1. **Report Title:** **AOI-2, A Novel Access Control Blockchain Paradigm for Cybersecure Sensor Infrastructure in Fossil Power Generation Systems**

2. **Report Type:** Final Technical

3. **Reporting Period:** Start Date: 09/01/2019
End Date: 09/30/2022

4. **Federal Agency:** Department of Energy

5. **Principal Author(s):**

Rahul Panat (PI)
Russell V. Trader Associate Professor
Department of Mechanical Engineering
Carnegie Mellon University
Pittsburgh, PA
Tel: 412-268-2501; Fax: 412-268-3348
E-mail: rpanat@andrew.cmu.edu

Vipul Goyal (co-PI)
Associate Professor
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA
E-mail: vipul@cmu.edu

6. **Date of Report:** Jan 30, 2023

7. **DOE Award Number:** DE-FOA-0001991

8. **Name and Address of Submitting Organization:**

Carnegie Mellon University
Room 8113 Wean Hall,
5000 Forbes Avenue,
Pittsburgh, PA 15213

ABSTRACT

Fossil power generation systems are increasingly vulnerable to attack from both cybercriminals as well as internal threats. These vulnerabilities demand that emerging technologies such as blockchains be utilized to secure the data involved in the information flows within the Supervisory Control and Data Acquisition (SCADA) systems of the fossil power generation plants. The publicly accessible blockchain protocols, although secure, are visible to everyone. Even private blockchains currently are unable to support different levels of access to different participants, which is a critical requirement for the existing SCADA systems running the power plants. In light of the above, novel blockchain protocols that are specifically adapted to fossil power generation environments need to be developed in order to achieve the goal of cybersecure sensor networks.

In this work, we address this question by creating a novel blockchain technology, namely smart private ledger, for cybersecure communication within the fossil power generation systems. A lab-scale sensor network consisting of strain and temperature sensors is constructed to develop the ledger. The technology has hierarchical access control which is compatible with the existing SCADA systems in fossil power plants. The sensor data is used with cryptographic digital signatures and secret sharing protocols within the nodes of the blockchain technology. The research results will lead to cybersecurity for machine-to-machine interactions, infrastructure for secure data logging for sensors, decentralized data storage, and second-layer technologies for high volume machine-to-machine interactions in the power plants. The work aims to largely address the concerns for the security of distributed sensor networks in such systems that can be compromised by insider threats and by cybercriminals. The research has led to the training of the next generation of engineers and scientists in the important areas of sensor engineering and blockchain technology.

TABLE OF CONTENTS

Page

EXECUTIVE SUMMARY

Fossil power generation systems are increasingly vulnerable to attack from both cybercriminals as well as internal threats. These vulnerabilities demand that emerging technologies such as blockchains be utilized to secure the data involved in the information flows within the Supervisory Control and Data Acquisition (SCADA) systems of the fossil power generation plants. The publicly accessible blockchain protocols, although secure, are visible to everyone. Even private blockchains currently are unable to support different level of access to different participants, which is a critical requirement for the existing SCADA systems running the power plants. In light of the above, novel blockchain protocols that are specifically adapted to fossil power generation environments need to be developed in order to achieve the goal of cybersecure sensor networks. To address this important problem, we created a novel blockchain technology, namely smart private ledger, for cybersecure communication within the fossil power generation systems. The technology has hierarchical access control which is compatible with the existing SCADA systems in fossil power plants. The sensor data was used with cryptographic digital signatures and secret sharing protocols within the nodes of the blockchain technology. The research generates cybersecurity for machine-to-machine interactions, infrastructure for secure data logging for sensors, decentralized data storage, and second-layer technologies for high volume machine-to-machine interactions in the power plants. The work has addressed the concerns for the security of distributed sensor networks in such systems that can be compromised by insider threats and by cybercriminals. The project starts with a conceptual phase (TRL 2) and involves creating a sensor infrastructure and a prototype private blockchain structure; and also concludes with a laboratory scale test (TRL 3) such that it is ready for implementation in the field. The project also integrates research into education and prepares next generation of scientists and engineers as relevant to the mission and goals of the Department of Energy (DOE).

CHAPTER ONE

Introduction

1.1 Background and Research Need

Cyberattacks against power grids, gas pipelines, and other critical infrastructure are increasing in frequency and severity.[1-4] These threats come from both national and international cybercriminals as well as stakeholders operating within the networks (insider threats). In addition to damaging the physical infrastructure, such attacks can cause widespread disruption in essential services, leading to a detrimental economic and social impact and in some cases, the loss of human lives. These vulnerabilities demand the use of emerging technologies such as blockchains to secure the data involved in the information flows of the individual infrastructure nodes.

Blockchains are shared and distributed data structures or ledgers that can securely store data and digital transactions without using a central point of authority. In distributed systems, each node has the same capabilities and levels of access to complete information. Since the data is stored on all the nodes of the system, there is no single point of failure in the event of a cyber-attack. A blockchain instance can be public or private depending on who is granted access to the ledger and is authorized to maintain it.

The publicly accessible blockchain protocols (e.g. Ethereum [5]), although secure, are visible to everyone. For data in critical infrastructure nodes such as power plants, such public access to data is impractical. Even current private blockchains are unable to support different levels of access to different participants, which is a critical requirement for existing systems that monitor and control field devices at remote sites (e.g., Supervisory Control and Data Acquisition, or SCADA system). In addition, energy transactions are typically stored in a centralized database and managed by a single entity. For example, a third-party operator manages all financial and energy transactions in the wholesale energy market (Gestore dei Mercati Energetici SpA or GME in Italy, Commission de régulation de l'énergie or CRE in France, etc). Cyberattacks could also compromise the trust and safety of such energy transactions. We note that blockchain technology needs to prove that it can offer the scalability, speed, and security required for the proposed use cases. They also face additional risks such as possible malfunctions at early stages of development due to lack of experience with large-scale applications.

Another important challenge is that blockchain systems currently have high development costs. For example, although the information in blockchain systems can be transferred for very low costs, the validation and verification of data requires considerable hardware and energy costs. Further, significant regulatory and legal barriers exist in the adoption of blockchain technology. Standards for blockchain architectures need to be developed to allow interoperability between the technology solutions.

Blockchain-based smart contracts have been adopted in the recent past for different use-cases in the heavily regulated power industry such as in automation of supplier roles, data aggregation, and data protection.[6] Aitzhan and Svetinovic [7] describe a token-based peer-to-peer system for anonymous negotiation of energy prices and Horta et al. [9] describe ongoing work on a blockchain-based distributed energy market test system [8]. LO3Energy's Brooklyn microgrid project has demonstrated electricity metering using smart contracts. Ponton has claimed the first blockchain-based energy trade in Europe [10]. There is therefore an urgent need to develop the next generation of blockchain technology for critical infrastructure to overcome the limitations of current blockchain protocols.

The research work in this report was undertaken with two clear aims in mind. First, we wanted to create a private blockchain architecture with cryptographic digital signatures and hierarchical access control with the ability to securely process signal data and other information flows within distributed sensor networks. Our focus was on enabling low-cost, commercially available laptops and personal computers to function as the blockchain nodes. The second aim was to create a lab-scale front-end system for the blockchain, consisting of a sensor network with secure data transmission that would test this protocol. The focus was on modelling the SCADA systems of commercial power plants in the lab-scale network and carrying out simulated cyberattacks on the blockchain network and testing its resilience.

1.2 Significance of the Work

Our protocol addresses security concerns for cyberattacks on distributed sensor networks that are vulnerable to threats from a given machine in the network. This is achieved by integrating the Smart Private Ledger into SCADA systems of the sensor networks that ensures strong security guarantees on sensor data using fundamental ideas from secret sharing and cryptographic digital signature schemes. The use of ordinary laptops and desktops as the nodes of the Smart Private Ledger leads to a blockchain network with minimal cost, lowering the barrier to access this important security technology for underserved areas/regions. This work thus establishes Smart Private Ledger as the next generation of blockchain technology for cybersecure sensor network compatible with existing SCADA systems and easily deployable for decentralized data storage and retrieval required for various applications.

1.3 Report Organization

The report is organized as follows. Chapter two describes the lab-scale sensor infrastructure created for the blockchain development. Chapters three to five describe the blockchain development. Chapter six provides a summary of the approach, while chapter seven describes the future directions.

CHAPTER TWO

Lab-scale Sensor Network for Blockchain Development

Operation of critical infrastructure such as smart grids, gas pipelines, and power plants depends upon the collection and analysis of a large amount of sensor data for their regular operation. This process involves the transfer of information within and across the networks through their Supervisory Control and Data Acquisition (SCADA) systems. Such systems, however, are increasingly vulnerable to malicious attacks from both, cybercriminals, and insider threats. In this paper, we develop a new blockchain protocol, namely, Smart Private Ledger, to realize cyber-secure sensor networks. The proposed technology securely transmits sensor data to the blockchain nodes using cryptographic digital signatures and secret sharing protocols. The protocol is a private blockchain network with hierarchical access control for data storage and retrieval to address the potential security risks associated with typical SCADA systems. Simulated cyber-attacks are performed on the system which establish the Smart Private Ledger as a highly secure, tamper-proof, and yet, simple decentralized data storage system. Given the concepts introduced in the past three chapters, the demonstration of the blockchain sensor network is shown in this chapter.

To simulate the information flows in infrastructure nodes such as power plants, we created a lab-scale sensor network with secure data transmission to the Smart Private Ledger blockchain. Two types of sensors pertaining to fossil energy power plants were chosen, namely, strain and temperature sensors.

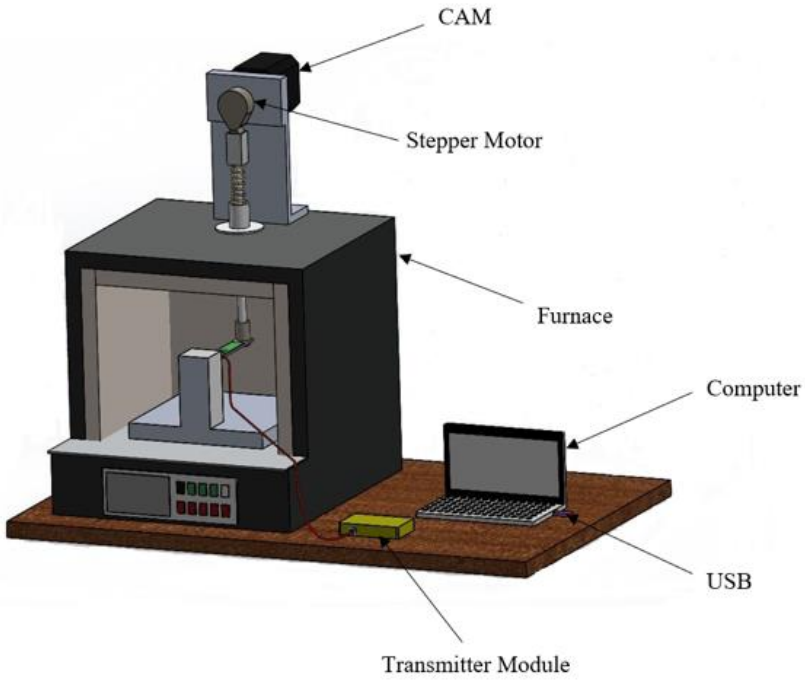


Figure 2.1: CAD model of high temperature strain sensing setup

2.1 Strain Sensors

The strain sensing apparatus consisted of a cantilever beam integrated with a high-temperature oven (Ney Vulcan 3-550 oven, Degussa-Ney Dental, Inc., Bloomfield, CT) as shown in Figure 1a. The components for this set-up consisted of a stepper motor, a driver to command the motor, cantilever base made of high temperature Inconel alloy, a ceramic pushrod driven by a CAM that deflected the cantilever beam. A stepper motor (HT34-490, Applied Motion, Watsonville, CA) was mounted onto the oven that drove the CAM. The motor was controlled by a driver (STAC6-Q, Applied Motion, Watsonville, CA) using a PC with the Q-programmer software. The CAM and the driver motor were mounted on the oven while the ceramic pushrod entered the oven and pushed down on the free end of the cantilever beam with the strain sensor. The base of the cantilever was clamped onto the Inconel base and the sensor assembly was placed inside the oven. The system was designed to provide a strain of 1000 με and a temperature range of up to 350 °C (or the maximum allowed by the sensor rather than the apparatus). Figure 1b shows a commercial strain sensor attached to the cantilever beam with wires connected to a transmission module to be described later. This strain sensor attachment procedure is similar to that described in our earlier work [11] and has been calibrated using commercial sensors. To measure strain, we installed a commercial strain sensor (VY4 Shear/Torsion full bridge strain gauge, HBM, Marlborough, MA, USA).

