with similar structure, PoA can be expressed as a *disjunctive* ZK proof where the statement is a set of hashed public keys (often called addresses), and the witness is one or more secret keys that correspond to some of the public keys. The challenge when computing a ZK proof for PoA is bifold: (a) the size of the statement grows with the total size of the Bitcoin UTXO set, which has hundreds of millions of elements, and (b) the statement is a combination of an algebraic circuit – the discrete log relation between $(\mathsf{sk}, \mathsf{pk})$ – and a Boolean hash function, since the prover needs to prove that it knows the secret key for one of the hashed public keys. Concrete protocols for the PoA application have been designed in the literature [17,100], but as explained in related work (Sections 4.1 and 5.1.1), they fall short in addressing the two main design challenges simultaneously.

Our Construction. We first focus on the challenge of dealing with very large statements. Specifically, we are interested in statements of the following form: for a publicly known circuit C, and a publicly known set of values $Y = \{y_1, \ldots, y_n\}$, the prover wishes to prove that it knows a witness x such that $C(x) = y_1 \vee C(x) = y_2 \vee \cdots \vee C(x) = y_n$. A simple restructuring of this statement allows us to remove the n copies of C, greatly reducing the statement size. The prover witness is modified to be a pair of values, (x, y), such that $(C(x) = y) \wedge (y = y_1 \vee \cdots \vee y = y_n))$. As we discuss below, this provides significant improvement even for existing proof systems that support disjunctions, as the size of an equality circuit is much smaller than $|C|$.[1] Once re-written in this form, we are able to reduce the disjunction of equalities to 1-out-of-n Oblivious Transfer (OT).

[1]Generically, a proof system that efficiently supports disjunctions might not support the conjunction of C with the disjunction of n equalities. In practice, existing systems seem to handle this change without significant complication.

We build a ZK proof using the MPC in the head paradigm (MPCitH). Our main observation is that when proving that y is equal to one of the elements in the public set Y, it suffices to enforce constraints on the Prover's set of inputs to the MPC in the head. We do that by having the Verifier prepare an encoding of all n possible inputs y_i for the MPCitH and having the Prover select a single input encoding obliviously, using 1-out-of-n OT. The Verifier creates these encodings such that the portions of the encoded input that are revealed to the verifier in the opened MPCitH views are identical for all y_i. This ensures that the view can be safely opened for verification, without revealing the index i. We implement 1-out-of-n OT using Private Information Retrieval (PIR). Note that PIR is a relaxation of 1-out-of-n OT, in that it potentially allows the receiver to learn more than one value. We strengthen PIR to OT by performing a zero-knowledge proof on the Prover's PIR query to show that it is well-formed (i.e. a valid ciphertext encrypting a query for only a single database element). In our 1-out-of-n OT protocol (as well as in the rest of the protocol) we enforce honest Verifier behavior by committing to, and later revealing, the Verifier's random tape, allowing the Prover to check that the Verifier's messages have all followed the protocol. We can only do this because the Verifier's inputs are random challenges that do not require privacy beyond the end of the protocol; for general purpose 1-out-of-n OT, this approach cannot be used for ensuring the honest behavior of the sender. By building 1-out-of-n OT from PIR, we achieve communication complexity of $O(\log n)$, which allows us to achieve an overall communication cost of $O(\log n) + \Pi_{Proofsize}$ where $\Pi_{Proofsize}$ denotes the proof size of the MPCitH protocol. We present our protocol in Section 5.3.

ZK Proofs on Mixed Statements. An efficient disjunctive proof however, is not enough to efficiently prove the concrete PoA statement, i.e. "I know sk such that: $(H(\mathsf{pk}) = y) \wedge (y = y_1 \vee \cdots \vee y = y_n))$" where $\{y_1, \ldots, y_n\}$ is a list of hashed public keys using function H and pk is the public key that corresponds to sk. One could convert our algebraic statement (on the relation between $(\mathsf{sk}, \mathsf{pk})$) to a Boolean circuit, but, as we discuss later in Section 5.5, this would result in a circuit with millions of gates. A number of works examined the problem of efficiently combining algebraic to non-algebraic statements in a ZK proof. Chase et al.

[49] provided one of the first techniques based on Garbled Circuits, MACs and Oblivious Transfer which was further optimized in [100], and later Backes et al. [135] presented a technique on an MPCitH Σ-protocol inspired from ZKBoo [136], however these works were not taking a disjunctive statement into account. In Section 5.4 we present an extension of our disjunctive proof that supports mixed statements, and in Figure 5.9 we further extend our mixed statements protocol to also handle the value that corresponds to the secret key (i.e. witness) of the Prover (a property needed towards designing a PoA protocol).

Evaluation Results. We evaluate gOTzilla on top of an existing PIR implementation, namely SealPIR [137], by implementing our techniques to derive a 1-out-of-n OT protocol, while treating the underlying MPCitH protocol as a black box. Our results are presented in detail in Section 5.5, where we show that we can prove knowledge over a disjunctive statement of $n = 2^{20}$ elements in 14.89 seconds, with 6.18 MB of communication and 10 seconds network latency at the worst case, for statistical security parameter $\lambda = 40$. Our evaluation shows a significant improvement in total protocol run-time over Mac'n'Cheese [138] the state of the art in disjunctive proofs, which has similar asymptotic costs to us as discussed in the Related Work section below.

5.1.1 Related Work

We provide a short, non exhaustive, overview of common ZK proof types for disjunctive proofs and mixed statements, as well as an overview of solutions specific to the PoA problem.

Standard ZK Techniques. Σ-protocols form a well studied class of efficient protocols specifically for algebraic statements, such as discrete logarithms and roots [139, 140], while garbled-circuit approaches were used to efficiently prove Boolean circuits [141]. We note that if one attempted to use a Σ-protocol to prove a statement about a function represented as a Boolean (or arithmetic) circuit C, both proving and verification costs would grow linearly with the size of the circuit (a simple SHA256 evaluation would result in tens of thousands of exponentiations) which makes them prohibitive for a PoA like statement.

ZK-SNARKs are better suited for Boolean or arithmetic circuits and while they could

be used for algebraic statements, they would require circuits with thousands or millions of gates for a simple computation like an exponentiation exploding the prover's cost.

Some recent works [45, 142–145] use techniques such as interactive oracle proofs (IOP), vector oblivious linear evaluation (VOLE) and the MPC in-the-head paradigm without relying on a setup phase, yet, they still impose high computational costs on the prover side thus are not directly relevant to the considered PoA application.

Disjunctive ZK Proofs. A number of related works have examined the general problem of building efficient disjunctive ZK proofs. Following the seminal work by Cramer et al. on constructing standard disjunctive proofs [91], Stacked Garbling [146] proposed a garbled-circuit approach for creating a disjunctive proof with sublinear communication complexity, based on Jawurek et al. [141]. Later, Stacking Sigmas [147] provided a generic compiler for reducing communication complexity (i.e "stacking") of disjunctive Sigma-protocols satisfying a specific "stackable" property, and is compatible with recent MPCitH style ZK protocols such as KKW [143] and Ligero [142].

Table 5.1: Asymptotic comparison of disjunctive ZK proof systems for n statements for a single circuit C. NI = Non-Interactive. Π denotes an MPCitH protocol, $\Pi_{Runtime}$ and $\Pi_{Proofsize}$ denote Runtime and Proofsize of Π, respectively.

	No setup	NI	Prover Runtime	Proof size
Ligero [142]	✔	✔	$O((n+\|C\|)\cdot\log(n+\|C\|))$	$O(\sqrt{n+\|C\|})$
Π + Stacking Sigmas [147]	✔	✔/✘	$O(n)+\Pi_{Runtime}$	$O(\log n)+\Pi_{Proofsize}$
Mac'n'cheese [138]	✔	✘	$O(n+\|C\|)$	$O(\log n+\|C\|)$
Goel et al. [148]	✔	✔	$O(n)+\Pi_{Runtime}$	$O(\log n)+\Pi_{Proofsize}$
gOTzilla	✔	✘	$O(n)+\Pi_{Runtime}$	$O(\log n)+\Pi_{Proofsize}$

Recently, Mac'n'Cheese [138] proposed a new, VOLE based approach to build generic zero-knowledge proofs for disjunctive statements of the form $(x, i) : C_i(x) = y_i$ for $i \in \{1, \ldots, n\}$ with communication cost of $max_i\{|C_i|\} + \log n$. In the general case where each C_i is a different circuit, both the prover and verifier have to execute all branches to construct or verify the proof, causing the total computation cost to be $O(\sum_{i=1}^{n} |C_i|)$. In the special case where all the circuits C_i are identical, using our observation above that restructures the

disjunctive portion, their construction can be slightly modified to improve the computational cost to $O(n + |C|)$.

In a concurrent to ours work, Goel et al. [148] provided a membership proof protocol towards building a ring signature, which is equivalent to our observation discussed above (i.e. restructuring a disjunctive proof statement to a disjunction of equalities). Similar to our construction, this work relies on an underlying MPCitH protocol and has equivalent asymptotic costs, however it follows a cut and choose approach which naturally implies higher concrete computational costs, while having reduced concrete communication costs. In addition, being public-coin, it can be converted to a non-interactive protocol in the random oracle model using the Fiat-Shamir transform. However, as we later discuss in Section 5.4.1, interaction is naturally implied for our application scenario, and we discuss the tradeoffs between computation and communication costs in Section 5.5.

In Table 5.1 we provide a comparison of basic techniques for disjunctive statements. We note that although asymptotically we might have similar performance as Mac'n'Cheese [138] or Stacking Sigmas [147] combined with a suitable MPCitH protocol Π, we have significant concrete improvements. In Section 5.5.1 we present a concrete comparison of our disjunctive protocol with Mac'n'Cheese [138] to showcase our improvement by 4x in runtime for the case of "mixed" disjunctive statements. There is no available implementation of Stacking Sigmas [147] for a direct comparison, however Stacking sigmas is expected to be more expensive than Mac'n'Cheese concretely due to its underlying techniques (Stacking sigmas relies on commitments with elliptic curve operations which are more expensive than VOLE used in Mac'n'cheese). Also, there is no available implementation for Goel et al. [148] therefore our comparison is based on their evaluation. We note that a caveat of our approach is that we generally have larger memory requirements as opposed to Mac'n'Cheese where the prover and verifier are not required to store the entire proof statement in memory. However, as we discuss in Section 5.5.1, gOTzilla can optimize RAM usage by generating the required data on the fly as needed to improve our storage costs.

1-out-of-n Oblivious Transfer and PIR The connection between PIR and Oblivious

Transfer was studied before in [149,150] (where 1-out-of-n OT was also referenced as "Symmetric PIR" or SPIR). These works provided transformations of PIR to SPIR, which however have an overhead in computational and/or communication costs.

While most 1-out-of-n OT protocols require linear communication, Zhang et al. [151] presented a protocol with $O(\sqrt{n})$ communication costs, while also proposing using PIR in conjunction with the appropriate ZK proofs. The protocol is quite practical (for short messages) in terms of computation time, however, the communication cost is high. For $n = 10^6$ and the message size of 192 bits, it took their protocol 30 seconds and 480 MB on an Intel Core i5-2400 CPU running at 3.10 GHz in LAN setting. Beside the high communication cost, another drawback of their protocol is that the message space is restricted by the size of the group used in their protocol. When the message length is at least 10000 bits (as in our use case), it is not clear how to modify [151] to make it work while still being practical[2].

5.2 Oblivious Transfer from Private Information Retrieval

A Private Information Retrieval (PIR) protocol [152] between a receiver R and a server S which owns a database D consisting of items $y_1, \cdots y_n$, enables R to retrieve some item y_i from D without S learning any information about i. Intuitively, PIR is similar to a 1-out-of-n OT protocol, with the main difference being that it only protects the privacy of R's input and assumes semi-honest behavior from both parties. In this section we construct a protocol for 1-out-of-n OT built on top of SealPIR [153]. Note that this construction cannot generically be applied to arbitrary PIR protocols, as it relies on properties of SealPIR's construction.

Privacy against semi-honest receivers. SealPIR is constructed from the additive homomorphic encryption scheme BFV [154,155] based on Ring-LWE. As privacy is not a concern in a PIR protocol, SealPIR packs many y_i to fully utilize the large plaintext supported by Ring-LWE, and computes $f(b, y) = \sum_{j=1}^{n/k} b_j \cdot Y_j$ where $Y_j = (y_{1+(j-1)k}, \ldots, y_{jk})$, k is

[2]It will be too costly to use a group of size 10000 bits.

the number of y_i's that can be fitted into one plaintext, $b_j = 1$ if the selected item is in $[1 + (j-1)k; jk]$ and $b_j = 0$ everywhere else. In the protocol, b_j's are encrypted and compressed by the receiver, decompressed by the sender who sends back the encrypted of $f(b, y)$. Finally the receiver decrypts the ciphertext and obtains $f(b, y) = Y_j$.

We observe that without packing the y_i, the protocol actually computes $f(b, y) = \sum_{j=1}^{n} b_j \cdot y_j$ and realizes a semi-honest 1-out-of-n OT protocol (with less efficiency if the plaintext has too much empty space).

Security against a malicious receiver. To achieve security against malicious receivers, after sending its query the receiver performs a zero-knowledge proof that the query is "well-formed" (i.e. is an encryption of a plaintext with exactly 1 nonzero index). We describe this protocol in figure 5.1.

First, $\Pi_{ZK}^{WellFormed}$ guarantees that the encrypted query c_v is a correctly-constructed ciphertext of a known plaintext with bounded noise using techniques proposed by Chen et al. [156]. The rest of the protocol proves that the underlying plaintext is a well-formed.

The server creates a challenge by sampling a random element r_i and random polynomial Q_i and an $(n-1)$-out-of-n shamir-sharing of Q_i (denote the vector of shares as q_i). The server then homomorphically computes $c_i := r_i \cdot c_b + Enc(q_i)$ and sends c_i to the receiver.

If the query is well-formed, then the decryption of c_i will have enough unmodified shares to reconstruct Q_i. More specifically, the decrypted plaintext will contain $n - ||b||$ shares of Q_i where $||b||$ is the number of nonzero elements in the plaintext query. Hence if the query contains > 1 non-zero elements the receiver is unable to reconstruct Q_i. We repeat this process (in parallel) to achieve the desired level of soundness.

Security against a malicious sender. To make $\Pi_{ZK}^{WellFormed}$ malicious-secure, we observe that the server only needs to keep Q_i and r_i private until the receiver has sent its response. Additionally, once the receiver knows Q_i, r_i and the randomness used to encrypt q_i it can deterministically recompute the honestly-generated c_i to verify honest behavior of the server.

Along with the challenges c_i, the server now sends a commitment to a PRG seed s from which all other random values are sampled. The receiver commits to its responses, then the server opens the seed to the receiver. The receiver recomputes the challenges and verifies that they match what the server originally sent. If so, the receiver opens its responses to the server.

For the overall 1-out-of-n OT protocol we apply a similar methodology. First the server commits to its PRG seed and database input, and later opens this commitment so that the receiver may check for honest behavior. In order to preserve the receiver's input privacy, this check must occur before any computations based on the received value are revealed to the sender. To preserve soundness, it must occur after all relevant prover computations have been run and their outputs committed. Because of this, our implementation of $\Pi_{OT}^{1:n}$ only attains security against semi-honest S and malicious R. When using $\Pi_{OT}^{1:n}$ in a larger protocol, we augment it with the PRG trick to reach malicious S security. As an optimization, we use a single seed for all instances of 1-out-of-n OT, allowing the server to reveal the databases for all instances simultaneously.

Theorem 2. *Protocol* $\Pi_{ZK}^{WellFormed}$ *is a Zero Knowledge proof that an encrypted PIR query* $Enc(\mathbf{b})$ *satisfies the condition:* $\nexists (i \neq j)$ *s.t.* $b_i \neq 0 \wedge b_j \neq 0$.

Proof (Sketch). Soundness: If the prover cheats by setting more than one entry of $\mathbf{b}$ to be non-zero, it will not have enough information to reconstruct $\widetilde{Q_i}$. As a_{ij} are sampled uniformly at random, $Q_i(0) = a_{i,0}$ is also uniformly random. In order to pass the check, the prover can only guess $a_{i,0}$ and has the probability of $1/t$ to guess it correctly. In overall, the chance that the verifier passes the check if it cheats is $t^{-\sigma} < 2^{-\lambda}$.

Zero-knowledge: It is clear that the encryption of $\mathbf{b}$ preserves the privacy of the selection index. This is due to the property of the encryption scheme. In the protocol, the prover only reveals $Q_i'(0)$ after seeing the seed used to generate the challenges by the verifier. Thus, the verifier has no way to deviate from the protocol without being caught. If the verifier abides by the protocol, it learns nothing from the answers. If the verifier cheats, it will not

$\Pi_{ZK}^{WellFormed}$

Setup. Ring-LWE scheme with parameters (N, t, q) where N is the degree of the cyclotomic polynomial, t the plaintext modulus, and q the ciphertext modulus. The prover has the key pair (sk, pk), while the verifier has the public key pk. σ is the soundness amplifier such that $t^{-\sigma} < 2^{-\lambda}$.

Prover's input. $\mathbf{b} \in R_t[X]/(X^N + 1)$ and $k \in [0, N)$ such that $b_k \neq 0$ and $b_i = 0$ $\forall i \neq k$.

Commom input. $c_b = \mathsf{Enc}(\mathsf{pk}; \mathbf{b})$ where $\mathbf{b} \in R_t[X]/(X^N + 1)$.

Protocol.

1. The prover sends a proof on the validity of ciphertext c_b which includes that the Ring-LWE noise is bounded.
2. The verifier samples a random seed $s \in \{0,1\}^\kappa$. For $i \in \{1, \cdots, \sigma\}$ the verifier samples $r_i, Q_i(X) \leftarrow \text{PRG(s)}$ where $r_i \in \mathbb{Z}_t$, $Q_i(X) = \sum_{j=0}^{n-1} a_{ij} X^j$, $a_{ij} \leftarrow \mathbb{Z}_t$, and computes $\mathbf{q}_i = (Q_i(1), \cdots, Q_i(N)) \in \mathbb{Z}_t^N$. It uses the additive homomorphic property of Ring-LWE to compute $c_i \leftarrow \mathsf{Enc}(\mathsf{pk}, r_i \cdot \mathbf{b} + \mathbf{q}_i)$. After that, the verifier sends $\mathsf{Com}(s)$ and c_i to the prover.
3. The prover decrypts c_i, obtains $\mathbf{c}_i = r_i \cdot \mathbf{b} + \mathbf{q}_i$, interpolates ${Q'}_i$ from the points (j, c_{ij}), where $j \in \{1, \cdots, N\}$, $j \neq k$. It then sends $\mathsf{Com}({Q'}_i(0))$ to the verifier.
4. The verifier decommits s to the prover.
5. The prover verifies that c_i is correctly generated. If so, it decommits ${Q'}_i(0)$ to the verifier.
6. The verifier checks that ${Q'}_i(0) = a_{i,0}$.

Figure 5.1: Zero-knowledge proof to prove that the encrypted ciphertext c_v is well formed and at most one of $b_i \neq 0$.

$\Pi_{\text{OT}}^{\text{1:n}}$

Receiver input. $b = \{b_1, \ldots, b_n\}$ where $\exists i \in \{1 \ldots n\} : b_i \neq 0 \wedge b_j = 0 \forall j \neq i$
Sender input. $y = \{y_1, \ldots, y_n\}$.

Setup. R generates a BFV keypair $(\mathsf{sk}, \mathsf{pk})$ and sends pk to S.

Protocol.

1. R computes $c_b \leftarrow \mathsf{Enc}(b)$ and sends c_b to S.
2. R and S run $\Pi_{ZK}^{WellFormed}$ on c_b.
3. S homomorphically computes $c'_b \leftarrow f(c_b, y)$ and sends c'_b to R
4. R computes $b' \leftarrow \mathsf{Dec}(c'_b)$ and outputs b'

Figure 5.2: 1-out-of-n OT protocol

see the answers, thus, there is no risk of leaking information to the verifier due to selective failure attacks. □

Theorem 3. *Let* Com *be a binding and hiding commitment scheme, and let SealPIR be the protocol described in [153], modified to not use any packing. Then the protocol described in Figure 5.2 implements* $\mathcal{F}_{OT}^{1:n}$ *with security against a malicious receiver and semi-honest sender in the* Com*-hybrid model.*

Proof. (Sketch) The security against semi-honest senders follows directly from the semantic security of the cryptosystem and the zero-knowledge property of $\Pi_{ZK}^{WellFormed}$.

As for the security against malicious receivers, the soundness of $\Pi_{ZK}^{WellFormed}$ ensures that b is a valid query. Given that b is a valid query (i.e. only one i is nonzero) the sender's response $c'_b = f(c_b, y) = \mathsf{Enc}\left(\sum_{j=1}^{n} b_j \cdot y_j\right) = \mathsf{Enc}(b_i \cdot y_i)$.

We construct a simulator $\mathcal{S}$ which interacts with R and the ideal functionality (shown in Figure 5.3). The indistinguishability of R's view when interacting with S in the real world versus R's view when interacting with $\mathcal{S}$ in the ideal world follows directly from the soundness of the $\Pi_{ZK}^{WellFormed}$ ZKPoPK subprotocol [156].

$\mathcal{S}_{\mathrm{OT}}^{1:n}$

1. R sends c_b to $\mathcal{S}$.
2. Perform $\Pi_{ZK}^{WellFormed}$ with R. Using the ZKPoPK extractor from the subprotocol in step 1 [156] of $\Pi_{ZK}^{WellFormed}$ extract b and learn R's secret index i.
3. Send i to $\mathcal{F}_{OT}^{1:n}$; receive y_i in return.
4. Compute $c'_b \leftarrow \mathsf{Enc}(b_i \cdot y_i)$ and send c'_b to R

Figure 5.3: Malicious-receiver simulator for $\Pi_{OT}^{1:n}$

□

5.3 Disjunctive proofs from 1:N OT

5.3.1 MPCitH Disjunctive Proof

Towards our goal of constructing an efficient disjunctive proof, we first define a helper function $\mathsf{ShareAt}(m, x, J, r)$ which pseudorandomly samples m-out-of-m additive secret shares of x from the seed r, outputting a vector of values $\{\mathrm{X}_1, \ldots \mathrm{X}_m\}$ such that $\bigoplus_{j=1}^{m} \mathrm{X}_j = x$ (we omit m as explicit input to $\mathsf{ShareAt}$ for the rest of our work). In addition, $\mathsf{ShareAt}$ has the property that for a fixed index $J \in \{1, \ldots m\}$ and r, $\mathsf{ShareAt}(m, x, J, r)$ and $\mathsf{ShareAt}(m, x', J, r)$ will have identical outputs at every index except the J-th index. This function is shown in Figure 5.4.

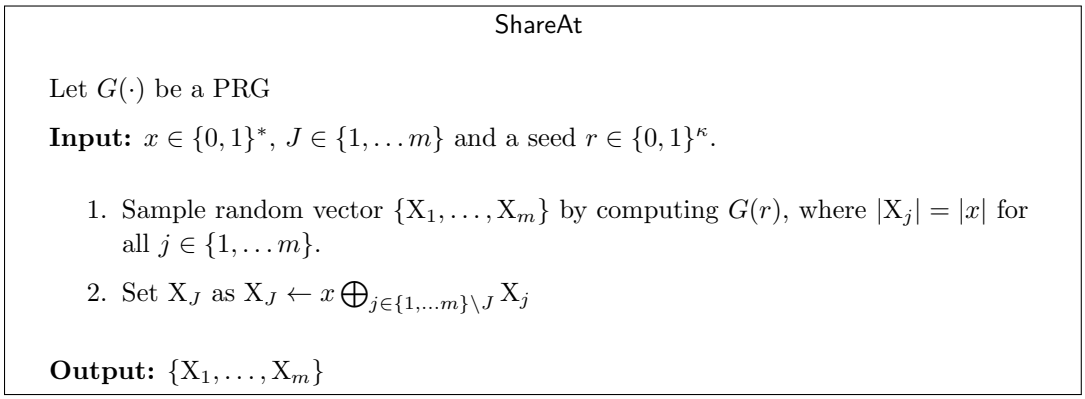


Figure 5.4: Secret sharing with specific offset index

We define our main protocol $\Pi_f^{\mathrm{MPCitH-OR}}$ in Figure 5.5. The common input is a function f, and a set of values $y_1, \ldots, y_n$. The Prover wishes to prove, in zero knowledge, that it knows input (x, y) such that $f(x, y) = 1 \wedge y \in \{y_1, \ldots, y_n\}$. We let $\Pi_{\mathcal{F}}$ denote an m-party protocol for securely computing $f(\bigoplus_j x_j, \bigoplus_j y_j)$. The verifier, $\mathcal{V}$, begins by generating and committing to a PRG seed s from which all further random values are sampled. This seed will allow the prover to confirm the verifier's honest behavior using the technique described in Section 5.2.

Next, the verifier $\mathcal{V}$ generates τ different sets of input encodings – each containing m

shares – for each of the y_i common input values. This results in a 3D table of shares $(n \times m \times \tau)$, denoted Y_{ij}^k, as shown in Figure 5.6. This 3D table is divided into n 2D "slices" of dimension $(m \times \tau)$, each corresponding to an input y_i. These slices are encodings of y_i such that all slices are identical in every position except for 1 per row (τ non-identical positions total). The non-identical position for the k-th row is denoted ε_k.

Then, prover $\mathcal{P}$ selects a "slice" of this 3D table through the 1-out-of-n OT protocol; the choice of the slice is its secret input, ℓ.

For each row of the slice (i.e. a chunk which was generated from the same value of k and adds up to y_ℓ) $\mathcal{P}$ generates additive shares of x and computes MPCitH views using the shares of y_ℓ from the slice and the newly-generated shares of x as input. Once all views are generated, $\mathcal{P}$ commits these views to $\mathcal{V}$.

Finally, $\mathcal{V}$ reveals the committed seed s to $\mathcal{P}$, who regenerates the whole 3D table, and verifies the honest behavior of the verifier. $\mathcal{P}$ aborts if there is a mismatch, otherwise $\mathcal{P}$ decommits the MPCitH views (except Party ε_k) to $\mathcal{V}$ who verifies the honest execution of the MPCitH protocol, and accepts the proof.

Note that since the slices are identical in each row except for at ε_k (i.e. the unrevealed MPCitH input) the privacy of the MPCitH protocol protects the prover from revealing its choice of index ℓ.

Theorem 4. *Let $m \geq 3$, let $f(\mathbf{Y}, \mathbf{X})$ be the function defined in Figure 5.5, let Com be a binding and hiding commitment scheme, let $\Pi_{\mathcal{F}}$ implement f with correctness and $(m-1)$-privacy, and let $\Pi_{\mathrm{OT}}^{1:n}$ implement $\mathcal{F}_{\mathrm{OT}}^{1:n}$ with security against a malicious receiver and a semi-honest sender. Then the protocol described in Figure 5.5 is a zero-knowledge proof protocol for the language $\{((y_1, \ldots, y_n), x) : \exists \ell \in \{1, \ldots, n\}$ such that $f(x, y_\ell) = 1\}$ in the $(\mathsf{Com}, \mathcal{F}_{\mathrm{OT}}^{1:n})$-hybrid model.*

Proof. Zero Knowledge: First we claim that the security of the protocol in the presence of a dishonest verifier reduces to the security of the protocol in the presence of a semi-honest verifier.

$\Pi_f^{\text{MPCitH}-\text{OR}}$

Setup. Let $f(x, y)$ be some function, and let $\mathcal{F}$ be an m-party functionality that takes input $(\mathrm{X}_j, \mathrm{Y}_j)$ from each party P_j and outputs $f(\oplus_{j=1}^m \mathrm{X}_j, \oplus_{j=1}^m \mathrm{Y}_j) \stackrel{?}{=} 1$ to all parties. Let $\Pi_{\mathcal{F}}$ be an m-party protocol that securely realizes $\mathcal{F}$ with correctness and $(m-1)$-privacy.

Common inputs. τ total number of repetitions, n values $\{y_1, \ldots, y_n\} \in \{0,1\}^\kappa$, and m, the number of parties involved in the MPC protocol, run in the head of the Prover, and $m^{-\tau} < 2^{-\lambda}$, λ is a security parameter.

Prover's input. $x \in \{0,1\}^*$ such that $f(x, y_\ell) = 1$ for some $\ell \in \{1, \ldots n\}$. [a]

1. Verifier $\mathcal{V}$ generates random seed s and sends $\mathsf{C}_s \leftarrow \mathsf{Com}(s)$ to the prover $\mathcal{P}$. Throughout the rest of the protocol, all randomness of the Verifier is generated by applying a PRG, $G(s)$. (We will, imprecisely, refer to these as "random" values.)
2. $\mathcal{V}$ uses the random seed s to sample the following values uniformly at random:
   (a) $\varepsilon_k \stackrel{\$}{\leftarrow} \{1, \ldots m\}$ for $k \in \{1, \ldots \tau\}$.
   (b) $r_k \stackrel{\$}{\leftarrow} \{0,1\}^\kappa$ for $k \in \{1, \ldots \tau\}$.
3. For $i \in \{1, \ldots n\}, k \in \{1, \ldots \tau\}$, $\mathcal{V}$ compute vector $\{\mathrm{Y}_{i,1}^{(k)}, \ldots, \mathrm{Y}_{i,m}^{(k)}\} \leftarrow \mathsf{ShareAt}(y_i, \varepsilon_k, r_k)$, which results in a 3D table, as in Figure 5.6. (Note that the same r_k is used for every y_i.)
4. For every $i \in \{1, \ldots n\}$, $\mathcal{V}$ sends the 2D table $\{\{\mathrm{Y}_{i,1}^{(1)}, \ldots \mathrm{Y}_{i,m}^{(1)}\}, \ldots \{\mathrm{Y}_{i,1}^{(\tau)}, \ldots \mathrm{Y}_{i,m}^{(\tau)}\}\} = \{\mathbf{Y}_i^{(1)}, \ldots \mathbf{Y}_i^{(\tau)}\}$
   to $\Pi_{\text{OT}}^{1:n}$.
5. $\mathcal{P}$ sends ℓ to $\Pi_{\text{OT}}^{1:n}$.
6. $\Pi_{\text{OT}}^{1:n}$ outputs $\{\mathbf{Y}_\ell^{(1)}, \ldots \mathbf{Y}_\ell^{(\tau)}\}$ to $\mathcal{P}$.
7. For every $k \in \{1, \ldots \tau\}$, $\mathcal{P}$:
   (a) Computes $\mathbf{X}^{(k)} = \{\mathrm{X}_1^{(k)}, \ldots, \mathrm{X}_m^{(k)}\}$ as a random additive share of x, i.e. $(x = \bigoplus_{j=1}^m \mathrm{X}_j^{(k)})$
   (b) Computes $\{view_{1,k}, \ldots, view_{m,k}\} \leftarrow \Pi_{\text{MPCitH}}(\Pi_{\mathcal{F}}, (\mathbf{X}^{(k)}, \mathbf{Y}_\ell^{(k)}))$.
8. For all $j \in \{1, \ldots m\}$, $k \in \{1, \ldots \tau\}$, $\mathcal{P}$ sends $\mathsf{C}_{view_{j,k}} \leftarrow \mathsf{Com}(view_{j,k})$ to $\mathcal{V}$
9. $\mathcal{V}$ de-commits s, which $\mathcal{P}$ uses to derive $\{\varepsilon_1, \ldots, \varepsilon_\tau\}$ and to reconstruct the 3D table as in above steps 2 - 3 [b].

[a] The input length $|x|$ can be either fixed or arbitrary.

[b] All messages sent by the receiver during $\Pi_{OT}^{1:n}$ are computed deterministically from the seed s and the common input $\{y_1, \ldots, y_n\}$.

Figure 5.5

10. $\mathcal{P}$ verifies the following properties hold for all $i, i' \in \{1, \ldots, n\}, k \in \{1, \ldots, \tau\}, j \in \{1, \ldots, m\} \backslash \epsilon_k$:

$$\bigoplus_{j=1}^{m} Y_{i,j}^{(k)} = y_i. \qquad\qquad Y_{i,j}^{(k)} = Y_{i',j}^{(k)}$$

If these properties do not hold, $\mathcal{P}$ aborts.

$\mathcal{P}$ decommits each $\{view_{j,k}\}_{k \in \{1,\ldots\tau\}, j \in \{1,\ldots m\} \backslash \varepsilon_k}$

11. $\mathcal{V}$ checks that the decommitted views are consistent with honest executions of $\Pi_{\mathcal{F}}$, include an output of 1, and that $view_{j,k}$ has a Y input equal to $Y_{1,j}^{(k)}$.

Figure 5.5: OR proof using MPC-in-the-Head

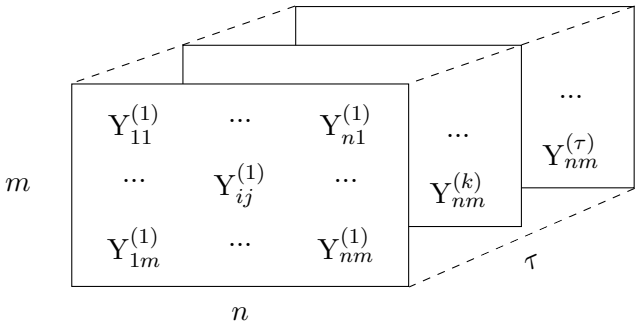


Figure 5.6: Notation for protocol $\Pi_f^{\text{MPCitH-OR}}$.

Consider the view of $\mathcal{V}$ after step 9. The messages $\mathcal{V}$ has received are encrypted queries of $\mathcal{P}$'s private input ℓ (step 1 of $\Pi_{OT}^{1:n}$), a transcript of a zero-knowledge proof (step 2 of $\Pi_{OT}^{1:n}$), and commitments to MPCitH views (step 8 of $\Pi_f^{\text{MPCitH}-\text{OR}}$).

Suppose $\mathcal{V}$ does not follow the protocol honestly. In the honest protocol, $\mathcal{V}$'s messages are generated deterministically based on s, the common input, and $\mathcal{P}$'s messages. Hence after $\mathcal{V}$ opens s in step 9, the prover can compare each message that $\mathcal{V}$ sent against the expected message in the honest verifier protocol. By our assumption one of these messages is inconsistent, hence $\mathcal{P}$'s next action is to abort the protocol.

$\mathcal{S}^{\mathsf{MPCitH}-\mathsf{OR}}_{\mathsf{MaliciousVerifier}}$

1. Emulating step 1 of $\Pi_{OT}^{1:n}$, $\mathcal{S}$ samples a random unit vector b and sends $\mathsf{Enc}(b)$ to $\mathcal{V}$.
2. Emulating step 2 of $\Pi_{OT}^{1:n}$, $\mathcal{S}$ runs the zero-knowledge simulator for $\Pi_{ZK}^{WellFormed}$ with $\mathcal{V}$.
3. Emulating step 8 of $\Pi_f^{\text{MPCitH}-\text{OR}}$, $\mathcal{S}$ samples $m \times \tau$ random strings of the same length as an MPCitH view and sends a commitment to each string to $\mathcal{V}$.
4. Emulating step 9 of $\Pi_f^{\text{MPCitH}-\text{OR}}$, $\mathcal{S}$ aborts the protocol after receiving $\mathcal{V}$'s decommitment.

Figure 5.7: Malicious-verifier simulator for $\Pi_f^{\mathsf{MPCitH}-\mathsf{OR}}$

We construct a simulator $\mathcal{S}$ for $\mathcal{V}$'s view after step 9 (shown in Figure 5.7) for the case where $\mathcal{V}$ deviates from the honest protocol and argue that the view when interacting with the simulator is indistinguishable from $\mathcal{V}$'s view when running $\Pi_f^{\text{MPCitH}-\text{OR}}$. The encrypted query's indistinguishability follows from the semantic security of the encryption scheme. The zero-knowledge transcript's indistinguishability follows from the zero-knowledge property of $\Pi_{ZK}^{WellFormed}$. The indistinguishability of the committed views follows from the hiding property of Com. Finally, as discussed above a verifier that deviates from the honest protocol will always cause the protocol to abort after step 9.

Next, we claim that, when given access to an ideal functionality for $\mathcal{F}_{\text{OT}}^{1:n}$, our protocol is zero-knowledge against a verifier that does not deviate from the semi-honest protocol. Since $\Pi_{\text{OT}}^{1:n}$ implements $\mathcal{F}_{\text{OT}}^{1:n}$ with semi-honest sender security and we are assuming the verifier

behaves semi-honestly, we may freely replace $\Pi_{\mathrm{OT}}^{1:n}$ in our protocol with $\mathcal{F}_{\mathrm{OT}}^{1:n}$. The simulator for $\Pi_f^{\mathrm{MPCitH-OR}}$ then proceeds as follows.

1. Accept the verifier's inputs to the OT functionality, recovering all input encodings, $\{\mathbf{Y}_i^{(1)}, \ldots \mathbf{Y}_i^{(\tau)}\}$ for all i, as well as $(\epsilon_1, \ldots, \epsilon_\tau)$. If any of these values are badly formed, the simulation sets an abort flag: $\mathsf{abort} = 1$. Otherwise, for each $k \in \{1, \ldots, \tau\}$, the simulator chooses random input shares, $\{X_1^{(k)}, \ldots, X_m^{(k)}\}$, then discards $X_{\epsilon_k}^{(k)}$ and $\mathrm{Y}_{i,\epsilon_k}^{(k)}$, for arbitrary i. (Note that, excluding $\mathrm{Y}_{i,\epsilon_k}^{(k)}$ and $\mathrm{Y}_{j,\epsilon_k}^{(k)}$, the remaining $n-1$ shares of y_i and y_j are identical).

2. Let $\mathcal{S}$ be the $(n-1)$-privacy simulator from Definition 6. The ZK simulator runs $\mathcal{S}$ using the remaining $n-1$ shares of x, and y_i. It commits to the resulting $n-1$ views and sends them to the verifier.

3. The simulator runs steps 9-11 honestly: it accepts the decommitment to the verifier's randomness, performs the described correctness checks, and decommits to the simulated views if everything passes, and $\mathsf{abort} \neq 1$.

By definition, the output of $\mathcal{S}$ is indistinguishable from the decommitted views of the prover in the hybrid-world proof. The reader can verify that the remainder of the simulation is perfect.

Soundness: Suppose the Prover does not know a valid input x_i for any y_i. By the hiding property of Com, the prover learns nothing about the verifier's state other than the queried row from $\Pi_{\mathrm{OT}}^{1:n}$ before it sends the commitments to its MPCitH views. And by the binding property of Com the prover's MPCitH views must be fixed before the prover receives the rest of the verifier's state. Since the proof is accepted or rejected based on these views, we can reduce the soundness of the protocol to the security properties of the MPCitH protocol.

We consider 3 cases, depending on how many simulated MPCitH parties perform malicious behavior (as measured by comparing inconsistencies in the parties' views).

In the first case, all MPCitH parties act honestly. By the completeness of $\Pi_{\mathcal{F}}$, either all parties output 0 with all but negligible probability (in which case the proof is rejected) or the input (x_i, y_i) is a valid witness for f. By our previous assumption, if $f(x_i, y_i) = 1$, then y_i is not a member of the common input set, hence the input encoding $Y_{i,j}$ must be different from the encodings created by the verifier in at least one position. If any opened view's input encoding of y_i does not match the verifier's encodings the proof will be rejected, thus in order to create valid proof, $\mathcal{P}$ must modify only the share of y_i that is not opened by $\mathcal{V}$. By the hiding property of Com, $\mathcal{P}$ has no information about ϵ_j, hence the prover has at best $\frac{1}{m}$ chance of guessing correctly for a single iteration. Amplified over τ iterations the cheating prover's probability of success is $\leq \frac{1}{m^\tau}$

The second case we consider is where exactly one simulated party performs malicious behavior. In this case, the malicious party's view must be inconsistent with at least one other view. Since $(n-1)$ views are opened, the verifier will see this inconsistency unless the malicious view, or the receiver of the malicious message, is the one left unopened. Therefore, the cheating prover's probability of correctly guessing ϵ_j for every iteration is $(\frac{2}{m})^\tau$. As an aside, it should be straightforward to see that some smaller success probability is achieved if the prover mixes-and-matches the strategies of the first and second cases between iterations.

The final case is that there are two or more malicious parties. In this case it is guaranteed that there will be an inconsistent pair of views in any $(n-1)$-subset, giving the cheating prover a success probability of 0 in this case.

□

5.4 Disjunctive Proofs for Mixed Statements

In the proof of assets problem, the Prover and Verifier hold as common input a list of hashed public keys, $L = \{y_1, \ldots y_n\}$, where $y_i = H(x_i)$, for some hash function H. The Prover wishes to prove that it knows secret key z such that (x, z) is a legitimate output of some cryptographic key generation algorithm, and, for some $y_i \in L$, $H(x) = y_i$.

More generally, we consider mixed statements of the form $f(x, y_i) = 1 \wedge g(x, z) = 1$, where $y_i \in L$, f is a Boolean (or "non-algebraic") function – in our application, a hash function – and $g(x, z)$ is an algebraic function – in our application, one verifying that z is the secret key corresponding to x.

Chase et al. [49] consider this question without the disjunction. That is, they assume $|L| = 1$, and focus solely on the challenge of constructing an efficient proof for mixed statements. They do this by leveraging the specific proof system for f, built from Garbled Circuits, in the following way. The prover begins by committing to input x with $\mathsf{Com}(x)$. The Verifier then prepares a garbled circuit for the Boolean circuit that outputs both $f(x)$, as well as a one-time MAC on the Prover's input: $t = ax+b$. The prover is allowed to decode t, and commits to this as well. Finally, the prover provides a proof, using an algebraic proof system, that it knows (x, t) such that $f(x, y) = 1$, x is consistent with $\mathsf{Com}(x)$, t is consistent with $\mathsf{Com}(t)$, and $t = ax + b$. In this way, the MAC on x that was derived while proving $f(x, y) = 1$ ensures that the same input x is used when proving that $g(x, z) = 1$.

Note that Chase et al. compute $t = ax + b$ inside a Boolean circuit, which requires $O(|a||x|)$ AND gates. We improve on their solution by using the oblivious transfer to avoid performing integer multiplication and addition in a Boolean circuit. As in the previous Section, we use MPC-in-the-head, rather than garbled circuits. Specifically, let x_i be the ith bit of x, and for each $u \in \{1, \ldots, |x|\}$, let $\mathbf{0_u}$ and $\mathbf{1_u}$ denote the input encodings generated by the verifier for that input bit. The verifier chooses a at random, and samples b by choosing random b_u for each $u \in \{1, \ldots, |x|\}$ and setting $b = \sum_u b_u$. When obliviously sending the encoding of the ith input bit, the verifier sends $(\mathbf{0_u}, b_u)$ and $(\mathbf{1_u}, 2^{u-1}a + b_u)$ to the OT functionality. By summing the 2nd value received in each of the $|x|$ received ordered pairs, the Prover recovers $ax + b$, and no computation in the circuit is required. We note that this leads to significant improvement over Chase et al. even when $|L| = 1$. In Figure 5.8, we present the full protocol, highlighting the changes in our disjunctive proof from Figure 5.5, in order to support mixed statements.

$\Pi_f^{\text{MPCitH}-\text{OR}-\text{Mix}}$

Setup. Let $f(x, y)$ be a Boolean function and $g(x, z)$ an algebraic function. Let $\mathcal{F}$ be an m-party functionality that takes input $(\mathrm{X}_j, \mathrm{Y}_j)$ from each party P_j and outputs $f(\oplus_{j=1}^m \mathrm{X}_j, \oplus_{j=1}^m \mathrm{Y}_j) \stackrel{?}{=} 1$ to all parties. Let $\Pi_{\mathcal{F}}$ be an m-party protocol that securely realizes $\mathcal{F}$ with correctness and $(m-1)$-privacy.
Common inputs. τ total number of repetitions, n values $\{y_1, \ldots, y_n\} \in \{0,1\}^\kappa$, and m, the number of parties in the MPC protocol run in the head of the Prover. m and τ are set such that $m^{-\tau} < 2^{-\lambda}$, where λ is a security parameter.
Prover's input. x, z, ℓ such that: $\ell \in \{1, \ldots n\}$, $f(x, y_\ell) = 1$ and $g(x, z) = 1$. We denote as $(x_1, \ldots x_{|x|})$ the bit representation of x (i.e. $x = \sum_{u=1}^{|x|} 2^{u-1} x_u$).
Verifier's input. $C_x = \mathsf{Com}(x)$, $C_z = \mathsf{Com}(z)$.

1. Verifier $\mathcal{V}$ generates its random tape s and sends $C_s \leftarrow \mathsf{Com}(s)$ to the prover $\mathcal{P}$. Throughout the rest of the protocol, all randomness of the Verifier is generated by applying a PRG, $G(s)$. (We will, imprecisely, refer to these as "random" values.).

2. $\mathcal{V}$ uses the random seed s to sample the following values uniformly at random:

   (a) $\varepsilon_k \stackrel{\$}{\leftarrow} \{1, \ldots m\}$ for $k \in \{1, \ldots \tau\}$.

   (b) $r_k \stackrel{\$}{\leftarrow} \{0,1\}^\kappa$ for $k \in \{1, \ldots \tau\}$.

   (c) $w_{u,k} \stackrel{\$}{\leftarrow} \{0,1\}^\kappa$ for $u \in \{1, \ldots, |x|\}, k \in \{1, \ldots \tau\}$

   (d) $a \stackrel{\$}{\leftarrow} \{0,1\}^\lambda, b_u \stackrel{\$}{\leftarrow} \{0,1\}^{|x|+\lambda}$ for $u \in \{1, \ldots, |x|\}$. Define $b := \sum_{u=1}^{|x|} b_u$

3. For $u \in \{1, \ldots |x|\}, k \in \{1, \ldots \tau\}$, $\mathcal{V}$ computes $\{\mathbf{0}_{u,1}^{(k)} \ldots \mathbf{0}_{u,m}^{(k)}\} \leftarrow \mathsf{ShareAt}(0, \varepsilon_k, w_{u,k})$ and $\{\mathbf{1}_{u,1}^{(k)} \ldots \mathbf{1}_{u,m}^{(k)}\} \leftarrow \mathsf{ShareAt}(1, \varepsilon_k, w_{u,k})$

4. For $i \in \{1, \ldots n\}, k \in \{1, \ldots \tau\}$, $\mathcal{V}$ computes vector $Y_i^{(k)} := \{\mathrm{Y}_{i,1}^{(k)}, \ldots \mathrm{Y}_{i,m}^{(k)}\} \leftarrow \mathsf{ShareAt}(y_i, \varepsilon_k, r_k)$

5. **Exchange labels for inputs.**

   For $i \in \{1, \ldots n\}$ denote 2D table $\mathbf{Y_i} := \{\mathrm{Y}_i^{(1)}, \ldots \mathrm{Y}_i^{(\tau)}\}$, and for $u \in \{1, \ldots, |x|\}$ denote 2D tables $\mathbf{0_u} := \{\{\mathbf{0}_{u,1}^{(1)}, \ldots \mathbf{0}_{u,m}^{(1)}\}, \ldots \{\mathbf{0}_{u,1}^{(\tau)}, \ldots \mathbf{0}_{u,m}^{(\tau)}\}\}$, $\mathbf{1_u} := \{\{\mathbf{1}_{u,1}^{(1)}, \ldots \mathbf{1}_{u,m}^{(1)}\}, \ldots \{\mathbf{1}_{u,1}^{(\tau)}, \ldots \mathbf{1}_{u,m}^{(\tau)}\}\}$

Figure 5.8

5. (a) $\mathcal{V}$ sends $\{\mathbf{Y_1}, \dots \mathbf{Y_n}\}$ to $\Pi_{\text{OT}}^{1:n}$.
   (b) $\mathcal{P}$ sends ℓ to $\Pi_{\text{OT}}^{1:n}$.
   (c) $\Pi_{\text{OT}}^{1:n}$ outputs $\mathbf{Y}_\ell$ to $\mathcal{P}$.
   (d) For every $u \in \{1, \dots |x|\}$
      i. $\mathcal{V}$ sends $\{(\mathbf{0_u}, b_u), (\mathbf{1_u}, 2^{u-1}a + b_u)\}$ to $\Pi_{\text{OT}}^{1:2}$.
      ii. For every $u \in \{1, \dots |x|\}$, $\mathcal{P}$ sends x_u to $\Pi_{\text{OT}}^{1:2}$.
      iii. If $x_u = 0$ then $\Pi_{\text{OT}}^{1:2}$ outputs $(\mathbf{0_u}, b_u)$ to $\mathcal{P}$, otherwise it outputs $(\mathbf{1_u}, 2^{u-1}a + b_u)$. $\mathcal{P}$ denotes whichever output it receives as $\{\{X_{u,1}^{(1)}, \dots X_{u,m}^{(1)}\}, \dots \{X_{u,1}^{(\tau)}, \dots X_{u,m}^{(\tau)}\}, M_u\}$
   (e) For $k \in \{1, \dots \tau\}$ denote the 2D table $\mathrm{X}^{(k)} := \{(X_{1,1}^{(k)} || \dots || X_{|x|,1}^{(k)}), \dots, (X_{1,m}^{(k)} || \dots || X_{|x|,m}^{(k)})\}$
6. For every $k \in \{1, \dots \tau\}$, $\mathcal{P}$ computes $(view_{1,k}, \dots, view_{m,k}) \leftarrow \Pi_{MPCitH}(\Pi_{\mathcal{F}}, (\mathbf{X}^{(k)}, \mathbf{Y}_\ell^{(k)}))$.
7. $\mathcal{P}$ computes $\text{MAC}(x) = \sum_{u=1}^{|x|}(M_u) = a \cdot x + b$ and $C_{MAC(x)} \leftarrow \mathsf{Com}(MAC(x))$.
8. For all $j \in \{1, \dots m\}$, $k \in \{1, \dots \tau\}$, $\mathcal{P}$ sends $C_{MAC(x)}$ and $C_{view_{j,k}} \leftarrow Com(view_{j,k})$ to $\mathcal{V}$.
9. $\mathcal{V}$ decommits s, which $\mathcal{P}$ uses to reconstruct $\{\varepsilon_1, \dots, \varepsilon_\tau\}$, $\{\mathbf{0_u}, \mathbf{1_u}\}$, $\{\mathbf{Y_1}, \dots \mathbf{Y_n}\}$
10. $\mathcal{P}$ verifies the following properties hold for all $i, i' \in \{1, \dots, n\}, u \in \{1, \dots, |x|\}, k \in \{1, \dots, \tau\}, j \in \{1, \dots, m\} \backslash \epsilon_k$:

$$\bigoplus_{j=1}^{m} Y_{i,j}^{(k)} = y_i. \qquad Y_{i,j}^{(k)} = Y_{i',j}^{(k)} \qquad \bigoplus_{j=1}^{m} \mathbf{0}_{u,j}^{(k)} = 0 \qquad \bigoplus_{j=1}^{m} \mathbf{1}_{u,j}^{(k)} = 1 \qquad \mathbf{0}_{u,j}^{(k)} = \mathbf{1}_{u,j}^{(k)}$$

11. $\mathcal{P}$ decommits each $\{view_{j,k}\}_{k \in \{1, \dots \tau\}, j \in \{1, \dots m\} \backslash \varepsilon_k}$
12. $\mathcal{V}$ checks that the decommitted views are consistent with honest executions of $\Pi_{\mathcal{F}}$, and, if so, outputs 1.
13. $\mathcal{P}$ and $\mathcal{V}$ execute the following ZK proof protocol: $\pi = \{(x, MAC(x), z) : C_x = \mathsf{Com}(x) \wedge C_{MAC(x)} = \mathsf{Com}(MAC(x)) \wedge MAC(x) = ax + b \wedge C_z = \mathsf{Com}(z) \wedge g(x, z) = 1\}(\mathsf{Com}(x), \mathsf{Com}(z), \mathsf{Com}(MAC(x)), a, b)$
14. If π verifies, $\mathcal{V}$ accepts, else $\mathcal{V}$ rejects.

Figure 5.8: Disjunctive protocol via MPCitH for mixed statements. We denote by colored text the additional elements introduced compared to Fig.5.5

Theorem 5. *Let $m \geq 3$, let $f(\mathbf{Y}, \mathbf{X})$ be the function defined in Figure 5.8, let* Com *be a binding and hiding commitment scheme, $MAC(x)$ an unforgeable one-time MAC and let $\Pi_{\mathcal{F}}$ implement f with correctness and $(m-1)$-privacy. Then the protocol described in Figure 5.8 is a zero-knowledge proof protocol for the language $\{((y_1, \ldots, y_n), z, x) : \exists \ell \in \{1, \ldots, n\}$ such that $f(x, y_\ell) = 1 \wedge g(x, z) = 1\}$ in the $(Com, \mathcal{F}_{COT}^{1:n})$-hybrid model.*

Proof (Sketch). Zero Knowledge: The simulator runs in the same fashion as in Theorem 4, and it inherits the same procedure for the ZK proof π.

Soundness: Our protocol inherits the soundness properties as in Theorem 4. Here the prover can also cheat by using an inconsistent witness x for functions f and g. However this is prevented from the unforegability property of the one-time MAC, the binding property of the commitment scheme and the soundness property of the ZK proof π. □

5.4.1 Proving the Value of Assets

When proving ownership of assets in a cryptocurrency such as Bitcoin, the exchange (i.e. the prover) needs to prove knowledge of a number of secret keys for the respective hashed public keys among those in the UTXO set. Additionally, they might wish to prove something about the values assigned to such keys, e.g. that their total value exceeds some minimum. For simplicity, we treat the UTXO set as a list of tuples, where each tuple $(H_i||v_i)$ represents a hashed[3] public key and value pair, where $H_i = H(\mathsf{pk}_i)$. Therefore, given a common input of a tuple list $L = \{(H_1||v_1), (H_2||v_2), ...(H_n||v_n)\}$, $\mathcal{P}$ must prove knowledge of secret keys $\{\mathsf{sk}_k\}_{k=1}^t$ corresponding to a set of public keys $S = \{\mathsf{pk}_k\}_{k=1}^t$ such that $\forall k \in \{1, \ldots, t\}$, $(\mathsf{sk}_k, \mathsf{pk}_k)$ is a valid output of the appropriate key generation algorithm, $(H(\mathsf{pk}_k), v_k) \in L$, and, $\Sigma_{k=1}^t v_i \geq v$.

The protocol as discussed above, is not sufficient for the Proof of Assets application as it does connect the provers' keys to their corresponding "coin" value stored on the blockchain. In Figure 5.9, we present an extension of our protocol that also supports values. The main

[3]We also treat the double hashing RIPEMD160(SHA256()) of public keys in Bitcoin as a single hash function H.

change is to provide an additional MAC on the input values, in order to bind them with their respective hashed keys. Also for simplicity, we do not provide a k-out-of-n OR proof but rather a 1-out-of-n OR proof (i.e. we only assume that the exchange only controls one address in the UTXO set), however the protocol can be naturally extended to accommodate multiple keys.

Interaction. Note that while gOTzilla is interactive, interaction is still acceptable for the Proof of Assets application. Recall that PoA is not typically executed on its own, but rather in parallel with a "Proof of Liabilities" (PoL) protocol [17, 25, 27] in order to *Prove Solvency*. PoA proves that an organization owns more than X assets, and PoL proves that an organization's total liabilities towards its clients are *less* than some value Y. For the organization to be solvent, its assets should exceed its liabilities $X > Y$. While PoA is executed between an organization and an auditor, PoL is executed between an organization and its clients. In PoL, the organization publishes a digest on its total liabilities, and each client needs to check the inclusion of their value (which they store with the organization) in that published digest. This check can only happen interactively which in turn makes the complete Proof of Solvency protocol interactive as well (for the Organization side). Therefore, our interactive protocol is still in-line with the requirements of this application, while it is the first one to provide an efficient way to prove assets even among many million hashed public key - value tuples.

5.5 Implementation

gOTzilla implementation is based on SealPIR [137] and MP-SPDZ [157][4] libraries. As discussed in Section 5.2, we use SealPIR for our needed OT functionality, and we provide more implementation details below.

[4]MP-SPDZ provides the ZK proof of plaintext knowledge needed for $\Pi_{ZK}^{WellFormed}$. The proof implemented in this library is for the BGV cryptosystem [158], whereas to be compatible with SealPIR we need a proof in the BFV cryptosystem [155]. However, a BFV-compatible version of this proof has been designed [156], and claims theoretically cheaper cost than the BGV version [159], but currently has no publicly available implementation. For this reason we believe the existing BGV implementation provides a good estimate of the cost.

Table 5.2: Evaluation for protocol in Fig 5.8. PIR preprocessing: NTT transform. Note: a) MPCitH encoding is needed by both the $\mathcal{P}$ and $\mathcal{V}$, therefore this column is counted twice in the total runtime b) PIR preprocessing is executed by both $\mathcal{P}$ and $\mathcal{V}$ in parallel.

Number of elements	MPCitH encoding (both $\mathcal{P}$ and $\mathcal{V}$)	PIR Preprocessing ($\mathcal{P}$ and $\mathcal{V}$)	PIR Query ($\mathcal{P}$)	PIR Reply ($\mathcal{V}$)	PIR Decode ($\mathcal{P}$)	$\Pi_{ZK}^{WellFormed}$ (polynomial interpolation) ($\mathcal{P}$)	$\Pi_{ZK}^{WellFormed}$ (bounded noise) ($\mathcal{P}$)	**Total Runtime**	**Runtime** of [138]
2^{13}	7ms	74ms	<1ms	199ms	1ms	1ms	163ms	453ms	30.23s
2^{16}	31ms	392ms	<1ms	880ms	2ms	1ms	151ms	1.64s	31.87s
2^{18}	86ms	1.3s	2ms	2.2s	3ms	1ms	152ms	3.83s	37.5s
2^{20}	358ms	6.47s	2ms	7.07s	5ms	1ms	163ms	14.89s	60s
2^{22}	1.61s	9.7s	3ms	9.6s	10ms	2ms	165ms	22.7s	150s
2^{24}	7.4s	38.59s	4ms	32.6s	10ms	3ms	167ms	86.09s	510s

5.5.1 Evaluation

Our first set of benchmarks for protocol $\Pi_f^{\text{MPCitH}-\text{OR}}$ is performed locally, with the prover and verifier running on the same host. We run our benchmarks on a z1d.metal AWS instance using 48 threads (24 physical cores) and 384 GB of RAM. We performed our evaluations for a range of disjunctive elements between 2^{16} and 2^{24}. As we require $m^\tau \simeq 2^{-40}$ soundness, we pick $m = 3$ MPCitH (minimum required) parties and $\tau = 25$ repetitions as this choice of parameters minimizes $(m-1)\cdot\tau$. Concretely, if we consider Limbo [160] as an efficient underlying MPCitH protocol, this implies about 50ms additional runtime cost on top of the rest of our protocol's runtime, which does not depend on n, and is dwarfed by the overall runtime costs of our protocol (therefore we do not take it into account). Assuming $|f(x)| = 256$ bits, the size of each slice of the 3D table is $256 \cdot m \cdot (\tau - 1)$ (we only need to send $\tau - 1$ shares as the remaining one can be inferred) which implies a 1600 byte size per element. As shown in Table 5.3, for $n = 2^{20}$ the total communication between the prover and verifier is 6.18 MB, plus 3.45 MB for communicating the PIR Galois keys beforehand. However, both the prover and verifier need to generate a version of the 3D table in their local memory (or storage) based on the seed s (steps 9 and 3 of the protocol $\Pi_f^{\text{MPCitH}-\text{OR}}$ respectively). In our current implementation, we store the entire table in RAM (requiring about 18GB memory for 2^{20} such elements), however this table can be offloaded

to disk and retrieved as needed at the cost of additional I/O operations, or alternatively, each slice of the cube can be generated on demand as needed.

Table 5.2 shows our local benchmarks in detail, where we provide a break down of the protocol's total runtime as follows:

1. Verifier's secret share phase (step 3 for ShareAt() of Fig. 5.5).

2. The 1-out-of-n OT phase (steps 4 - 6 of Fig. 5.5) which include:

    - Preprocessing, Query, Reply and Decode costs of SealPIR.
    - Polynomial interpolation (step. 3 of Fig. 5.1).
    - Proof of bounded noise (step. 1 of Fig. 5.1).

Note for brevity, we don't include the costs of PRG generation, commitment and decommitment costs, one-time MAC and ZK proof protocol as these are negligible compared to the overall runtime costs (which as shown in Table 5.2, are dominated by MPCitH encoding and PIR preprocessing and reply).

We also performed a second set of benchmarks over a network for $n = 2^{20}$, as shown on Table 5.4, where we take the additional network latency into account as well. In particular, the PIR Query - Reply would replace the 5th and 6th column together on Table 5.2, and similarly the rest of our network measurements. This shows that although our protocol has many rounds of interaction (most of them during $\Pi_{ZK}^{WellFormed}$), the overall impact to its total run-time is very small.

Comparison. We now make a concrete comparison of our protocol's runtime and communication costs with Mac'n'cheese [138], which also aims for disjunctive Zero-knowledge proofs and has similar asymptotic costs as shown in Fig. 5.1. We observe that Mac'n'cheese is not explicitly tailored for disjunctive proofs comprised of circuits with the same structure, however as discussed in Section 5.1, in such a case its disjunctive statement $C(x) = y_1 \vee C(x) = y_2 \vee \cdots \vee C(x) = y_n$ can be modified as $(C(x) = y) \wedge (y = y_1 \vee \cdots \vee y = y_n)$ to avoid many circuit evaluations, resulting in a conjunctive statement which one part consists

of a disjunctive statement of equality checks.

In addition, as discussed in Section 5.4, our protocol can be naturally extended to accommodate mixed statements, with only an overhead of $|x|$ 1-out-of-2 OTs, which only cost roughly 25ms in total for $|x| = 256$ [161], a negligible cost compared to the main protocol's benchmarks presented above.

Given those observations, we first compare with Mac'n'cheese's needed runtime for n equality checks plus proving $(C(x) = y)$ where C is a 250 million gate Boolean circuit, which is equivalent to converting from an arithmetic circuit representing the algebraic statement. Specifically when $n = 2^{20}$, the estimated reported cost of C for Mac'n'Cheese is 30 seconds, and the cost for 2^{20} equality checks (256 million gates), is another 30 seconds, running on a system with equivalent specifications[5], adding to a total runtime of 60 seconds, with a total communication cost of 63 MB. Given our measurements, we observe significant improvements even when comparing with the disjunctive equality checks part of Mac'n'cheese. In addition, Mac'n'cheese can handle only Boolean or arithmetic circuits, therefore as an example, a mixed statement in the form of SHA256$(g^x) = y$ would need around 250 million gates, while gOTzilla is compatible with techniques combining algebraic and non-algebraic statements similar to the work by Chase et al. [49] as discussed in Section 5.4, and therefore we don't need to convert between circuit types. Finally, in comparison to the concurrent work of [148], and based on their reported numbers our protocol is roughly 6x more efficient in computational costs (assuming the reported runtime t in Table 2 of [148] only takes the prover's *or* the verifier's costs into account, but not both simultaneously as we do), namely for 2^{13} elements our total runtime for 256 bits of statistical security (which implies a parameter $\tau = 80$) is 644ms while the total runtime for [148] in an equivalent system would be 3960ms. However, our protocol has higher communication costs, primarily because of the required proof of bounded Ring-LWE noise.

[5]These estimates were provided by Mac'n'cheese authors assuming cost per gate is 120ns.