Figure 2.2: Image of the strain sensing setup integrated with an oven

2.2 Temperature Sensors

To integrate temperature measurement into our sensor network, we installed a commercial temperature sensor (100 Ohm RTD with a temperature range of 0 to 900 ºF, Grainger, USA) and connected it with the same transmitter as that used for the strain sensors. RTD (Resistance Temperature Detector) element consisted of a length of wire typically made up of platinum, wrapped around by a glass core. Electrical current is transmitted through the RTD element and the resistance value of the RTD element is measured. This resistance value is then correlated to temperature based upon the known resistance characteristics of the RTD element. Due to higher accuracy and repeatability of RTDs, they are slowly replacing thermocouples in industrial applications below 600 °C. Figure 2 shows an image of the temperature sensor attached to the transmission module

2.3 Transmission Module

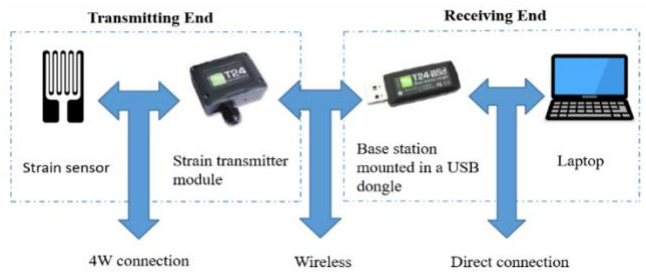
The sensors were integrated with a commercially available transmitter and a base station to securely transfer the data to the blockchain. Figure 3a shows the transmitter module (T24 module, Mantracourt Plc, UK) connected with the sensor and the base station, and a portable USB device connected to a laptop computer configured to act as a blockchain node. In this set-up, a single receiver could obtain data from up to 8 transmitters at once (each on a dedicated transmission channel). Furthermore, the transmitter and the receiver could be up to 800 meters

apart, which is sufficient in power plants or similar use-cases. We note that the base station can transmit signals over a secure wireless network to a receiving SCADA system as necessary.

We used a license-free 2.4 GHz direct sequence spread spectrum (DSSS) radio technology provided by Mantracourt, UK for data transmission between the transmitter to the base station. This technology uses a proprietary protocol based on 802.15.4 chip, allowing T24 range to co-exist with Bluetooth, Zigbee & Wi-Fi devices without conflicts. The system also had 128-bit AES data encryption for complete security during transmission. Other commercial systems can also be used to transmit data from the sensors to the blockchain.

We also note that the transmission frequency can be reduced (current value is 3Hz) to improve the battery life of the transmitter module. The data transmitted through the transmitter module is in a CSV format (Figure 3b), which is required for running the encryption algorithms on the blockchain. Note that the data is converted to a byte array format on the blockchain node.

Figure 2.3: Image of transmission module connected with temperature sensor




```
CE3A,10133102,127.9730,Friday, March 13, 2020 6:15:25 PM:689
CE3A,10133436,127.9682,Friday, March 13, 2020 6:15:26 PM:24
CE3A,10133771,127.9411,Friday, March 13, 2020 6:15:26 PM:359
CE3A,10134108,127.9202,Friday, March 13, 2020 6:15:26 PM:696
CE3A,10134445,127.9365,Friday, March 13, 2020 6:15:27 PM:32
CE3A,10134781,127.9551,Friday, March 13, 2020 6:15:27 PM:368
CE3A,10135118,127.9278,Friday, March 13, 2020 6:15:27 PM:705
CE3A,10135453,127.9365,Friday, March 13, 2020 6:15:28 PM:41
CE3A,10135786,127.9202,Friday, March 13, 2020 6:15:28 PM:374
CE3A,10136125,127.9411,Friday, March 13, 2020 6:15:28 PM:712
CE3A,10136460,127.9020,Friday, March 13, 2020 6:15:29 PM:47
CE3A,10136795,127.9305,Friday, March 13, 2020 6:15:29 PM:383
CE3A,10137132,127.9232,Friday, March 13, 2020 6:15:29 PM:720
CE3A,10137469,127.9051,Friday, March 13, 2020 6:15:30 PM:57
CE3A,10137804,127.8869,Friday, March 13, 2020 6:15:30 PM:391
CE3A,10138140,127.8946,Friday, March 13, 2020 6:15:30 PM:727
CE3A,10138475,127.8674,Friday, March 13, 2020 6:15:31 PM:62

[B@511d50c0
```

Figure 2.4: Schematic of secure wireless data transmission for the blockchain (left) and the data output from the transmission system in CSV format and later converted to a byte array format. This data format enables the encryption algorithm to be implemented on the blockchain. (right)

CHAPTER THREE

Blockchain for Encryption

In recent years, new blockchain-based cryptographic constructions have (theoretically) enabled exciting applications thought to be impossible to achieve in the standard model. For example, Liu et al. [11] propose a time-lock encryption scheme that allows one to encrypt a message such that it can only be decrypted once a certain deadline has passed, without relying on trusted third parties or imposing high computational overhead on the receiver. The construction of Choudhuri et al. [12] achieves fairness in multi-party computation against a dishonest majority. Goyal and Goyal [13] present the first construction for one-time programs (that run only once and then "self-destruct") that does not use tamper-proof hardware.

These exciting constructions have something in common — they all rely on blockchains and the notion of extractable witness encryption. Indeed, the combination of blockchains and witness encryption has proven remarkably powerful. Introduced by Garg et al. [14], a witness encryption scheme is, roughly, a primitive that allows one to encrypt a message with respect to a problem instance. Such a problem instance could be a sudoku puzzle in a newspaper or an allegedly bug-free program, or more generally, any NP search problem. If the decryptor knows a valid witness for the corresponding problem instance, such as a sudoku solution or a bug in the program, she can decrypt the ciphertext. Extractable security is a strong notion of security for witness encryption. If a witness encryption scheme is extractable, then an adversary who is able to learn any non-trivial information about the encrypted message is also able to provide a witness for the corresponding problem instance.

Unfortunately, extractable witness encryption typically relies on a strong and expensive form of obfuscation, the differing-inputs obfuscation [14, 15]. Indeed currently, there are no known practical extractable witness encryption schemes. In fact, there are no schemes based on standard assumptions, and Garg et al. [16] suggest that it may be impossible to achieve extractable witness encryption when the adversary has access to arbitrary auxiliary inputs.

In this work, we use a blockchain to achieve a functionality that is essentially equivalent to extractable witness encryption. Roughly, we allow users to encode a secret along with a release condition. A predefined set of n blockchain users (in the following, miners) jointly and securely store the encoding and later privately release the secret to the user who satisfies the release condition. We introduce a formal definition for extractable Witness Encryption on a Blockchain (eWEB) and a protocol that can augment existing blockchains with this functionality. By making only small changes to the code run by miners, it is now possible to easily implement the many applications that use extractable witness encryption as a building block. Conveniently, many such schemes [11-13] already rely on the guarantees provided by a blockchain, so building extractable witness encryption into the blockchain does not change their assumptions.

We formally prove our construction secure, relying on the guarantees provided by the blockchain setting. Specifically, we select a set (or rather, a multiset) of miners such that the majority of the selected miners are honest. As pointed out by Goyal and Goyal [13], one way to select such set of miners is by selecting miners who were responsible for mining the last n blocks (where n is large enough). Indeed, a single miner might mine multiple blocks and if so, appear multiple times in the set. For proof of work blockchains such as Bitcoin, the probability of successful mining is proportional to the amount of computational power, and hence if a majority of the computing power is honest, the majority of the selected miners can be expected to be honest. For proof of stake blockchains, where the probability of successful mining is proportional to the amount of currency possessed by the miner, this property follows from the assumption that honest miners possess the majority of stake in the system.

To ensure that dishonest miners cannot leak the user's secret, our eWEB scheme is built on top of a secret sharing scheme. A secret sharing scheme enables one party to distribute shares of a secret to n parties and ensures that an adversary in control of t out of n parties will learn no information about the secret. In our protocol, the miners hold the shares of the secret. Since we assume that the majority of the selected miners are honest, using a threshold of $t = \frac{n}{2} - 1$ enables us to securely store the secret.

However, traditional secret sharing schemes are insufficient for eWEB, since the set of parties (miners) who hold secret shares is constantly changing. To achieve security in this case, dynamic proactive secret sharing (DPSS) is required ([17-22]). DPSS schemes proactively update the secret shares held by the parties and allow changes to the set of parties holding the secrets.

Scheme	Dynamic setting	Adversary	Threshold	Network	Comm. (amort.)	Comm. (non-amort.)
Herzberg et al.	No	Active	$t/n < 1/2$	synch.	$O(n^2)$	$O(n^2)$
Cachin et al.	No	Active	$t/n < 1/3$	asynch.	$O(n^4)$	$O(n^4)$
Desmedt et al.	Yes	Passive	$t/n < 1/3$	asynch.	$O(n^2)$	$O(n^2)$
Wong et al.	Yes	Active	$t/n < 1/2$	synch.	$exp(n)$	$exp(n)$
Zhou et al.	Yes	Active	$t/n < 1/3$	asynch.	$exp(n)$	$exp(n)$
Schultz-MPSS	Yes	Active	$t/n < 1/3$	asynch.	$O(n^4)$	$O(n^4)$
Baron et al.	Yes	Active	$t/n < 1/2-\epsilon$	synch.	$O(1)$	$O(n^3)$
CHURP	Yes	Active	$t/n < 1/2$	synch.	$O(n^2)$	$O(n^2)$
This work	Yes	Active	$t/n < 1/2$	synch.	$O(n)$	$O(n^2)$

Figure 3.1: Comparison of PSS Schemes. The Comm. columns show the communication cost/secret in a hand-off round.

Part of our work is a new and highly optimized batched DPSS scheme. Batched setting is crucial for eWEB as there might be thousands of secrets stored in the system at any given time, and we need an efficient way to update all of those secrets in parallel. In contrast to previous work on batched DPSS [20], which focused on a single client submitting a batch of secrets and does not allow storing and releasing secrets independently, we allow multiple different clients to dynamically share and release secrets. Our protocol is the most efficient DPSS scheme that allows

the highest-possible adversarial threshold of $\frac{1}{2}$ (Figure 3.1). We have formally proven secure and implemented our scheme and believe that it is of independent interest.

We demonstrate the concrete practicality of our scheme, with all operations completing in seconds.

We outperform a prior state-of-the-art DPSS scheme by over 6×. In summary, we make the following contributions:

- We propose a new cryptographic primitive - extractable witness encryption on a blockchain.
- We design and formally prove secure a protocol which satisfies the notion of extractable witness encryption on blockchain.
- We present and formally prove secure a highly efficient batched dynamic proactive secret sharing scheme.
- We implement and evaluate several applications built atop eWEB.

3.1 DPSS

We start by discussing dynamic proactive secret sharing (DPSS), the key building block in eWEB. We informally explain the DPSS process and give an overview of our scheme and security proof.

3.1.1 DPSS Background

A DPSS scheme allows a client to distribute shares of a secret to n parties, so that an adversary in control of some threshold number of parties t learns no information about the secret. The set of parties holding secrets is constantly changing, and the adversary can "release" some parties (users regain control of their systems) and corrupt new ones.

A DPSS scheme consists of the following three phases.

Setup. In each setup phase, one or more independent clients secret- share a total of m secrets to a set of n parties, known as a committee, denoted by C = {$P1, \ldots, Pn$ }-. After each setup phase, each commit- tee member holds one share for each secret s distributed during this phase.

Hand-off. As the protocol runs, the hand-off phase is periodically invoked to provide the new committee with updated shares in such a way that the adversary cannot use information from multiple committees to learn anything about the secret. This process reflects parties leaving and joining the committee. After the hand-off phase, all parties in the old committee delete their shares, and all parties in the new committee hold a sharing for each secret s. The hand-off phase is particularly challenging, since during the hand-off a total of $2t$ parties may be corrupted (t parties in the old committee and t parties in the new committee).

3.1.1.1 Adversary Model

We consider a computationally bounded fully malicious adversary A with the power to adaptively choose parties to corrupt at any time. A can corrupt any number of clients distributing secrets and learn the secrets held by the corrupted clients. For each committee C with a threshold $t < |C|/2$, A can corrupt at most t parties in C. When a party Pi is corrupted by A, A fully controls the behavior of Pi and can modify Pi 's memory state. Even if A releases its control of Pi, its memory may have already been modified, e.g., Pi 's share might have been erased.

For a party Pi in both the old committee C and the new commit- tee C′, if A has the control of Pi during the hand-off phase, then Pi is considered to be corrupted in both committees. If A releases its control before the hand-off phase in which the secret sharing is passed from C to C′, then Pi is only considered corrupted in the old committee C. If A only corrupts Pi after the hand-off phase, Pi is only considered corrupted in the new committee C′.

For simplicity, in the following, we assume that there exist secure point-to-point channels between the parties and the corruption threshold is a fixed value t. Our protocol can be easily adapted to allow different thresholds for different committees.