$\Pi_f^{\text{MPCitH}-\text{OR}-\text{BTC}}$

Setup. $f(x,y)$ is a Boolean function and $g(x,z)$ is an algebraic function. $\Pi_{\mathcal{F}}$ is a semi-honest m-party protocol implementing the functionality $\mathcal{F}$ which takes as input $(\mathrm{X}_j, \mathrm{Y}_j)$ from each party P_j and outputs to all parties $f(\oplus_{j=1}^m \mathrm{X}_j, \oplus_{j=1}^m \mathrm{Y}_j) \stackrel{?}{=} 1$ with correctness and $(m-1)$-privacy..
Common inputs. τ total number of repetitions, n public key and value pairs $(y_1, v_1), \ldots, (y_n, v_n)$ and a minimum asset value v_0. $y_i \in \{0,1\}^\kappa$ and $m^{-\tau} < 2^{-\lambda}$, λ is a security parameter.
Prover's input. x, z, ℓ such that: $f(x, y_\ell) = 1$, $v_\ell \geq v_0$, and $g(x,z) = 1$. We denote as $(x_1, \ldots x_{|x|})$ the bit representation of x (i.e. $x = \sum_{u=1}^{|x|} 2^{u-1} x_u$).
Verifier's input. $C_x = \mathsf{Com}(x)$, $C_z = \mathsf{Com}(z)$, $C_v = \mathsf{Com}(v_\ell)$.

1. Verifier $\mathcal{V}$ generates its random tape s and sends $C_s \leftarrow Com(s)$ to the prover $\mathcal{P}$. Throughout the rest of theprotocol, all randomness of the Verifier is generated by applying a PRG,G(s). (We will, imprecisely, refer to these as "random" values.).
2. $\mathcal{V}$ uses the random seed s to sample the following values uniformly at random:
   (a) $\varepsilon_k \stackrel{\$}{\leftarrow} \{1, \ldots m\}$ for $k \in \{1, \ldots \tau\}$.
   (b) $r_k \stackrel{\$}{\leftarrow} \{0,1\}^\kappa$ for $k \in \{1, \ldots \tau\}$.
   (c) $w_{u,k} \stackrel{\$}{\leftarrow} \{0,1\}^\kappa$ for $u \in \{1, \ldots |x|\}, k \in \{1, \ldots \tau\}$
   (d) $a \stackrel{\$}{\leftarrow} \{0,1\}^\lambda, b_u \stackrel{\$}{\leftarrow} \{0,1\}^{|x|+\lambda}$ for $u \in \{1, \ldots, |x|\}$. Define $b := \sum_{u=1}^{|x|} b_u$
   (e) $c \stackrel{\$}{\leftarrow} \{0,1\}^\lambda, d \stackrel{\$}{\leftarrow} \{0,1\}^{|v|+\lambda}$
3. For $u \in \{1, \ldots |x|\}, k \in \{1, \ldots \tau\}$, $\mathcal{V}$ computes $\{\mathbf{0}_{u,1}^{(k)} \ldots \mathbf{0}_{u,m}^{(k)}\} \leftarrow \mathsf{ShareAt}(0, \varepsilon_k, w_{u,k})$ and $\{\mathbf{1}_{u,1}^{(k)} \ldots \mathbf{1}_{u,m}^{(k)}\} \leftarrow \mathsf{ShareAt}(1, \varepsilon_k, w_{u,k})$
4. For $i \in \{1, \ldots n\}, k \in \{1, \ldots \tau\}$, $\mathcal{V}$ computes vector $\{\mathrm{Y}_{i,1}^{(k)}, \ldots \mathrm{Y}_{i,m}^{(k)}\} \leftarrow \mathsf{ShareAt}(y_i, \varepsilon_k, r_k)$
5. **Exchange labels for inputs.**
   For $i \in \{1, \ldots n\}$ denote 2D table $\mathbf{Y_i} := \{\mathrm{Y}_i^{(1)}, \ldots \mathrm{Y}_i^{(\tau)}, cv_i + d\}$, and for $u \in \{1, \ldots, |x|\}$ denote 2D tables $\mathbf{0_u} := \{\{\mathbf{0}_{u,1}^{(1)}, \ldots \mathbf{0}_{u,m}^{(1)}\}, \ldots \{\mathbf{0}_{u,1}^{(\tau)}, \ldots \mathbf{0}_{u,m}^{(\tau)}\}\}$, $\mathbf{1_u} := \{\{\mathbf{1}_{u,1}^{(1)}, \ldots \mathbf{1}_{u,m}^{(1)}\}, \ldots \{\mathbf{1}_{u,1}^{(\tau)}, \ldots \mathbf{1}_{u,m}^{(\tau)}\}\}$
   (a) $\mathcal{V}$ sends $\{(\mathbf{Y_1}, c \cdot v_1 + d), \ldots, (\mathbf{Y_n}, c \cdot v_n + d)\}$ to $\Pi_{\mathrm{OT}}^{1:n}$.
   (b) $\mathcal{P}$ sends ℓ to $\Pi_{\mathrm{OT}}^{1:n}$.
   (c) $\Pi_{\mathrm{OT}}^{1:n}$ outputs $(\mathbf{Y}_\ell, c \cdot v_\ell + d)$ to $\mathcal{P}$.
   (d) For every $u \in \{1, \ldots, |x|\}$
      i. $\mathcal{V}$ sends $\{(\mathbf{0_u}, b_u), (\mathbf{1_u}, 2^{u-1}a + b_u)\}$ to $\Pi_{\mathrm{OT}}^{1:2}$.
      ii. $\mathcal{P}$ sends x_u to $\Pi_{\mathrm{OT}}^{1:2}$.
      iii. If $x_u = 0$ then $\Pi_{\mathrm{OT}}^{1:2}$ outputs $(\mathbf{0_u}, b_u)$ to $\mathcal{P}$, otherwise it outputs $(\mathbf{1_u}, 2^{u-1}a + b_u)$. $\mathcal{P}$ denotes whichever output it receives as $\{\{X_{u,1}^{(1)}, \ldots X_{u,m}^{(1)}\}, \ldots \{X_{u,1}^{(\tau)}, \ldots X_{u,m}^{(\tau)}\}, M_u\}$
   (e) For $k \in \{1, \ldots \tau\}$ denote the 2D table $\mathrm{X}^{(k)} := \{(X_{1,1}^{(k)} || \ldots || X_{|x|,1}^{(k)}), \ldots, (X_{1,m}^{(k)} || \ldots || X_{|x|,m}^{(k)})\}$

Figure 5.9

6. For every $k \in \{1, \dots \tau\}$, $\mathcal{P}$ computes $(view_{k,1}, \dots, view_{k,m}) \leftarrow \Pi_{MPCitH}(\Pi_{\mathcal{F}}, (\mathbf{X}^{(k)}, \mathbf{Y}_{\ell}^{(k)}))$.

7. $\mathcal{P}$ computes $\text{MAC}(x) = \sum_{u=1}^{|x|} M_u$ and $C_{MAC(x)} \leftarrow \mathsf{Com}(MAC(x))$. Similarly compute $\text{MAC}(v_\ell) = cv_\ell + d$ and $C_{MAC(v_\ell)} \leftarrow \mathsf{Com}(MAC(v_\ell))$.

8. For all $j \in \{1, \dots m\}$, $k \in \{1, \dots \tau\}$, $\mathcal{P}$ sends $C_{MAC(x)}$, $C_{MAC(v_\ell)}$, and $C_{view_{j,k}} \leftarrow \mathsf{Com}(view_{j,k})$ to $\mathcal{V}$.

9. $\mathcal{V}$ decommits s, which $\mathcal{P}$ uses to reconstruct $\{\varepsilon_1, \dots, \varepsilon_\tau\}$, $\{\mathbf{0_u}, \mathbf{1_u}\}$, and $\{\mathbf{Y_1}, \dots \mathbf{Y_n}\}$

10. $\mathcal{P}$ verifies the following properties hold for all $i, i' \in \{1, \dots, n\}, u \in \{1, \dots, |x|\}, k \in \{1, \dots, \tau\}, j \in \{1, \dots, m\} \backslash \epsilon_k$:

$$\bigoplus_{j=1}^{m} Y_{i,j}^{(k)} = y_i. \qquad Y_{i,j}^{(k)} = Y_{i',j}^{(k)} \qquad \bigoplus_{j=1}^{m} \mathbf{0}_{u,j}^{(k)} = 0 \qquad \bigoplus_{j=1}^{m} \mathbf{1}_{u,j}^{(k)} = 1 \qquad \mathbf{0}_{u,j}^{(k)} = \mathbf{1}_{u,j}^{(k)}$$

11. $\mathcal{P}$ decommits each $\{view_{j,k}\}_{k \in \{1, \dots \tau\}, j \in \{1, \dots m\} \backslash \varepsilon_k}$

12. $\mathcal{V}$ checks that the decommitted views are consistent with honest executions of $\Pi_{\mathcal{F}}$ output 1.

13. $\mathcal{P}$ and $\mathcal{V}$ execute the following ZK proof protocols:

    (a) $\pi_1 = \{(x, MAC(x), z) : C_x = \mathsf{Com}(x) \wedge C_{MAC(x)} = \mathsf{Com}(MAC(x)) \wedge MAC(x) = ax + b \wedge C_x = \mathsf{Com}(x) \wedge C_z = \mathsf{Com}(z) \wedge g(x, z) = 1\}(\mathsf{Com}(x), \mathsf{Com}(z), \mathsf{Com}(MAC(x)), a, b)$

    (b) $\pi_2 = \{(v_\ell, MAC(v_\ell)) : C_v = \mathsf{Com}(v_\ell) \wedge C_{MAC(v_\ell)} = \mathsf{Com}(MAC(v_\ell)) \wedge MAC(v_\ell) = cv_\ell + d \wedge v_\ell \geq v_0\}(C_v, C_{MAC(v_\ell)}, c, d, v_0)$

14. If any of π_1,π_2 do not verify, $\mathcal{V}$ rejects, else accepts.

Figure 5.9: Disjunctive Composite protocol via MPCitH for Proving Assets in Bitcoin. We denote by colored text the additional elements introduced compared to Fig.5.8

Table 5.3: Communication costs for Fig. 5.5 protocol (including Fig. 5.1 subroutine) for $n = 2^{20}$

PIR Query (Step 5)	64.14 KB
PIR Response (Step 4)	320.71 KB
$\Pi_{ZK}^{WellFormed}$ poly interp.	192.04 KB
$\Pi_{ZK}^{WellFormed}$ bounded noise	5.62 MB
Committed views (Step 8)	2.45 KB
Total	6.18 MB

Table 5.4: Latency costs for Fig. 5.5 protocol (including Fig. 5.1 subroutine) for $n = 2^{20}$

	PIR Query + Reply	$\Pi_{ZK}^{WellFormed}$ polynomial interpol.	$\Pi_{ZK}^{WellFormed}$ bounded noise	MPCitH views	Total
US East - East	7.405s	7ms	298ms	370ms	8.08s
US East - West	7.563s	126ms	892ms	381ms	8.962s
US East - Japan	7.738s	292ms	1.59s	421ms	10.04s

5.6 Disjunctive proofs using Garbled Circuits

As discussed in Section 5.4, Chase et al. [49] aims to provide ZK proof protocols for mixed statements in the form of $f(x, y) = 1 \wedge g(x, z) = 1$. Towards this goal, it constructs protocols in two different ways. The first (and more efficient) assumes the existence of a bit-wise commitment as part of a larger protocol, while the second requires a separate sub-circuit to compute a one-time MAC $t = a \cdot x + b$ which has O($|x||a|$) AND gates, where $|x|$ and $|a|$ is the bit length of x and a respectively. For instance, if $|x| = |a| = 512$, then $|x||a| = 262144$, which is about 10 times the size of a SHA256 circuit.

We observe that this one-time MAC value can be computed during a COT step using a similar process to how we encode the input (shown in $\Pi_{\text{MAC,f}}$ GC), where $\mathcal{F}_{\text{COT}}$ is equivalent to $\mathcal{F}_{\text{OT}}$ plus an opening phase to the receiver (refer to [49] for the ideal functionality). In this protocol, the prover and verifier compute $MAC(x) = a \cdot x + b$ as follows. The prover computes the bit decomposition of x: $x_1, \ldots, x_{|x|}$. The verifier creates additive shares of b: $b_1, \ldots, b_{|x|}$. For each bit x_u in x the two parties perform a 1-out-of-2 OT. If $x_u = 0$ the prover

Table 5.5: Comparison of ZK proof systems for Proof of Assets for a *single* proof. $|x|$ is length of input, $|F|$ circuit size, λ security parameter. (for BTC UTXO $|x| = 512$ (BTC public key 256bits + 256 bits of padding for MD), probably $\lambda < x$, we can consider 128 or 256 sec bits. Circuit F' might be 10 times larger than F)

	No setup	NI	Prover	Verifier	Proof size	Notes
Chase et al. [49] (w/o MAC)	✔	✕	$O(\|x\|$ pub + $\|F\|$ sym)	$O(\|x\|$ pub + ($\|F\|$ sym)	$O((\|F\|+\|x\|)\lambda)$	Σ-protocol + GC
Chase et al. [49] (w/ MAC)	✔	✕	$O(\lambda$ pub + ($\|F'\|$ + $\|x\|\lambda$) sym)	$O(\lambda$ pub + ($\|F'\|$ + $\|x\|\lambda$) sym)	$O((\|F'\|$ + $\|x\|\lambda)\lambda)$	Σ-protocol + GC
Backes et al. [135]	✔	✔	$O(\|x\| + \lambda$ pub + $\|F\|\lambda$ sym)	$O(\|x\| + \lambda$ pub + $\|F\|\lambda$ sym)	$O((\|F\|\lambda$ + $\|x\|)\lambda)$	Ped. Comm. + ZKBoo
Figure 5.10 with value	✔	✕	$O(\lambda$ pub + ($\|F\|$ sym)	$O(\lambda$ pub + ($\|F\|$) sym)	$O((\|F\|+\|x\|)\lambda)$	Σ-protocol + GC

receives $M_u \leftarrow b_u$, otherwise the prover receives $M_u \leftarrow (2^{u-1}a + b_u)$. From $M_1, \ldots, M_{|x|}$ the prover is able to compute $MAC(x)$ as $\sum_{u=1}^{|x|} M_u = \sum_{u=1}^{|x|} 2^{u-1}a \cdot x_u + b_u = a \cdot x + b = MAC(x)$.

However, even with the above optimization, the costs for a disjunctive proof remain linear in the size of the circuit. In Figure 5.10 we show an optimized OR Proof using Garbled Circuits with MAC, while in Figure 5.10 we extend the previous protocol with value.

5.7 Conclusion

We presented gOTzilla, a novel protocol for disjunctive Zero-Knowledge proofs, tailored for large disjunctions. While our protocol has equivalent asymptotic communication costs with recent works, we show that gOTzilla offers a concrete improvement over the state-of-the-art, especially when the disjunctions include mixed (i.e. algebraic and non-algebraic) statements, since our protocol is more "mixed statement-friendly". Finally, as the Bitcoin's UTXO count is roughly 80 million at the time of writing [162], gOTzilla can serve as a basis for a Proof of Assets over Bitcoin's blockchain, where an exchange can interactively prove its assets to an auditor in a few minutes.

$\Pi_f^{\text{GC}-\text{OR}-\text{Mix}}$

Setup. Group $\mathbb{G}$ where DDH assumption holds. $Com(\cdot)$ is a commitment scheme. Let $\mathcal{G} = (\text{Gb}, \text{En}, \text{De}, \text{Eval}, \text{Ve})$ be a garbling scheme.

Commom input. $y_1, \cdots, y_n$ where $y_i \in \{0,1\}^\kappa$.

Prover's input. $x \in \mathbb{G}$, where x is the witness to the statement y_ℓ.

Verifier's input. $\mathsf{C}_x = \mathsf{Com}(x)$.

Protocol.

1. The verifier constructs a garbled circuit for F.

$$(GC, e, d) \leftarrow \text{Gb}(1^\kappa, F(x, y) = y \oplus f(x)$$

2. The prover sends (i, x_i) for all $i \in [n]$ to $\mathcal{F}_{\text{COT}}$.
3. The verifier sends $(i, (K_i^0, b_i), (K_i^1, 2^i a + b_i))$ for all $i \in [n]$ to $\mathcal{F}_{\text{COT}}$ where a and b_i has length of λ and $|x| + \lambda$ bits respectively.
4. $\mathcal{F}_{\text{COT}}$ outputs $(K_i^{x_i}, 2^i a x_i + b_i)$ for all $i \in [n]$ to the prover.
5. The prover sends ℓ to $\mathcal{F}_{\text{COT}}^{1:n}$.
6. The verifier sends $(y_1, \cdots, y_n)$ to $\mathcal{F}_{\text{COT}}^{1:n}$.
7. $\mathcal{F}_{\text{COT}}^{1:n}$ outputs y_ℓ to the prover.
8. The verifier sends the garbled circuit GC to the prover.
9. The prover evaluates the garbled circuit

$$Z \leftarrow \text{Eval}(GC, \{K_i^{x_i}\}_{i \in [n]}, y_\ell)$$

10. The prover computes $t = \sum_{i=0}^{n-1} (2^i a x_i + b_i) = ax + b$.
11. The prover commits to the garbled output Z and t by sending $\mathsf{Com}(Z), \mathsf{Com}(t)$ to the verifier and proves knowledge of opening.
12. The verifier sends open to $\mathcal{F}_{\text{COT}}$ and $\mathcal{F}_{\text{COT}}^{1:n}$.
13. $\mathcal{F}_{\text{COT}}$ sends (K_i^0, K_i^1) and $\mathcal{F}_{\text{COT}}^{1:n}$ sends $(y_1, \cdots, y_n)$ to the prover for all $i \in [n]$.
14. The prover verifies that the circuit was garbled correctly by running $\text{Ve}(GC, \{K_i^0, K_i^1\}_{i \in [n]}, F)$ and the garbled inputs for $x, y_1, \cdots, y_n$ are correct. If the check fails, the prover terminates. Else, it opens Z to the verifier.
15. The verifier checks that $\text{De}(d, Z) = 0$. Otherwise, it rejects and terminates.
16. The prover and the verifier execute a ZK proof to prove the following. $\pi = \{(x, t) : \mathsf{C}_x = \mathsf{Com}(x) \wedge \mathsf{C}_t = \mathsf{Com}(t) \wedge t = ax + b\}(\mathsf{C}_x, \mathsf{C}_t)$
17. If π does not verify, the verifier terminates.

Figure 5.10: OR Proof using Garbled Circuits with MAC. We denote by colored text the additional elements introduced compared to the protocol of Chase et al.

$\Pi_f^{\text{GC-OR-Mix-v}}$

Setup. Group $\mathbb{G}$ where DDH assumption holds. $\mathsf{Com}(\cdot)$ is a commitment scheme. Let $\mathcal{G} = (\text{Gb}, \text{En}, \text{De}, \text{Eval}, \text{Ve})$ be a garbling scheme.

Commom input. $(y_1, v_1), \cdots, (y_n, v_n)$ where $y_i \in \{0,1\}^\kappa, v_i \in [0, L]$.

Prover's input. $x \in \mathbb{G}$, where x is the witness to statement y_ℓ such that $v_\ell \geq v_0$.

Verifier's input. $\mathsf{C}_x = \mathsf{Com}(x)$ and $\mathsf{C}_{v_\ell} = \mathsf{Com}(v_\ell)$.

Protocol.

1. The verifier constructs a garbled circuit for F.
$$(GC, e, d) \leftarrow \text{Gb}(1^\kappa, F(x, y) = y \oplus f(x)$$
2. The prover sends (i, x_i) for all $i \in [n]$ to $\mathcal{F}_{\text{COT}}$.
3. The verifier sends $(i, (K_i^0, b_i), (K_i^1, 2^i a + b_i))$ for all $i \in [n]$ to $\mathcal{F}_{\text{COT}}$ where a and b_i has length of λ and $|x| + \lambda$ bits respectively.
4. $\mathcal{F}_{\text{COT}}$ outputs $(K_i^{x_i}, 2^i a x_i + b_i)$ for all $i \in [n]$ to the prover.
5. The prover sends ℓ to $\mathcal{F}_{\text{OT}}^{1:n}$.
6. The verifier sends $((y_1, a'v_1 + b'), \cdots, (y_n, a'v_n + b'))$ to $\mathcal{F}_{\text{OT}}^{1:n}$ where a' and b' has length of λ and $|x| + \lambda$ bits respectively.
7. $\mathcal{F}_{\text{COT}}^{1:n}$ outputs $(y_\ell, a'v_\ell + b')$ to the prover.
8. The verifier sends the garbled circuit GC to the prover.
9. The prover evaluates the garbled circuit
$$Z \leftarrow \text{Eval}(GC, \{K_i^{x_i}\}_{i \in [n]}, y_\ell)$$
10. The prover computes $t = \sum_{i=0}^{n-1} (2^i a x_i + b_i) = ax + b$ and $v = a'v_\ell + b'$.
11. The prover commits to the garbled output Z and t by sending $\mathsf{Com}(Z), \mathsf{Com}(t)$, $\mathsf{Com}(v)$ to the verifier and proves knowledge of opening.
12. The verifier sends open to $\mathcal{F}_{\text{COT}}$ and $\mathcal{F}_{\text{COT}}^{1:n}$.
13. $\mathcal{F}_{\text{COT}}$ sends (K_i^0, K_i^1) and $\mathcal{F}_{\text{COT}}^{1:n}$ sends $((y_1, a'v_1 + b'), \cdots, (y_n, a'v_n + b'))$ to the prover for all $i \in [n]$.

14. The prover verifies that the circuit was garbled correctly by running $\mathrm{Ve}(GC, \{K_i^0, K_i^1\}_{i\in[n]}, F)$ and the garbled inputs for $x, y_1, \cdots, y_n$ are correct. If the check fails, the prover terminates. Else, it opens Z to the verifier.

15. The verifier checks that $\mathrm{De}(d, Z) = 0$. Otherwise, it rejects and terminates.

16. The prover and the verifier execute the following ZK proofs: $\pi_1 = \{(x,t) : \mathsf{C}_x = \mathsf{Com}(x) \wedge \mathsf{C}_t = \mathsf{Com}(t) \wedge t = ax + b\}$ $\pi_2 = \{(v_\ell, v) : \mathsf{C}_{v_\ell} = \mathsf{Com}(v_\ell) \wedge \mathsf{C}_v = \mathsf{Com}(v) \wedge v = a'v_\ell + b' \wedge v_\ell \geq v_0\}$

17. If any of π_1, π_2 do not verify, the verifier terminates.

Figure 5.10: OR Proof with MAC and value Protocol using Garbled Circuits. We denote by colored text the additional elements introduced compared to Fig. 5.10.

Chapter 6: Efficient signatures for auditing IoT devices

6.1 Introduction

Blockchain applications are not restricted into cryptocurrency and payment systems. In fact, the use of distributed, immutable ledgers has been proposed as a prominent solution in the IoT setting, allowing rapid detection of inconsistencies in sensory data and network communications, providing a conflict resolution mechanism without relying on a trusted authority [163]. A number of relevant schemes has been proposed in the literature [164, 165] (later discussed in Section 6.6), which propose various ways to integrate distributed ledgers with IoT.

In addition, the commercial success of low Size Weight and Power (SWaP) sensors and IoT devices has given rise to new sensor-centric applications transcending traditional industrial and closed-loop systems [166, 167]. In their most recent Annual Internet Report [168], CISCO estimates that there will be 30 billion networked devices by 2023, which is more than three times the global population. While very different in terms of their hardware and software implementations, Industrial IoT (IIoT) systems share common functional requirements: they are designed to collect data from a large number of low-SWaP sensor nodes deployed at the edge. These nodes, which we refer to as edge *sensors*, are resource-constrained devices used in volume to achieve a broader sensing coverage while maintaining low cost. Thus, while capable of performing simple operations, low-SWaP sensors usually depend on battery power, are equipped with limited storage, and have low processing speed [169].

In practice, edge sensors are usually controlled by and report to more powerful gateway devices (which we refer to as *aggregators*) that process and aggregate the raw sensory data. For instance, in an Industrial (IIoT) environment, sensors are devices such as temperature

sensors are broadcasting their measurements to the network router, which in turn submits it to the cloud through the Internet. Until recently, due to processing and storage constraints, many IoT designs were geared towards cloud aggregation and data processing. However, latency, bandwidth, autonomy and data privacy requirements for IoT applications keep pushing the aggregation and processing of data towards the edge [170]. In addition, in most use cases, IoT devices need to be mutually *authenticated* to maintain system integrity and the data origin has to be verified to prevent data pollution attacks [171,172] and in "model poisoning" where an attacker has compromised a number of nodes acting cooperatively, aiming to reduce the accuracy or even inject backdoors to the resulting analysis models [173, 174].

The Challenge: One of the main roadblocks for using Blockchain-based systems as "decentralized" databases for sharing and storing collected data is their dependency on asymmetric authentication techniques. Typically, to produce authenticated data packets, sensors have to digitally sign the data by performing public key cryptographic operations, which are associated with expensive sign and verification computations and large bandwidth requirements. Although some high-end consumer sensor gateways and integrated sensors might be capable of performing cryptographic operations, a large number of edge sensors have limited computational power, storage and energy [175, 176]. To make matters worse, sensors try to optimize their power consumption by entering a "sleep" state to save power resulting in intermittent network connectivity and lack of synchronicity. Given such tight constraints, an important challenge is allowing low-SWaP devices being extremely constrained both in terms of computational power and memory (categorized as Class 0 in RFC 7228 [177] ref. Section 6.5.1), to authenticate and utilize a blockchain-based data sharing infrastructure.

Our Contributions: We design and implement BBox-IoT, a complete blockchain-based system for Industrial IoT devices aimed to create a decentralized, immutable ledger of sensing data and operations while addressing the sensor and data authentication challenge for extremely constrained devices. We aim to use our system as a "black-box" that empowers operators of an IIoT enclave to audit sensing data and operational information such as IIoT

communications across all IIoT devices.

To perform sensor and data authentication operations *without* relying on heavy cryptographic primitives, we introduce a novel hash-based digital signature that uses an onetime hash chain of signing keys. While our design is inspired by TESLA broadcast authentication protocol [178,179], our approach *does not* require any timing and synchronicity assumptions between signer and verifier. Overcoming the synchronicity requirement is critical for low-SWaP devices since their internal clocks often drift out of synchronization (especially those using low cost computing parts) [180, 181]. Our signature construction is proven secure assuming a preimage resistant hash function. Also, we achieved logarithmic storage and computational (signature/verification) costs using optimizations [182, 183]. Our proposed scheme further benefits by the broadcast nature of the wireless communication. Indeed, in combination with the immutable blockchain ledger, we are able to ferret out man-in-the-middle attacks in all scenarios where we have more than one aggregators in the vicinity of the sensors. To bootstrap the authentication of sensor keys, we assume an operator-initiated device bootstrap protocol that can include either physical contact or wireless pairing using an operator-verified ephemeral code between sensors and their receiving aggregators. Our bootstrap assumptions are natural in the IoT setting, where sensors often "report" to specific aggregators and allows us to overcome the requirement for a centralized PKI. Note that our signature scheme is of independent interest, in-line with recent efforts by NIST for lightweight cryptography [184].

For the blockchain's consensus protocol, we consider a *permissioned* setting, where a trusted party authorizes system participation at the aggregator level. Our system supports two main types of IoT devices: low-SWaP sensors who just broadcast data and self-reliant aggregators who collect the data and serve as gateways between sensors and the blockchain. While our system is initialized by a trusted operator, the operator is not always assumed present for data sharing and is only required for high-level administrative operations including adding or removing sensors from the enclave. We build the consensus algorithms for BBox-IoT using a modified version of Hyperledger Fabric [185], a well known permissioned

blockchain framework, and leverage blockchain properties for constructing our protocols tailored for constrained-device authentication. However, BBox-IoT operations are designed to be lightweight and do not use public key cryptography based on the RSA or discrete logarithm assumptions, which are common, basic building blocks of popular blockchain implementations. We describe our system in detail considering interactions between all participants and argue about its security.

We implemented and tested a BBox-IoT prototype in an IIoT setting consisting of extremely constrained sensors (Class 0 per RFC 7228). We employed 8-bit sensor nodes with 16MHz micro controllers and 2KB RAM, broadcast data every 10 seconds to a subset of aggregators (e.g. IIoT gateways) which in turn submit aggregated data to a cloud infrastructure. The evaluation shows that the IIoT sensors can compute our 64-byte signature in 50ms, making our signature scheme practical even for the least capable of IIoT platforms. Our evaluation section shows results by considering a sensor/gateway ratio of 10:1. When compared with ECDSA signing operations, our scheme is significantly more efficient offering two (2) and three (3) orders of magnitude speedup for signing and verification respectively. Our theoretical analysis and implementation shows that we can achieve strong chained signatures with half signature size, which permits accommodating more operations in the same blockchain environment. BBox-IoT is also over 50 times more energy-efficient, which makes our system ideal for edge cost-efficient but energy-constrained IIoT devices and applications.

Finally, we adopt the same evaluation for Hyperledger Fabric considered in previous work [185] and estimate the end-to-end costs of BBox-IoT when running on top of our Hyperledger modification, showing it is deployable in our considered use-cases.

The work presented in this chapter has been published in [31].

6.2 Background

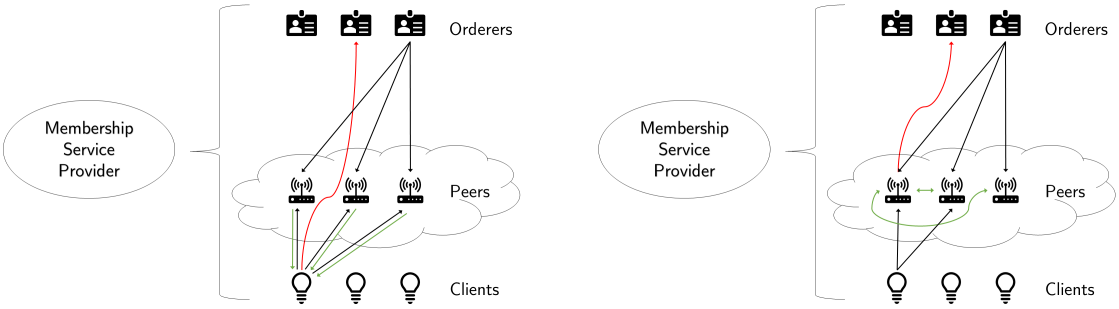
BBox-IoT Permissioned Consensus: The consensus properties discussed in Section 2.2 while necessary, are not sufficient for our system consensus. For instance, most "classical" consensus algorithms such as PBFT [55] have not been widely deployed due to various practical issues including lack of scalability. With BBox-IoT requirements in mind, the system's consensus algorithm needs to satisfy the following *additional* properties:

i. Dynamic membership: In BBox-IoT, there is no a priori knowledge of system participants. New members might want to join (or leave) after bootstrapping the system. We highlight that the vast majority of permissioned consensus protocols assume a static membership [186]. Decoupling "transaction signing participants" from "consensus participants" is a paradigm that circumvents this limitation [187].

ii. Scalable: BBox-IoT might be deployed in wide-area scenarios (e.g. IIoT), so the whole system must support in practice many thousands of participants, and process many operations per second (more than 1000 op/s) [56].

iii. DoS resistant: For the same reason above, participants involved in consensus should be resilient to denial-of-service attacks [58].

Hyperledger. One of the most promising examples of permissioned blockchains is the Hyperledger project, established by the Linux Foundation and actively supported by companies such as IBM and Intel [188]. The Hyperledger project aims to satisfy a wide range of business blockchain requirements, and has developed several frameworks with different implementation approaches, such as Hyperledger Fabric, Indy, Iroha and Sawtooth. Each framework uses a different consensus algorithm, or even supports "pluggable" (rather than hardcoded) consensus like Hyperledger Fabric [185]. To make pluggable consensus possible, Hyperledger Fabric introduces the "execute-order-validate" paradigm in its architecture, instead of the traditional "order-execute" [185]. In this paradigm, the "maintainer"



(a) Original architecture. Clients collect signatures from peers for a transaction, then submit the signed transaction to the ordering service which then returns a block containing packaged transactions to peers.