3.1.1.2 DPSS Security Definition

A dynamic proactive secret-sharing scheme is required to satisfy two security properties: (1) it should always be possible to recover the secret, and (2) an adversary should not learn any further information about the secret beyond what has been learned before running the protocol. While our formal definition is slightly different from the original PSS definition [23], we can easily satisfy this definition as well. Our definition is most similar to the one used by Baron et al. [24]. However, we consider not only the scenario of one client submitting secrets, but many different clients submitting (and requesting the release of) secrets independently at different points in time. Additionally, we allow secret release not only to a single public, but also to the public.

3.1.2 Overview: Our DPSS Construction

We now outline our DPSS scheme. We first discuss the hand-off phase of our scheme in the semi-honest case and then explain how it can be modified for the fully malicious case. In a nutshell, the semi-honest case is solved primarily through the careful use of ideas from the MPC literature [21]. For the fully malicious case, unlike in the MPC world, we must marry these techniques with polynomial commitment schemes. We present the setup phase as a special case of our hand-off phase, summarize our reconstruction phase, and provide intuition for our construction's security proof.

In the following, we assume the corruption threshold for each committee is fixed to t. The scheme is based on Shamir Secret Sharing [25]. We use $[x]d$ to denote a degree-d sharing, i.e., (d + 1)-out-of-n Shamir sharing. It requires at least d + 1 shares to reconstruct the secret, any d or fewer shares do not leak any information about the secret. Note that Shamir's scheme is additively homomorphic.

3.1.2.1 Our Construction: Semi-honest Case

We first explain the high-level idea of our protocol in the semi-honest setting, i.e., all parties honestly follow the protocol. The crux of our construction is that both the old and the new committee hold a sharing of a random value. While the sharing is different for the two committees, the value this sharing corresponds to is the same. Let $([r]_t, [r\tilde{}]_t)$ denote these two sharings, where $[r]_t$ is held by the old committee, $[r\tilde{}]_t$ is held by the new committee, and $r = r\tilde{}$. Suppose the secret sharing we want to refresh is $[s]t$, held by the old committee. Then the old committee will compute the sharing $[s + r]_t = [s]_t + [r]_t$ and reconstruct the secret $s + r$. Since r is a uniform element, $s + r$ does not leak any information about s. Now, the new committee can compute $[s\tilde{}]_t = (s + r) - [r\tilde{}]_t$. Since $r\tilde{} = r$, we have $s\tilde{} = s$. This whole process is split into preparation and refresh phases:

- In the preparation phase, parties in the new committee prepare two degree-t sharings of the same random value r $(= r\tilde{})$, denoted by $[r]_t$ and $[r\tilde{}]_t$. The old committee receives the shares of $[r]_t$ and the new committee holds the shares of $[r\tilde{}]_t$. We refer to these two sharings as a *coupled sharing*.
- In the refresh phase, the old committee reconstructs the sharing $[s]_t + [r]_t$ and publishes the result. The new committee sets $[s\tilde{}]_t = (s + r) - [r\tilde{}]_t$.

We start by explaining the preparation phase with the goal of generating a coupled sharing of a random value. In the following, let C denote the old committee and C′ denote the new committee. Intuitively, the new committee can prepare a coupled sharing as follows:

(1) Each party $P \in C'$ prepares a coupled sharing $([u^{(i)}]_t, [u\tilde{}^{(i)}]_t)$ of a random value and distributes $[u^{(i)}]_t$ to the old committee and $[u\tilde{}^{(i)}]_t$ to the new committee.

(2) All parties in the old committee compute $[r]_t = \acute{I}^{n}_{i=1}[u^{(i)}]_t$. All parties in the new committee compute $[r\tilde{}] = \acute{I}^{n}_{i=1}[u\tilde{}^{(i)}]_t$.

Since for each i, $u(i) = u\tilde{}(i)$, we have $r = r\tilde{}$.

However, this way of preparing coupled sharings is wasteful since at least $(n - t)$ coupled sharings are generated by honest parties, which appear uniformly random to corrupted parties. In order to get $(n - t)$ random coupled sharings instead of just 1, we borrow an idea from Damgård and Nielsen [21].

In their work, parties need to prepare a batch of random sharings which will be used in an MPC protocol. All parties first agree on a fixed and public Vandermonde matrix $\boldsymbol{M}$T of size $n \times (n - t)$. An important property of a Vandermonde matrix is that any $(n - t) \times (n - t)$ submatrix of $\boldsymbol{M}$T is invertible. To prepare a batch of random sharings, each party Pi generates and distributes a random sharing $[u(i)]t$. Next, all parties compute

$$([r^{(1)}]_t, [r^{(2)}]_t, \ldots, [r^{(n-t)}]_t)^{\mathrm{T}} = \boldsymbol{M}([u^{(1)}]_t, [u^{(2)}]_t, \ldots, [u^{(n)}]_t)^{\mathrm{T}}$$

and take $[r^{(1)}]_t, [r^{(2)}]_t, \ldots, [r^{(n-t)}]_t$ as output. Since any $(n-t) \times (n-t)$ submatrix of $\boldsymbol{M}$ is invertible, given the sharings provided by corrupted parties, there is a one-to-one map from the output sharings to the sharings distributed by honest parties. Since the input sharings of the honest parties are uniformly random, the one-to-one map ensures that the output sharings are uniformly random as well [21].

Note that any linear combination of a set of coupled sharings is also a valid coupled sharing. Thus, in our protocol, instead of computing $([r]_t, [\tilde{r}]_t) = \sum_{i=1}^{n} ([u^{(i)}]_t, [\tilde{u}^{(i)}]_t)$, parties in the old committee can compute

$$([r^{(1)}]_t, [r^{(2)}]_t, \ldots, [r^{(n-t)}]_t)^{\mathrm{T}} = \boldsymbol{M}([u^{(1)}]_t, [u^{(2)}]_t, \ldots, [u^{(n)}]_t)^{\mathrm{T}}$$
$$([\tilde{r}^{(1)}]_t, [\tilde{r}^{(2)}]_t, \ldots, [\tilde{r}^{(n-t)}]_t)^{\mathrm{T}} = \boldsymbol{M}([\tilde{u}^{(1)}]_t, [\tilde{u}^{(2)}]_t, \ldots, [\tilde{u}^{(n)}]_t)^{\mathrm{T}}$$

and parties in the new committee can compute

Now all parties get $(n-t)$ random coupled sharings. The amortized communication cost per such sharing is $O(n)$.

We now describe the refresh phase. For each sharing $[s]t$ of a client secret which needs to be refreshed, one random coupled sharing ($[r]_t$, $[r\tilde{}]_t$) is consumed. Parties in the old committee first select a special party *P*king. To reconstruct $[s]_t + [r]_t$, parties in the old committee locally compute their shares of $[s]_t + [r]_t$, and then send the shares to *P*king. Then, *P*king uses these shares to reconstruct $s + r$ and publishes the result. Finally, parties in the new committee can compute $[s\tilde{}]_t = (s + r) - [r\tilde{}]_t$.

3.1.2.2 Moving to a Fully Malicious Setting

In a fully malicious setting, three problems might arise.

- During preparation, a party distributes an inconsistent degree-t sharing or incorrect coupled sharing.
- During refresh, a party provides an incorrect share to P_{king}, causing a reconstruction failure.
- P_{king} provides an incorrectly reconstructed value.

We address these problems by checking the correctness of coupled sharings in the preparation phase and relying on polynomial commitments to transform a plain Shamir secret sharing into a verifiable one.

Checking the Correctness of Coupled Sharings. Recall that any linear combination of coupled sharings is also a valid coupled sharing. Thus, to increase efficiency, instead of checking the correctness of each coupled sharing, it is possible to check a random linear combination of the coupled sharings distributed by each party.

To protect the privacy of the coupled sharing $([u^{(i)}]_t, [\tilde{u}^{(i)}]_t)$ generated by P_i' , P_i' will generate one additional random coupled sharing as a random mask, which is denoted by $([\mu^{(i)}]_t, [\tilde{\mu}^{(i)}]_t)$.

Consider the following two sharings of polynomials of degree-$(2n-1)$:

$$[F(X)]_t = \sum_{i=1}^{n} ([\mu^{(i)}]_t + [u^{(i)}]_t \cdot X) X^{2(i-1)},$$
$$[\tilde{F}(X)]_t = \sum_{i=1}^{n} ([\tilde{\mu}^{(i)}]_t + [\tilde{u}^{(i)}]_t \cdot X) X^{2(i-1)}.$$

If all coupled sharings are correct, then $([F(\lambda)]_t, [\tilde{F}(\lambda)]_t)$ is also a coupled sharing for any λ. Otherwise, the number of λ such that $([F(\lambda)]_t, [\tilde{F}(\lambda)]_t)$ is a coupled sharing is bounded by $2n-1$. Thus, it is sufficient to test $([F(\lambda)]_t, [\tilde{F}(\lambda)]_t)$ at a random evaluation point λ. Since each individual coupled sharing $([u^{(i)}]_t, [\tilde{u}^{(i)}]_t)$ is masked by $([\mu^{(i)}]_t, [\tilde{\mu}^{(i)}]_t)$, revealing $([F(\lambda)]_t, [\tilde{F}(\lambda)]_t)$ does not leak any information about the individual coupled sharings.

Therefore, all parties first generate a random challenge λ. Parties in the old committee compute $[F(\lambda)]_t$ and publish their shares. Parties in the new committee compute $[\tilde{F}(\lambda)]_t$ and publish their shares. Finally, all parties check whether $([F(\lambda)]_t, [\tilde{F}(\lambda)]_t)$ is a valid coupled sharing.

If the check fails, we need to pinpoint the parties who distributed incorrect coupled sharings. Each coupled sharing $([u^{(i)}]_t, [\tilde{u}^{(i)}]_t)$ is masked by $([\mu^{(i)}]_t, [\tilde{\mu}^{(i)}]_t)$. Therefore it is safe to open the whole sharing $([\mu^{(i)}]_t + [u^{(i)}]_t \cdot \lambda, [\tilde{\mu}^{(i)}]_t + [\tilde{u}^{(i)}]_t \cdot \lambda)$ and check whether it is a valid coupled sharing. For each i, parties in the old committee compute $[\mu^{(i)}]_t + [u^{(i)}]_t \cdot \lambda$ and publish their shares, and parties in the new committee compute $[\tilde{\mu}^{(i)}]_t + [\tilde{u}^{(i)}]_t \cdot \lambda$ and publish their shares. This way, we can tell which coupled sharings are inconsistent. This inconsistency in the coupled sharing distributed by some party P_i' (in the following, *dealer*) has two possible causes:

- The dealer P_i' distributed an invalid coupled sharing (either the secrets were not the same or one of the t-sharings was invalid).
- Some corrupted party $P_j \in \mathcal{C} \cup \mathcal{C}'$ provided an incorrect share during the verification of the sharing distributed by the dealer P_i'.

The first case implies that the dealer is a corrupted party. To distinguish the first case from the second, we will rely on polynomial commitments, which can be used to transform a plain Shamir secret sharing into a verifiable one so that an incorrect share (e.g., in case 2) can be identified and rejected by all parties.

Relying on Polynomial Commitments. A degree-t Shamir secret sharing corresponds to a degree-t polynomial $f(\cdot)$ such that: (a) the secret is $f(0)$, and (b) the i-th share is $f(i)$. Thus, each dealer can commit to f by using a polynomial commitment scheme to add verifiability.

A polynomial commitment scheme allows the dealer to open one evaluation of f (which corresponds to one share of the Shamir secret sharing) and the receiver can verify the correctness of this evaluation value. Essentially, whenever a dealer distributes a share it also provides a *witness* which can be used to verify this share. Informally, a polynomial commitment scheme should satisfy three properties:

- Polynomial Binding: A commitment cannot be opened to two different polynomials.
- Evaluation Binding: A commitment cannot be opened to two different values at the same evaluation point.
- Hiding: A commitment should not leak any information about the committed polynomial.

We use polynomial commitments as follows: in the beginning, each dealer first commits to the sharings it generated and opens the shares to corresponding parties. To ensure that each party is satisfied with the shares it received, there is a following accusation-and-response phase:

(1) Each party publishes (accuse, P_i') if the share received from P_i' does not pass the KZG verification algorithm.

(2) For each accusation made by P_j, P_i' opens the j-th share to all parties, and P_j uses the new share published by P_i' if it passes the verification. Otherwise, P_i' is regarded as a corrupted party by everyone else.

Note that an honest party will never accuse another honest party. Also, if a malicious party accuses an honest party, no more information is revealed to the adversary than what the adversary knew already. Thus, it is safe to reveal the share sent from P_i' to P_j. *After this step, all i-parties should always be able to provide valid witnesses for their shares.*

Recall that parties need to do various linear operations on the shares. In our protocol we use the KZG commitment scheme [26], which is linearly homomorphic. Thus, even if a share is a result of a number of linear operations, it is still possible for a party to compute the witness for this share. From now on, each time a party sends or publishes a share, this party also provides *the associated witness* to allow other parties verify the correctness of the share. Since honest parties will always provide shares with valid witnesses and there are at least $n - t \geq t + 1$ honest parties, all parties will only use shares that pass verification. Intuitively, this solves the problem of incorrect shares provided by corrupted parties since corrupted parties cannot provide valid witnesses for those shares. Similarly, it should solve the problem of a malicious P_{king}, since he cannot provide a valid witness for the incorrectly reconstructed value. However, due to a subtle limitation of the KZG commitment scheme, we actually need to add an additional minor verification step (see Appendix C for details).