(b) Modified architecture. Clients only broadcast the transaction to the peers, who are then responsible themselves for signing it before submitting it to the ordering service.

Figure 6.1: Modified Hyperledger Fabric architecture.

participants are decoupled from the "consensus" participants (called *Peers* and *Orderers* respectively as we will see below), which eventually leads to satisfying dynamic membership and scalability.

In the following we focus on Hyperledger Fabric which seems to fit best in our system and provide a high-level description of its architecture (shown in Figure 6.1a). Its main components are categorized as follows:

1. **Clients** are responsible for creating a transaction and submitting it to the peers for signing. After collecting a sufficient number of signatures (as defined by the system policy), they submit their transaction to the orderers for including it in a block. Client authentication is delegated to the application.

2. **Peers** are the blockchain maintainers, and are also responsible for endorsing clients' transactions. Notice that in the context of Hyperledger, "Endorsing" corresponds to the process of applying message authentication.

3. **Orderers** collectively form the "ordering service" for the system. After receiving signed transactions from the clients, the service establishes *consensus* on total order of a collected transaction set. Then the ordering service by delivering blocks to the

peers, and ensures the consistency and liveness properties of the system.

4. The **Membership Service Provider** (MSP) is responsible for granting participation privileges in the system.

In its initial version v0.6, Hyperledger Fabric used the Byzantine fault-tolerant PBFT consensus algorithm [189], which only supports static membership for the ordering service participants. It's current version (v2.2) offers the *Raft* [190] consensus algorithm, which provides crash fault tolerance but not Byzantine fault tolerance, thus preventing the system from reaching consensus in the case of malicious nodes. Hyperledger Fabric could potentially use BFT-SMART in the future [191, 192].

Regarding scalability, although the original version of Hyperledger Fabric has several potential bottlenecks in its architecture, proposals exist to improve its overall performance in terms of operations per second [193]. These proposals suggest storing blocks in a distributed peer cluster for further scalability improvements. Also while operations (or transactions) per second is one scalability aspect in our setting, the other one is the actual number of peers the system can practically support without heavily impacting its performance. To the best of our knowledge, this has only been experimentally shown for up to 100 peers [185]. This experiment indicates however a low impact of the number of peers, especially if the network latency is low. Our setting allows such an assumption, since our use-case scenarios are all instantiated in a specific geographical location, as later discussed in section 6.5.

Modifying Hyperledger Fabric. While Hyperledger Fabric's *execute-order-validate* architecture offers several advantages discussed previously, we cannot directly use it in our BBox-IoT system, since we assume that lightweight devices (which for Hyperledger Fabric would have the role of "clients") are limited to only broadcasting data without being capable of receiving and processing. In Hyperledger Fabric, clients need to collect signed transactions and send them to the ordering service, which is an operation that lightweight devices are typically not capable of performing.

To address this issue, we propose a modification in Hyperledger architecture. In our

modified version, as shown in Figure 6.1b, a client broadcasts its "transaction" message to all nearby peer nodes. However, the transaction is handled by a *specific* peer (which is equivalent to an aggregator as we discuss in the next section), while peers not "responsible" for that transaction disregard it. That specific peer then assumes the role of the "client" in the original Hyperledger architecture simultaneously, while also continuing functioning as a peer node. As a client, it would be responsible for forwarding this transaction to other peers, and collecting the respective signatures, as dictated by the specified system policy, in a similar fashion to original Hyperledger Fabric. It would then forward the signed transaction to the ordering service, and wait for it to be included in a block. The ordering service would send the newly constructed block to all peers, which would then append it to the blockchain.

Security of our modifications. The proposed modifications of Hyperledger do not affect the established security properties (i.e. Consistency and Livenessas we define them in Section 2.2 and interpreted in [185]), since a peer node simultaneously acting as a client, can only affect the signing process by including a self-signature in addition to other peers' signatures. However, because the signing requirements are dynamically dictated by the system policy, these could be easily changed to require additional signatures or even disallow self-signatures to prevent any degradation in security. We also note that while this modified version of Hyperledger effectively becomes agnostic to the original client, which otherwise has no guarantees that its broadcasted transaction will be processed honestly, our threat model discussed in the next section captures the above trust model.

6.3 BBox-IoT system properties

In BBox-IoT there are five main types of participants, most of them inherited by Hyperledger Fabric: the MSP, orderers, local administrators, aggregators and sensors. Aggregators are equivalent to *peers* and sensors to *clients* in our modified Hyperledger Fabric architecture discussed in the previous section. We provide a high level description of each participant's role in the system and include detailed definitions in Section 6.10 .

- The MSP is a trusted entity who grants or revokes authorization for orderers, local administrators and aggregators to participate in the system, based on their credentials. It also initializes the blockchain and the system parameters and manages the system configuration and policy.

- Orderers (denoted by O) receive signed transactions from aggregators. After verifying the transactions as dictated by the system policy they package them into blocks. An orderer who has formed a block invokes the consensus algorithm which runs among the set of orderers $\mathcal{O}$. On successful completion, it is transmitted back to the aggregators with the appropriate signatures.

- Local administrators (denoted by LAdm, are lower-level system managers with delegated authority from the MSP. Each LAdm is responsible for creating and managing a local device group G, which includes one or more aggregators and sensors. He grants authorization for aggregators to participate in the system with the permission of the MSP. He is also solely responsible for granting or revoking authorization for sensors in his group, using aggregators to store their credentials.

- Aggregators (denoted by Ag) are the blockchain maintainers. They receive blocks from orderers and each of them keeps a copy of the blockchain. They store the credentials of sensors belonging in their group and they pick up data broadcasted by sensors. Then they create blockchain "transactions" based on their data (after possible aggregation), and periodically collect signatures for these transactions from other aggregators in the system, as dictated by the system policy. Finally, they send signed transactions to the ordering service, and listen for new blocks to be added to the blockchain from the orderers.

- Sensors (denoted by S) are resource-constrained devices. They periodically broadcast signed data blindly without waiting for any acknowledgment. They interact with local administrators during their initialization, while their broadcasted data can potentially be received and authenticated by multiple aggregators.

We then define the security and operational properties of BBox-IoT, in accordance with evaluation principles adopted in [165, 194–196].

6.3.1 Threat model & Assumptions

Physical layer attacks and assumptions. While our system cannot prevent physical tampering with sensors that might affect data correctness, any data discrepancies can be quickly detected through comparisons with adjacent sensors given the blockchain immutability guarantees [197]. Similarly, any malicious or erroneous data manipulation by an aggregator will result in detectable discrepancies even when one of the aggregators is not compromised simultaneously. Of course, if all aggregators become compromised instantaneously, which is hard in a practical setting, our system will not detect any discrepancies. This raises the bar significantly for an adversary who might not be aware or even gain access to all aggregator nodes at the same time. Finally, attacks such as flooding/jamming and broadcast interception attacks are out of scope in this thesis.

Trust Assumptions. We assume that MSP is honest during system bootstrapping only, and that device group participants (Local administrators, aggregators and sensors) may behave unreliably and deviate from protocols. For instance, they might attempt to statically or dynamically interfere with operations of honest system participants (e.g. intercept/inject own messages in the respective protocols), even colluding with each other to do so. This behavior is expected which our system is designed to detect and thwart.

Consensus Assumptions. As in Hyperledger, we decouple the security properties of our system from the consensus ones. For reference, this implies tolerance for up to 1/3 Byzantine orderer nodes, with a consensus algorithm satisfying at least the fundamental and additionally required properties discussed in Section 6.2.

Given the above adversarial setting, we define the following security properties[1]:

S-1 Only authenticated participants can participate in the system. Specifically:

[1]We do not consider data confidentiality in our system, however as discussed later our model could be extended to satisfy confidentiality as well.

a. An orderer non-authenticated by the MSP is not able to construct blocks (i.e., successfully participate in the consensus protocol). The ordering service can tolerate up to f malicious (byzantine) orderers.

b. An LAdm non-authenticated by the MSP is not able to form a device group G.

c. If an aggregator is not authenticated by the MSP, then its signatures on transactions cannot be accepted or signed by other aggregators.

S-2 *Sensor health:* Sensors are resilient in the following types of attacks:

a. Cloning attacks: A non-authenticated sensor cannot impersonate an existing sensor and perform operations that will be accepted by aggregators.

b. Message injection - MITM attack: A malicious adversary cannot inject or modify data broadcasted by sensors.

S-3 *Device group safety:* Authenticated participants in one group cannot tamper with other groups in any way, i.e.:

a. An LAdm cannot manage another group, i.e. add or revoke participation of an aggregator or sensor in another device group, or interfere with the functionalities of existing aggregators or sensors at any time.

b. An aggregator (or a coalition of aggregators) cannot add or remove any sensor in device group outside of their scope, or interfere with the functionalities of existing aggregators or sensors at any time.

c. A sensor (or a coalition of sensors) cannot interfere with the functionalities of existing aggregators or other sensors at any time.

S-4 *Non-repudiation and data provenance:* Any BBox-IoT node cannot deny sent data they signed. For all data stored in BBox-IoT, the source must be identifiable.

S-5 *DoS resilient:* BBox-IoT continues to function even if MSP is offline and not available, or an adversary prevents communication up to a number of orderers (as dictated

by the consensus algorithm), a number of aggregators (as dictated by the system policy) and up to all but one sensor. Also an adversary is not able to deny service to any system node (except through physical layer attacks discussed before).

S-6 *System policy and configuration security:* BBox-IoT policy and configuration can only be changed by MSP.

S-7 *Revocation:* The system is able to revoke authentication for any system participant, and a system participant can have its credentials revoked only by designated system participants.

6.4 Constructions

6.4.1 Our Hash-based signature scheme

Our construction is inspired by Lamport passwords [198] and TESLA [178, 179] but *avoids the need for any synchronization* between senders and receivers which is a strong assumption for the IoT setting. Instead, we assume the existence of a constant-sized state for both the sender and receiver between signing operations. Our scheme allows for a fixed number of messages to be signed, and has constant communication and logarithmic computation and storage costs under the following requirements and assumptions:

- There's *no* requirement for time synchronization, and a verifier should only need to know the original signer's pk.
- The verifier should immediately be able to verify the authenticity of the signature (i.e. without a "key disclosure delay" that is required in the TESLA family protocols, described in more detail in Section 6.6.2).
- Network outages, interruptions or "sleep" periods can be resolved by requiring computational work from the verifier, proportional to the length of the outage.

- We do not protect against Man-in-the-Middle attacks in the signature level, instead, we use the underlying blockchain to detect and mitigate such attacks as we discuss later in Section 6.4.3.

- The signer has very limited computation, power and storage capabilities, but can outsource a computationally-intensive pre-computation phase to a powerful system.

Our scheme, presented in Construction 6.1, is a chain-based one-time signature schemesecure under an adaptive chosen-message attack as formally defined in Definition 15, with each key derived from its predecessor as $k_i \leftarrow h(k_{i+1})$, $i \in \{n-1, n-2, \ldots, 0\}$ and h is a preimage resistant hash function. The keys when used in pairs (k_i, k_{i-1}) can be viewed as a public-private key pair for a one-time signature scheme, then forming a one-way hash chain with consecutive applications of h. The key k_n serves as the "private seed" for the entire key chain. In the context of integrity, a signer with a "public key" $k_{i-1} = h(k_i)$ would have to use the "private key" k_i to sign his message. Since each key can only be used once, the signer would then use $k_i = h(k_{i+1})$ as his "public key" and k_{i+1} as his "private key", and continue in this fashion until the key chain is exhausted.

For example as shown in Figure 6.2, we can construct a hash chain from seed k_5. For signing the 1st message m_1, the signer would use $(\mathsf{pk_1}, \mathsf{sk_1}) = (k_0, k_1)$ and output signature $\sigma = h(m_1||k_0)||k_1$. Similarly, for the 2nd message he would use $(\mathsf{pk_2}, \mathsf{sk_2}) = (k_1, k_2)$ and for the 5th message $(\mathsf{pk_5}, \mathsf{sk_5}) = (k_4, k_5)$.

Constructing the one-way hash-chain described above, given the seed k_n, would require $O(n)$ hash operations to compute $k_0 = h^n(k_n)$, which might be a significant computational cost for resource-constrained devices, as the length of the hash chain n is typically large to offset the constraint of single-use keys. While we could pre-compute all the keys, which would cost a $O(1)$ lookup operation, we would then require $O(n)$ space, which is also a limited resource in such devices. Using efficient algorithms [182,183], we can achieve logarithmic storage and computational costs by placing "pebbles" at positions $2^j = 1 \cdots \lceil log_2(n) \rceil$, which as shown in Section 6.5.3 makes our construction practical for resource-constrained

Let $h : \{0,1\}^* \to \{0,1\}^\lambda$ be a preimage resistant hash function.

$(\mathsf{pk}, \mathsf{sk_n}, s_0) \leftarrow \mathsf{OTKeyGen}(1^\lambda, n)$

- sample a random "private seed" $k_n \leftarrow \{0,1\}^*$
- generate hash chain $\mathsf{pk} = k_0 = h(k_1) = h(h(k_2)) = ... = h^i(k_i) = h^{i+1}(k_{i+1}) = ... = h^{n-1}(k_{n-1}) = h^n(k_n)$
- hash chain creates n pairs of $(\mathsf{pk_i}, \mathsf{sk_i})$ where:
  $(\mathsf{pk_1}, \mathsf{sk_1}) = (k_0, k_1) = (h(k_1), k_1)$,
  $(\mathsf{pk_2}, \mathsf{sk_2}) = (k_1, k_2) = (h(k_2), k_2)$,
  ... ,
  $(\mathsf{pk_i}, \mathsf{sk_i}) = (k_{i-1}, k_i) = (h(k_i), k_i)$,
  ...,
  $(\mathsf{pk_n}, \mathsf{sk_n}) = (k_{n-1}, k_n) = (h(k_n), k_n)$
- initialize a counter $\mathsf{ctr} = 0$, store ctr and pairs as $[(\mathsf{pk_i}, \mathsf{sk_i})]_1^n$ to initial state s_0
- output $(\mathsf{pk} = \mathsf{pk_1}, \mathsf{sk_n}, s_0)$.

Note: Choosing to store only $(\mathsf{pk}, \mathsf{sk_n})$ instead of the full key lists introduces a storage-computation trade-off, which can be amortized by the "pebbling" technique we discuss in this section.

$(\sigma, \mathsf{sk_i}, s_i) \leftarrow \mathsf{OTSign}(\mathsf{sk_{i-1}}, m, s_{i-1})$

- parse s_{i-1} and read $\mathsf{ctr} \to i - 1$
- compute one-time private key $\mathsf{sk_i} = k_i$ from $n - i$ successive applications of the hash function h on the private seed k_n (or read k_i from $[\mathsf{sk}]_1^n$ if storing the whole list)
- compute $\sigma = h(m||\mathsf{pk_i})||\mathsf{sk_i} = h(m||k_{i-1})||k_i = h(m||h(k_i))||k_i$
- increment $\mathsf{ctr} \to \mathsf{ctr} + 1$, store it to updated state s_i

$\mathsf{OTVerify}(\mathsf{pk}, n, m, \sigma) := b$

- parse $\sigma = \sigma_1||\sigma_2$ to recover $\sigma_2 = k_i$
- Output $b = (\exists j < n : h^j(k_i) = \mathsf{pk}) \wedge (h(m||h(k_i)) = \sigma_1)$

Note: The verifier might choose to only store the most recent k_i which verified correctly, and replace pk with k_i above resulting in fewer hash iterations.

Construction 6.1: n-length Chain-based Signature Scheme

$$k_0 \xleftarrow{h} k_1 \xleftarrow{h} k_2 \xleftarrow{h} k_3 \xleftarrow{h} k_4 \xleftarrow{h} k_5$$

Figure 6.2: Key generation for $n = 5$ and seed k_5. First signature uses as $\mathsf{pk} = k_0$ and $\mathsf{sk} = k_1$.

Table 6.1: Hash-based scheme comparison.

Scheme	Architecture	NoSync	NoDelay
TESLA [178, 179]	Chain	✗	✗
μTESLA 2-level chain [199]	Chain	✗	✗
Sandwich, 1-level, light chain [200]	Chain	✗	✗
Comb Skipchain [200]	Chain	✔	✗
Short Hash-Based Signatures [201]	Chain	✔	✔
XMSS [202]	Tree	✔	✔
BPQS [203]	Chain	✔	✔
SPHINCS [204]	Tree	✔	✔
Our construction	Chain	✔	✔

Table 6.2: Hash-based scheme comparison for 256-bit messages and 256-bit security parameter. Sizes in bytes. $\mathbb{M}$,$\mathbb{F}$ and $\mathbb{H}$ denote MAC, PRF and hash operations respectively. n denotes length of chain-based schemes.

Scheme	$\lvert\sigma\rvert$	$\lvert\mathsf{pk}\rvert$	$\lvert\mathsf{sk}\rvert$	$\mathsf{Sign}()$	$\mathsf{Verify}()$
Short Hash-Based Signatures [201]	$128 + log_2 n$	32	$64(\lceil log_2(n) \rceil + 1)$	$(\lceil log_2(n) \rceil + 3)\mathbb{H} + 3\mathbb{F}$	$\lceil log_2(n) \rceil$
XMSS [202]	2692 (4963)	1504 (68)	64	$747\mathbb{H} + 10315\mathbb{F}$	$83\mathbb{H} + 1072\mathbb{F}$
BPQS [203]	2176	68	64	1073 $\mathbb{H}$	1073 $\mathbb{H}$
SPHINCS [204]	41000	1056	1088	386$\mathbb{F}$, 385 PRGs, 167519 $\mathbb{H}$	14060 $\mathbb{H}$
Our Construction	32(64)	32	32	$\lceil log_2(n) \rceil$ $\mathbb{H}$	1 $\mathbb{H}$

devices. The verifier's cost is $O(1)$ when storing the most recently-used k.

Comparison and Discussion. Our scheme is directly comparable with the TESLA Broadcast Message Authentication Protocol [178, 179], which follows a similar chain-based paradigm but requires some synchronicity between the sender and receiver, and the receiver can only verify a message after some delay. Several other chain-based schemes have been proposed [199–201], forming a "hierarchy" of chains aiming to improve their efficiency in various aspects. However, most of them do not prevent the synchronicity requirement and delayed verification, in fact some even introduce additional requirements, e.g. special "commitment distribution" messages [199], where a verifier won't be able to verify a long series of signatures if those are lost. As our scheme is hash-based, we compare with another family of hash-based signatures schemes that follow a tree structure, e.g. XMSS [204] and SPHINCS [202]. While these schemes do not have any synchronicity assumptions,

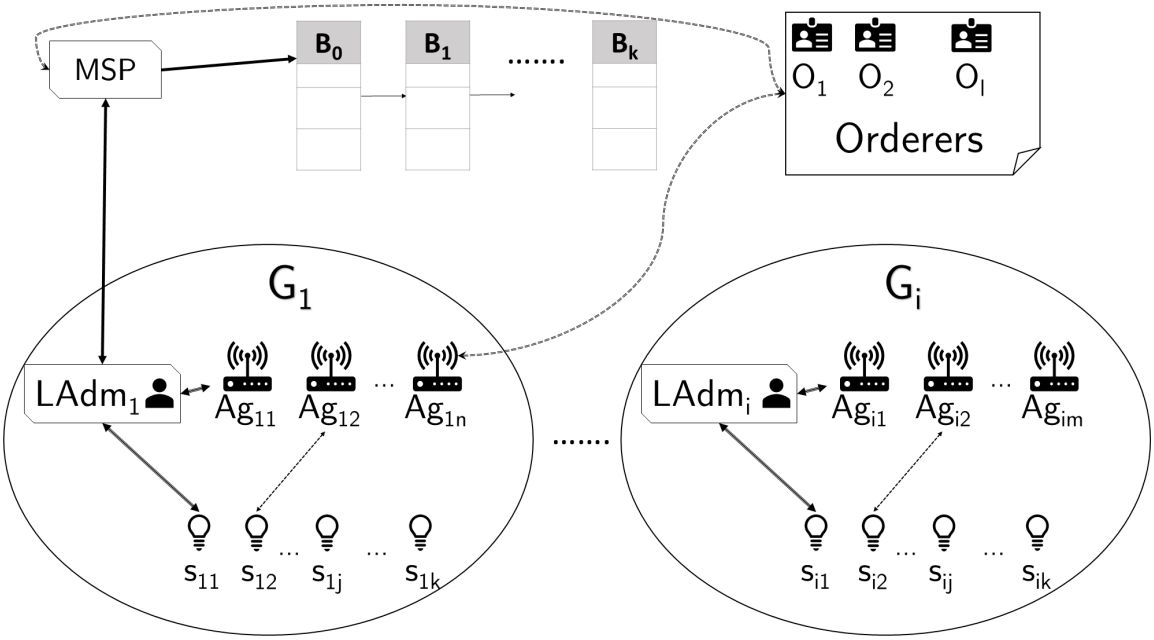


Figure 6.3: BBox-IoT construction overview

their performance is not suited for the low SWaP sensors we consider (even with resource-constrained device optimizations [205]which we compare in detail in Section 6.8.1). In Table 6.1 we compare with other hash-based schemes in terms of properties (i.e. no synchronicity or delays, denoted as NoSync and NoDelay respectively). In Table 6.2 we provide a concrete comparison with the rest of the schemes satisfying the above properties. In Section 6.6 we discuss some of the above schemes in more detail.

The caveat in our scheme is that it is susceptible to Man-in-the-Middle attacks. Specifically, an attacker might intercept a signature packet in transit (thus learning the "ephemeral" private key) and replace it with an arbitrary message and signature. Nevertheless such attacks are highly unlikely to be successful in our setting as discussed later in Section 6.4.3.

6.4.2 Overall BBox-IoT construction

Our BBox-IoT system consists of the following components as shown in Figure 6.3 illustrating our modifications to the Hyperledger Fabric architecture.

- A (trusted) Membership Service Provider[2] MSP, which resembles a Trusted Party, and is responsible for authorizing participation in the system. The MSP bootstraps the system and forms the genesis block, which contains hardcoded information on its public key and the consensus algorithm. The genesis block also initializes the authorized system participants and the system policy (denoted by Pol), both of which can be changed later.

- A permissioned blockchain BC, which consists of normal "transaction" blocks and special "configuration" blocks.

- A configuration Config for BC, containing membership information for local administrator, orderer and aggregators, as well as system policy data. As in Hyperledger Fabric, Config is stored in the configuration blocks.

- A set of orderer nodes $\mathcal{O} : \{\mathsf{O}_1, \mathsf{O}_2, ..., \mathsf{O}_\ell\}$, responsible for achieving consensus on forming new blocks. These nodes are assumed static, although it can be extended to handle dynamic membership.

- A set of device groups $\mathcal{G} : \{\mathsf{G}_1, \mathsf{G}_2, ..., \mathsf{G}_\mathsf{n}\}$. On each group G_i there exist:

  - A local administrator LAdm_i, responsible for its group membership, which includes a set of aggregators and sensors. In order for LAdm_i to add or remove an aggregator in the system must also have consent from the MSP, however he does not need permission to handle sensor membership.

  - A set of aggregators $\mathcal{AG}_i : \{\mathsf{Ag}_{\mathsf{i}1}, \mathsf{Ag}_{\mathsf{i}2}, ..., \mathsf{Ag}_{\mathsf{im}}\}$, which also have the role of *peers* in Hyperledger Fabric. We assume aggregators can perform regular cryptographic operations and aggregate data received from sensors. As discussed in our modified Hyperledger, they also briefly take the role of a "client".

  - A set of sensors $\mathcal{S}_i : \{\mathsf{S}_{\mathsf{i}1}, \mathsf{S}_{\mathsf{i}2}, ..., \mathsf{S}_{\mathsf{ik}}\}$, which are assumed to be resource-constrained devices. These would be the equivalent of *clients* in the original Hyperledger Fabric architecture, but here they are assumed to only broadcast their data to nearby

[2]The MSP also includes the system administrator.

group aggregators, without expecting a confirmation. The only step where interaction occurs is during initial setup, where they exchange their public key and other initialization data with the group administrator. We also assume that sensors can only perform basic cryptographic operations (i.e. hashing), meaning they can't perform public key cryptography operations that use exponentiations.

We first describe the initialization process for the system's MSP and genesis block B_0. After generating its keys, MSP bootstraps the system with pre-populated participation whitelists of orderers, local group administrators, and aggregators (denoted by OL, LL and PL respectively) and a pre-defined system policy. Sensors do not need to be tracked from the MSP, as participation authorization for sensors is delegated to the group local administrators. Local administrators control authorization privileges with a respective sensor whitelist denoted by SL, and they also keep a whitelist of group aggregators denoted by AL.

Furthermore, we detail the functionality of reading or updating the system's configuration, including the permissioned participants and the system policy. Orderers and local administrators can only be authorized for participation by the MSP, while aggregators need their local administrator's approval as well. As discussed above, sensor participation is handled by the local administrators, however, group aggregators also keep track of group participation for sensors in a passive manner. The local administrators are also responsible for revoking participation rights for aggregators and sensors belonging in their group. In general, granting or revoking participation privileges is equivalent to adding or removing the participant's public key from the respective whitelist. Note that membership verification can also be handled by accumulators [206, 207] instead of whitelists to achieve lists of constant size, however we keep whitelists for simplicity purposes.

Furthermore, on a high-level, sensors "blindly" broadcast their data as signed transactions. Nearby aggregators (belonging to the same device group) receive and verify the data and collect the required amount of signatures from other aggregators in the system (as defined by the system policy), and then submit the signed transaction to the ordering service. The orderers then by running the consensus protocol, "package" the collected transactions

to form a blockchain block. Finally, the block is sent back to the aggregators, who as the blockchain "maintainers", append it to the blockchain. The core system functionalities are shown in Construction 6.2 and we provide a detailed description of all system algorithms and protocols in Section 6.10. .

Sensor join: Defined by SensorJoin() protocol between a sensor and a Local administrator. This is the only phase when a sensor is interacting with the system, as the LAdm generates a new hash chain and its associated pebbles in a powerful device. The pebbles are then loaded to the sensor, and LAdm updates the group aggregators with the new sensor's public key.

Sensor broadcast: Defined by SensorSendData() protocol between a sensor and group aggregators. For some data m, the sensor computes the one-time hash-based signature using OTSign() and the signed data m, σ is broadcasted to all group aggregators. If there is any aggregator who receives a different signed message m', σ, the message is discarded, else it remains in the aggregator's pending memory for processing.

Aggregator transaction: Defined by AggrSendTx() protocol between aggregators and orderers. For an aggregator to submit aggregated data to the blockchain, it first needs to collect the needed signatures from other aggregators. Then it submits the signed transaction to the ordering service, which in turn executes the Consensus() algorithm to construct a block with a set of signed transactions. Finally, the block is transmitted to the aggregators, who append the block as the blockchain maintainers.

Sensor transfer: Defined by SensorTransfer algorithm, executed when a sensor is transferred to a new location or device group. The handing-over aggregator saves its state of our signature scheme w.r.t. that sensor and encrypts it on the blockchain under the receiving aggregator's public key. After sensor transfer, the receiving aggregator decrypts that state and resumes message verification.

Optionally in our construction, a symmetric group key $\mathsf{K_G}$ can be shared between each group's local administrator, aggregators and sensors for confidentiality purposes. However,

SensorJoin

- Sensor generates a seed uniformly at random, and generates hash chain through OTKeyGen algorithm. (computation is out sourced to a powerful device)
- Sensor stores hash chain "pebbles" in its memory and outputs the last element of the chain as public key to the LAdm

SensorSendData

- Sensor computes signature σ for broadcasted data m using OTSign algorithm
- S_{ij} broadcasts σ to aggregators in group.
- Each aggregator after verifying the signature through OTVerify, checks if any other aggregator received a conflicting message. It adds the message - signature pair in its local state, pending for blockchain submission.

AggrSendTx

- Aggregator parses its local state for pending blockchain operations as a transaction.
- Aggregator computes signature on transaction and sends it to other aggregators.
- Each aggregator after verifying signature and sender membership in the system, signs the transaction.
- The sending aggregator submits signed transaction to ordering service after reaching the necessary number of signatures, as dictated by system policy.
- Each orderer after verifying signatures, runs consensus algorithm which outputs a blockchain update operation.
- The blockchain operation is received by orderers who update the blockchain state.

SensorTransfer

- Aggregator encrypts the state for the sensor under the receiving aggregator's pk (i.e. the most recently received sk_i) and submits it to the blockchain using AggrSendTx. Sensor is removed from the device group and is transferred to a new group.
- Receiving aggregator decrypts state from the blockchain and resumes verification of received data from sensor.

Construction 6.2: BBox-IoT core algorithms and protocols

the additional encryption operations have an impact mainly on sensors, which have constrained computational and storage resources. Note that using such a key for authentication or integrity would be redundant since these properties are satisfied using public keys existing in the appropriate membership lists and revocation operations can still be performed at an equivalent cost using those lists.

6.4.3 Security analysis

Theorem 6 (informal)**.** *The construction in Section 6.4.2 satisfies participant authentication (S-1), sensor health S-2 and device group safety properties (S-3) assuming* (SignGen, Sign, SVrfy) *is an existentially unforgeable under a chosen-message attack signature scheme,* (OTKeyGen, OTSign, OTVerify) *is an unforgeable one-time chain based signature scheme,* MSP *is honest and not compromised and the consensus scheme* (TPSetup, PartyGen, TPMembers, Consensus) *satisfies the consistency property.*

Proof Sketch. We now provide a proof sketch arguing about the security of our scheme.
S-1 Participant Authentication. We require that only authenticated participants can participate in the different functions of our protocol. We argue that if an adversary breaks the participant authentication property then it could break the unforgeability of the underlying signature scheme. Specifically:

1. For property S-1a., in protocol OrdererAdd (coupled with ConfigUpdate) the use of an unforgeable signature scheme guarantees that no one but the MSP can authenticate orderers, while in protocol Consensus the same scheme guarantees that only the authenticated orderers can perform this core functionality. Also recall from the previous property that an adversary being able to authenticate orderers could break the immutability property.

2. For property S-1b., in protocol LAdminAdd (coupled with ConfigUpdate) the use of an unforgeable signature scheme guarantees that no one but the MSP can authenticate Local Administrators, while AggrSetup, AggrAdd, AggrUpd, SensorSendData and

GroupRevoke guarantee that only the authenticated local administrators can perform these functionalities.

3. For property S-1c., in protocol AggrAdd (coupled with ConfigUpdate) the use of an unforgeable signature scheme guarantees that no one but the MSP can authenticate Aggregators, while AggrUpd and AggrSendTx the same scheme guarantees that only the authenticated aggregators can perform these functionalities.

S-2 Sensor health. In order for an adversary to impersonate/clone a sensor, it would either have to break the unforgeability of our signature scheme, or launch a MITM attack which is a potential attack vector as discussed in Section 6.4.1.

As discussed in Section 6.3.1, we consider jamming attacks at the physical layer outside the scope of this thesis. Given the nature of our setting where a sensor's broadcast has typically short range, we consider MITM and message injection attacks hard and unlikely to launch but we still consider them as part of our threat model. Even in these unlikely scenarios, MITM attacks can be easily mitigated in BBox-IoT. A first approach for detecting such attacks is to leverage blockchain properties, where aggregators can compare data received from a sensor at the blockchain level. Our assumption here is that sensor data can be received by more than one aggregator in the vicinity of the sensor which is a reasonable scenario for typical dense IoT deployments. If there's even one dissenting aggregator, probably victim of a MITM attack, all the associated data would be considered compromised and disregarded and the operator will be notified of the data discrepancy detected. The above approach, while simple, still permits a MITM attacker to "eclipse" a sensor from the system using a jamming attack.

An alternative approach is to make a proactive check at a group level, where each aggregator would verify the validity of its received data by comparing it with other aggregators before even submitting it to the blockchain. In both above strategies, the attacker's work increases significantly because he would need to launch simultaneous MITM attack between the sensor and all aggregators in the vicinity. We adopt the second approach in

our Construction in Section 6.10.

The above properties and strategies ensure that only data broadcasted by authenticated sensors are accepted by aggregators in SensorSendData.

S-3 Device group safety. An adversary wanting to break device group safety would either have to add or revoke aggregators or sensors in an existing group through AggrAdd, AggrUpd or GroupRevoke thus breaking unforgeability of the signature scheme used in these protocols or interfere with existing authenticated sensors in a group through SensorSendData by breaking unforgeability of the one-time chain based signature scheme. □

Integrity, non-repudiation and data provenance requirements (S-4) are core properties of any digital signature scheme thus directly satisfied in BBox-IoT.

Additionally, we argue that our system is DOS resilient (S-5) in the following scenarios:

- MSP offline or not available: The core system functionality is not affected, although there can be no configuration changes in the system. All algorithms and protocols (except those involving adding or revoking orderers, local administrators or aggregators or those involving system policy changes) perform authentication through the configuration blocks and not the MSP itself.
- Orderers unavailable: Reduces to tolerance properties of the consensus algorithm.
- LAdm unavailable: The core system functionality is not affected, although there can be no administrative operations in the respective group.
- Ag unavailable: Transactions are not processed only in the respective groups. However if more than τ aggregators are unavailable as required in AggrSendTx, no transactions can be processed in the whole system.

Also an adversary might attempt to flood an aggregator by broadcasting messages and arbitrary signatures. In this scenario the aggregator would be overwhelmed since by running OTVerify for each message-signature pair separately, it would have to check the signature against all hash chain values up to the first public key. To mitigate this we propose checking

only for a few hashes back to the chain (defined by a system parameter "maxVerifications" as shown in Algorithm 2. This parameter can be set by the local administrator but should be carefully selected. A small value might result in need of frequent re-initializations for the sensors - if a long network outage occurs between a sensor and an aggregator and they lose "synchronization", the local administrator should reinitialize the sensor in the device group. On the other hand, a large value would amplify the impact of DOS attacks.

Policy and configuration security (S-6) is ensured by algorithms ConfigUpdate and ReadConfig, as the first algorithm creates a special configuration transaction signed by the MSP and the second returns configuration data originating from such a transaction.

Revocation (S-7) is made possible by NodeRevoke and GroupRevoke (in conjunction with whitelists used throughout all system protocols and algorithms). Also the unforgeability of the underlying signature scheme ensures that only the MSP (and the LAdm respectively only for aggregators and sensors) can revoke these credentials.

Remark. One might suggest using MACs instead of our proposed signature scheme for sensor authentication, however this would compromise sensor auditing (we discuss this in more detail in Section 6.7).

6.5 Performance evaluation & measurements

6.5.1 The IIoT setting with constrained devices

IIoT environments are complex systems consisting of heterogeneous devices that can be tracked at different organizational layers, namely (a) computational, (b) network, (c) sensor/edge layers [208]. Devices at the higher levels are powerful servers dedicated to the analysis of data, storage, and decision making. They frequently reside outside the factory premises, i.e., in cloud infrastructures. On the other hand, on-site and at the edge layer, a myriad of low-SWaP devices such as sensors and actuators reside, assigned with the tasks of posting their data or reconfiguring their status based on received instructions. On typical real-life IIoT deployments, the processing speed of such devices ranges from tens (e.g.,

Table 6.3: Classes of Constrained Devices in terms of memory capabilities according to RFC 7228.

Name	RAM	Flash
Class 0	<<10 KiB	<<100 KiB
Class 1	10 KiB	100KiB
Class 2	50KiB	250KiB

Atmel AVR family) to hundreds of MHz (e.g., higher-end models of ARM Cortex M series). Diving even deeper, at the lower end of the spectrum, one may observe sensor-like devices that are severely constrained in memory and processing capabilities.

Such extremely constrained devices have been considered by RFC 7228 [177] which underlines that "most likely they will not have the resources required to communicate directly with the Internet in a secure manner". Thus, the communication of such nodes must be facilitated by stronger devices acting as gateways that reside at the network layer. In Table 6.3 we provide a taxonomy of constrained devices residing at the edge of IIoT according to RFC 7228.

In this work, we consider a generic IIoT application scenario that involves Class 0 devices which are connected to more powerful IoT gateways in a sensor/gateway ratio of 10:1. The chosen platforms and all experimental decisions were made to provide a realistic scenario under the following assumptions: (a) devices severely constrained in terms of computational power and memory resources (Class 0) and (b) moderately demanding in terms of communication frequency (i.e. transmission once every 10 seconds).

6.5.2 Evaluation setup

Our testbed consists of Arduino UNO R3 [209] open-source microcontroller boards equipped with ATmega328P 16 MHz microcontroller and 2KB SRAM fitted with a Bluetooth HC-05 module. These devices are really constrained and they represent the minimum of capabilities in all of IoT sensors utilized in our experimental scenarios (Class 0 in Table 6.3). For the gateways, we use Raspberry Pi 3 Model B devices equipped with a Quad Core

1.2GHz BCM2837 64bit CPU and 1GB RAM.

We first focus on evaluating our system in a device group level[3]. We use the one-time signature scheme outlined in Construction 6.1 and SHA256 as the hash function $h()$. The length of the hash chain as defined in section 6.6.2 sets the upper bound on the number of one-time signatures each sensor $\mathsf{S_i}$ can generate. In the case where the sensor's available signatures are depleted, it would enter an "offline" state and the Local Administrator LAdm would need to manually renew its membership in the system through the SensorJoin protocol. In a large-scale deployment of our system however, frequent manual interventions are not desirable, so our goal is to pick a sufficiently large n such that the available one-time signatures to the sensor last for the sensor's lifetime. As discussed above and taking similar schemes' evaluations into account [210], we consider a frequency of one (1) signing operation per 10 seconds for simplicity. We consider sensor lifetimes between 4 months as an lower and 21 years as a upper estimate (as shown in Table 6.5), which imply a hash chain between 2^{20} and 2^{26} elements respectively.

In the setup phase, we pre-compute the hash-chain as needed by the pebbling algorithm [182] and load the initial pebble values into the sensor. We first measure the actual needed storage on the sensor for various values of n. Note that for $n = 2^{26}$, the lower bound for needed storage using a 256-bit hash function is about $26 \cdot 256 = 832$ bytes of memory. Then we set the sensor device to communicate with the aggregator through Bluetooth in broadcast-only mode and measure the maximum number of signing operations that can be transmitted to the aggregator for various values of n, as well as the verification time needed on the aggregator side since it will need to verify a large number of sensor messages. The fact that we are able to run BBox-IoT on Class 0 devices demonstrates the feasibility of our approach for all low-SWaP sensors.

[3]Our code is available at `https://github.com/PanosChtz/Black-Box-IoT`

6.5.3 Signing and verification

We run our experiments under different scenarios and multiple times. Our evaluation results, which are shown in Table 6.5, represent the statistical average across all measurements. Note that for measuring the average signature verification time on the aggregator side, we assume that the aggregator is able to receive all the data broadcasted by the sensor. If a network outage occurs between them (and the sensor during the outage keeps transmitting), the aggregator after reestablishing connection would have to verify the signature by traversing the hash chain back up to the last received secret key, which incurs additional computation time (in Figure 6.4 we show the associated verification cost in such occasions). As expected, the verification time is relatively constant in all measurements, about 0.031ms on average. This suggests that such an aggregator could still easily handle 10^5 sensors transmitting data for verification (as we considered one transmission every 10 seconds for each sensor).

Table 6.5, shows that the pebbles data structure consumes most of the required memory storage in our implementation, while the remaining program requires a constant amount of memory for any number of pebbles. We also observe a slight impact of the number of pebbles on the total verification time, which is mainly affected by the sensor's capability to compute the signature on its message and the next secret key. For example, the sensor needs 50ms to compute the next signature with $n = 2^{26}$ and 49.95ms for $n = 2^{24}$. Also by comparing the total verification time with the signature computation time, we conclude the extra 14.3 msec are needed for transmitting the signature.

In Table 6.4 we provide a series of measurement results for the average verification time of 1 signature on the aggregator. By T2 we denote the verification time of a signature and by T3 the total verification time by an aggregator (we provide detailed algorithms for our measurements in Section 6.11.) The average total verification time (denoted by maxV) increases significantly as we require more verification operations from the Arduino device. This happens because of dynamic memory fragmentation as the pebbling algorithm updates the pebble values.

Table 6.4: Evaluation for sensor-aggregator protocol - Average verification times

	T2 (μsec)				T3 (msec)			
maxV	20	22	24	26	20	22	24	26
100	28.12	31.18	31.34	28.95	42.83	42.84	42.91	43.08
500	30.78	31.94	30.31	30.63	51.25	51.23	51.37	51.39
1000	31.39	30.96	31.14	30.74	55.27	55.35	55.36	55.41
2500	30.57	30.97	32.39	30.86	60.61	60.65	60.7	60.78
5000	33.26	31.7	31.66	31.43	64.66	64.74	64.79	64.83
10000	33.34	33.38	33.6	31.41	68.68	68.75	68.78	68.86

Comparison with ECDSA We compare our lightweight scheme with ECDSA, which is commonly used in many blockchain applications. We assume IoT data payloads between 50 and 220 bytes, which can accommodate common data such as timestamps, attributes, source IDs and values. In Table 6.6 we show that our scheme is more efficient compared to ECDSA by 2 and 3 orders of magnitude for signing and verification respectively. Even when considering larger payload sizes which impact hash-based signature operations, our scheme remains much more efficient. However, verification cost for our scheme increases linearly during network outages, and as shown in Figure 6.4 it might become more expensive than ECDSA when more than 2400 signature packets are lost.

Another metric we consider is energy efficiency, which is of particular importance in IoT applications that involve a battery as power source. Our experiments depicted in Figure 6.5 show that our ATmega328P microcontroller can perform more than 50x hash-based signing operations compared to the equivalent ECDSA operations for the same amount of power. Finally, while our hash-based signature normally has a size of 64 bytes (as shown in Table 6.5), we can "compress" consecutive signatures along a hash chain to 32 bytes by only publishing the most recent k_i. The verifier would then generate the previous hash chain values at a minimal computational cost. This makes it possible to store more authenticated data in the blockchain, as we show below.

Table 6.5: Evaluation for sensor-aggregator protocol (average values for 5000 verifications)

Hash Chain length n	2^{20}	2^{22}	2^{24}	2^{26}
Sensor lifetime for 1sig/10sec (m: months, y: years)	4 m	16 m	5 y	21 y
Pebble Gen time (seconds)	1.62	6.49	24.57	95.33
Verification time per signature (msec)	0.031			
Signature size (bytes)	64+ $\|m\|$			
Total dynamic memory usage (bytes)	1436	1520	1604	1678
Pebble struct memory usage (bytes)	840	924	1008	1082
Program memory usage (bytes)	596			
Signature computation time (msec)	49.82	49.88	49.95	50.00
Average total verification time per signature (msec)	64.15	64.25	64.26	64.32
Communication cost (msec)	14.3			

6.5.4 Consensus performance

Considering the use-case scenario discussed in Section 6.5.1, we discuss the performance of our BBox-IoT system as a whole. We show that the most important metric in the system is the transaction throughput which heavily depends on the ability of the SWaP sensors to transmit data in a group setting. Of course, the scalability of the system overall is also directly proportional to the number of system active participants it can support simultaneously.

Sensors. Our measurements indicate that the aggregator - which is a relatively powerful device - is not the bottleneck in the protocol execution. Based on the measurements in Table 6.5, we can safely assume that a single aggregator can verify over a thousand sensors' data being continuously broadcasted, since the signature computation time by a sensor is

three (3) orders of magnitude larger than the verification time by an aggregator. This is still a pessimistic estimation, since we previously assumed that a sensor broadcasts (and signs) data every 10 seconds, which implies that the aggregator can accommodate even more sensors.

Orderers. Since orderers only participate in the consensus protocol to sign blocks, we only need a few orderers such that our system remains resilient to attacks at the consensus level should a subset of orderers become compromised. Orderers can be strategically distributed over a geographical area to minimize the network latency between an aggregator and the ordering service, controlled by the main organization (which also controls the MSP). Evaluations performed in previous works have shown that by having 3 orderers, 3000 transactions/second can be easily achieved using the consensus protocol used in the current version of Hyperledger Fabric (with a potential of further improvement in a future adoption of BFT-SMART), and even considering up to 10 orderers in the system does not greatly affect its performance [185, 192].

Aggregators. The expected number of aggregators in the system depends on the use case as it is expected. As discussed in Section 6.5.1, where gateways play the role of BBox-IoT aggregators, we consider a sensor/gateway ratio of 10:1 for our evaluation purposes. To our knowledge, no evaluation of Hyperledger Fabric has ever been performed to consider such a great number of peers, which would require a great amount of resources to perform. However, by adopting the evaluation performed in [185] which measured the throughput in terms of number of peers up to 100 (which as discussed, are the aggregators in our system), we can extrapolate this evaluation to the order of thousands, which shows that with the aid of a "peer gossip" protocol, the system remains scalable if the peers are in the same approximate geographical area which implies low average network latency.

Blockchain operations. As discussed, aggregators' role is to aggregate sensor data into blockchain transactions. Assuming that aggregators perform no "lossy" operations (such as averaging techniques), they would just package many collected sensor data along with the

Table 6.6: Signing and verification costs (in milliseconds) compared with message and signature sizes (in bytes). Note we assume hash-based signatures are aggregated as discussed in Section 6.5.3. Signer is ATmega328P microcontroller and verifier is RPi 3.

	BBox-IoT		ECDSA	
Message length	Sensor Sign	Aggr Vrfy	Sensor Sign	Aggr Vrfy
50	50.43	0.0339	4200	42.55
100	53.47	0.0349		
150	56.40	0.0357		
202	59.33	0.03687		
218	60.06	0.0369		
Signature size	32		64	

respective signatures into a transaction which in turn would be submitted to the ordering service. If we assume as in [185] a block size of 2MB, we can estimate how much signed sensor data a block can hold. Given the discussion in Section 6.5.3, a Hyperledger block could hold (at most) about 15800 signed sensor data using our hash-based scheme vs. 12700 using ECDSA.

Latency. We also wish to estimate the time from a value being proposed by an aggregator until consensus has been reached on it (assuming the block contains a single transaction). Again we can adopt previous evaluations in Hyperledger Fabric [185], which show an average of 0.5 sec for the complete process. Finally, considering that the previous evaluations mentioned above were all performed on the original Hyperledger Fabric (while our architecture requires a slight modification as discussed in Section 6.2), for our purposes we assume that the expected performance of aggregators (which are essentially Hyperledger peers also having client application functionalities) is not affected by this additional functionality, since the main affecting factor that can potentially become a bottleneck for the scalability of the whole system is network latency and not computational power.

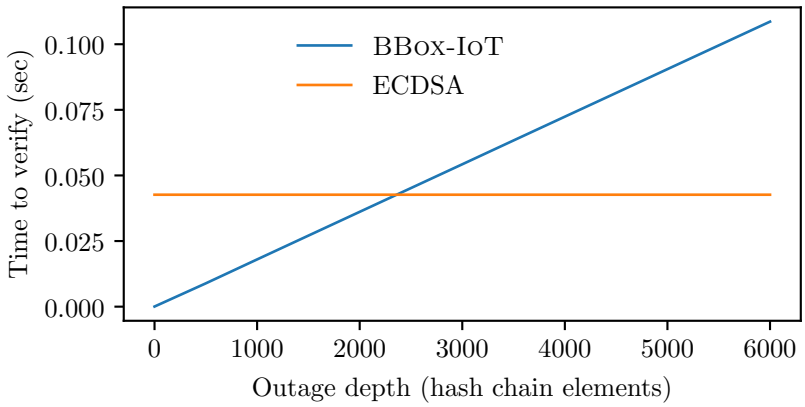


Figure 6.4: Aggregator verification costs in network outages. BBox-IoT is more expensive when more than about 2400 signature packets are lost.

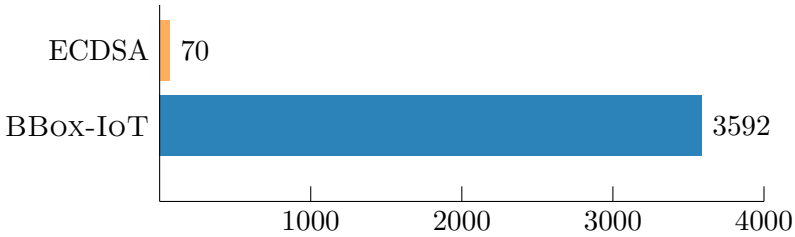


Figure 6.5: Number of signing operations for a 20mWh battery.

6.6 Related work

We now discuss a number of works that connect IoT to the blockchain setting or works which build cryptographic primitives to optimize different parts of computation for resource-constrained IoT devices. Note that none of these works addresses the problem of authentication for extremely constrained (Class 0) devices.

6.6.1 IoT and blockchain

Shafagh et al. [165] presented an architecture aiming to handle IoT data in a decentralized manner while achieving confidentiality, authenticity and integrity. This proposed system defines itself as "IoT compatible" being append-only by a single writer and can be accessed by many readers, and consists of a layered design on top of an existing public

blockchain to store access permissions and hash pointers for data, while storing the actual data off-chain using decentralized P2P storage techniques. Other approaches [211–213] also used a similar "layering" paradigm. While these approaches are simpler than ours, they ultimately rely heavily on the performance and properties of the underlying public blockchain and are not specifically tailored to handle resource-constrained IoT devices.

Dorri, Kanhere, and Jurdak [195] considered a "local" private blockchain maintained by a capable device, managed by the on-site owner and containing the local IoT device transactions. These lower-tier elements would be overlaid by a shared blockchain that can handle hashed data originating from the local blockchain and stored in a cloud storage service, and can enable access to local data. The above approach also offers confidentiality and integrity for submitted data and is suitable for resource-constrained IoT devices, however it is more complex than BBox-IoT and requires managing and replicating data over several points in the system.

More recently, AlTawy and Gong [214] presented a blockchain-based framework in the supply chain setting using RFIDs. This model considered blockchain smart contracts interacting with an overlay application on the RFID readers and a centralized server that handles membership credentials. This framework offers anonymity for the participating entities, which prove their membership in zero-knowledge, while their anonymity remains revocable by the server. It also provides confidentiality for its transactions and enforces a notion of "forward secrecy" which enables future product owners in the supply chain to access its entire history. BBox-IoT differs from the above work in several ways, since it is tailored to handle resource-constrained devices. Our work does not have confidentiality or anonymity as a main goal, although it can be added as an option using symmetric keys. We also do not require any smart contract functionality from the blockchain, and we operate exclusively in the permissioned setting.

IoTLogBlock [164] shares a common goal with our work: enabling the participation of low-power devices in a distributed fashion, and similarly uses Hyperledger as a "cloud service" in a IoT setting. The crucial difference with our work, is that IoTLogBlock is

evaluated on a Class 2 device using ECDSA signatures, which are far more expensive than our proposed hash-based signature and could not have been supported at all by a Class 0 device, while having much larger power consumption (Fig 6.5). Our proposed signature scheme is a key component for efficient implementations of blockchain-based systems in the IIoT setting.

Several more approaches have been presented which augmented an IoT infrastructure with a blockchain, focusing on providing two-factor authentication [215], managing or improving communication among IoT devices [216, 217], implementing a trust management system in vehicular networks [218], providing edge computing services [219], data resiliency [220], providing secure and private energy trade in a smart-grid environment [221] and implementing a hierarchical blockchain storage for efficient industrial IoT infrastructures [222] and all of which are orthogonal to our work. We point the reader to [223, 224] for extensive reviews on the related literature.

6.6.2 Hash-based signatures

Early works such as Lamport's One-Time Signatures (OTS) [225] allowed the use of a hash function to construct a signature scheme. Apart from being one-time however, this scheme suffered from large key sizes. Utilizing tree-based structures such as Merkle trees [48], enabled to sign many times while keeping a constant-sized public key as the Merkle root. Winternitz OTS and later WOTS+ [226][227] introduced a way of trading space for computation for the underlying OTS, by signing messages in groups. XMSS [202] further optimized the Merkle tree construction using Winternitz OTS as an underlying OTS. Other works such as HORS [228] enabled signing more than once, and more recently SPHINCS and SPHINCS+ [204,229] enabled signing without the need to track state. Using HORS [228] as a primitive combined with a hash chain, Time Valid One-Time Signature (TV-HORS) [230] improves in signing and verification computational efficiency, but assuming "loose" time synchronization between the sender and the verifier. All of the above scheme families while only involving hash-based operations, still incur either large computational and/or space

costs, and cannot be implemented in Class 0 resource-constrained devices we consider. Follow-up work exists for implementing SPHINCS on resource-constrained devices [205] which we discuss later in this section and compare in Section 6.8.1.

The TESLA Broadcast Message Authentication Protocol [178, 179] follows a "one-way" chain-based approach for constructing a hash-based message authentication scheme. Based on a "seed" value, it generates a one-way chain of n keys, which elements are used to generate temporal MAC keys for specified time intervals. The protocol then discloses each chain element with some time delay Δ, then authenticity can be determined based on the validity of the element in the chain as well as the disclosure time. The "pebbling" algorithms [182, 183] enable logarithmic storage and computational costs as discussed in Section 6.4.1. Its main drawback however is that it also requires "loose" time synchronization between the sender and the receiver for distinguishing valid keys. In an IoT setting this would require the frequent execution of an interactive synchronization protocol, since IoT devices are prone to clock drifting [180, 181]. Also we assume in Section 6.3 that IoT devices function in a broadcast-only mode, which would not allow the execution of such interactive protocol in the first place. Furthermore, TESLA introduces a "key disclosure delay" which might be problematic in certain IoT applications, and gives up the non-repudiation property of digital signatures.

Several modifications and upgrades to the TESLA protocol have been proposed, with most of them maintaining its "key disclosure delay" approach which is also associated with the loose time synchronization requirement [199, 200]. A notable paradigm is the "hierarchical" (or two-dimensional) one-way chain structure, where the elements of a "primary" hash chain serve as seeds for "secondary" chains in order to reduce communication costs. [200] includes several such proposals. For instance, its Sandwich-chain uses two separate one-way chains. The first one-way chain is used as a "primary" chain, which generates intermediate "secondary" chains using the elements of the second one-way chain as salts. However to maintain efficiency, it still assumes some weak time synchronicity between the signer and the verifier by disclosing each element of the "primary" chain with some time

delay (else the verifier in case of a network outage would have to recompute all the previous secondary chains as well which would defeat its efficiency gains). More importantly however, this construction has much larger storage requirements than ours. In the same work, the Comb Skipchain construction is asymptotically more efficient in signing costs than our scheme and does not require time synchronicity, but has worse concrete storage requirements which are prohibitive for low-end IoT devices, and still suffers from delayed verification. This work includes other interesting modifications such as the "light" chains where the secondary chains are generated using a lower security parameter and a standard one-dimensional TESLA variant which does not require a MAC.

6.6.3 Cryptographic operations in IoT

In the context of improving cryptographic operations in the IoT setting, Ozmen and Yavuz [231] focused on optimizing public key cryptography for resource-constrained devices. This work exploited techniques in Elliptic Curve scalar multiplication optimized for such devices and presented practical evaluations of their scheme on a low-end device. Even though the device used in this work can be classified as a Class 1 or Class 2 device, our construction signing is more efficient both in terms of computation cost and storage by at least an order of magnitude.

As discussed above, Hülsing, Rijneveld and Schwabe [205] showed a practical evaluation of the SPHINCS hash-based signature scheme [204] on a Class 2 device. At first glance this implementation could also serve our purposes, however our proposed construction, while stateful, is much cheaper in terms of runtime, storage and communication costs, without such additional assumptions. We directly compare with their scheme in Section 6.8.1.

Kumar et al. [232] propose an integrated confidentiality and integrity solution for large-scale IoT systems, which relies on an identity-based encryption scheme that can distribute keys in a hierarchical manner. This solution also uses similar techniques to our work for signature optimization for resource-constrained devices, however, it requires synchronicity between the system participants. Portunes [233] is tailored for preserving privacy (which is

not within our main goals in our setting), and requires multiple rounds of communication (while we consider a "broadcast-only" setting)

Wander et al. [234] quantified the energy costs of RSA and Elliptic Curve operations as public key cryptography algorithms in resource-constrained devices. In a similar context, Potlapally et al. [235] performed a comprehensive analysis of several cryptographic algorithms for battery-powered embedded systems. However as discussed in Section 6.6.2, we consider hash-based algorithms that are lighter and more efficient.

Finally we mention an extensive IoT authentication survey [236]. In this work, our authentication scheme is comparable to [237] which utilizes hashing for one-way authentication in a distributed architecture, however our scheme is more storage-efficient, suited for low-SWaP (Class 0) sensors.

6.7 On MACs for sensor authentication

One might suggest using MAC authentication in our scheme instead of one-time hash based signatures, which might be slightly more efficient in terms of computation cost for generating a signature, are simpler in usage and do not expire. The question of whether its preferable using symmetric primitives in resource-constrained IoT devices instead of public key cryptography has been raised in academic works [231], and several motivations to provide efficient public key cryptography techniques in such devices were outlined, most of which are also applicable to our system as follows.

Firstly, signatures provide non-repudiation, which as discussed previously is a needed security property S-4. Although a way to achieve non-repudiation through MACs could be to use a separate MAC key for each sensor, each key would need to be shared with each group aggregator separately since they should be all able to verify data from all sensors in the group. This would increase the attack surface since an attacker compromising any aggregator could also send bogus data for all sensors. Also considering that aggregators might have to verify data from a great number of sensors, our hash-based verification cost

(which involves one hash operation) is cheaper than one MAC operation. Although for sensors a MAC operation is cheaper than a hash-based signature, as we show in section 6.5 a hash-based signature which involves a few hashes and a Quicksort operation is still relatively efficient even for the weakest types of sensors.

Secondly, our chain hash-based scheme has a built-in "replay protection" against an attacker, since that signature is by definition valid for one time only. A MAC scheme would require extra layers of protection (nonces and/or timers) against replay attacks.

Lastly, by using our hash-based signature scheme we enable public verifiability of signed sensor data on the blockchain, even by entities not authorized to participate in the system.

6.8 Definition and Security proof of our signature scheme

We first define the API of a chain based signature for a fixed number of messages n.

- $(\mathsf{pk}, \mathsf{sk_n}, s_0) \leftarrow \mathsf{OTKeyGen}(1^\lambda, n)$: Outputs a pair of keys $\mathsf{pk}, \mathsf{sk_n}$ and an initial state s_0, where $\mathsf{pk} = h^n(\mathsf{sk_n})$ and $h()$ is a collision resistant hash function.

- $(\sigma, \mathsf{sk_i}, s_i) \leftarrow \mathsf{OTSign}(\mathsf{sk_{i-1}}, m, s_{i-1})$: Takes as input the system state s_{i-1}, a private key $\mathsf{sk_{i-1}}$ and a message m, generates a signature σ and updates the signer's private key to $\mathsf{sk_i}$ where $\mathsf{sk_{i-1}} = h(\mathsf{sk_i})$ and his state to s_i where $i \leq n$.

- $\mathsf{OTVerify}(\mathsf{pk}, m, \sigma) := b$: Takes as input a public key pk, a message m and a signature σ, and outputs a bit b where $b = 1$ indicates successful verification.

To formalize security for chain-based signatures with length of chain n, we describe the following experiment $\mathsf{OTSigForge}(\lambda, n)$:

1. $(\mathsf{pk}, \mathsf{sk_n}, s_0) \leftarrow \mathsf{OTKeyGen}(1^\lambda, n)$

2. $\mathcal{A}$ on input (pk, n) makes up to $q \leq n$ queries to the signing oracle. Let $Q : [m_i, \sigma_i]_{i=1}^q$ the set of all such queries where m_i is the queried message and σ_i is the signature returned for m_i.

3. $\mathcal{A}$ outputs (m_{q+1}, σ_{q+1}).

4. $\mathcal{A}$ wins if $\mathsf{OTVerify}(\mathsf{pk}, m_{q+1}, \sigma_{q+1}) := 1$ and $h^i(\mathsf{sk_i}) \neq \mathsf{pk}\ \forall i \leq q$ where $\mathsf{OTSigForge}$ outputs "1", else it outputs "0".

Note in the above experiment by $h^i(\mathsf{sk_i}) \neq \mathsf{pk}\ \forall i \leq q$ we restrict $\mathcal{A}$ from winning the game by reusing a secret key $\mathsf{sk_i}$ existing in the chain up to distance q from the public key pk.

Definition 15. *A chain-based one-time digital signature scheme is existentially unforgeable under an adaptive chosen-message attack, if* $\forall$ *ppt* $\mathcal{A}$, $\Pr[\mathsf{OTSigForge}(\lambda, n) = 1]$ *is negligible in* λ.

Given the formal definition above we now prove the security of Construction 6.1.

Theorem 7. *Let* $h : \{0,1\}^* \rightarrow \{0,1\}^\lambda$ *be a preimage resistant hash function. Then Construction 6.1 is an existentially unforgeable chain-based one-time signature scheme.*

Proof. Let $\mathcal{A}$ be an adversary who wins the $\mathsf{OTSigForge}$ game described in Section 6.6.2 and therefore can forge signatures using the above scheme with non-negligible probability $p(\lambda)$. That is, $\exists\ \mathcal{A}$ which after performing q queries $\{m_i\}_{i=1}^q$ where $q \leq n$, can output a signature σ_{q+1} for a message m_{q+1} where $\mathsf{OTVerify}(\mathsf{pk}, m_{q+1}, \sigma_{q+1}) = 1$ and $h^i(\mathsf{sk_i}) \neq \mathsf{pk}$ $\forall i \leq q$.

Then, an algorithm $\mathcal{B}$ running the PreIm experiment would use $\mathcal{A}$ to break preimage resistance of h as follows: On input (λ, y), $\mathcal{B}$ would generate a hash chain of length n with seed y as $(y, h(y), ..., h^n(y))$ and forward $(h^n(y), n)$ to $\mathcal{A}$. Then $\mathcal{A}$ makes up to $q \leq n$ queries to $\mathcal{B}$. When $\mathcal{A}$ queries for m_i (where i denotes the query number), $\mathcal{B}$ returns $\sigma_i = h(m_i || h^{n-i+1}(y)) || h^{n-i}(y)$ to $\mathcal{A}$. If $q = n$ and $\mathcal{A}$ does not output a forgery, $\mathcal{B}$ returns $\perp$ and starts over. If $\mathcal{A}$ eventually outputs a forgery (m^{q+1}, σ^{q+1}) to $\mathcal{B}$ and $q < n$, then $\mathcal{B}$ returns $\perp$ as output of PreIm experiment and starts over, else if $q = n$, $\mathcal{B}$ would parse $\sigma^{n+1} = \sigma^A || \sigma^B$ and return σ^B. Assuming a uniform probability distribution of the number of queries q, $\Pr[\mathsf{PreIm}(\lambda, y) = 1] = \frac{\Pr[\mathsf{OTSigForge}(\lambda, n) = 1]}{n} = \frac{p(\lambda)}{n}$ which is non-negligible.

□

6.8.1 Evaluation comparison with modified SPHINCS

As discussed in section 6.6.2, the modified SPHINCS scheme tailored for resource-constrained devices [205] could be a candidate scheme for our system. Here we make a direct comparison between modified SPHINCS and our proposed scheme for our system's purposes.

Assuming a hash chain length of 2^{26} elements (which as discussed is only exhausted after 21 years assuming generation of one signature every 10 seconds), a signature generation only requires 27 hashing operations in the worst case, which according to our measurements on a 8-bit 16Mhz CPU Arduino device outlined in Section 6.5.3, would only need 50 ms on average. On the other hand, modified SPHINCS' evaluation performed on a resource-constrained device (32-bit 32Mhz Cortex M3 which is more powerful than our Arduino Uno R3) needs 22.81 seconds for signature generation. Also our signature size (excluding the payload) is only 64 bytes for the signature and the program storage requirement 1082 bytes, while modified SPHINCS generates a 41KB signature, streamed serially. Our only additional requirement is an initial precomputation phase using a powerful device, which will have to pre-compute the 2^{26} hash chain elements and then send the "pebbles" to the resource-constrained device.