3.1.2.3 DPSS Setup Phase

The setup phase uses a similar approach to the hand-off phase. First, the committee prepares random sharings. As in the hand-off phase, the validity of the distributed shares is verified using the KZG commitment scheme. For each secret s distributed by a client, one random sharing $[r]_t$ is consumed. The client receives the whole sharing $[r]_t$ from the committee and reconstructs the value r.

Finally, the client publishes $s + r$. The committee then computes $[s]_t = s + r - [r]_t$.

3.1.2.4 DPSS Reconstruction Phase

When a client asks for the reconstruction of some secret $s^\star$, all parties in the current committee simply send their shares of $[s^\star]_t$ and the associated witnesses to the client. The client then reconstructs the secret using the first $t + 1$ shares that pass the verification checks.

3.1.2.5 Security of Our Construction

We give a high-level idea of our proof. The goal is to construct a simulator to simulate the behaviors of honest parties. For each sharing, corrupted parties receive at most t shares, which are independent of the secret. Thus, when an honest party needs to distribute a random sharing, the simulator can send random elements to corrupted parties as their shares without fixing the shares of honest parties. Since we use the perfectly hiding variant of the KZG commitment, the commitment is independent of the secret, and can be generated using the trapdoor of the KZG scheme. Furthermore, we can adaptively open t shares chosen by the adversary after the commitment is generated. This makes our scheme secure against adaptive corruptions.

3.2 Defining eWEB

Witness encryption [14] allows a party to encrypt a message to an instance x of an NP language. Another party can then decrypt the ciphertext using a witness that x is in the language. The traditional notion of security for witness encryption, introduced by Garg et al. [14], is *soundness security* and states that if a message was encrypted to some instance x that is *not* in the language, then no adversary can learn any non-trivial information about the message. However, this is often insufficient. Many constructions [11, 13, 27] consider the case where the instance x *is* in the language, and must rely on a stronger notion of *extractable* security. Informally, if an adversary is able to distinguish between two different ciphertexts encrypted to the same problem instance, then he is also able to provide a witness to this problem instance.

In our work we use a permissioned blockchain to achieve a similar goal. Although they may overlap in practice, for ease of exposition, we distinguish between users who deposit secrets (depositors), users who request that a secret be released (requesters), and a changing set of blockchain nodes (miners) who are executing these requests. A depositor who wishes to securely store a secret until some condition is satisfied will distribute the encoded secret

among the miners and specify the release condition. When a requester wishes to learn the secret, they must provide a witness for the release condition. The miners will check the witness, and if it is valid, securely provide the secret to the requester (the secret is *not* released publicly). In addition to storing and releasing secrets, we require a hand-off procedure to be periodically executed by the miners, since the set of miners is constantly changing. During the hand-off, all the deposited secrets are handed from the old set of miners (*old committee*) to the new set of miners (*new committee*). The full process is depicted in Figure 2. Intuitively, no adversary should learn any non-trivial information about a user's secret unless he knows a witness for the corresponding release condition. Further, no one should be able to change stored secrets.

We provide formal syntax for the primitive. Using this syntax, we define the security of extractable witness encryption on a blockchain (*eWEB*) via a security game then presents our eWEB protocol.

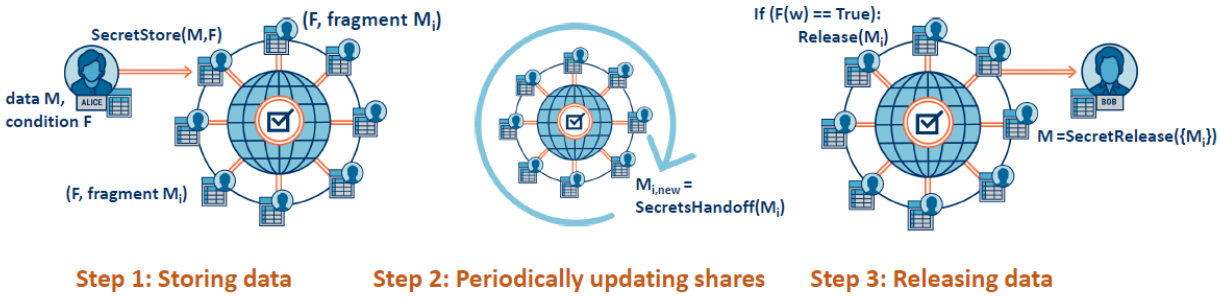


Figure 3.2: High-level overview of eWEB

3.2.1 Syntax

An eWEB system consists of the following, possibly randomized and interactive, subroutines:

$SecretStore(M, F) \rightarrow (id, \{frag_1, .., frag_n\}, F)$: A depositor stores a secret M which can be released to a requester who knows a witness w s.t. F (w) is true. After interacting with the depositor, each of the n miners obtains a "fragment", frag $_i$, of the secret that is associated with i the secret storage request with the identifier id.

$SecretsHandoff(\{frag_1^1, .., frag_n^1\}, .., \{frag_1^m, .., frag_n^m\}) \rightarrow (\{frag_1^1, .., frag_n^1\}, .., \{frag_1^m, .., frag_n^m\})$:

Miners periodically execute the SecretsHandoff function to hand over all m stored secrets from the old committee to the new committee. Each miner i of the old committee possesses m fragments (one for each secret) frag , .., frag at the end of the protocol.

SecretRelease(id,w) $\rightarrow M$ or $\perp$: A requester uses this function to request the release of the secret with the identifier id. The requester specifies the witness w to the release condition. Miners check whether the requester holds a valid witness and if so, as a result of the interaction

with the miners, the requester obtains the secret M. Otherwise, the function returns $\perp$ (i.e., attempt failed).

3.2.2 Security Game Definition

We define security via a game.

Definition 3.1 (Security Game). The game is played between the adversary, A, and a challenger, C. A is a probabilistic, polynomial-time adversary who controls t parties of the old committee and t parties of the new committee during each handoff round. The game is parametrized by the number of parties n participating in each round as well as the number of rounds d. The challenger creates a set of $(n - t) * d$ unique party identifiers and a corresponding set PK_H of the public keys for the honest parties. These IDs and keys are known to A. The decryption condition F is public information as well. The game takes as input a list W of witnesses for F such that $\forall w \in W: F(w) = true$. Our security game is quantified over all choices of this list. Note that the same witness can occur multiple times in this list, and so if there exists at least one witness for F we assume w.l.o.g. that W is of length d. If no witness for F exists, the list is empty. One can iterate through the list using the $next$ () function, which returns the next witness in the list or $\perp$, if no such witness exists. Whenever messages are exchanged between parties, the adversary sees them (including messages sent between honest parties).

(1) A chooses two strings M_1 and M_2 of the same length and submits both to the challenger.

(2) The challenger flips a coin, $b \in \{0, 1\}$, uniformly at random, which is fixed for the duration of the game.

(3) In each round, until A finishes the game:

A new committee is chosen, and secrets (if any exist) are handed over

- A chooses a set T of t indices for the adversarial parties of the new committee and generates a public key pk_i for each adversarial member of this committee. Then, A sends T and the public keys $\{pk_i\}_{i \in T}$ to C.
- A chooses a set H of $n - t$ public keys from the set PK_H for the honest parties of the new committee and sends H to C.
- If at least one secret is stored on the blockchain, C carries out *SecretsHandoff* for each honest member of the old and the new committee.

(4) Additionally, until A finishes the game, one of the following can happen in each round:

(a) A may ask the challenger to create a secret storage request for the challenge (only once)

• If this is the first time A makes such a request, the challenger carries out *SecretStore* for the secret M_b and the secret release condition F.

(b) If the challenge secret storage request was executed, A may ask the challenger to have an honest party (with identifier pid) create a release request for the challenge.

• C executes *SecretRelease* for the challenge using the public key pk of the party with the identifier p_{id} and a witness $w = W.next()$. If $w = \perp$, A loses.

(c) A may create new secret deposits

• C carries out *SecretStore* for the new request for each honest member of the committee using the information supplied by the adversary.

(d) A may create new release requests for any number of storage requests with IDs id_i

• C carries out *SecretRelease* for each request id_i for each honest member of the committee using the information supplied by the adversary.

(e) A may end the game with its guess, b', for b.

Definition 3.2. (Secrecy) For a security parameter λ, number of parties participating in each round n, number of rounds d, corruption threshold t, decryption condition F, and list of

$$\mathsf{Adv}[\mathcal{A}, 1^\lambda, n, d, t, F, W] = \left| Pr[b' = b] - \frac{1}{2} \right|$$

witnesses W of size d (or 0 if no witness for F exists) s.t. $\forall w \in W : F(w) = true$, let the advantage Adv[A, 1^λ,n,d,t, F,W] of the adversary A in the game introduced above be defined as follows:

The system provides secrecy if for any polynomial-time probabilistic adversary A and any predicate function F that is verifiable in PPT, there exists a PPT extractor E that uses A as an oracle such that:

where $poly(n)$, $poly'(n)$ are polynomials.

$$\mathsf{Adv}[\mathcal{A}, 1^\lambda, n, d, t, F, W] \geq \frac{1}{poly(n)} \Rightarrow Pr[E^{\mathcal{A}}(F, 1^n) = w \wedge F(w) = True] \geq \frac{1}{poly'(n)},$$

Practically, this definition means that if an adversary is able to distinguish between the protocol executed with secret M_0 and the protocol executed with secret M_1, then we can extract a valid witness for the release condition F using this adversary. Note that E does not get access to the list of witnesses. Intuitively, this notion is quite similar to the extractable security of witness encryption, which states that if an adversary can distinguish between two ciphertexts, then he can also extract a witness from the corresponding problem instance.

We define the robustness of an eWEB scheme as follows:

Definition 3.3. (Robustness) For any PPT adversary A with corruption threshold t, it holds that after *SecretStore*(M, F), there exists a fixed secret (identified by id) for the distributed

fragmentation $\{frag_1, .., frag_n\}$. In particular, the fragmentation dealt by an honest depositor has the same secret M as the one chosen by this depositor. When at some point a requester executes *SecretRelease*(id, w), where $F(w) = true$, the requester reconstructs the correct secret M.

Remark 3.1. Removing step (4b) in the game above produces a slightly relaxed security notion we dub *Public Witness* security. Here, the secret is made public after a single successful secret release. As shown, this notion proves quite useful in a number of applications.

3.3 Our eWEB Protocol Design

In this section, we introduce our construction. First, we provide an overview of the assumptions that we rely on in our scheme. Then, we describe our eWEB construction. Finally, we provide a security proof sketch, with a formal proof.

In we provide a PublicWitness scheme that is similar to our eWEB protocol, but targets the use case in which the secret is made public after a single successful release request (rather than released privately only to requesters with valid witness as in eWEB).

3.3.1 Assumptions

Adversary model. We assume that the adversary is able to control a polynomial number of users and miners, subject to the constraint that at any time the majority of the miners who are eligible to participate in the protocol are honest. We rely on permissioned blockchains and assume that eWEB is a core functionality, which allows us to focus on the fundamental construction without worrying about selfish mining or bribery attacks. Honest majority assumptions are very common in the blockchain space [13, 22, 27-29]. Especially permissioned blockchains often rely on BFT replication protocols, which in turn usually assume honest supermajority[30].

We assume that once an adversary corrupts a party it remains corrupted. The adversary cannot adaptively corrupt previously honest parties. When a party is corrupted by the adversary, the adversary fully controls this party's behaviour and internal memory state. We do not distinguish between adversarial and honest parties who behave maliciously unintentionally; e.g., those who have connection issues and cannot access the blockchain to participate.

Infrastructure model. It is common for public keys to be known in blockchains. We require that additionally each party p_i has a *unique identifier*, denoted by pid_i, that is known to all other parties. In practice, this identifier can be the hash of the party's public key. For simplicity, we present the scheme as if there were authenticated channels between all parties in the system. In practice, these channels can be realized using standard techniques such as signatures.

Communication model. Our DPSS scheme assumes secure point-to-point communication channels. In the decentralized blockchain setting of eWEB we prefer not to make such an assumption, since using point-to-point channels could compromise nodes' anonymity and lead to targeted attacks [22] . Instead, we assume that parties communicate via an existing

blockchain. We distinguish between posting a message on the blockchain (expensive) and using the blockchain's peer-to-peer network for broadcast (cheap). Point-to-point channels can be simulated using IND-CCA secure encryption and broadcasting the ciphertexts.