6.8.2 Collision probability analysis

Although we assumed a collision resistant hash function in our hash-based signature construction, given the length of the hash chain (typical length 2^{26}) there is an increased likelihood of a collision along that chain through the birthday paradox (especially for lower levels of security where the output size of the hash function is small), which would result in "cycles" of hashes. If such cycles occur, an adversary could then trivially break the security of our scheme and sign bogus sensor data.

Assume a hash chain of length 2^n and a security parameter λ. From the birthday

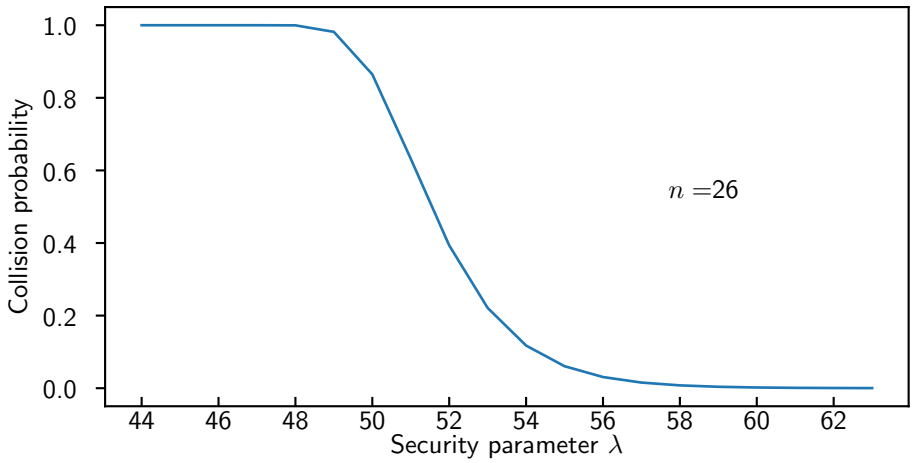


Figure 6.6: Collision probability for hash chain length 2^{26}.

paradox, the probability of a collision on the hash chain is approximated by $p = 1 - e^{\frac{-2^n(2^n-1)}{2^{\lambda+1}}}$. In Figure 6.6 we show that given a chain length of 2^{26} as previously discussed, the output size of the hash function $h()$ should be at least 64, and SHA256 which we used in our evaluations satisfies these requirements.

Nevertheless, if birthday attacks become an issue for a small security parameter, we can apply the technique from [200] where the chain index is used as salt to prevent such attacks for a small overhead in cost. However since we show that the birthday attack is negligible, we prefer to keep the costs as low as possible.

6.9 An instantiation for consensus algorithm

In the generic construction of our scheme, we assumed a "pluggable" consensus algorithm, decoupled from our construction, similar to the original Hyperledger architecture. Recall that this algorithm, which is executed among all orderers O_i, on input of a blockchain BC and some orderer state st_i, outputs an agreed new updated blockchain BC'. Here we provide a concrete instantiation of a consensus algorithm for the modified Hyperledger used

in BBox-IoT that matches the PBFT consensus protocol [55] as follows:

1. O_i parses TXL from its $\mathsf{st}_{\mathsf{O_i}}$ extracting a set of transactions $\{\mathsf{tx}_i\}$.

2. O_i based on the current BC and $\{\mathsf{tx}_i\}$ constructs a new block $\mathsf{B_i}$ which would create $\mathsf{BC}||\mathsf{B_i} \rightarrow \mathsf{BC}'$.

3. O_i computes $\sigma := \mathsf{Sign}(\mathsf{sk}_{\mathsf{O_i}}, \mathsf{B_i})$. and sends σ to all orderers in $\mathcal{O}$ (equivalent to "pre-prepare" phase in PBFT).

4. All the other orderers $\mathsf{O}_x \in \mathcal{O}$ parse $(\mathsf{OL}_{\mathsf{BC}})$ from the output of $\mathsf{ReadConfig}(\mathsf{BC})$. Check if $\mathsf{pk}_{\mathsf{O_i}} \in \mathsf{OL}_{\mathsf{BC}}$ and $\mathsf{SVrfy}(\mathsf{pk}_{\mathsf{O_i}}, \sigma, \mathsf{B_i}) = 1$. Then it verifies that the proposed block was formed correctly (i.e., it is a valid extension of the current blockchain BC). If all verifications pass, it computes $\sigma_x := \mathsf{Sign}(\mathsf{sk}_{\mathsf{O_x}}, \mathsf{B_i})$ and sends σ_x to all orderers in $\mathcal{O}$ (equivalent to "prepare" phase in PBFT).

5. Each $\mathsf{O}_x \in \mathcal{O}$ (including O_i) checks if $\mathsf{SVrfy}(\mathsf{pk}_{\mathsf{O_x}}, \sigma_x, \mathsf{B_i}) = 1$. If it collects sufficient number of signatures (specific to each consensus protocol) it computes $\sigma'_x := \mathsf{Sign}(\mathsf{sk}_{\mathsf{O_x}}, \mathsf{B_i}, 1)$ and sends σ'_x to all orderers in $\mathcal{O}$ (equivalent to PBFT "commit" phase).

6. Each $\mathsf{O}_x \in \mathcal{O}$ (including O_i) checks if $\mathsf{SVrfy}(\mathsf{pk}_{\mathsf{O_{x''}}}, \sigma'_x, \mathsf{B_i}, 1) = 1$. If it receives sufficient number of signatures (specific to each consensus protocol) it updates its state to $\mathsf{st}_{\mathsf{O_x}}{}'$ and outputs "1". It outputs "0" in all other cases.

The above instantiation satisfied the basic consensus properties in Definition 8.

6.10 Construction algorithms

For our construction we assume an existentially unforgeable signature scheme ($\mathsf{SignGen}$, Sign, SVrfy) and an unforgeable one-time chain based signature scheme as defined in Section

Table 6.7: Hash-based schemes concrete comparison, 256-bit security

Scheme	Stateful	Public key (bytes)	Secret key (bytes)	Signature (bytes)	Sign (msec)	Verify (msec)	Remarks
XMSS	Yes	68		4963	610	160	Cortex M3 32MHz 32-bit 16KB RAM [238, 239]
SPHINCS	No	1056	1088	41000	18410	513	Cortex M3 32MHz 32-bit 16KB RAM [205]
BBox-IoT	Yes	32	32	32 (64)	52	0.035	ATmega328P 16MHz 8-bit 2KB RAM

6.6.2 ($\mathsf{OTKeyGen}$, OTSign, $\mathsf{OTVerify}$). We also assume an authenticated blockchain consensus scheme ($\mathsf{TPSetup}, \mathsf{PartyGen}, \mathsf{TPMembers}, \mathsf{Consensus}$) satisfying the properties outlined in Section 2.2.

1. $\mathsf{SystemInit}(1^\lambda, \mathsf{LL}, \mathsf{OL}, \mathsf{PL}, \mathsf{Pol})$ allows the MSP to initialize the BBox-IoT system. The initialization is optionally based on predetermined initial system participants, where $\mathsf{LL}, \mathsf{OL}, \mathsf{PL}$ are lists containing public keys for Local administrators, orderers and peers respectively, as well as a preselected system policy Pol.

   (a) MSP sets as pp the system parameters for the signature and the hash function, as well as the consensus algorithm by running $\mathsf{TPSetup}$.

   (b) Computes a random key pair $(\mathsf{pk}_{\mathsf{MSP}}, \mathsf{sk}_{\mathsf{MSP}}) \leftarrow \mathsf{SignGen}(1^\lambda)$.

   (c) Assembles and outputs the genesis block B_0 (serving as the initial configuration block) by copying $\mathsf{pk}_{\mathsf{MSP}}, \mathsf{pp}$ and $[\mathsf{LL}, \mathsf{OL}, \mathsf{PL}, \mathsf{Pol}]$ from the algorithm inputs.

   (d) Initializes empty lists in MSP memory
   $[\mathsf{LL}_{\mathsf{MSP}}, \mathsf{OL}_{\mathsf{MSP}}, \mathsf{PL}_{\mathsf{MSP}}, \mathsf{Pol}, \mathsf{oper}]$ where oper denotes a pending revoke operation list.

   (e) Copies Pol to $\mathsf{Pol}_{\mathsf{MSP}}$.

The genesis block $\mathsf{B_0}$ (as the blockchain BC in general) is public, while the rest of the outputs remain private to MSP. For the following algorithms and protocols we assume that the security parameter and the system parameters are a default input.

2. $\mathsf{ConfigUpdate}(\mathsf{BC}, \mathsf{sk_{MSP}}, \mathsf{st_{MSP}})$ enables MSP to read system configuration information from its memory that is pending to be updated, and construct a new configuration block to make the new system configuration readable and valid in the blockchain by all system participants.

   (a) MSP parses $\mathsf{st_{MSP}}$ as $[\mathsf{LL_{MSP}}, \mathsf{OL_{MSP}}, \mathsf{PL_{MSP}}, \mathsf{Pol_{MSP}}]$.

   (b) Assembles a configuration update transaction $\mathsf{tx}_u = [\mathsf{LL_{MSP}}, \mathsf{OL_{MSP}}, \mathsf{PL_{MSP}}, \mathsf{Pol_{MSP}}]$.

   (c) Parses $\mathsf{ReadConfig}(\mathsf{BC})$ as $\mathsf{PL_{BC}}$.

   (d) Sends signed transaction $\sigma_{\mathsf{MSP}}(\mathsf{tx}_u)$ to all $\mathsf{Ag_i} \in \mathsf{PL_{BC}}$.

   Since Pol does not apply to transactions signed by MSP, the configuration update transaction is promptly appended to BC by all aggregators, resulting in public output BC'.

3. $\mathsf{PolicyUpdate}(\mathsf{st_{MSP}}, \mathsf{Pol})$ enables MSP to update system policy parameters. On input of a new system policy Pol, MSP copies it to $\mathsf{st_{MSP}}[\mathsf{Pol}]$, overwriting the previous policy. The algorithm outputs the new updated $\mathsf{st_{MSP}}'$.

4. $\mathsf{ReadConfig}(\mathsf{BC})$ can be run by any system participant to recover the current system configuration.

   (a) Parses BC as a series of blocks $\mathsf{B_i}$.

   (b) From the set of blocks marked as "configuration" blocks where $\mathsf{B_i}[type = "C"]$, selects the block $\mathsf{B_c}$ with the greatest height c.

   (c) Parses and outputs $\mathsf{B_c}$ as $([\mathsf{LL_{BC}}, \mathsf{OL_{BC}}, \mathsf{PL_{BC}}], \mathsf{Pol_{BC}})$.

5. $\mathsf{OrdererSetup}()$ is run by an orderer O_i initializing its credentials and state. It computes and outputs signing keys as $(\mathsf{pk}_{\mathsf{O_i}}, \mathsf{sk}_{\mathsf{O_i}}) \leftarrow \mathsf{SignGen}(1^\lambda)$ and initializes a signed transaction list in memory $\mathsf{st}_{\mathsf{O_i}}[TXL]$.

6. $\mathsf{OrdererAdd}\{\mathsf{O}_i(\mathsf{pk}_{\mathsf{O_i}}, \mathsf{sk}_{\mathsf{O_i}}) \leftrightarrow$
   $\mathsf{MSP}(\mathsf{pk}_{\mathsf{MSP}}, \mathsf{sk}_{\mathsf{MSP}}, \mathsf{st}_{\mathsf{MSP}}[\mathsf{OL}_{\mathsf{MSP}}], \mathsf{BC})\}$ is an interactive protocol between an orderer O_i and the system MSP in order to add that orderer in the system:

   (a) O_i first creates a physical identity proof π, then submits π and $\mathsf{pk}_{\mathsf{O_i}}$ to MSP.

   (b) MSP verifies π. Then it parses $(\mathsf{OL}_{\mathsf{BC}})$ from the output of $\mathsf{ReadConfig}(\mathsf{BC})$. Check that $(\mathsf{pk}_{\mathsf{O_i}} \notin \mathsf{OL}_{\mathsf{MSP}}) \wedge (\mathsf{pk}_{\mathsf{O_i}} \notin \mathsf{OL}_{\mathsf{BC}})$. If all verifications hold, add $\mathsf{pk}_{\mathsf{O_i}}$ to its local orderer list $\mathsf{OL}_{\mathsf{MSP}}$ and return "1" to O_i, else return "0" with an error code.

7. $\mathsf{LAdminSetup}()$ is an algorithm run by a LAdm to initialize its credentials and state and create a new device group G. A Local Administrator computes and outputs signing keys as $(\mathsf{pk}_{\mathsf{LAdm_i}}, \mathsf{sk}_{\mathsf{LAdm_i}}) \leftarrow \mathsf{SignGen}(1^\lambda)$. Allocates memory for storing group aggregators' and sensors' public keys as $\mathsf{st}_{\mathsf{LAdm_i}}[\mathsf{AL}, \mathsf{SL}]$.

8. $\mathsf{LAdminAdd}\{\mathsf{LAdm_i}(\mathsf{pk}_{\mathsf{LAdm}}, \mathsf{sk}_{\mathsf{LAdm}}) \leftrightarrow$
   $\mathsf{MSP}(\mathsf{pk}_{\mathsf{MSP}}, \mathsf{sk}_{\mathsf{MSP}}, \mathsf{st}_{\mathsf{MSP}}[\mathsf{LL}_{\mathsf{MSP}}], \mathsf{BC})\}$ is an interactive protocol between a local group administrator $\mathsf{LAdm_i}$ and MSP in order to add $\mathsf{LAdm_i}$ in the system.

   (a) $\mathsf{LAdm_i}$ creates a physical identity proof π, then submits π and $\mathsf{pk}_{\mathsf{LAdm}}$ to MSP.

   (b) MSP verifies π. Then it parses $(\mathsf{LL}_{\mathsf{BC}})$ from the output of $\mathsf{ReadConfig}(\mathsf{BC})$. Check that $(\mathsf{pk}_{\mathsf{LAdm_i}} \notin \mathsf{LL}_{\mathsf{BC}}) \wedge (\mathsf{pk}_{\mathsf{LAdm_i}} \notin \mathsf{LL}_{\mathsf{MSP}})$. If all verifications hold, add $\mathsf{pk}_{\mathsf{LAdm_i}}$ to $\mathsf{LL}_{\mathsf{MSP}}$ in $\mathsf{st}_{\mathsf{MSP}}$ and return "1" to $\mathsf{LAdm_i}$, else return "0" with an error code.

9. $\mathsf{AggrSetup}\{\mathsf{LAdm_i}(\mathsf{pk}_{\mathsf{LAdm_i}}, \mathsf{sk}_{\mathsf{LAdm_i}}, \mathsf{st}_{\mathsf{LAdm_i}}) \leftrightarrow \mathsf{Ag}_{\mathsf{ij}}()\}$ is an interactive protocol between an $\mathsf{LAdm_i}$ and an aggregator $\mathsf{Ag}_{\mathsf{ij}}$ wishing to join group $\mathsf{G_i}$.

(a) $\mathsf{Ag_{ij}}$ computes signing keys as $(\mathsf{pk_{A_{ij}}}, \mathsf{sk_{A_{ij}}}) \leftarrow \mathsf{SignGen}(1^\lambda)$ and initializes pending and write transaction sets $\mathsf{pset_i} \rightarrow \emptyset, \mathsf{txset_i} \rightarrow \emptyset$ in its $\mathsf{st_{Ag_{ij}}}$.

(b) $\mathsf{Ag_{ij}}$ creates a physical identity proof π, then submits π and $\mathsf{pk_{A_{ij}}}$ to $\mathsf{LAdm_i}$.

(c) $\mathsf{LAdm_i}$ verifies $(\mathsf{pk_{A_{ij}}} \notin \mathsf{AL})$ and π. If these verifications hold, it invokes $\mathsf{AggrAdd}(\mathsf{pk_{A_{ij}}})$ with MSP. If MSP outputs "1", it adds $\mathsf{pk_{A_{ij}}}$ to AL, sends an updated copy of AL to all $\mathsf{Ag_{ij}} \in \mathsf{AL}$ and $\mathsf{pk_{LAdm_i}}$ to $\mathsf{Ag_{ij}}$. In all other cases it returns "0".

(d) $\mathsf{Ag_{ij}}$ copies $\mathsf{pk_{LAdm_i}}$ in its memory in $\mathsf{st_{Ag_{ij}}}$.

10. $\mathsf{AggrAdd}\{\mathsf{LAdm_i}(\mathsf{pk_{LAdm_i}}, \mathsf{sk_{LAdm_i}}, \mathsf{pk_{A_{ij}}}) \leftrightarrow \mathsf{MSP}(\mathsf{pk_{MSP}}, \mathsf{sk_{MSP}}, \mathsf{st_{MSP}}[\mathsf{PL_{MSP}}], \mathsf{BC})\}$ is an interactive protocol between a local administrator $\mathsf{LAdm_i}$ wishing to add an aggregator to the system and MSP.

(a) $\mathsf{LAdm_i}$ computes $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk_{LAdm_i}}, \mathsf{pk_{A_{ij}}})$. Send σ to MSP.

(b) MSP computes $\mathsf{SVrfy}(\mathsf{pk_{LAdm_i}}, \mathsf{pk_{A_{ij}}}, \sigma) := b$. Checks that $(\mathsf{pk_{LAdm_i}} \in \mathsf{LL_{MSP}}) \wedge b \wedge (\mathsf{pk_{A_{ij}}} \notin \mathsf{PL_{MSP}}) == 1$. If the verification holds, it parses $(\mathsf{PL_{BC}})$ from the output of $\mathsf{ReadConfig}(\mathsf{BC})$. Check that $(\mathsf{pk_{A_{ij}}} \notin \mathsf{PL_{BC}})$. If the verification holds, add $\mathsf{pk_{A_{ij}}}$ to $\mathsf{PL_{MSP}}$ and returns "1" to $\mathsf{LAdm_i}$. It returns "0" in all other cases.

11. $\mathsf{AggrUpd}\{\mathsf{LAdm_i}(\mathsf{sk_{LAdm}}, \mathsf{pk_S}) \leftrightarrow \mathsf{Ag_{ij}}(\mathsf{st_{Ag_{ij}}}[\mathsf{CL}])\}$ is an interactive protocol between $\mathsf{LAdm_i}$ and an aggregator $\mathsf{Ag_{ij}}$, both belonging to Group i. It is used when $\mathsf{LAdm_i}$ wants to add a sensor public key $\mathsf{pk_S}$ to $\mathsf{Ag_{ij}}$ and update its sensor list CL.

(a) $\mathsf{LAdm_i}$ computes $\sigma := \mathsf{Sign}(\mathsf{sk_{LAdm_i}}, \mathsf{pk_S})$. Send σ to $\mathsf{Ag_{ij}}$.

(b) $\mathsf{Ag_{ij}}$ computes $\mathsf{SVrfy}(\mathsf{pk_{LAdm_i}}, \mathsf{pk_S}, \sigma) := b$. Checks that $(\mathsf{pk_S} \notin \mathsf{st_{Ag_{ij}}}) \wedge b == 1$. If the verification holds, it adds $\mathsf{pk_S}$ to CL[4] and returns "1" to $\mathsf{LAdm_i}$. It returns "0" in all other cases.

[4] $\mathsf{LAdm_i}$ should run the protocol with every aggregator in the group, however we present this with one aggregator for simplicity.

12. $\mathsf{SensorJoin}\{\mathsf{LAdm_i}(\mathsf{pk_{LAdm_i}}, \mathsf{sk_{LAdm_i}}, \mathsf{st_{LAdm_i}}[\mathsf{SL}]) \leftrightarrow \mathsf{S_{ij}}(n)\}$ is an interactive protocol between $\mathsf{LAdm_i}$ of Group i and a sensor $\mathsf{S_{ij}}$ wishing to join the system.

    (a) $\mathsf{S_{ij}}$ using the one-time signature scheme described in Section 6.6.2:

        i. Samples $k \leftarrow (1^\lambda)$. and stores it in $\mathsf{st_{S_{ij}}}$.

        ii. Runs $(\mathsf{pk_{S_{ij}}}, \mathsf{sk_{S_{ij}}}, \ell = 1) \leftarrow \mathsf{OTKeyGen}(1^\lambda, n)$[5]

        iii. Stores $\ell = 1$ to $\mathsf{st_{S_{ij}}}$ where ℓ denotes the current "index" in the hash chain.

        iv. Creates a physical identity proof π

        v. Sends $(\pi, \mathsf{pk_{S_{ij}}})$ to $\mathsf{LAdm_i}$

    (b) $\mathsf{LAdm_i}$ checks $\mathsf{Vrfy}(\pi) \wedge (\mathsf{pk_{S_{ij}}} \notin \mathsf{SL})$

    $\wedge \mathsf{AggrUpd}(\mathsf{sk_{LAdm_i}}, \mathsf{pk_{S_{ij}}}) == 1 \; \forall \mathsf{Ag_{ij}} \in \mathsf{G_i}$. $\mathsf{LAdm_i}$ adds $\mathsf{pk_{S_{ij}}}$ to SL, else it outputs "0".

13. $\mathsf{SensorSendData}\{\mathsf{S_{ij}}(\mathsf{pk_{S_{ij}}}, \mathsf{sk_{S_{ij}}}, m, \mathsf{st_{S_{ij}}}) \leftrightarrow$

    $\mathcal{AG}_x(\mathsf{st_{Ag}}[\mathsf{CL}, \mathsf{pset}, \mathsf{txset}])\}$ is a protocol between sensor $\mathsf{S_{ij}}$ broadcasting data and a subset of aggregators $\mathcal{AG}_x \subseteq \{\mathcal{AG}_i\}$ (where $\{\mathcal{AG}_i\}$ is the aggregator set in $\mathsf{G_i}$).

    (a) For sending data m, $\mathsf{S_{ij}}$ computes $(\sigma, \mathsf{sk_S}, \mathsf{st_{S_{ij}}}') \leftarrow \mathsf{OTSign}(\mathsf{sk_S}, m, \mathsf{st_{S_{ij}}})$

    (b) $\mathsf{S_{ij}}$ broadcasts σ to $\mathcal{AG}_x$.

    (c) $\mathsf{Ag_k}$ runs $\mathsf{OTVerify}(\mathsf{pk_{S_{ij}}}, m, \sigma) := b$.[6] If $b == 1$ it runs $\mathsf{AggrAgree}$ with all other aggregators in the group. If no "alarm" message m^A from some other aggregator is received within some time δ, it adds m, σ to $\mathsf{pset_i}$. If at least one "alarm" message is received, it outputs $\perp$.

14. $\mathsf{AggrAgree}\{\mathsf{Ag_k}(\mathsf{sk_{A_k}}, \mathsf{AL}, \mathsf{pk_{S_{ij}}}, m, \sigma) \leftrightarrow \mathcal{AG}([\mathsf{sk_{A_i}}, \mathsf{pk_{S_{ij}}}, m'])\}$ is a protocol between an aggregator in $\mathsf{G_i}$ and all other aggregators in the group. The purpose of this protocol is

[5] This step is typically computed by a powerful device.

[6] To avoid redundancy, the protocol can be improved by deterministically defining a "responsible" aggregator for each transaction as discussed previously in this section.

to detect any MITM attacks, and verifies that no aggregator in the group has received any message m', σ' from $\mathsf{pk}_{\mathsf{S}_{ij}}$ where $m \neq m'$[7].

(a) $\mathsf{Ag_k}$ for payload $\mu = (\mathsf{pk}_{\mathsf{S}_{ij}}, m, \sigma)$ computes $s = \mathsf{Sign}(\mathsf{sk}_{\mathsf{A}_k}, \mu)$ using an EU-CMA signature scheme and sends s, μ to all $\mathsf{Ag_i} \in \mathcal{AG}$.

(b) Each $\mathsf{Ag_i}$ checks if it received a message m' with signature σ from sensor $\mathsf{pk}_{\mathsf{S}_{ij}}$ where $m \neq m'$. If there's no such message, it outputs $\perp$. Else it sends an "alarm" message m^A and respective signature s to $\mathsf{Ag_k}$ and keeps a record in its log.

15. $\mathsf{AggrSendTx}\{\mathcal{AG}([\mathsf{pk}_{\mathsf{A}_i}, \mathsf{sk}_{\mathsf{A}_i}, \mathsf{st}_{\mathsf{Ag_i}}, \mathsf{BC}]) \leftrightarrow \mathcal{O}(\mathsf{pk}_{\mathsf{O}_j}, \mathsf{sk}_{\mathsf{O}_j}, \mathsf{st_j})\}$ is an interactive protocol between all aggregators $\mathsf{Ag_i} \in \mathcal{AG}$ and all orderers $\mathsf{O}_j \in \mathcal{O}$. It is initiated when an aggregator wishes to submit a transaction for validation in the system and eventually store it in the blockchain.

(a) An $\mathsf{Ag_i} \in \mathcal{AG}$ parses $\mathsf{pset_i}$ in $\mathsf{st}_{\mathsf{Ag_i}}$ as a set of transactions $\{\mathsf{tx}\}$.

(b) $\mathsf{Ag_i}$ samples a nonce $n \leftarrow (1^\lambda)$ and appends it to $\{\mathsf{tx}\}$.

(c) $\mathsf{Ag_i}$ computes $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_{\mathsf{A}_i}, \{\mathsf{tx}\})$. Send σ to all other $\mathsf{Ag_j} \in \mathcal{AG}_x$.

(d) Each $\mathsf{Ag_j}$, parses $(\mathsf{PL}_{\mathsf{BC}})$ from the output of $\mathsf{ReadConfig}(\mathsf{BC})$. Computes $\mathsf{SVrfy}(\mathsf{pk}_{\mathsf{A}_i}, \{\mathsf{tx}\}, \sigma) := b$. If $(\mathsf{pk}_{\mathsf{A}_i} \in \mathsf{PL}_{\mathsf{BC}}) \wedge b == 1$, compute
$\sigma_j \leftarrow \mathsf{Sign}(\mathsf{sk}_{\mathsf{A}_j}, \{\mathsf{tx}\})$. Send σ_j to $\mathsf{Ag_i}$.

(e) $\mathsf{Ag_i}$ parses $\mathsf{ReadConfig}(\mathsf{BC}) \rightarrow \mathsf{Pol}_{\mathsf{BC}} \rightarrow \tau$ where τ the minimum required number of signatures for a transaction to be submitted on the blockchain, as defined by policy Pol.

(f) If $|\{\sigma_j\}| > \tau$, select a reachable orderer O, send $\{\sigma_j\}$, copy $\{\mathsf{tx}\} \rightarrow \mathsf{txset}$ and set $\mathsf{pset} \rightarrow \emptyset$.

(g) The orderer O parses $\mathsf{st_j} \rightarrow TXL$,
$\{\mathsf{tx}\} \rightarrow n$, $\mathsf{ReadConfig}(\mathsf{BC}) \rightarrow \mathsf{PL}_{\mathsf{BC}}, \mathsf{Pol}_{\mathsf{BC}}$ and $\mathsf{Pol}_{\mathsf{BC}} \rightarrow \tau$ then checks:

[7]This protocol does not require that all other aggregators in the group are reachable, therefore it does not require a reply from all aggregators to complete.

i. $|\{\sigma_j\}| > \tau$ and $n \notin TXL$

ii. Compute $\mathsf{SVrfy}(\mathsf{pk}_{\mathsf{A_j}}, \{\mathsf{tx}\}, \sigma_j) := b, \forall j$ then

$\prod b_j == 1$

iii. $\prod_j (\{\mathsf{pk}_{\mathsf{A_j}}\} \in \mathsf{PL}_{\mathsf{BC}}) == 1$

If the checks are valid, stores $|\{\sigma_j\}|$ in its $\mathsf{st}_{\mathsf{O_i}}$ and replies "1" to $\mathsf{Ag_i}$ as a confirmation, else it replies "0".

(h) If O_i has created a new block containing ordered transactions, it runs Consensus to update the blockchain.

(i) If Consensus succeeds, it runs UpdateBC with all aggregators to update to the new BC'.

16. $\mathsf{Consensus}([[\mathsf{pk}_{\mathsf{O_j}}]_{j=1}^{n}, \mathsf{sk}_{\mathsf{O_i}}, \mathsf{st_i}, \mathsf{BC}]_{i=1}^{n}) := \mathsf{BC}'$

The exact protocol functionality is described in the system parameters pp[8] and follows the definition provided in Section 2.2. In general this protocol is executed among all orderers $\mathsf{O}_i \in \mathcal{O}$ where they agree on a new updated blockchain BC'. In Section 6.9 we provide a concrete instantiation of a consensus algorithm for our construction.

17. $\mathsf{UpdateBC}\{\mathsf{O}_i(\mathsf{pk}_{\mathsf{O_i}}, \mathsf{sk}_{\mathsf{O_i}}, \mathsf{st}_{\mathsf{O_i}}, \mathsf{BC}') \leftrightarrow$

$\mathcal{AG}([\mathsf{pk}_{\mathsf{A_x}}, \mathsf{sk}_{\mathsf{A_x}}, \mathsf{st}_{\mathsf{Ag_x}}, \mathsf{BC}])\}$ is initiated by an orderer O_i to append a new block in the blockchain.

(a) O_i parses its $\mathsf{st}_{\mathsf{O_i}}$ to retrieve the agreed blockchain update signature set $\{\sigma'_x\}$

(b) O_i computes $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_{\mathsf{O_i}}, (\mathsf{B_i}, \{\sigma'_x\}))$ where $\mathsf{BC}' := \mathsf{BC}||\mathsf{B_i}$ and sends σ to all $\mathsf{Ag_x} \in \mathcal{AG}$.

(c) Each $\mathsf{Ag_x}$ computes $\mathsf{SVrfy}(\mathsf{pk}_{\mathsf{O_i}}, \mathsf{B_i}||\{\sigma'_x\}, \sigma) := b$ and checks if $b == 1$. Then it parses σ as a transaction set $\{\mathsf{tx}\}$ and removes these from $\mathsf{txset_x}$. Then it updates BC to BC', else it outputs $\perp$.

[8] This is equivalent to Hyperledger's "pluggable" consensus, which is defined in the genesis block.

18. $\mathsf{NodeRevoke}(\mathsf{pk_i}, \sigma, \mathsf{st_{MSP}}, \mathsf{BC})$ is initiated by MSP to revoke credentials of any system participant.

    MSP parses $\mathsf{ReadConfig}(\mathsf{BC})$ as $[\mathsf{LL_{BC}}, \mathsf{OL_{BC}}, \mathsf{PL_{BC}}]$. It verifies σ (if the remove operation was initiated by a LAdm) and checks if $\mathsf{pk_i}$ exists in $[\mathsf{LL_{BC}}, \mathsf{OL_{BC}}, \mathsf{PL_{BC}}]$ or in its $[\mathsf{LL_{MSP}}, \mathsf{OL_{MSP}}, \mathsf{PL_{MSP}}] \in \mathsf{st_{MSP}}$ in case participation privileges for $\mathsf{pk_i}$ have not yet been updated on the BC through $\mathsf{ConfigUpdate}$. If it finds a match in the blockchain lists, it creates a remove operation $R := (\mathsf{pk_i}, \text{“}rm\text{”})$ and adds R to oper, else if it finds a match in its state lists it removes it from the respective list, else it outputs $\perp$. If $\mathsf{pk_i} \in \mathsf{PL_{BC}} \vee \mathsf{pk_i} \in \mathsf{PL_{MSP}}$, it also informs $\mathsf{LAdm_i}$.