Storage. We assume that, in addition to parties' internal storage, there exists some publicly accessible off-chain storage that is cheaper than on-chain one. Thus, we store data off-chain and save only data *hashes* on-chain. Our system's robustness depends on the robustness of the off-chain storage. Thus, storage systems with a reputation for high availability should be chosen. However malicious off-chain storage does not impact the secrecy properties of our system. Alternatively, at a higher cost, we can simply use on-chain storage for everything.

Permissionless setting. In future our work could be extended to permissionless blockchains that have the *chain quality*, meaning that for each n or more continuous blocks mined in the system, more than half were mined by honest parties. The committees are then formed by the miners that mined the most recent n blocks. The chain quality is particularly easy to achieve in *fair* blockchains such as Fruitchains [31], where the fraction of honestly mined blocks is close to the fraction of honest work or stake. For fair proof-of-work blockchains, where the probability of successful mining is proportional to the amount of computational power,chain quality follows from the assumption that honest miners possess the majority of the computational power in the system [13]. For fair proof-of-stake blockchains, where the probability of successful mining is proportional to the amount of coins possessed by the miner, it follows from the assumption that honest miners possess the majority of stake in the system.

Miners that behave honestly w.r.t the blockchain protocol might need further incentivization to behave honestly w.r.t. eWEB; otherwise they might try to disrupt the execution of the eWEB protocol or leak their secret shares. Our DPSS scheme has numerous checks that identify parties disrupting correct protocol execution, which could translate to economic disincentivization. Traitor-tracing secret sharing as well as trusted hardware that can verify correct share deletion could be used as mechanisms that ensure that miners are punished for leaking secrets entrusted to them. We leave exploring these directions for future work.

3.3.2 Our eWEB Construction

We now describe our eWEB scheme. Its key building block is a DPSS scheme used in a black-box way. The initial committee are miners currently eligible to participate in the permissioned blockchain we are building upon.

Given a secret message M and a release condition F, the depositor stores the release condition F on the blockchain and secret-shares M among the miners using the secret storage (setup) algorithm of the DPSS scheme.

Whenever the set of miners eligible to participate in the blockchain protocol changes, the hand-off phase is executed and the secrets are passed from the miners of the old committee to the miners of the new committee using the DPSS hand-off algorithm. Note that our DPSS

scheme supports changes in the committee sizes as long as the honest majority assumption is satisfied

To retrieve a stored secret, a requester U needs to prove that they are eligible to do so. This poses a challenge. An insecure solution is to just send a valid witness w ($F(w) = true$) to the miners. One obvious problem with this solution is that a malicious miner can use the provided witness to construct a new secret release request and retrieve the secret himself. To solve this problem, instead of sending the witness in clear, the user *proves that they know a valid witness*. Thus, while the committee members are able to check the validity of the request and privately release the secret to U, the witness remains hidden. In our scheme we rely on non-interactive zero knowledge proofs (NIZKs) [13]. Such proofs allow one party (the prover) to prove validity of some statement to another party (the verifier), such that nothing except for the validity of the statement is revealed. In eWEB we specifically use simulation extractable non-interactive zero knowledge *proofs of knowledge*, which allow the prover convince the verifier that they know a witness to some statement. Note that extractability can be added to any NIZK [32, 33]. We use NIZKs for relation $R = \{(pk, w) \mid F(w) = true$ and $pk = pk\}$, where $F(\cdot)$ is the release condition specified by the depositor and pk is the public key of user U and is used to identify the user eligible to receive the secret. After the miners verify the validity of the request, they engage in the DPSS's secret reconstruction with requester U to release the secret to U.

We provide the full secret storage protocol in Figure 3. The hand-off protocol is given in Figure 4. The secret release protocol is in Figure 5. The full construction is given in Figure 6. Note that the asymptotics of eWEB match those of our underlying DPSS scheme. Below, we elaborate on additional details of our construction.

3.3.2.1 Subtleties of Point-to-Point Channels

As mentioned, while our DPSS protocol assumes secure point-to-point channels, we do not make such an assumption in eWEB. Instead, we rely on authenticated encryption and Protocols 1 and 2, executed whenever a message needs to be securely sent from one party to another. It is used for all messages exchanged in eWEB, including the underlying DPSS protocol. Whenever a party receives an encrypted message, it performs an authentication check via Protocol 2 to ensure that a ciphertext received from some party was generated by that party. This prevents the following malleability issue - a malicious user desiring to learn a secret with the identifier id could generate a new secret storage request with a function F' for which he knows a witness, copy the DPSS messages sent by the user who created the storage request id to the miners and later on prove his knowledge of a witness for F' to release the corresponding secret. Without the authentication check our scheme would be insecure, and our security proof would not go through.

3.3.2.2 *Storage Identifiers*

Each storage request has a unique identifier id. This can be, e.g., the address of this particular transaction in the blockchain. It is used for practical reasons, and is not relevant for the security of our construction.

3.3.2.3 *Handling Large Secrets*

Since the secret itself might be very large, it is also possible to first encrypt the secret using a symmetric encryption scheme, store the ciphertext publicly off chain and then secret-share the symmetric key instead. Also, we store request parameters (such as release conditions or proofs) off-chain, saving only the hash of the message on-chain.

Protocol 1 MESSAGEPREPARATION

1. For a message m to be sent by party P_s to party P_r, P_s computes the ciphertext $c \leftarrow Enc_{pk_r}(m|pid_s)$, where pk_r is the public key of P_r and pid_s is the party identifier of P_s.
2. P_s prepends the storage identifier id of his request and sends the tuple (id, c) to P_r.

Protocol 2 AUTHENTICATEDDECRYPTION

1. Upon receiving a tuple (id, c) from party P_s over an authenticated channel, the receiving party P_r decrypts c using its secret key sk to obtain $m \leftarrow Dec_{sk}(c)$.
2. P_r verifies that m is of the form $m'|pid_s$ for some message m', where pid_s is the identifier of party P_s.
3. If the verification check fails, P_r stops processing c and outputs an error message.

3.3.3 Security Proof Intuition

We provide a formal proof of security in Appendix G, showing that our scheme satisfies the security definition for eWEB given. In this proof, we rely on the zero-knowledge and simulation-sound extractability properties of the NIZK scheme to switch from providing honest proofs to using simulated proofs. Next, we rely on the collision-resistance of the hash function to show that any modification of the data stored offchain will be detected. Then, we rely on the multi-message IND-CCA security of the encryption scheme to change all encrypted messages exchanged between honest parties to encryptions of zero. Finally, we rely on the secrecy property of our DPSS scheme to switch from honestly executing the DPSS protocol to using a DPSS simulator. At this point, we can show that either the adversary was able to provide a valid secret release request for the challenge's secret-release function, in which case we are able to extract a witness from the provided NIZK proof (relying on the NIZK's proof-of-knowledge property), or the adversary did not provide a valid secret release request and in this case we are able to "forget" the secret altogether, since it is never used.

Protocol 3 SECRETSTORE

1. The depositor executes NIZK's KeyGen protocol to obtain a CRS: $\sigma \leftarrow \mathsf{KeyGen}(1^k)$.
2. The depositor computes hash $\mathsf{requestHash} \leftarrow H(F|\sigma)$, and publishes $\mathsf{requestHash}$ on the blockchain. Let id be the storage identifier of the published request.
3. The depositor stores the tuple $(id, F|\sigma)$ offchain.
4. The depositor and the current members of the miner committee engage in the DPSS **Setup Phase**.
5. Each committee member retrieves $\mathsf{requestHash}$ from the blockchain, $F|\sigma$ from the offchain storage, and verifies that $\mathsf{requestHash}$ is indeed the hash of $F|\sigma$:
$$\mathsf{requestHash} \stackrel{?}{=} H(F|\sigma)$$
If this is not the case, the committee member aborts.
6. C_i stores $(id, \mathsf{dpss\text{-}data}_i)$ internally, where $\mathsf{dpss\text{-}data}_i$ is the data obtained from the DPSS **Setup Phase**.

Protocol 4 SECRETSHANDOFF

1. For each secret storage identifier id, the miners of the old and the new committee engage in the DPSS **Handoff Phase** for the corresponding secret. Let $\mathsf{dpss\text{-}data}_i^{id}$ denote the resulting DPSS data corresponding to the storage identifier id of party C_i of the new committee after the handoff phase.
2. For each secret storage identifier id, each miner of the new committee stores $(id, \mathsf{dpss\text{-}data}_i^{id})$ internally.

Finally, we rely on the secrecy property of our DPSS scheme to switch from honestly executing the DPSS protocol to using a DPSS simulator. At this point, we can show that either the adversary was able to provide a valid secret release request for the challenge's secret-release function, in which case we are able to extract a witness from the provided NIZK proof (relying on the NIZK's proof-of-knowledge property), or the adversary did not provide a valid secret release request and in this case we are able to "forget" the secret altogether, since it is never used.

3.4 Application Examples

In this section, we present some motivational application examples and briefly explain the key ideas behind implementing each of them using our construction.

Time-lock Encryption. Time-lock encryption, related to timed-release encryption introduced by Rivest et al. [34], allows one to encrypt a message such that it can only be decrypted after a certain deadline has passed. Time-lock encryption must satisfy a number of properties [11], such as the encrypter needs not be available for decryption and trusted parties are not allowed. Time-lock encryption can be easily implemented using the PublicWitness scheme. Using this scheme, the encrypter executes SecretStore with a secret release condition F specifying the time t when the data can be released. Once the time has passed, a user who wishes to see the message submits a SecretRelease request with the witness "The deadline has passed". Miners check whether the time is indeed past t and if so, release their fragments of the secret. With a slight modification to our scheme, it is also possible to enable automatic decryption - upon receiving a secret storage request with an "automatic" tag, miners would place the identifier in a list and periodically check whether the release condition holds for any request in this list.

Note that our scheme evades the issue that some time-lock encryption schemes [11] have: even if the adversary becomes computationally more powerful, it does not allow him to receive

the secret message earlier. Additionally, we avoid the computational waste of timed-release

Protocol 5 SECRETRELEASE

1. To request the release of a secret with identifier id, the requester retrieves requestHash from the blockchain, $F|\sigma$ from off-chain storage, and verifies that requestHash is indeed the hash of $F|\sigma$:

$$\mathsf{requestHash} \stackrel{?}{=} H(F|\sigma)$$

   If this is not the case, the requester aborts.
2. The requester computes a NIZK proof of knowledge of the witness for F and his identifier p_{id}:

$$\pi \leftarrow P(\sigma, p_{id}, w),$$

3. The requester computes hash of the storage identifier, his identifier and the proof to obtain $\mathsf{requestHash}^* \leftarrow H(id|p_{id}|\pi)$ and publishes $\mathsf{requestHash}^*$ on blockchain. Let id^* be the identifier of the published request.
4. The requester stores $(id^*, id|p_{id}|\pi)$ offchain.
5. Each committee member retrieves $\mathsf{requestHash}^*$ from the blockchain request with the identifier id^*, $id|p_{id}|\pi$ from the offchain storage, and verifies that:

$$\mathsf{requestHash}^* \stackrel{?}{=} H(id|p_{id}|\pi)$$

   If not, the committee member aborts.
6. Each committee member retrieves requestHash from the blockchain request with the identifier id, $F|\sigma$ from the offchain storage, and verifies the following:

$$\mathsf{requestHash} \stackrel{?}{=} H(F|\sigma)$$

   If not, the committee member aborts.
7. Each committee member C_i retrieves its share of the secret, $\mathsf{dpss\text{-}data}_i$, from its internal storage.
8. Each committee member C_i checks if π is a valid proof using the NIZK's verification algorithm V:

$$V(\sigma, p_{id}, \pi) \stackrel{?}{=} true$$

   If so, C_i and party p_{id} engage in the DPSS **Reconstruction Phase** using $\mathsf{dpss\text{-}data}_i$.

Protocol 6 eWEB

1. User initiate secret storage via SECRETSTORE (3).
2. User initiate secret release via SECRETRELEASE (5).
3. Each round, miners execute SECRETSHANDOFF (4).

encryption schemes [34], which often require the decrypter to, say, compute a long series of repeated modular squarings.

Dead-man's Switch. A dead-man's switch is designed to be activated when the human operator becomes incapacitated. Software versions of the dead-man's switch typically trigger a process such as making public (or deleting) some data. The triggering event, for centralized software versions, can be a user failing to log in for three days, a GPS-enabled mobile phone that does not move for a period of time, or a user failing to respond to an automated email. A dead-man's switch can be seen as insurance for journalists and whistleblowers.

A dead-man's switch can use our PublicWitness protocol as follows: the user who wishes to setup the switch generates a SecretStore request with the desired release condition. Such condition can be failing to post a signed message on the blockchain for several days or anything publicly verifiable. As in the time-lock example, we can either use the standard scheme where a person (e.g., a relative or a friend) proves to the miners that the release condition has been satisfied or define an "automatic" request where the miners periodically check the release condition.

Fairness. eWEB can be used to support fair exchange, which ensures that two parties receive each other's inputs atomically. Using eWEB, Alice specifies a release condition that requires a signature from her and from Bob, while Bob's release condition requires only a signature from Bob. Once both secrets are posted, Alice verifies Bob's release condition and posts her signature. When Bob posts his signature, the committee releases both their secrets atomically. Fair exchange can be used to build fair MPC [19, 35].

Multi-party computation (MPC) is considered fair if it ensures that either all parties receive the output of the protocol, or none. In the standard model, fair MPC was proven to be impossible to achieve for general functions when a majority of the parties are dishonest [36]. However, we can achieve it by simply adapting the construction of Choudhuri et al. [27] to use our eWEB protocol, instead of traditional witness encryption. Conveniently, Choudhuri et al.'s scheme relies on a public bulletin board, which is most readily realized in practice via a blockchain-based ledger. Thus, by replacing witness encryption with our blockchain-based scheme, we do not add any extra assumptions to Choudhuri et al.'s construction.

One-time Programs. A one-time program is a program that runs only once and then "self-destructs". This notion was introduced by Goldwasser et al. [37]. In the same work they presented a proof-of-concept construction that relies on tamper-proof hardware. Considerable work on one-time programs followed [15, 38-40], but all such schemes relied on tamper-proof hardware. Goyal and Goyal [13], however, present the first construction for one-time programs that does not rely on tamper-proof hardware (but does rely on extractable witness encryption). As with fair MPC and Choudhuri et al.'s construction, by replacing the witness encryption scheme with our eWEB protocol in the Goyal and Goyal's one-time program construction with public inputs, we are not adding any extra assumptions since they already rely on blockchains.

Note that since eWEB reveals whether a secret was retrieved, additional mechanisms are needed in the case where the inputs submitted to the one-time program must be kept private.

Non-repudiation/Proof of Receipt. A protocol allows repudiation if one of the entities involved can deny participating in all or part of the communication. With eWEB, it is easy to provide a proof that a person received certain data. In this case, the user providing the data stores it using the SecretStore protocol. To satisfy the release condition F, a user with public identifier pid publishes a signed message "User pid requests the message". The miners then securely release a secret to the user pid as specified by SecretRelease. The publicly verifiable signature on the message "User pid requests the message" then serves as a proof that party pid indeed received the data.

3.4.1 Voting Protocol

As a more detailed example, we show how eWEB can support a "yes-no" voting application. Specifically, using eWEB, each voter can independently and asynchronously cast their vote by secret sharing a −1 for a "no" or a 1 for a "yes" (note that (0, 1) voting can be supported as well). When voting closes, the miners release an aggregate of the votes. The vote of any specific client must be kept private (guaranteed by eWEB's secrecy), and no client should be able to manipulate the result more than with his own vote.

To prevent improper votes, the committee must verify the correctness of the secrets shared by the clients, i.e., that each $s \in \{-1, 1\}$. Our key idea is to let each client first commit to its secret and then prove its correctness to the miners. However, this requires the client to prove that the committed value is the same as the value the client shared to the committee. To avoid this expensive check, committee members will instead compute the necessary commitment using the secret shares they receive from the client (guaranteeing consistency by construction).

We show that the committee members can prepare Pedersen commitments [41] for all of the clients with constant amortized cost.

For a client's secret s, the resulting commitment is of the form $c = g_s h_z$, where z is a random value (known to the client) and g,h are publicly known generators with $h = g^{\beta}$ for some unknown β.

With such a commitment, the user can prove $s \in \{-1, 1\}$ by proving $s^2 = 1$. To prove that $s^2 = 1$, the client (who knows s and z) computes $w = g^{2sz}h^{z2}$ and publishes w to all parties. To check that $s^2 = 1$, anyone can check that:

$$e(c, c) = e(g, g) \cdot e(h, w).$$

Correctness. To show correctness, note that

$$\begin{aligned} \text{LHS} &= e(g^{s+\beta z}, g^{s+\beta z}) = e(g, g)^{s^2+2\beta sz+\beta^2 z^2} \\ \text{RHS} &= e(g, g) \cdot e(g^{\beta}, g^{2sz+\beta z^2}) = e(g, g)^{1+2\beta sz+\beta^2 z^2}. \end{aligned}$$

Therefore, if the equation holds, then we have $s = 1$ and thus the vote submitted by the client is valid.

To compute the voting result the committee computes the sharing of the result relying on the linear homomorphism of KZG commitments and Shamir's secret sharing, and then follows the usual SecretRelease procedure.

3.5 Implementation

We implement both our eWEB scheme and our new DPSS scheme in about 2000 lines of Python code. To perform the underlying field and curve operations, we add Python wrappers around the C++ code of the Ate-Pairings library. For networking, we rely on gRPC, and for hashing, we use SHA256. For our NIZK scheme, we currently use Schnorr's proof of knowledge. We make it non-interactive via the Fiat-Shamir heuristic, thus simultaneously making it simulation extractable.

Polynomial arithmetic is done over the polynomial ring $F_p[X]$ for a 254-bit prime p. For the KZG commitment scheme, we use an ate pairing over Barreto-Naehrig curves of the form $y = x + b$ for constant b over F_p with a 254-bit prime p. We implement polynomial interpolation for polynomials of degree n in time $O(n \log^2 n)$ using an algorithm presented by Aho et al..

3.6 Experimental Evaluation

Our eWEB scheme offers a practical option for extractable witness encryption. We evaluate its costs and show that:

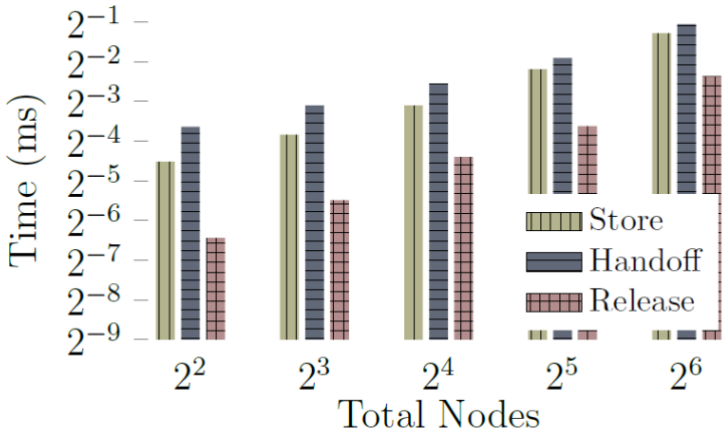


Figure 3.3: Time required for high-level eWEB steps on a LAN. Non-DPSS operations are too small to see.

Our eWEB prototype's performance matches the expected asymptotics with small constants, making it practical to integrate with existing blockchains.

eWEB's performance is dominated by our new DPSS scheme, which outperforms the state-of-the-art.

Setup. We run experiments using CloudLab [20], an NSF-sponsored testbed that provides compute nodes along with a configurable networking substrate. We run experiments in both a LAN setting (~0.2 ms ping) to focus on the CPU overhead of our cryptography and a WAN setting (~40 ms ping) to demonstrate the networking overhead. In the LAN setting we use up to 128 machines each with 8-core 2.00GHz CPUs and 4 GB RAM. In the WAN setting we use up to 128 machines split between Salt Lake City, Utah and Madison, Wisconsin. These machines have 8–10 cores and 2.00–2.4GHz CPUs with 2–4GB RAM.

Since eWEB is compatible with a wide range of blockchains, we abstract away the blockchain and simulate it via a single trusted node. In practice, writes to the blockchain will incur additional blockchain-specific latency.

3.6.1 eWEB Performance

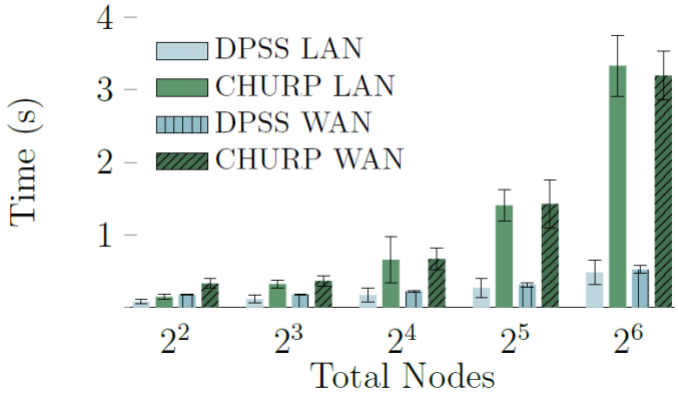


Figure 3.4: Handoff Times for our DPSS vs. CHURP. Error bars represent 95% confidence interval.

Our first experiment measures the costs of eWEB's three top-level operations (SecretStore, SecretsHandoff , and SecretRelease) for the minimal Schnorr identification application over an increasing number of committee members. In particular, given a public key, committee members release the secret if a client proves (in zero-knowledge) that they possess the associated secret key.

Figure 3.3 summarizes the average time for 150 runs (note the log-log scale). Each bar shows the split between eWEB operations (e.g., preparing the NIZK proof) and the underlying DPSS operations. Note that the time for SecretsHandoff includes the amortized cost for the preparation phase that produces coupled sharings of random value used during the refresh phase. Similiarly the time for SecretStore includes the amortized cost for the preparation phase that produces sharings for random values used to distribute the initial secret.

Unsurprisingly, the DPSS costs dominate, to the point where the time for eWEB operations cannot be seen. Note that the performance results match our expectation of linear asymptotic growth, and concretely costs ~7.3 milliseconds/node, ~10.7 milliseconds/node, and ~3.0 milliseconds/node for the store, refresh, and release secret operations respectively. This suggests if CloudLab allowed us to scale beyond 64 nodes per committee, we would expect eWEB to

store, refresh, release secrets in 7.3 s, 10.7 s, and 3.0 s respectively, even with a 1000-node committee.

3.6.2 DPSS Comparison

As discusses, the most efficient prior DPSS schemes are CHURP and that of Baron et al. Since CHURP reports that their performance dominates that of Baron et al., we focus our comparison on CHURP.

In our experiment, we measure the time required for each scheme to handoff secrets to a new committee in the optimistic case where parties behave honestly. Both schemes have a fallback path for when malfeasance is detected; it adds an $O(n)$ factor to both schemes.

Figure 3.4 summarizes the average time for 50 runs. As expected from our asymptotic analysis, our DPSS scheme increasingly out-performs CHURP as the number of nodes increases, to the point where our scheme is ∼7× faster than CHURP with 64 nodes. Naturally, the absolute difference will increase as committee sizes grow.

Note that the additional networking overhead in the WAN setting (∼40 ms latency) only significantly affects the end-to-end latency for committees with less than 8 members for both our DPSS scheme and CHURP. For larger committees, computation dominates networking costs even with realistic latencies.

3.6.3 Applicaitons

We implement several applications on top of our eWEB protocol in order to demonstrate practicality and efficiency for common use cases. As a baseline we implement the minimal Schorr identification application: Given a public key, committee members release a secret when provided a (zero-knowledge) proof that a client possess the associated private key. Because the Schnorr identification protocol only requires a few additional group operations for both the client and committee members, this gives us the best view of the core eWEB operational cost.

We additionally implement time-lock encryption and dead-man's switch as described. In both applications, a claim that the prescribed amount of time has passed is treated as the "witness". In the latter application, we additionally implement an update functionality that allows an operator to extend the secret-release timeout if they provide a valid signature.

We implement the fair exchange protocol as described. In particular, given valid signatures from two clients, the committee releases both their secrets atomically.

	Committee Size				
	4	8	16	32	64
Schnorr Identification	0.15 s	0.22 s	0.37 s	0.67 s	1.22 s
Time Lock Encryption	0.15 s	0.22 s	0.36 s	0.66 s	1.25 s
Dead-man's Switch	0.15 s	0.23 s	0.37 s	0.68 s	1.23 s
Fair Exchange	0.19 s	0.27 s	0.44 s	0.78 s	1.45 s

Table 3.1: Cost of eWEB applications. End-to-end latency including secret store, a single handoff, and secret release. (50 trials)

3.6.4 Microbenchmarks

To better understand the sources of overhead in our protocol, we measure the costs of the underlying primitives (Table 3.1). Because EC operations on the "twist" curve are about twice as expensive as on the base curve, we perform most of the protocol operations over the latter.

Ultimately, a party only needs to perform $2n$ operations on the twist curve whenever a shared secret needs to be reconstructed.

Operation	Time
Field Addition	6.45 μs
Field Multiplication	7.35 μs
Field Division	5.52 μs
Curve Addition	6.27 μs
Curve Scalar Mult	234 μs
Pairing	443 μs

Table 3.2: Cost of Field & Curve Operations. (500 trials)

	Degree		
	16	32	64
Setup	5.0 ms	9.5 ms	18.6 ms
Polynomial Commit	4.9 ms	9.7 ms	18.7 ms
Polynomial Eval	5.3 ms	10.7 ms	21.0 ms
Polynomial Verify	1.7 ms	1.7 ms	1.7 ms

Table 3.3: Cost of KZG Commitment Scheme. eWEB uses KZG to verify share corrections (500 trials)

We also measure and report the costs of the core operations in our implementation of the KZG commitment scheme (Table 3.2). Setup refers to the time required to generate a public key for a polynomial with a given degree. This is a one-time cost across all committed polynomials.