19. $\mathsf{GroupRevoke}(\mathsf{pk_i}, \mathsf{st_{LAdm}}[\mathsf{AL}, \mathsf{SL}])$ is initiated by a Local Administrator to revoke credentials of an aggregator or sensor in its group. LAdm checks if $\mathsf{pk_i} \in [\mathsf{AL}, \mathsf{SL}]$. If it finds a match and $\mathsf{pk_i} \in \mathsf{AL}$, it $\mathsf{LAdm_i}$ computes $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk_{LAdm_i}}, \mathsf{pk_{A_i}}, \text{“}R\text{”})$. Sends $(\sigma, \mathsf{pk_{A_i}}, \text{“}R\text{”})$ to MSP. On receiving successful removal from MSP (after it invokes $\mathsf{NodeRevoke}$), LAdm removes $\mathsf{pk_i}$ from AL. If $\mathsf{pk_i} \in \mathsf{SL}$, it invokes $\mathsf{AggRevokeSensor}$ with all $\mathsf{Ag} \in \mathsf{AL}$. After successful completion, it removes $\mathsf{pk_i}$ from SL.

20. $\mathsf{AggRevokeSensor}\{\mathsf{LAdm_i}(\mathsf{sk_{LAdm}}, \mathsf{pk_S}) \leftrightarrow \mathsf{Ag_{ij}}(\mathsf{st_{Ag_{ij}}}[\mathsf{CL}])\}$ is initiated by a Local Administrator as a subroutine of $\mathsf{GroupRevoke}$ to revoke credentials of a sensor in its group.

    (a) $\mathsf{LAdm_i}$ computes $\sigma := \mathsf{Sign}(\mathsf{sk_{LAdm_i}}, \mathsf{pk_S}\ \text{“}R\text{”})$. Send σ to $\mathsf{Ag_{ij}}$.

    (b) $\mathsf{Ag_{ij}}$ computes $\mathsf{SVrfy}(\mathsf{pk_{LAdm_i}}, \mathsf{pk_S}, \sigma) := b$. Checks that $(\mathsf{pk_S} \notin \mathsf{st_{Ag_{ij}}}) \wedge b == 1$. If the verification holds, it removes $\mathsf{pk_S}$ from CL and returns “1” to $\mathsf{LAdm_i}$. It returns “0” in all other cases.

6.11 Evaluation details

Algorithms 1 and 2 show the pseudocode for our evaluations on the sensor and aggregator side respectively. We denote by T1 the signature computation time, by T2 the verification

time and by T3 the total verification time, as previously shown in Table 6.5.

Algorithm 1 Sensor send data

```
1: tempkey ← k_0
2: initPebbles()
3: while True do
4:    m ← readSensor()
5:    output.type ← "payload"
6:    output.data ← m
7:    transmit(output)
8:    T1.start()
9:    hashedData ← h(m——tempkey)
10:   output.type ← "hash"
11:   output.data ← hashedData
12:   transmit(output)
13:   tempkey ← computePebbles() {as in [182]}
14:   output.type ← "secretKey"
15:   T1.end()
16:   output.data ← tempkey
17:   transmit(output)
18: end while
```

6.12 Conclusions

We designed and implemented BBox-IoT, a block-chain inspired approach for Industrial IoT sensors aiming at offering a transparent and immutable system for sensing and control information exchanged between IIoT sensors and aggregators. Our approach guarantees blockchain-derived properties to even low-Size Weight and Power (SWaP) devices. Moreover, BBox-IoT acts as a "black-box" that empowers the operators of any IoT system to detect data and sensor tampering ferreting out attacks against even SWaP devices. We posit that enabling data auditing and security at the lowest sensing level will be highly beneficial to critical infrastructure environments with sensors from multiple vendors.

Finally, we envision that our approach will be implemented during the sensor manufacturing stage: having industrial sensors shipped with pre-computed pebbles and their key material labeled using QR-code on the sensor body will allow for a seamless and practical deployment of BBox-IoT.

Algorithm 2 Aggregator receive data

```
1: publickey ← k_0
2: verifications ← 0
3: while verifications < maxVerifications do
4:    check1 ← False
5:    check2 ← False
6:    read ← input()
7:    if read.type = "payload" then
8:       T3.start()
9:       m ← read.data
10:   else if read.type = "hash" then
11:      s_1 ← read.data
12:   else if read.type = "secretKey" then
13:      s_2 ← read.data
14:      tempkey ← s_2
15:      i ← 0
16:      T2.start()
17:      while i < maxVerification ∧ doWhile = True do
18:         if h(tempkey) = publickey then
19:            check1 ← True
20:            if h(m——publickey) = s_1 then
21:               check2 ← True
22:               publickey ← secretkey
23:            end if
24:            doWhile = False
25:         else
26:            tempkey ← h(tempkey)
27:            i++
28:         end if
29:      end while
30:      if check1 ∧ check2 = True then
31:         print("Payload m is valid")
32:         verifications++
33:         T2.end()
34:      else
35:         print("Verification failed")
36:      end if
37:      T3.end()
38:   end if
39: end while
```

Chapter 7: Conclusions and Future Work

In this thesis, we considered the problem of incorporating auditability-related protocols in blockchain-based payment systems that can facilitate the needed regulatory functionalities, without deviating from the original system's core characteristics. To that end, we first presented MiniLedger as a solution at a microscopic level, which is a distributed payment system offering both strong privacy and auditability protocols for its participants, while also remaining efficient in terms of storage costs for the public ledger. Then at a macroscopic level, we focused on Proof of Assets protocols, which are needed as a part of a solvency proof, and presented a framework tailored for the Diem cryptocurrency, as well as gOTzilla, an efficient protocol that enables to prove assets of large organizations in large-scale permissionless systems such as Bitcoin. In addition, we considered the problem of auditing data collected from resource-constrained devices in a blockchain-based Industrial IoT setting.

As discussed in our recent paper [63], there are still a number of research gaps in this field. For instance, proving solvency of an organization on top of a distributed payment system with privacy-preserving characteristics remains an open problem, while considering "off-chain" approaches that have becoming increasingly popular due to the scalability properties offered in blockchain systems is also challenging, as their main ledger footprint is minimal by nature, making the design of such protocols particularly challenging. We believe this thesis will inspire such future work needed towards a greater adoption of blockchain-based payment systems.

Bibliography

Bibliography

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2009. [Online]. Available: http://bitcoin.org/bitcoin.pdf

[2] E. Androulaki, G. Karame, M. Roeschlin, T. Scherer, and S. Capkun, "Evaluating user privacy in Bitcoin," in *FC 2013*, ser. LNCS, A.-R. Sadeghi, Ed., vol. 7859. Springer, Heidelberg, Apr. 2013, pp. 34–51.

[3] S. Meiklejohn, M. Pomarole, G. Jordan, K. Levchenko, D. McCoy, G. M. Voelker, and S. Savage, "A fistful of bitcoins: characterizing payments among men with no names," *Commun. ACM*, vol. 59, no. 4, pp. 86–93, 2016. [Online]. Available: https://doi.org/10.1145/2896384

[4] D. Deuber, V. Ronge, and C. Rückert, "Sok: Assumptions underlying cryptocurrency deanonymizations," *Proc. Priv. Enhancing Technol.*, vol. 2022, no. 3, pp. 670–691, 2022. [Online]. Available: https://doi.org/10.56553/popets-2022-0091

[5] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in *2014 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2014, pp. 459–474.

[6] N. Van Saberhagen, "Cryptonote v 2.0," 2013. [Online]. Available: https://cryptonote.org/whitepaper.pdf

[7] P. Fauzi, S. Meiklejohn, R. Mercer, and C. Orlandi, "Quisquis: A new design for anonymous cryptocurrencies," in *ASIACRYPT 2019, Part I*, ser. LNCS, S. D. Galbraith and S. Moriai, Eds., vol. 11921. Springer, Heidelberg, Dec. 2019, pp. 649–678.

[8] E. Cecchetti, F. Zhang, Y. Ji, A. E. Kosba, A. Juels, and E. Shi, "Solidus: Confidential distributed ledger transactions via PVORM," in *ACM CCS 2017*, B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, Eds. ACM Press, Oct. / Nov. 2017, pp. 701–717.

[9] G. Maxwell, "Coinjoin: Bitcoin privacy for the real world," 2013. [Online]. Available: https://bitcointalk.org/index.php?topic=279249.0

[10] E. Heilman, L. Alshenibr, F. Baldimtsi, A. Scafuro, and S. Goldberg, "TumbleBit: An untrusted bitcoin-compatible anonymous payment hub," in *NDSS 2017*. The Internet Society, Feb. / Mar. 2017.

[11] "IRS is trying to deanonymize privacy coins like monero and zcash." [Online]. Available: https://www.forbes.com/sites/shehanchandrasekera/2020/07/06/irs-is-trying-to-deanonymize-privacy-coins-like-monero-and-zcash/#4607506c4174

[12] "Deloitte COINIA and the future of audit." [Online]. Available: https://www2.deloitte.com/us/en/pages/audit/articles/impact-of-blockchain-in-accounting.html

[13] "Deloitte's 2020 global blockchain survey," 2020. [Online]. Available: https://www2.deloitte.com/content/dam/insights/us/articles/6608_2020-global-blockchain-survey/DI_CIR%202020%20global%20blockchain%20survey.pdf

[14] "FATF travel rule: What you need to know." [Online]. Available: https://complyadvantage.com/knowledgebase/fatf-travel-rule/

[15] "Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation)," *Official Journal of the European Union L119*, pp. 1–88, 2016.

[16] S. Brands, "Untraceable off-line cash in wallets with observers (extended abstract)," in *CRYPTO'93*, ser. LNCS, D. R. Stinson, Ed., vol. 773. Springer, Heidelberg, Aug. 1994, pp. 302–318.

[17] G. G. Dagher, B. Bünz, J. Bonneau, J. Clark, and D. Boneh, "Provisions: Privacy-preserving proofs of solvency for bitcoin exchanges," in *ACM CCS 2015*, I. Ray, N. Li, and C. Kruegel, Eds. ACM Press, Oct. 2015, pp. 720–731.

[18] C. Garman, M. Green, and I. Miers, "Accountable privacy for decentralized anonymous payments," in *FC 2016*, ser. LNCS, J. Grossklags and B. Preneel, Eds., vol. 9603. Springer, Heidelberg, Feb. 2016, pp. 81–98.

[19] N. Narula, W. Vasquez, and M. Virza, "zkledger: Privacy-preserving auditing for distributed ledgers," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. Renton, WA: USENIX Association, Apr. 2018, pp. 65–80. [Online]. Available: https://www.usenix.org/conference/nsdi18/presentation/narula

[20] K. Wüst, K. Kostiainen, V. Capkun, and S. Capkun, "PRCash: Fast, private and regulated transactions for digital currencies," in *FC 2019*, ser. LNCS, I. Goldberg and T. Moore, Eds., vol. 11598. Springer, Heidelberg, Feb. 2019, pp. 158–178.

[21] "Tether: Fiat currencies on the bitcoin blockchain." [Online]. Available: https://tether.to/wp-content/uploads/2016/06/TetherWhitePaper.pdf

[22] T. Moore and N. Christin, "Beware the middleman: Empirical analysis of Bitcoin-exchange risk," in *FC 2013*, ser. LNCS, A.-R. Sadeghi, Ed., vol. 7859. Springer, Heidelberg, Apr. 2013, pp. 25–33.

[23] "Tether's bank says it invests customer funds in bitcoin." [Online]. Available: https://www.coindesk.com/tethers-bank-says-it-invests-customer-funds-in-bitcoin

[24] "CSBS state regulatory requirements for virtual currency activities." [Online]. Available: https://www.csbs.org/sites/default/files/2017-11/CSBS%20Draft%20Model%20Regulatory%20Framework%20for%20Virtual%20Currency%20Proposal%20--%20Dec.%2016%202014.pdf

[25] K. Chalkias, K. Lewi, P. Mohassel, and V. Nikolaenko, "Distributed auditing proofs of liabilities," Cryptology ePrint Archive, Report 2020/468, 2020, https://eprint.iacr.org/2020/468.

[26] P. Chatzigiannis, F. Baldimtsi, and K. Chalkias, "Sok: Auditability and accountability in distributed payment systems," pp. 311–337, 2021. [Online]. Available: https://doi.org/10.1007/978-3-030-78375-4_13

[27] Y. Ji and K. Chalkias, "Generalized proof of liabilities," Cryptology ePrint Archive, Report 2021/1350, 2021, https://ia.cr/2021/1350.

[28] P. Chatzigiannis and K. Chalkias, "Proof of assets in the diem blockchain," in *Applied Cryptography and Network Security Workshops - ACNS 2021 Satellite Workshops, AIBlock, AIHWS, AIoTS, CIMSS, Cloud S&P, SCI, SecMT, and SiMLA, Kamakura, Japan, June 21-24, 2021, Proceedings*, ser. Lecture Notes in Computer Science, J. Zhou, C. M. Ahmed, L. Batina, S. Chattopadhyay, O. Gadyatskaya, C. Jin, J. Lin, E. Losiouk, B. Luo, S. Majumdar, M. Maniatakos, D. Mashima, W. Meng, S. Picek, M. Shimaoka, C. Su, and C. Wang, Eds., vol. 12809. Springer, 2021, pp. 27–41. [Online]. Available: https://doi.org/10.1007/978-3-030-81645-2_3

[29] "The libra blockchain," 2020. [Online]. Available: https://developers.libra.org/docs/assets/papers/the-libra-blockchain/2020-05-26.pdf

[30] F. Baldimtsi, P. Chatzigiannis, S. D. Gordon, P. H. Le, and D. McVicker, "gotzilla: Efficient disjunctive zero-knowledge proofs from mpc in the head, with application to proofs of assets in cryptocurrencies," Cryptology ePrint Archive, Paper 2022/170, 2022, https://eprint.iacr.org/2022/170. [Online]. Available: https://eprint.iacr.org/2022/170

[31] P. Chatzigiannis, F. Baldimtsi, C. Kolias, and A. Stavrou, "Black-box iot: Authentication and distributed storage of iot data from constrained sensors," in *IoTDI '21: International Conference on Internet-of-Things Design and Implementation, Virtual Event / Charlottesville, VA, USA, May 18-21, 2021*. ACM, 2021, pp. 1–14. [Online]. Available: https://doi.org/10.1145/3450268.3453536

[32] J. Katz and Y. Lindell, *Introduction to Modern Cryptography, Second Edition*, 2nd ed. Chapman & Hall/CRC, 2014.

[33] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," in *CRYPTO'84*, ser. LNCS, G. R. Blakley and D. Chaum, Eds., vol. 196. Springer, Heidelberg, Aug. 1984, pp. 10–18.

[34] O. Ugus, A. Hessler, and D. Westhoff, "Performance of additive homomorphic ec-elgamal encryption for tinypeds," *6. Fachgespräch Sensornetzwerke*, 2007.

[35] Y. Chen, X. Ma, C. Tang, and M. H. Au, "PGC: Decentralized confidential payment system with auditability," in *ESORICS 2020, Part I*, ser. LNCS, L. Chen, N. Li, K. Liang, and S. A. Schneider, Eds., vol. 12308. Springer, Heidelberg, Sep. 2020, pp. 591–610.

[36] S. Goldwasser, S. Micali, and R. L. Rivest, "A digital signature scheme secure against adaptive chosen-message attacks," *SIAM Journal on Computing*, vol. 17, no. 2, pp. 281–308, Apr. 1988.

[37] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *CRYPTO'91*, ser. LNCS, J. Feigenbaum, Ed., vol. 576. Springer, Heidelberg, Aug. 1992, pp. 129–140.

[38] J. Camenisch and M. Stadler, "Efficient group signature schemes for large groups (extended abstract)," in *CRYPTO'97*, ser. LNCS, B. S. Kaliski Jr., Ed., vol. 1294. Springer, Heidelberg, Aug. 1997, pp. 410–424.

[39] G. Maxwell and A. Poelstra, "Borromean ring signatures," 2015. [Online]. Available: https://github.com/Blockstream/borromean_paper/blob/master/borromean_draft_0.01_34241bb.pdf

[40] A. Poelstra, A. Back, M. Friedenbach, G. Maxwell, and P. Wuille, "Confidential assets," in *FC 2018 Workshops*, ser. LNCS, A. Zohar, I. Eyal, V. Teague, J. Clark, A. Bracciali, F. Pintore, and M. Sala, Eds., vol. 10958. Springer, Heidelberg, Mar. 2019, pp. 43–63.

[41] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for confidential transactions and more," in *2018 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2018, pp. 315–334.

[42] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *CRYPTO'86*, ser. LNCS, A. M. Odlyzko, Ed., vol. 263. Springer, Heidelberg, Aug. 1987, pp. 186–194.

[43] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: Nearly practical verifiable computation," in *2013 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2013, pp. 238–252.

[44] J. Groth, "On the size of pairing-based non-interactive arguments," in *EUROCRYPT 2016, Part II*, ser. LNCS, M. Fischlin and J.-S. Coron, Eds., vol. 9666. Springer, Heidelberg, May 2016, pp. 305–326.

[45] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Scalable zero knowledge with no trusted setup," in *CRYPTO 2019, Part III*, ser. LNCS, A. Boldyreva and D. Micciancio, Eds., vol. 11694. Springer, Heidelberg, Aug. 2019, pp. 701–732.

[46] F. Baldimtsi, J. Camenisch, M. Dubovitskaya, A. Lysyanskaya, L. Reyzin, K. Samelin, and S. Yakoubov, "Accumulators with applications to anonymity-preserving revocation," in *2017 IEEE European Symposium on Security and Privacy, Paris, France, April 26-28, 2017*. IEEE, 2017, pp. 301–315.

[47] D. Boneh, B. Bünz, and B. Fisch, "Batching techniques for accumulators with applications to IOPs and stateless blockchains," in *CRYPTO 2019, Part I*, ser. LNCS, A. Boldyreva and D. Micciancio, Eds., vol. 11692. Springer, Heidelberg, Aug. 2019, pp. 561–586.

[48] R. C. Merkle, "A digital signature based on a conventional encryption function," in *CRYPTO'87*, ser. LNCS, C. Pomerance, Ed., vol. 293. Springer, Heidelberg, Aug. 1988, pp. 369–378.

[49] M. Chase, C. Ganesh, and P. Mohassel, "Efficient zero-knowledge proof of algebraic and non-algebraic statements with applications to privacy preserving credentials," in *CRYPTO 2016, Part III*, ser. LNCS, M. Robshaw and J. Katz, Eds., vol. 9816. Springer, Heidelberg, Aug. 2016, pp. 499–530.

[50] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai, "Zero-knowledge from secure multiparty computation," in *39th ACM STOC*, D. S. Johnson and U. Feige, Eds. ACM Press, Jun. 2007, pp. 21–30.

[51] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Trans. Program. Lang. Syst.*, 1982. [Online]. Available: http://doi.acm.org/10.1145/357172.357176

[52] J. A. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," in *EUROCRYPT 2015, Part II*, ser. LNCS, E. Oswald and M. Fischlin, Eds., vol. 9057. Springer, Heidelberg, Apr. 2015, pp. 281–310.

[53] J. A. Garay and A. Kiayias, "SoK: A consensus taxonomy in the blockchain era," in *CT-RSA 2020*, ser. LNCS, S. Jarecki, Ed., vol. 12006. Springer, Heidelberg, Feb. 2020, pp. 284–318.

[54] Y. Lindell, A. Lysyanskaya, and T. Rabin, "On the composition of authenticated byzantine agreement," Cryptology ePrint Archive, Report 2004/181, 2004, https://eprint.iacr.org/2004/181.

[55] M. Castro and B. Liskov, "Practical byzantine fault tolerance," in *OSDI '99*. Berkeley, CA, USA: USENIX Association, 1999. [Online]. Available: http://dl.acm.org/citation.cfm?id=296806.296824

[56] M. Vukolic, "The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication," in *IFIP WG 11.4 International Workshop, iNetSec 2015*, ser. Lecture Notes in Computer Science, J. Camenisch and D. Kesdogan, Eds., vol. 9591. Springer, 2015, pp. 112–125. [Online]. Available: https://doi.org/10.1007/978-3-319-39028-4_9

[57] C. Cachin and M. Vukolic, "Blockchain consensus protocols in the wild," *CoRR*, vol. abs/1707.01873, 2017. [Online]. Available: http://arxiv.org/abs/1707.01873

[58] S. Bano, A. Sonnino, M. Al-Bassam, S. Azouvi, P. McCorry, S. Meiklejohn, and G. Danezis, "Consensus in the age of blockchains," *CoRR*, vol. abs/1711.03936, 2017. [Online]. Available: http://arxiv.org/abs/1711.03936

[59] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," http://bitcoin.org/bitcoin.pdf," 2008.

[60] N. Stifter, A. Judmayer, P. Schindler, A. Zamyatin, and E. Weippl, "Agreement with satoshi — on the formalization of nakamoto consensus," Cryptology ePrint Archive, Report 2018/400, 2018, https://eprint.iacr.org/2018/400.

[61] H. Chen, M. Pendleton, L. Njilla, and S. Xu, "A survey on ethereum systems security: Vulnerabilities, attacks and defenses," *CoRR*, vol. abs/1908.04507, 2019. [Online]. Available: http://arxiv.org/abs/1908.04507

[62] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse attacks on bitcoin's peer-to-peer network," in *USENIX Security 2015*, J. Jung and T. Holz, Eds. USENIX Association, Aug. 2015, pp. 129–144.

[63] P. Chatzigiannis, F. Baldimtsi, and K. Chalkias, "Sok: Auditability and accountability in distributed payment systems," in *Applied Cryptography and Network Security - 19th International Conference, ACNS 2021, Kamakura, Japan, June 21-24, 2021, Proceedings, Part II*, ser. Lecture Notes in Computer Science, K. Sako and N. O. Tippenhauer, Eds., vol. 12727. Springer, 2021, pp. 311–337. [Online]. Available: https://doi.org/10.1007/978-3-030-78375-4_13

[64] B. Bünz, S. Agrawal, M. Zamani, and D. Boneh, "Zether: Towards privacy in a smart contract world," in *FC 2020*, ser. LNCS, J. Bonneau and N. Heninger, Eds., vol. 12059. Springer, Heidelberg, Feb. 2020, pp. 423–443.

[65] G. Fuchsbauer, M. Orrù, and Y. Seurin, "Aggregate cash systems: A cryptographic investigation of Mimblewimble," in *EUROCRYPT 2019, Part I*, ser. LNCS, Y. Ishai and V. Rijmen, Eds., vol. 11476. Springer, Heidelberg, May 2019, pp. 657–689.

[66] G. Maxwell, "Confidential transactions," 2015. [Online]. Available: https://people.xiph.org/~greg/confidential_values.txt

[67] Y. Li, G. Yang, W. Susilo, Y. Yu, M. H. Au, and D. Liu, "Traceable monero: Anonymous cryptocurrency with enhanced accountability," *IEEE Transactions on Dependable and Secure Computing*, 2019.

[68] Y. Jiang, Y. Li, and Y. Zhu, "Auditable zerocoin scheme with user awareness," in *Proceedings of the 3rd International Conference on Cryptography, Security and Privacy, Kuala Lumpur, Malaysia, January 19-21, 2019*, 2019, pp. 28–32.

[69] K. Naganuma, M. Yoshino, H. Sato, and T. Suzuki, "Auditable zerocoin," in *2017 IEEE European Symposium on Security and Privacy Workshops*, 2017, pp. 59–63.

[70] P. Chatzigiannis and F. Baldimtsi, "Miniledger: Compact-sized anonymous and auditable distributed payments," Cryptology ePrint Archive, Report 2021/869, 2021, https://ia.cr/2021/869.

[71] A. Centelles and G. Dijkstra, "Extending zkledger with private swaps." [Online]. Available: https://cdn2.hubspot.net/hubfs/6034488/privateledger.pdf

[72] J. Doerner, A. Shelat, and D. Evans, "Zeroledge: Proving solvency with privacy."

[73] K. Chalkias, K. Lewi, P. Mohassel, and V. Nikolaenko, "Distributed auditing proofs of liabilities," Cryptology ePrint Archive, Report 2020/468, 2020, https://eprint.iacr.org/2020/468.

[74] J. Frankle, S. Park, D. Shaar, S. Goldwasser, and D. J. Weitzner, "AUDIT: Practical accountability of secret processes," Cryptology ePrint Archive, Report 2018/697, 2018, https://eprint.iacr.org/2018/697.

[75] S. Goldwasser and S. Park, "Public accountability vs. secret laws: Can they coexist?" Cryptology ePrint Archive, Report 2018/664, 2018, https://eprint.iacr.org/2018/664.

[76] G. Wood, "Ethereum: A secure decentralized generalised transaction ledger," 2021, accessed: 2021-02-14. [Online]. Available: https://ethereum.github.io/yellowpaper/paper.pdf

[77] J. Bonneau, I. Meckler, V. Rao, and E. Shapiro, "Coda: Decentralized cryptocurrency at scale," Cryptology ePrint Archive, Report 2020/352, 2020, https://eprint.iacr.org/2020/352.

[78] A. Chepurnoy, C. Papamanthou, and Y. Zhang, "Edrax: A cryptocurrency with stateless transaction validation," Cryptology ePrint Archive, Report 2018/968, 2018, https://eprint.iacr.org/2018/968.

[79] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Secure distributed key generation for discrete-log based cryptosystems," in *EUROCRYPT'99*, ser. LNCS, J. Stern, Ed., vol. 1592. Springer, Heidelberg, May 1999, pp. 295–310.

[80] G. Danezis and S. Meiklejohn, "Centrally banked cryptocurrencies," in *NDSS 2016*. The Internet Society, Feb. 2016.

[81] E. Androulaki, J. Camenisch, A. De Caro, M. Dubovitskaya, K. Elkhiyaoui, and B. Tackmann, "Privacy-preserving auditable token payments in a permissioned blockchain system," Cryptology ePrint Archive, Report 2019/1058, 2019, https://eprint.iacr.org/2019/1058.

[82] W. Lueks, B. Kulynych, J. Fasquelle, S. L. Bail-Collet, and C. Troncoso, "zksk: A library for composable zero-knowledge proofs," in *Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society*, 2019, pp. 50–54.

[83] B. Schoenmakers, "Interval proofs revisited," in *Workshop on Frontiers in Electronic Elections*, 2005.

[84] D. Catalano and D. Fiore, "Vector commitments and their applications," in *PKC 2013*, ser. LNCS, K. Kurosawa and G. Hanaoka, Eds., vol. 7778. Springer, Heidelberg, Feb. / Mar. 2013, pp. 55–72.

[85] R. W. F. Lai and G. Malavolta, "Subvector commitments with application to succinct arguments," in *CRYPTO 2019, Part I*, ser. LNCS, A. Boldyreva and D. Micciancio, Eds., vol. 11692. Springer, Heidelberg, Aug. 2019, pp. 530–560.

[86] National Institute of Standards and Technology, *Recommendation for Key Management: NIST SP 800-57 Part 1 Rev 4*, USA, 2016.

[87] "Apache kafka," https://kafka.apache.org/.

[88] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. D. Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolic, S. W. Cocco, and J. Yellick, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proceedings of the Thirteenth EuroSys Conference, EuroSys 2018, Porto, Portugal, April 23-26, 2018*, 2018, pp. 30:1–30:15.

[89] "Libra roles and permissions," 2020. [Online]. Available: https://lip.libra.org/lip-2/

[90] J. Zhang, J. Gao, Z. Wu, W. Yan, Q. Wu, Q. Li, and Z. Chen, "Performance analysis of the libra blockchain: An experimental study," *CoRR*, vol. abs/1912.05241, 2019. [Online]. Available: http://arxiv.org/abs/1912.05241

[91] R. Cramer, I. Damgård, and B. Schoenmakers, "Proofs of partial knowledge and simplified design of witness hiding protocols," in *CRYPTO'94*, ser. LNCS, Y. Desmedt, Ed., vol. 839. Springer, Heidelberg, Aug. 1994, pp. 174–187.

[92] V. Shoup and R. Gennaro, "Securing threshold cryptosystems against chosen ciphertext attack," *Journal of Cryptology*, vol. 15, no. 2, pp. 75–96, Mar. 2002.

[93] "libpgc: a c++ library for pretty good confidential transaction system," https://github.com/yuchen1024/libPGC/tree/master/PGC_openssl/PGC.

[94] D. Shanks, "Five number-theoretic algorithms," 1973.

[95] D. J. Bernstein and T. Lange, "Computing small discrete logarithms faster," in *INDOCRYPT 2012*, ser. LNCS, S. D. Galbraith and M. Nandi, Eds., vol. 7668. Springer, Heidelberg, Dec. 2012, pp. 317–338.

[96] V. Mavroudis, "Computing small discrete logarithms using optimized lookup tables," 2015, USCB, Koç Lab.

[97] C. Decker and R. Wattenhofer, "Bitcoin transaction malleability and MtGox," in *ESORICS 2014, Part II*, ser. LNCS, M. Kutylowski and J. Vaidya, Eds., vol. 8713. Springer, Heidelberg, Sep. 2014, pp. 313–326.

[98] "Bitstamp proof of reserves." [Online]. Available: https://www.bitstamp.net/s/documents/Bitstamp_proof_of_reserves_statement.pdf

[99] D. Hopwood, S. Bowe, T. Hornby, and N. Wilcox, "Zcash protocol specification," *GitHub: San Francisco, CA, USA*, 2016.

[100] S. Agrawal, C. Ganesh, and P. Mohassel, "Non-interactive zero-knowledge proofs for composite statements," in *CRYPTO 2018, Part III*, ser. LNCS, H. Shacham and A. Boldyreva, Eds., vol. 10993. Springer, Heidelberg, Aug. 2018, pp. 643–673.

[101] A. Gabizon, K. Gurkan, P. Jovanovic, G. Konstantopoulos, A. Oines, M. Olszewski, M. Straka, and E. Tromer, "Plumo: Towards scalable interoperable blockchains using ultra light validation systems," *ZKProof*, 2020.

[102] A. Dutta and S. Vijayakumaran, "Mprove: A proof of reserves protocol for monero exchanges," in *2019 IEEE European Symposium on Security and Privacy Workshops, EuroS&P Workshops 2019, Stockholm, Sweden, June 17-19, 2019*. IEEE, 2019, pp. 330–339. [Online]. Available: https://doi.org/10.1109/EuroSPW.2019.00043

[103] H. Wang, D. He, and Y. Ji, "Designated-verifier proof of assets for bitcoin exchange using elliptic curve cryptography," *Future Gener. Comput. Syst.*, vol. 107, pp. 854–862, 2020. [Online]. Available: https://doi.org/10.1016/j.future.2017.06.028

[104] S. Roose, "Standardizing bitcoin proof of reserves." [Online]. Available: https://blockstream.com/2019/02/04/en-standardizing-bitcoin-proof-of-reserves/

[105] "Kraken proof of reserves." [Online]. Available: https://www.kraken.com/en-us/proof-of-reserves-audit

[106] "Ethereum wiki," 2019. [Online]. Available: https://web.archive.org/web/20190613115908if_/https://github.com/ethereum/wiki/wiki/JSON-RPC#eth_sign

[107] "Proof of solvency: Technical overview." [Online]. Available: https://medium.com/iconominet/proof-of-solvency-technical-overview-d1d0e8a8a0b8

[108] M. Baudet, A. Ching, A. Chursin, G. Danezis, F. Garillot, Z. Li, D. Malkhi, O. Naor, D. Perelman, and A. Sonnino, "State machine replication in the libra blockchain," *The Libra Assn., Tech. Rep*, 2019.

[109] S. Josefsson and I. Liusvaara, "RFC 8032: Edwards-Curve Digital Signature Algorithm (EdDSA)," Jan 2017.

[110] D. Khovratovich and J. Law, "Bip32-ed25519: Hierarchical deterministic keys over a non-linear keyspace," in *2017 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, 2017, pp. 27–31.

[111] K. Chalkias, K. Lewi, P. Mohassel, and V. Nikolaenko, "Distributed auditing proofs of liabilities," Cryptology ePrint Archive, Report 2020/468, 2020, https://eprint.iacr.org/2020/468.