3.7 Related Work

3.7.1 Prior Work on DPSS

Since the introduction of proactive secret sharing by Herzberg et al., many PSS schemes have been developed. These schemes vary in terms of security guarantees, network assumptions (synchronous or asynchronous), communication complexity and whether they can handle dynamic changes in the committee membership. In Figure 1 we provide a comparison of PSS schemes.

Below, we compare our DPSS construction in detail with the two constructions (CHURP, Baron et al.) that are most closely related to our protocol, as they are also the most efficient DPSS schemes to date.

In the best case, CHURP achieves communication complexity $O(n^2)$ plus $O(n) \cdot$ B to refresh each secret in the hand-off phase, where n is the number of parties and B denotes the cost of broadcasting a bit. In the worst case (where some corrupted party deviates from the protocol), it requires $O(n^2) \cdot$ B communication per secret.

In the single-secret setting, our protocol achieves the same asymptotic communication complexity as CHURP. However, our protocol achieves amortized communication complexity $O(n)$ plus $O(1) \cdot$ B per secret in the best case, and $O(n^2)$ plus $O(n) \cdot$ B per secret in the worst case. Batching is crucially important in eWEB since there may be thousands of secrets stored at any given time.

While CHURP uses asymmetric bivariate polynomials to refresh a secret during hand-off, we use a modified version of a technique of Damgård et al. to prepare a batch of random secret sharings. Generating random secret sharings is much more efficient than generating bivariate polynomials. Specifically, each bivariate polynomial requires $O(n)$ communication per party; i.e., $O(n^2)$ in total. On the other hand, we can prepare $O(n)$ random sharings with the same communication cost as preparing one bivariate polynomial. To benefit from it, we use a entirely different way to refresh secrets.

The work focuses on a slightly different setting from ours: they consider unconditionally secure DPSS with a $(1/2 - \epsilon)$ corruption threshold, where ϵ is a constant. Their scheme has $O(1)$ amortized communication per secret. However, in the single-secret setting, it requires $O(n^3)$ communication to refresh a secret.

Baron et al. use packed secret sharing (in contrast to our batched secret sharing), which allows a client to store $O(n)$ secrets in one sharing by encoding multiple secrets as distinct points of a single polynomial. Thus, refreshing each sharing effectively refreshes a batch of $O(n)$ secrets submitted by the same client at the same time. However, this means that secrets in the same batch come from a single client and can only be refreshed or reconstructed together. It is unclear if merging batches submitted by different clients is at all possible. Thus, even if in eWEB some secrets submitted by different clients were to be released at the same time, Baron

et al.'s scheme would not allow us to join these secrets in one batch to profit from their low amortized communication complexity. Our scheme has one secret per sharing: only supplementary random sharings are generated in a batch. Then, each secret is refreshed (and can be released) individually.

To reach $O(1)$ amortized communication per secret, Baron et al. need a $(1/2 - \epsilon)$ corruption threshold. Our scheme does not suffer from this corruption threshold loss.

3.7.2 Extractable Witness Encryption and Conditional Secret Release

The notion of witness encryption was introduced by Garg et al. [14]. Extractable security for witness encryption was proposed by Goldwasser et al. In their work a candidate construction was introduced that requires very strong assumptions over multilinear maps. According to Liu et al., existing witness encryption schemes have no efficient extraction methods. Garg et al. suggest that it even might be impossible to achieve extractable witness encryption with arbitrary auxiliary inputs.

Nevertheless, as mentioned in previous sections, the notion of extractable witness encryption has been extensively used in various cryptographic protocols, especially in conjunction with blockchains.

Benhamouda et al. recently published a manuscript that also proposes conditionally storing secrets on a blockchain. Unlike eWEB, their work is specific to proof-of-stake blockchains. Like us, they design a new DPSS scheme, but they target a very specific (albeit intriguing!) use case — in their setting, the members of a committee must remain anonymous, even to the previous committee. They consider a stronger adversary who can corrupt and uncorrupt previously honest parties, but they only tolerate 25% corruption, versus 50% for our scheme.

Finally, they do not explain how to release secrets without revealing witnesses to the miners, nor provide an implementation.

Kokoris-Kogias et al. proposed Calypso, a verifiable data-management solution that relies on blockchains and threshold encryption. Calypso targets a different use case than our eWEB system: it allows verifiable sharing of data to parties that are explicitly authorized (either by the depositor or by a committee of authorized parties) to have access rather than specifying a general secret release condition that allows anyone who is able to satisfy this condition to get the data. Kokoris-Kogias et al. do not provide a formal security definition or formal security proof of their system. A major part of their work focuses on the static committee of parties holding the secrets, the dynamic committee setting is only discussed very briefly.

eWEB could be seen as a special case of proactive secure multi-party computation (PMPC) [8, 22, 42]. However, while our DPSS scheme could be used for PMPC, eWEB targets a different use case than general PMPC. This allows for a much more efficient and largely non-interactive construction compared to PMPC protocols, which typically have very high round complexity.

3.8 Conclusion

We have introduced a new cryptographic primitive: extractable witness encryption on a blockchain, which allows the blockchain to store and release secrets. We designed a proof-of-concept eWEB protocol and implemented it. A key building block of this protocol is a highly efficient batched dynamic secret sharing scheme that may be of independent interest. This framework is compatible with the data from the sensor network described in chapter 2.

CHAPTER FOUR

Security Policies and Two-Factor Authentication

Bitcoin and blockchain-based cryptocurrencies brought us at the brink of a technological revolution: These systems allow us to bypass the need for centralized trusted entities to run currencies on a large-scale. While their decentralized and borderless nature make them appealing substitutes for Fiat currencies, a burning problem with this approach is the lack of a recovery mechanism if something goes wrong. What happens if one user's key is lost or stolen? There is no bank to call and no authority to rely upon to get your funds back. Indeed, the number of such high-profile attacks on cryptocurrencies has been rising steadily. Famous incidents like the Mt. Gox hack [42], the coincheck hack, and the Parity Technology code deletion saw funds upwards of several hundred million dollars being lost or stolen. The well-known DAO hack [43] forced Ethereum to adopt a hard fork which created two version of the currency: Ethereum and Ethereum classic. To deal with such problems in future, Ethereum developers have proposed EIP 867. An ethereum improvement protocol (EIP) is the process by which code changes get accepted onto the Ethereum platform. However, EIP 867 has proven to be controversial since it will again bifurcate the currency into two versions, the very problem it was trying to solve.

The problems of attackers stealing money or losing credentials are of course not unique to blockchain-based cryptocurrencies. The (traditional) banking industry has been dealing with such issues for decades where a number of mitigating approaches have worked pretty effectively. Examples of common security policies to deal with such attacks include: A waiting period (say 24 hours) for transferring money to a new recipient and an overall daily outgoing transfer limit. Bypassing these restrictions could either require additional verifications (such as two-factor authentication), or may not be allowed at all. Different banks could follow different security policies which might also vary depending on the type of account (business vs personal) and by the state/region and the local laws. This motivates the following question:

> *Can we take the lessons learnt in the traditional banking domain, and apply them fruitfully to blockchain-based systems, without compromising their decentralized nature?*

4.0.1 System Architecture

We propose a system where each user has the opportunity to delegate the custody of its funds to a smart contract. The security policy (which can be specified by the user itself) is hardwired in the smart contract and it governs its decision. All funds transfer will go through the approval of such a smart contract, which has the power to accept or reject (or even set on hold) each transaction, depending on the specified policy. In case of an exceptional event, such as key theft, or as an additional security safeguard in case of particularly large transactions or transactions to new addresses, the smart contract will require additional verification via two-

factor authentication (2FA). We stress that one does not need to authenticate all transactions but only a carefully chosen set, at the discretion of the policy specified by the owner of the funds.

In this work we implement the above idea by developing solutions for 2FA mechanisms that can coexist with the decentralized nature of blockchain-based currencies. To be aligned with the philosophy of decentralized system, our solutions are guided by the following design principles.

Distributed Trust. The 2FA must not rely on the existence of a trusted authority. Instead we propose to leverage a hardware token or distribute the trust among a (reasonably-sized) set of parties, which are asked to intervene in case of exceptional event. The members of such a committee can be chosen by the users or can be selected through a consensus protocol (e.g., the miners themselves can play this role). Security must be guaranteed even if a subset of the committee members behaves maliciously.

Reliability and Guaranteed Output Delivery. We require the 2FA mechanism to be resilient against (benign) disconnections and (malicious) denial of service attacks. That is, even if part of the members of the committee go offline or become corrupted, one should still be able to complete the authentication procedure (given that the set of online parties is large enough).

Low Latency. In distributed environments communication is typically expensive, as messages have to be broadcasted to all users in the network. For this reason it is of central importance to minimize the rounds of communication of a 2FA mechanism.

Computational Efficiency. General purpose cryptographic solutions are very powerful but are often computationally intensive. We aim to build a solution based on well-established cryptographic components which is efficient enough to be integrated in real-life systems.

4.0.2 Two-Factor Authentication Mechanisms

A popular approach to 2FA in the traditional banking system is that of security questions: When the 2FA is triggered, the user is prompted to answer a personal question and the verification process succeeds if the user's answer is correct. If we were to import this idea to the decentralized setting, the first question that arises is where to store the correct answers in absence of a trusted authority. Clearly the smart contract cannot hardwire the correct answers as otherwise they would be public knowledge (smart contracts do not offer any form of privacy). Since answers typically come from a low-entropy distribution, storing the hash of them is also not a good idea since it is vulnerable to off-line dictionary attacks, where an attacker recovers the hash and tests locally his guesses until he succeeds.

Our idea to bypass this obstacle is to secret share the correct answers among a set of n parties, in such a way that stealing the secret would require to corrupt at least t members of this committee (for some threshold parameter $t \leq n$). When the 2FA mechanism is triggered, the user is asked to authenticate its transaction τ by providing the correct answers to a set of predefined questions. The user then broadcasts a message to all committee members, who

perform some computation locally and output a partial response tied to the transaction τ. The smart contract then collects sufficiently many responses and publicly checks whether the authentication was successful. If this is the case the smart contract allows τ to go through, otherwise it rejects it. In terms of security, we require that the above process does not reveal anything beyond whether the user's guess was correct or not, even if the attacker corrupts a certain subset of committee members. Henceforth, we refer to this procedure as a distributed zero-tester (DZT).

We also consider an alternative setup where we leverage a physical two-factor authentication token. As an example, universal two-factor authentication (U2F) tokens allow a user to sign any message by querying the token. With this tool at hand, authentication is done as follows: The smart contract hardwires the verification key of the U2F token and, when the 2FA mechanism is invoked, sends the user a nonce. The authentication is successful if the user provides a correct signature on the nonce.

4.0.3 Security Policies

Our system is completely flexible in the policy that is enforced by the smart contract, which can be specified by the owner of the address. A more conservative usage of our system is just as an additional safeguard mechanism: Suspicious transactions (e.g., unusually large amounts or transactions to a new address) can trigger the 2FA mechanism and force the user to provide (human-memorable) answers to some security questions or a signature from a physical token. This gives an additional line of defense even against the catastrophic event where an attacker steals a user signing key.

On the other side of the spectrum, one user may want to set the policy such that some transactions are allowed given only the 2FA, i.e., they are not required to be digitally signed. This can be useful in scenarios where a user has lost the signing key for an address and wants to recover the funds: Answering some security questions allows him to transfer the coins to a fresh address (with a newly sampled signing key). However this liberal usage of our system requires careful thoughts. While on the one hand it improves the usability of the currency, on the other hand it opens the doors to a new attack vector: instead of stealing a secret key, an attacker can compromise the address by guessing the answers to some security questions (or stealing a physical token), which is typically a much easier task. This risk can be mitigated by rate-limiting the amount of attempts for unsigned transactions to, e.g., once per week. Since each attempt requires the user to post a message over the blockchain, the query limit can be enforced by the smart contract itself, without the need for the committee members to coordinate or to update the code of the physical token.

In this work we mainly focus on the former case, where the 2FA mechanism is invoked in addition to the standard digital signature check. An in-depth cost analysis of the security and usability tradeoff of unsigned transactions is beyond the scope of this work.

4.0.4 Our Contribution

In this work we propose a new method to safeguard transactions on blockchain-based cryptocurrencies, inspired by the solutions developed in the (traditional) banking domain. We then develop 2FA mechanisms amenable to the decentralized nature of cryptocurrencies, offering different trade-offs in terms of trust assumptions, physical capabilities, and computational efficiency. Our technical contributions can be summarized as follows.