[112] E. Syta, P. Jovanovic, E. Kokoris-Kogias, N. Gailly, L. Gasser, I. Khoffi, M. J. Fischer, and B. Ford, "Scalable bias-resistant distributed randomness," in *2017 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2017, pp. 444–460.

[113] H. Gjermundrød, K. Chalkias, and I. Dionysiou, "Going beyond the coinbase transaction fee: Alternative reward schemes for miners in blockchain systems," in *Proceedings of the 20th Pan-Hellenic Conference on Informatics*, 2016, pp. 1–4.

[114] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch, "Verifiable delay functions," in *Annual international cryptology conference*. Springer, 2018, pp. 757–788.

[115] "Provisions: Privacy-preserving proofs of solvency for bitcoin exchanges," Real World Crypto 2016. [Online]. Available: https://rwc.iacr.org/2016/Slides/Provisions%20talk%20RWC.pdf

[116] Z. Gao, Y. Hu, and Q. Wu, "Jellyfish merkle tree," 2021, https://developers.diem.com/papers/jellyfish-merkle-tree/2021-01-14.pdf.

[117] "Diem storage module," 2021, https://github.com/diem/diem/tree/master/storage.

[118] "Diem data structures specification," 2021, https://github.com/diem/diem/blob/main/specifications/common/data_structures.md.

[119] "Diem authenticated data structure specification," 2021, https://github.com/diem/diem/blob/main/specifications/common/authenticated_data_structures.md.

[120] "Diem proofs of assets," 2021. [Online]. Available: https://github.com/diem/diem/blob/main/client/assets-proof/README.md

[121] K. Chalkias, J. Brown, M. Hearn, T. Lillehagen, I. Nitto, and T. Schroeter, "Blockchained post-quantum signatures," in *2018 IEEE Blockchain*. IEEE, 2018, pp. 1196–1203.

[122] J. Nick, T. Ruffing, and Y. Seurin, "Musig2: Simple two-round schnorr multi-signatures," Cryptology ePrint Archive, Report 2020/1261, 2020, https://eprint.iacr.org/2020/1261.

[123] C. Komlo and I. Goldberg, "Frost: Flexible round-optimized schnorr threshold signatures," Cryptology ePrint Archive, Report 2020/852, 2020, https://eprint.iacr.org/2020/852.

[124] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the Weil pairing," in *ASIACRYPT 2001*, ser. LNCS, C. Boyd, Ed., vol. 2248. Springer, Heidelberg, Dec. 2001, pp. 514–532.

[125] K. Chalkias, F. Garillot, Y. Kondi, and V. Nikolaenko, "Non-interactive half-aggregation of eddsa and variants of schnorr signatures," *CT-RSA*, 2021.

[126] R. Gennaro, D. Leigh, R. Sundaram, and W. Yerazunis, "Batching schnorr identification scheme with applications to privacy-preserving authorization and low-bandwidth communication devices," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2004, pp. 276–292.

[127] S. Azouvi, G. Danezis, and V. Nikolaenko, "Winkle: Foiling long-range attacks in proof-of-stake systems," in *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, 2020, pp. 189–201.

[128] F. Breedon, L. Chen, A. Ranaldo, and N. Vause, "Judgement day: Algorithmic trading around the swiss franc cap removal," 2018.

[129] "Ethereum wiki (archive.org)." [Online]. Available: https://web.archive.org/web/20190613115908if_/https://github.com/ethereum/wiki/wiki/JSON-RPC#eth_sign

[130] S. Blackshear, B. Wilsion, and T. Zakian, "Diem improvement proposal 11," 2021, https://dip.diem.com/dip-11/.

[131] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof-systems (extended abstract)," in *17th ACM STOC*. ACM Press, May 1985, pp. 291–304.

[132] O. Goldreich, S. Micali, and A. Wigderson, "How to prove all NP-statements in zero-knowledge, and a methodology of cryptographic protocol design," in *CRYPTO'86*, ser. LNCS, A. M. Odlyzko, Ed., vol. 263. Springer, Heidelberg, Aug. 1987, pp. 171–185.

[133] M. Bellare and S. Goldwasser, "New paradigms for digital signatures and message authentication based on non-interactive zero knowledge proofs," in *CRYPTO'89*, ser. LNCS, G. Brassard, Ed., vol. 435. Springer, Heidelberg, Aug. 1990, pp. 194–211.

[134] J. Camenisch and A. Lysyanskaya, "An identity escrow scheme with appointed verifiers," in *CRYPTO 2001*, ser. LNCS, J. Kilian, Ed., vol. 2139. Springer, Heidelberg, Aug. 2001, pp. 388–407.

[135] M. Backes, L. Hanzlik, A. Herzberg, A. Kate, and I. Pryvalov, "Efficient non-interactive zero-knowledge proofs in cross-domains without trusted setup," in *PKC 2019, Part I*, ser. LNCS, D. Lin and K. Sako, Eds., vol. 11442. Springer, Heidelberg, Apr. 2019, pp. 286–313.

[136] I. Giacomelli, J. Madsen, and C. Orlandi, "ZKBoo: Faster zero-knowledge for Boolean circuits," in *USENIX Security 2016*, T. Holz and S. Savage, Eds. USENIX Association, Aug. 2016, pp. 1069–1083.

[137] "Sealpir: A computational pir library that achieves low communication costs and high performance." [Online]. Available: https://github.com/microsoft/SealPIR

[138] C. Baum, A. J. Malozemoff, M. B. Rosen, and P. Scholl, "Mac'n'cheese: Zero-knowledge proofs for boolean and arithmetic circuits with nested disjunctions," in *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part IV*, ser. Lecture Notes in Computer Science, T. Malkin and C. Peikert, Eds., vol. 12828. Springer, 2021, pp. 92–122. [Online]. Available: https://doi.org/10.1007/978-3-030-84259-8_4

[139] C.-P. Schnorr, "Efficient identification and signatures for smart cards," in *CRYPTO'89*, ser. LNCS, G. Brassard, Ed., vol. 435. Springer, Heidelberg, Aug. 1990, pp. 239–252.

[140] L. C. Guillou and J.-J. Quisquater, "A practical zero-knowledge protocol fitted to security microprocessor minimizing both trasmission and memory," in *EUROCRYPT'88*, ser. LNCS, C. G. Günther, Ed., vol. 330. Springer, Heidelberg, May 1988, pp. 123–128.

[141] M. Jawurek, F. Kerschbaum, and C. Orlandi, "Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently," in *ACM CCS 2013*, A.-R. Sadeghi, V. D. Gligor, and M. Yung, Eds. ACM Press, Nov. 2013, pp. 955–966.

[142] S. Ames, C. Hazay, Y. Ishai, and M. Venkitasubramaniam, "Ligero: Lightweight sublinear arguments without a trusted setup," in *ACM CCS 2017*, B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, Eds. ACM Press, Oct. / Nov. 2017, pp. 2087–2104.

[143] J. Katz, V. Kolesnikov, and X. Wang, "Improved non-interactive zero knowledge with applications to post-quantum signatures," in *ACM CCS 2018*, D. Lie, M. Mannan, M. Backes, and X. Wang, Eds. ACM Press, Oct. 2018, pp. 525–537.

[144] S. Setty, "Spartan: Efficient and general-purpose zkSNARKs without trusted setup," in *CRYPTO 2020, Part III*, ser. LNCS, D. Micciancio and T. Ristenpart, Eds., vol. 12172. Springer, Heidelberg, Aug. 2020, pp. 704–737.

[145] C. Weng, K. Yang, J. Katz, and X. Wang, "Wolverine: Fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits," in *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*. IEEE, 2021, pp. 1074–1091. [Online]. Available: https://doi.org/10.1109/SP40001.2021.00056

[146] D. Heath and V. Kolesnikov, "Stacked garbling for disjunctive zero-knowledge proofs," in *EUROCRYPT 2020, Part III*, ser. LNCS, A. Canteaut and Y. Ishai, Eds., vol. 12107. Springer, Heidelberg, May 2020, pp. 569–598.

[147] A. Goel, M. Green, M. Hall-Andersen, and G. Kaptchuk, "Stacking sigmas: A framework to compose σ-protocols for disjunctions," Cryptology ePrint Archive, Report 2021/422, 2021, https://ia.cr/2021/422.

[148] ——, "Efficient set membership proofs using mpc-in-the-head," Cryptology ePrint Archive, Report 2021/1656, 2021, https://ia.cr/2021/1656.

[149] M. Naor and B. Pinkas, "Oblivious transfer and polynomial evaluation," in *31st ACM STOC*. ACM Press, May 1999, pp. 245–254.

[150] G. Di Crescenzo, T. Malkin, and R. Ostrovsky, "Single database private information retrieval implies oblivious transfer," in *EUROCRYPT 2000*, ser. LNCS, B. Preneel, Ed., vol. 1807. Springer, Heidelberg, May 2000, pp. 122–138.

[151] B. Zhang, H. Lipmaa, C. Wang, and K. Ren, "Practical fully simulatable oblivious transfer with sublinear communication," in *FC 2013*, ser. LNCS, A.-R. Sadeghi, Ed., vol. 7859. Springer, Heidelberg, Apr. 2013, pp. 78–95.

[152] E. Kushilevitz and R. Ostrovsky, "Replication is NOT needed: SINGLE database, computationally-private information retrieval," in *38th FOCS*. IEEE Computer Society Press, Oct. 1997, pp. 364–373.

[153] S. Angel, H. Chen, K. Laine, and S. T. V. Setty, "PIR with compressed queries and amortized query processing," in *2018 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2018, pp. 962–979.

[154] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical GapSVP," in *CRYPTO 2012*, ser. LNCS, R. Safavi-Naini and R. Canetti, Eds., vol. 7417. Springer, Heidelberg, Aug. 2012, pp. 868–886.

[155] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," Cryptology ePrint Archive, Report 2012/144, 2012, https://eprint.iacr.org/2012/144.

[156] H. Chen, M. Kim, I. Razenshteyn, D. Rotaru, Y. Song, and S. Wagh, "Maliciously secure matrix multiplication with applications to private deep learning," Cryptology ePrint Archive, Report 2020/451, 2020, https://ia.cr/2020/451.

[157] "Multi-protocol spdz." [Online]. Available: https://github.com/data61/MP-SPDZ

[158] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," in *ITCS 2012*, S. Goldwasser, Ed. ACM, Jan. 2012, pp. 309–325.

[159] C. Baum, D. Cozzo, and N. P. Smart, "Using TopGear in overdrive: A more efficient ZKPoK for SPDZ," in *SAC 2019*, ser. LNCS, K. G. Paterson and D. Stebila, Eds., vol. 11959. Springer, Heidelberg, Aug. 2019, pp. 274–302.

[160] C. D. de Saint Guilhem, E. Orsini, and T. Tanguy, "Limbo: Efficient zero-knowledge mpcith-based arguments," in *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, Y. Kim, J. Kim, G. Vigna, and E. Shi, Eds. ACM, 2021, pp. 3022–3036. [Online]. Available: https://doi.org/10.1145/3460120.3484595

[161] "Efficient multi-party computation toolkit," 2022. [Online]. Available: https://github.com/emp-toolkit/emp-ot#iknp-style-protocols

[162] "Blockchain.com unspent transaction outputs," 2022. [Online]. Available: https://www.blockchain.com/charts/utxo-count

[163] J. Bell, T. D. LaToza, F. Baldimtsi, and A. Stavrou, "Advancing open science with version control and blockchains," in *12th IEEE/ACM International Workshop on Software Engineering for Science, SE4Science@ICSE 2017, Buenos Aires, Argentina, May 22, 2017*. IEEE, 2017, pp. 13–14. [Online]. Available: https://doi.org/10.1109/SE4Science.2017.11

[164] C. Profentzas, M. Almgren, and O. Landsiedel, "Iotlogblock: Recording off-line transactions of low-power iot devices using a blockchain," in *44th IEEE Conference on Local Computer Networks, LCN 2019, Osnabrueck, Germany, October 14-17, 2019*, K. Andersson, H. Tan, and S. Oteafy, Eds. IEEE, 2019, pp. 414–421. [Online]. Available: https://doi.org/10.1109/LCN44214.2019.8990728

[165] H. Shafagh, L. Burkhalter, A. Hithnawi, and S. Duquennoy, "Towards blockchain-based auditable storage and sharing of iot data," in *Proceedings of the 2017 on Cloud Computing Security Workshop*, ser. CCSW '17. New York, NY, USA: ACM, 2017, pp. 45–50. [Online]. Available: http://doi.acm.org/10.1145/3140649.3140656

[166] H. Zou, Y. Zhou, J. Yang, and C. J. Spanos, "Towards occupant activity driven smart buildings via wifi-enabled iot devices and deep learning," *Energy and Buildings*, vol. 177, pp. 12–22, 2018.

[167] H. Derhamy, J. Eliasson, J. Delsing, and P. Priller, "A survey of commercial frameworks for the internet of things," in *ETFA 2015*. IEEE, 2015.

[168] "Cisco Annual Internet Report ," 2020. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

[169] J. Bugeja, P. Davidsson, and A. Jacobsson, "Functional classification and quantitative analysis of smart connected home devices," in *2018 Global Internet of Things Summit, GIoTS 2018, Bilbao, Spain, June 4-7, 2018*. IEEE, 2018, pp. 1–6. [Online]. Available: https://doi.org/10.1109/GIOTS.2018.8534563

[170] L. Lin, X. Liao, H. Jin, and P. Li, "Computation offloading toward edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1584–1607, 2019.

[171] J. Liu, Y. Xiao, and J. Gao, "Achieving accountability in smart grid," *IEEE Systems Journal*, vol. 8, no. 2, pp. 493–508, 2014. [Online]. Available: https://doi.org/10.1109/JSYST.2013.2260697

[172] P. Tenti, H. K. M. Paredes, and P. Mattavelli, "Conservative power theory, a framework to approach control and accountability issues in smart microgrids," *IEEE Transactions on Power Electronics*, vol. 26, no. 3, pp. 664–673, March 2011.

[173] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. B. Calo, "Analyzing federated learning through an adversarial lens," in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 2019, pp. 634–643. [Online]. Available: http://proceedings.mlr.press/v97/bhagoji19a.html

[174] T. Gu, K. Liu, B. Dolan-Gavitt, and S. Garg, "Badnets: Evaluating backdooring attacks on deep neural networks," *IEEE Access*, vol. 7, pp. 47 230–47 244, 2019.

[175] K. Boyer, L. Brubaker, K. Everly, R. Herriman, P. Houston, S. Ruckle, R. Scobie, and I. Ulanday, "A distributed sensor network for an off-road racing vehicle." International Foundation for Telemetering, 2017.

[176] F. Karray, M. W. Jmal, A. G. Ortiz, M. Abid, and A. M. Obeid, "A comprehensive survey on wireless sensor node hardware platforms," vol. 144, 2018, pp. 89–110. [Online]. Available: https://doi.org/10.1016/j.comnet.2018.05.010

[177] C. Bormann, M. Ersue, and A. Keranen, "Terminology for constrained-node networks," Internet Requests for Comments, RFC Editor, RFC 7228, May 2014.

[178] A. Perrig, R. Canetti, J. D. Tygar, and D. Song, "Efficient authentication and signing of multicast streams over lossy channels," in *IEEE S & P 2000*, 2000, pp. 56–73.

[179] ——, "The tesla broadcast authentication protocol," 2002.

[180] F. Tirado-Andrés, A. Rozas, and Á. Araujo, "A methodology for choosing time synchronization strategies for wireless iot networks," vol. 19, no. 16, 2019, p. 3476. [Online]. Available: https://doi.org/10.3390/s19163476

[181] A. Elsts, X. Fafoutis, S. Duquennoy, G. Oikonomou, R. J. Piechocki, and I. Craddock, "Temperature-resilient time synchronization for the internet of things," *IEEE Trans. Industrial Informatics*, vol. 14, no. 5, pp. 2241–2250, 2018. [Online]. Available: https://doi.org/10.1109/TII.2017.2778746

[182] M. Jakobsson, "Fractal hash sequence representation and traversal," in *Information Theory, 2002.* IEEE, 2002, p. 437.

[183] D. H. Yum, J. W. Seo, S. Eom, and P. J. Lee, "Single-layer fractal hash chain traversal with almost optimal complexity," in *CT-RSA 2009*, ser. LNCS, M. Fischlin, Ed., vol. 5473. Springer, Heidelberg, Apr. 2009, pp. 325–339.

[184] M. S. Turan, K. A. McKay, Ç. Çalık, D. Chang, and L. Bassham, "Status report on the first round of the nist lightweight cryptography standardization process," 2019.

[185] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. D. Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolic, S. W. Cocco, and J. Yellick, "Hyperledger fabric: A distributed operating system for permissioned blockchains," *CoRR*, vol. abs/1801.10228, 2018. [Online]. Available: http://arxiv.org/abs/1801.10228

[186] N. Chondros, K. Kokordelis, and M. Roussopoulos, "On the practicality of practical byzantine fault tolerance," in *ACM/IFIP/USENIX 13th International Middleware Conference*, ser. Lecture Notes in Computer Science, P. Narasimhan and P. Triantafillou, Eds., vol. 7662. Springer, 2012, pp. 436–455. [Online]. Available: https://doi.org/10.1007/978-3-642-35170-9_22

[187] R. Rodrigues, B. Liskov, K. Chen, M. Liskov, and D. Schultz, "Automatic reconfiguration for large-scale reliable storage systems," *IEEE Trans. Dependable Sec. Comput.*, vol. 9, no. 2, pp. 145–158, 2012. [Online]. Available: https://doi.org/10.1109/TDSC.2010.52

[188] Hyperledger, "Hyperledger architecture volumes 1 and 2," Jun 2018. [Online]. Available: https://www.hyperledger.org/wp-content/uploads/2017/08/Hyperledger_Arch_WG_Paper_1_Consensus.pdf

[189] ——, "Hyperledger fabric documentation: Consensus algorithm, release 0.6," Jun 2018. [Online]. Available: https://fabricdocs.readthedocs.io/en/origin-v0.6/FAQ/consensus_FAQ.html

[190] D. Ongaro and J. K. Ousterhout, "In search of an understandable consensus algorithm," in *2014 USENIX Annual Technical Conference, USENIX ATC '14, Philadelphia, PA, USA, June 19-20, 2014.*, 2014, pp. 305–319. [Online]. Available: https://www.usenix.org/conference/atc14/technical-sessions/presentation/ongaro

[191] A. N. Bessani, J. Sousa, and E. A. P. Alchieri, "State machine replication for the masses with BFT-SMART," in *44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2014, Atlanta, GA, USA, June 23-26, 2014*. IEEE Computer Society, 2014, pp. 355–362. [Online]. Available: https://doi.org/10.1109/DSN.2014.43

[192] J. Sousa, A. Bessani, and M. Vukolic, "A byzantine fault-tolerant ordering service for the hyperledger fabric blockchain platform," in *DSN 2018*. IEEE Computer Society, 2018, pp. 51–58. [Online]. Available: https://doi.org/10.1109/DSN.2018.00018

[193] C. Gorenflo, S. Lee, L. Golab, and S. Keshav, "Fastfabric: Scaling hyperledger fabric to 20, 000 transactions per second," *CoRR*, vol. abs/1901.00910, 2019. [Online]. Available: http://arxiv.org/abs/1901.00910

[194] M. Bowman and C. Morhardt, "Blockchain must adapt to build trust in the internet of things," May 2018, retrieved June 24, 2018. [Online]. Available: https://www.coindesk.com/blockchain-must-adapt-build-trust-internet-things/

[195] A. Dorri, S. S. Kanhere, and R. Jurdak, "Blockchain in internet of things: Challenges and solutions," *CoRR*, vol. abs/1608.05187, 2016. [Online]. Available: http://arxiv.org/abs/1608.05187

[196] O. Garcia-Morchon, R. Rietman, S. Sharma, L. Tolhuizen, and J. Torre-Arce, "A comprehensive and lightweight security architecture to secure the iot throughout the lifecycle of a device based on himmo," in *ALGOSENSORS 2015*. Berlin, Heidelberg: Springer-Verlag, 2015, pp. 112–128. [Online]. Available: https://doi.org/10.1007/978-3-319-28472-9_9

[197] K. Wüst and A. Gervais, "Do you need a blockchain?" Cryptology ePrint Archive, Report 2017/375, 2017, https://eprint.iacr.org/2017/375.

[198] L. Lamport, "Password authentication with insecure communication," *Commun. ACM*, vol. 24, no. 11, pp. 770–772, Nov. 1981. [Online]. Available: http://doi.acm.org/10.1145/358790.358797

[199] D. Liu and P. Ning, "Efficient distribution of key chain commitments for broadcast authentication in distributed sensor networks," in *NDSS 2003*. The Internet Society, Feb. 2003.

[200] Y.-C. Hu, M. Jakobsson, and A. Perrig, "Efficient constructions for one-way hash chains," in *ACNS 05*, ser. LNCS, J. Ioannidis, A. Keromytis, and M. Yung, Eds., vol. 3531. Springer, Heidelberg, Jun. 2005, pp. 423–441.

[201] E. Dahmen and C. Krauß, "Short hash-based signatures for wireless sensor networks," in *CANS 09*, ser. LNCS, J. A. Garay, A. Miyaji, and A. Otsuka, Eds., vol. 5888. Springer, Heidelberg, Dec. 2009, pp. 463–476.

[202] J. A. Buchmann, E. Dahmen, and A. Hülsing, "XMSS - A practical forward secure signature scheme based on minimal security assumptions," in *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011*, B.-Y. Yang, Ed. Springer, Heidelberg, Nov. / Dec. 2011, pp. 117–129.

[203] K. Chalkias, J. Brown, M. Hearn, T. Lillehagen, I. Nitto, and T. Schroeter, "Blockchained post-quantum signatures," Cryptology ePrint Archive, Report 2018/658, 2018, https://eprint.iacr.org/2018/658.

[204] D. J. Bernstein, D. Hopwood, A. Hülsing, T. Lange, R. Niederhagen, L. Papachristodoulou, M. Schneider, P. Schwabe, and Z. Wilcox-O'Hearn, "SPHINCS: Practical stateless hash-based signatures," in *EUROCRYPT 2015, Part I*, ser. LNCS, E. Oswald and M. Fischlin, Eds., vol. 9056. Springer, Heidelberg, Apr. 2015, pp. 368–397.

[205] A. Hülsing, J. Rijneveld, and P. Schwabe, "ARMed SPHINCS - computing a 41 KB signature in 16 KB of RAM," in *PKC 2016, Part I*, ser. LNCS, C.-M. Cheng, K.-M. Chung, G. Persiano, and B.-Y. Yang, Eds., vol. 9614. Springer, Heidelberg, Mar. 2016, pp. 446–470.

[206] F. Baldimtsi, J. Camenisch, M. Dubovitskaya, A. Lysyanskaya, L. Reyzin, K. Samelin, and S. Yakoubov, "Accumulators with applications to anonymity-preserving revocation," Cryptology ePrint Archive, Report 2017/043, 2017, https://eprint.iacr.org/2017/043.

[207] J. Camenisch and A. Lysyanskaya, "Dynamic accumulators and application to efficient revocation of anonymous credentials," in *CRYPTO 2002*, ser. LNCS, M. Yung, Ed., vol. 2442. Springer, Heidelberg, Aug. 2002, pp. 61–76.

[208] Y. Wu, H.-N. Dai, and H. Wang, "Convergence of blockchain and edge computing for secure and scalable iiot critical infrastructures in industry 4.0," *IEEE Internet of Things Journal*, 2020.

[209] "Arduino uno rev3," 2019. [Online]. Available: https://store.arduino.cc/usa/arduino-uno-rev3

[210] D. Amiet, A. Curiger, and P. Zbinden, "FPGA-based accelerator for SPHINCS-256," *IACR TCHES*, vol. 2018, no. 1, pp. 18–39, 2018, https://tches.iacr.org/index.php/TCHES/article/view/831.

[211] L. Bai, M. Hu, M. Liu, and J. Wang, "Bpiiot: A light-weighted blockchain-based platform for industrial iot," *IEEE Access*, vol. 7, pp. 58 381–58 393, 2019.

[212] O. Novo, "Blockchain meets iot: An architecture for scalable access management in iot," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 1184–1195, April 2018.

[213] J. Wan, J. Li, M. Imran, and D. Li, "A blockchain-based solution for enhancing security and privacy in smart factory," *IEEE Transactions on Industrial Informatics*, June 2019.

[214] R. AlTawy and G. Gong, "Mesh: A supply chain solution with locally private blockchain transactions," *PoPETs*, vol. 2019, no. 3, pp. 149–169, 2019. [Online]. Available: https://doi.org/10.2478/popets-2019-0041

[215] L. Wu, X. Du, W. Wang, and B. Lin, "An out-of-band authentication scheme for internet of things using blockchain technology," in *2018 International Conference on Computing, Networking and Communications (ICNC)*, March 2018, pp. 769–773.

[216] M. Samaniego and R. Deters, "Internet of smart things - iost: Using blockchain and clips to make things autonomous," in *IEEE ICCC 2017*, June 2017, pp. 9–16.

[217] D. Miller, "Blockchain and the internet of things in the industrial sector," *IT Professional*, vol. 20, no. 3, pp. 15–18, May 2018.

[218] Z. Yang, K. Yang, L. Lei, K. Zheng, and V. C. M. Leung, "Blockchain-based decentralized trust management in vehicular networks," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 1495–1505, April 2019.

[219] Z. Xiong, Y. Zhang, D. Niyato, P. Wang, and Z. Han, "When mobile blockchain meets edge computing," *IEEE Communications Magazine*, vol. 56, no. 8, pp. 33–39, August 2018.

[220] X. Liang, J. Zhao, S. Shetty, and D. Li, "Towards data assurance and resilience in iot using blockchain," in *MILCOM 2017 - 2017 IEEE Military Communications Conference (MILCOM)*, Oct 2017, pp. 261–266.

[221] N. Z. Aitzhan and D. Svetinovic, "Security and privacy in decentralized energy trading through multi-signatures, blockchain and anonymous messaging streams," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 5, pp. 840–852, Sep. 2018.

[222] G. Wang, Z. Shi, M. Nixon, and S. Han, "Chainsplitter: Towards blockchain-based industrial iot architecture for supporting hierarchical storage," in *IEEE International Conference on Blockchain, 2019*. IEEE, 2019, pp. 166–175. [Online]. Available: https://doi.org/10.1109/Blockchain.2019.00030

[223] M. Ali, M. Vecchio, M.Pincheira, K. Dolui, F. Antonelli, and M. Rehmani, "Applications of blockchains in the internet of things: A comprehensive survey," *IEEE Communications Surveys and Tutorials*, vol. 21, no. 2, pp. 1676–1717, 2019. [Online]. Available: https://doi.org/10.1109/COMST.2018.2886932

[224] M. A. Ferrag, M. Derdour, M. Mukherjee, A. Derhab, L. A. Maglaras, and H. Janicke, "Blockchain technologies for the internet of things: Research issues and challenges," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 2188–2204, 2019. [Online]. Available: https://doi.org/10.1109/JIOT.2018.2882794

[225] L. Lamport, "Constructing digital signatures from a one-way function," SRI International Computer Science Laboratory, Technical Report SRI-CSL-98, Oct. 1979.

[226] J. Buchmann, E. Dahmen, S. Ereth, A. Hülsing, and M. Rückert, "On the security of the winternitz one-time signature scheme," Cryptology ePrint Archive, Report 2011/191, 2011, https://eprint.iacr.org/2011/191.

[227] A. Hülsing, "W-OTS+ - shorter signatures for hash-based signature schemes," in *AFRICACRYPT 13*, ser. LNCS, A. Youssef, A. Nitaj, and A. E. Hassanien, Eds., vol. 7918. Springer, Heidelberg, Jun. 2013, pp. 173–188.

[228] L. Reyzin and N. Reyzin, "Better than BiBa: Short one-time signatures with fast signing and verifying," in *ACISP 02*, ser. LNCS, L. M. Batten and J. Seberry, Eds., vol. 2384. Springer, Heidelberg, Jul. 2002, pp. 144–153.

[229] D. J. Bernstein, A. Hülsing, S. Kölbl, R. Niederhagen, J. Rijneveld, and P. Schwabe, "The SPHINCS$^+$ signature framework," in *ACM CCS 2019*, L. Cavallaro, J. Kinder, X. Wang, and J. Katz, Eds. ACM Press, Nov. 2019, pp. 2129–2146.

[230] Q. Wang, H. Khurana, Y. Huang, and K. Nahrstedt, "Time valid one-time signature for time-critical multicast data authentication," in *INFOCOM 2009. 28th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 19-25 April 2009, Rio de Janeiro, Brazil*. IEEE, 2009, pp. 1233–1241. [Online]. Available: https://doi.org/10.1109/INFCOM.2009.5062037

[231] M. O. Ozmen and A. A. Yavuz, "Low-cost standard public key cryptography services for wireless iot systems," in *Proceedings of the 2017 Workshop on Internet of Things Security and Privacy, IoT S&P@CCS,*, 2017, pp. 65–70. [Online]. Available: https://doi.org/10.1145/3139937.3139940

[232] S. Kumar, Y. Hu, M. P. Andersen, R. A. Popa, and D. E. Culler, "JEDI: many-to-many end-to-end encryption and key delegation for iot," *CoRR*, vol. abs/1905.13369, 2019. [Online]. Available: http://arxiv.org/abs/1905.13369

[233] H. Li, G. Dán, and K. Nahrstedt, "Portunes: Privacy-preserving fast authentication for dynamic electric vehicle charging," in *2014 IEEE International Conference on Smart Grid Communications, SmartGridComm 2014, Venice, Italy, November 3-6, 2014*. IEEE, 2014, pp. 920–925. [Online]. Available: https://doi.org/10.1109/SmartGridComm.2014.7007766

[234] A. Wander, N. Gura, H. Eberle, V. Gupta, and S. C. Shantz, "Energy analysis of public-key cryptography for wireless sensor networks," in *3rd IEEE International Conference on Pervasive Computing and Communications (PerCom 2005), 8-12 March 2005, Kauai Island, HI, USA*. IEEE Computer Society, 2005, pp. 324–328. [Online]. Available: https://doi.org/10.1109/PERCOM.2005.18

[235] N. R. Potlapally, S. Ravi, A. Raghunathan, and N. K. Jha, "Analyzing the energy consumption of security protocols," in *Proceedings of the 2003 International Symposium on Low Power Electronics and Design, 2003, Seoul, Korea, August 25-27, 2003*, I. Verbauwhede and H. Roh, Eds. ACM, 2003, pp. 30–35. [Online]. Available: https://doi.org/10.1145/871506.871518

[236] M. El-hajj, A. Fadlallah, M. Chamoun, and A. Serhrouchni, "A survey of internet of things (iot) authentication schemes," *Sensors*, vol. 19, no. 5, p. 1141, 2019. [Online]. Available: https://doi.org/10.3390/s19051141

[237] O. O. Bamasag and K. Youcef-Toumi, "Towards continuous authentication in internet of things based on secret sharing scheme," in *Proceedings of the 10th Workshop on Embedded Systems Security, WESS 2015, Amsterdam, The Netherlands, October 8, 2015*, S. A. Koubias and T. Sauter, Eds. ACM, 2015, p. 1. [Online]. Available: https://doi.org/10.1145/2818362.2818363

[238] P. Kampanakis and S. Fluhrer, "LMS vs XMSS: A comparison of the stateful hash-based signature proposed standards," Cryptology ePrint Archive, Report 2017/349, 2017, https://eprint.iacr.org/2017/349.

[239] A. Hülsing, L. Rausch, and J. Buchmann, "Optimal parameters for XMSS^{MT}," Cryptology ePrint Archive, Report 2017/966, 2017, https://eprint.iacr.org/2017/966.

Curriculum Vitae

Panagiotis (Panos) Chatzigiannis received his Bachelor's engineering degree from Hellenic Naval Academy and his Masters of Science in Computer Science from US Naval Postrgraduate School. During his PhD program, he was a summer intern with Facebook/Novi in 2020 and 2021. Currently, he is working as a staff research scientist at Visa Research.