Definitions. We develop the notion of distributed zero-tester (DZT), the central cryptographic building block that enables efficient 2FA in decentralized system. We give formal definitions for this primitive and we characterize the security requirements with a game-based definition.

Cryptographic Protocol. We propose a cryptographic instantiation of DZT from standard assumptions on bilinear groups. The scheme is round optimal, has guaranteed output delivery, and is concretely efficient. Along the way, we develop a new threshold homomorphic encryption scheme for linear predicates from bilinear groups, which might be of independent interest.

Implementation. We implement the DZT scheme as a smart contract in Ethereum. Our performance evaluation shows that, for reasonably-size committee, the resulting 2FA mechanism can be deployed by today's users at minimal additional cost (around 1$ per authenticated transaction). We also implement a U2F-based 2FA mechanism as a smart contract in Ethereum, thus enabling 2FA with a hardware token already available to the public. In terms of added cost, our scheme introduces (approximately) an additional 3¢to 28¢ per transaction, depending on the choice of curve.

4.1 Technical Overview

In the following section we give an informal exposition of the techniques developed in this work. We focus on the main goal of designing a crytpographic solution for a DZT: In a DZT a client secret shares an answer α (the primitive naturally extends to the settings of multiple answers) among a set of n parties. When the 2FA is invoked, the client (which has an attempt $\tilde{\alpha}$ in his head) crafts a single query q and sends q (together with the transaction identifier τ) to all parties, who locally compute a response pi. Given a large enough set of responses $\{pi\}_{i \in S}$, for some set S of size $|S| = t$, anyone can publicly determine the outcome of the authentication process. The transaction τ is successfully authenticated if and only if $\alpha = \tilde{\alpha}$ and the protocol should not leak any information beyond whether the authentication process succeeded or not.

4.1.1 A Generic Solution

We first discuss a high-level idea of our solution and then we present an efficient cryptographic instantiation. Threshold fully-homomorphic encryption [44], [45] allows us to compute any function over encrypted data and offers a general solution to our problem: The client can simply compute Enc(pk,α) and distribute the shares of the secret key (sk1,...,skn) to

the committee members. To authenticate a transaction, a user computes Enc(pk,α˜) and broadcasts it to all parties. Then the committee members locally compute, using the homomorphic properties of the scheme,

$$c = \text{Enc}\left(\text{pk}, \alpha \stackrel{?}{=} \tilde{\alpha}\right)$$

and compute and output the partial decryption using their share ski. The plaintext bit {0,1} of c can be publicly reconstructed using a large enough set of decryption shares and the output of the authentication is set to be this bit.

While this solution satisfies all of our security requirements, it introduces a prohibitively high computational overhead. As of today, implementing a generic fully-homomorphic encryption in an Ethereum smart contract is far off the reach of the current infrastructure. Thus, developing a solution that can be used in today's systems requires us to devise a different strategy.

4.1.2 A Flawed Attempt

In order to understand our solution, it is instructive to iterate through a simple construction and analyze its pitfalls. In the setup phase, the client samples an ElGamal [46] key pair (x,h = gx) and encrypts the answer α under such a key, making the resulting ciphertext

$$(c_0, c_1) = (g^r, h^r \cdot g^{\alpha})$$

publicly available to all parties. The secret key x is then secret shared (using Shamir scheme [47]) among n parties in such a way that any t shares are sufficient to reconstruct the secret. Let (xi,i) be the i-th share. To authentication a transaction τ, a user can compute the encryption of its attempt

$$(d_0, d_1) = (g_s, h_s \cdot g^{-\alpha})$$

and broadcast it to all members of the committee. Each party then computes and broadcasts $p_i = (c_0 \cdot d_0)^{x_i}$. Given a set S of responses, the outcome of the authentication can be publicly recovered by checking

$$\prod_{i \in S} p_i^{\lambda_i} \stackrel{?}{=} c_1 \cdot d_1$$

where λi is the i-th Lagrange coefficient. Note that if α = α˜, then c1 · d1 = hr+s and indeed the above equation holds true since the secret x is reconstructed in the exponent. Unfortunately, this solution has multiple flaws. First of all, there is no mechanism that ties the transaction τ to the authentication process so one can authenticate without the knowledge of α by simply replaying a valid ciphertext (c0,c1) and swapping the corresponding transaction τ with a new τ˜. Furthermore, even if a single member provides a malformed answer, the entire mechanism for verification fails.

While these issues can be resolved using non-interactive zero-knowledge proofs (NIZK) [48], there is a more serious problem: If the authentication is not successful, then the responses of the committee members reveals the exact difference $\alpha - \tilde{\alpha}$. This is because

$$\frac{c_1 \cdot d_1}{\prod_{i \in S} p_i^{\lambda_i}} = \frac{h^{r+s} \cdot g^{\alpha - \tilde{\alpha}}}{g^{(r+s)\sum_{i \in S} x_i \lambda_i}} = \frac{h^{r+s} \cdot g^{\alpha - \tilde{\alpha}}}{h^{r+s}} = g^{\alpha - \tilde{\alpha}}$$

which means that the attacker can always guess the correct α with at most two queries. This is a notorious issue in threshold cryptography, and current approaches to resolve this problem (see, e.g., [49-51]) require one to add two rounds of interaction. This is not acceptable for us, since we consider settings where communication is expensive (i.e., each message is posted on a blockchain) and thus we aim at a completely non-interactive solution.

4.1.3 Bilinear Maps at Rescue

The above vulnerability comes from the fact that the threshold version of ElGamal encryption does not satisfy the notion of simulation security for equality predicates, i.e., instead of revealing whether two strings are equal or not it reveals the exact difference. This problem can be bypassed by resorting to more powerful cryptographic machinery.

Our first observation is that the class of predicates that we need to evaluate is very restricted, so there is hope to build a solution without the full power of fully-homomorphic encryption. Our second observation is that the protocol is already secure if the authentication is successful, so we only need to worry about the case where the answer is not correct, i.e., $\alpha \neq \tilde{\alpha}$.

Our central idea is to revisit the ElGamal-based approach by adding a re-randomization factor to the plaintext of the evaluated ciphertext, which cancels out only if $\alpha = \tilde{\alpha}$. This can be done with the help of bilinear groups. In a bit more detail, in the setup phase, the client publishes

$$(c_0, c_1, c_2, c_3) = (g^r, h^r \cdot g^{\rho\alpha}, g^{\tilde{r}}, h^{\tilde{r}} \cdot g^{\rho}) \in \mathbb{G}_1^4$$

where h = gx. The secret key x is secret shared as before. Note that (c0,c1) serves the same role as before, except that there is an additional factor ρ multiplied by the answer α. The role of (c2,c3) will be clear in a moment. The authentication begins with the user computing and broadcasting the encryption

$$(d_0, d_1) = (g^s, h^s \cdot g^{-\rho\tilde{\alpha}}) \in \mathbb{G}_1^2$$

Note that the user does not know the factor ρ but can still compute a valid ciphertext for $-\rho\tilde{\alpha}$ by obliviously raising (c2,c3) to the power of $-\tilde{\alpha}$ and then re-randomizing the resulting ciphertext.

Each committee member computes $(c0 \cdot d0)^{xi}$ as before, except that it additionally samples a fresh element $k = g^{\kappa} \in G2$ using a random oracle and returns

$$p_i = e\left((c_0 \cdot d_0)^{x_i}, k\right) = e(g, g)^{(r+s)x_i\kappa} \in \mathbb{G}_T$$

as the decryption share. The outcome of the authentication process can be recovered by checking

$$\prod_{i \in S} p_i^{\lambda_i} \stackrel{?}{=} e(c_1 \cdot d_1, k).$$

To see why security is preserved even in the case of a failed authentication, observe that

$$\frac{e(c_1 \cdot d_1, k)}{\prod_{i \in S} p_i^{\lambda_i}} = \frac{h^{(r+s)\kappa} \cdot g^{\kappa\rho(\alpha-\tilde{\alpha})}}{g^{(r+s)\kappa \sum_{i \in S} x_i \lambda_i}} = \frac{h^{(r+s)\kappa} \cdot g^{\kappa\rho(\alpha-\tilde{\alpha})}}{h^{(r+s)\kappa}}$$

so in case $\alpha = \tilde{\alpha}$ the randomization factor $\kappa\rho$ cancels out and the above equality is verified. For the case case $\alpha \neq \tilde{\alpha}$, the factor $\kappa\rho$ completely masks the difference $\alpha - \tilde{\alpha}$ and prevents the attack as described above. Since the value of $k = g^{\kappa}$ is freshly sampled upon each attempt, the randomization factor is pseudorandom for each authentication query. Also note that k can be sampled locally by each party, without the need of any additional interaction.

4.1.4 Additional Challenges

The above presentation glosses over many important aspects that need to be taken into account when building an efficient protocol. As an example, the above protocol must be augmented with NIZK proofs to make sure that a malicious player cannot deviate from the specification of the protocol. However, using generic NIZK schemes for NP would vanish our efforts to build a practical system. Fortunately we show that our scheme can be slightly tweaked to allow us to implement the required NIZKs using exclusively Schnorr proofs for discrete logarithm equality [52], which results in a concretely efficient scheme.

Another set of challenges arises when implementing the protocol as a smart contract in Ethereum. In order to obtain an efficient protocol, we would like to leverage precompiled instructions to perform bilinear group operations. However, the semantic of operations supported by Solidity (the language of Ethereum smart contracts) is very limited: It only supports group operations in the source G1 and pairing-product equation checks. This turns out to be insufficient to implement the scheme as outlined above. To circumvent this issue, we further modify our construction to make it fully compatible with precompiled instructions in Solidity. This process is not hassle-free: The new scheme requires us to introduce a new (static) assumption over bilinear groups, the dual Diffie-Hellman assumption. We then show that such an assumption holds true in the generic group model.

Also, the integration of a U2F-based solution in Ethereum introduces some additional complications. As an example, a smart contract cannot implement a randomized algorithm, but we need to sample a random challenge to complete the U2F authentication protocol. Instead of a truly random string, we use the hash of a unique identifier of the transaction (to prevent replay attacks) to implement the challenge sampling procedure.

4.2 Related Work

In the following we survey some related work from the literature.

Multi-Sig Addresses. One of the most prominent approaches to safeguard accounts in cryptocurrencies is the creation of multi-sig addresses: A secret key of a digital signature scheme is secret shared among multiple parties using a threshold scheme [50, 53-55]. Approving transaction requires gathering a large enough set of users to jointly sign it. This means that compromising a single device is no longer sufficient to steal coins.

Our approach complements this idea and adds two important features: (1) Multi-sig address require to gather multiple parties to sign every transaction regardless of how small or big it is, whereas in our system a smart security policy can decide when additional verification is required. To the best of our knowledge, there has been no previous work which has this feature. (2) Our system supports also low-entropy secrets (e.g., human-memorable passwords or answers to security questions) that can be used as an additional verification, in case one of the parties loses his credentials.

Password-Protected Wallets. Another approach to improve the usability of blockchain systems is the so-called password-protected wallet[56], where the secret keys are encrypted using a human-memorable password. Unfortunately, these systems are vulnerable to exhaustive-search attacks where one can enumerate all plausible passwords and attempt to recover the secret. A DZT can be used to overcome this limitation and to construct a form of distributed password protected wallet across some committee members.

Vaults. Cryptocurrency vaults aim at disincentivizing key theft by delaying the acceptance transactions: Once an illegitimate transaction is placed on the blockchain, the legitimate owner has enough time to prevent the spending using an appropriate recovery key. The aim of this mechanism is to reduce the incentive for an attacker to steal keys, as it will likely result in no financial gain. However, it does not address the question of recovering access to an account, in case of unintentional credential losses.

Password-Authenticated Key Exchange (PAKE). An area which is closely related to our DZT is that of threshold PAKE [57, 58], where a human-memorable password is used to authentication a user against a set of servers, who collectively know the password but can be individually corrupted without compromising security. The main difference with respect to our settings is that we do not require to exchange any key, instead we want to tie the authentication process with a transaction chosen by the client. Additionally, we require that the outcome of the authentication process is publicly verifiable even by external parties, which is typically not the case for PAKE protocols.

Zero-Testing of ElGamal Ciphertexts. One of our main technical innovations is a new protocol where parties can jointly test whether an ElGamal ciphertext is an encryption of 0, without revealing any additional information. While this is a well-known problem in threshold

cryptography [59], known efficient solutions (see, e.g., [49-51]) are based on a commit-and-prove approach and require at least three rounds of interaction. In contrast, our solution is completely non-interactive: Each party broadcasts a single message, and the verification procedure is public. This is particularly suitable for settings where communication is expensive, e.g., where parties exchange messages by posting them on a blockchain.

Password-Protected Secret-Sharing (PPSS). A PPSS allows a user to share a secret that can be reconstructed with the sole knowledge of a password. Recent efficient protocols can be used as a substitute for DZT in our 2FA mechanism. However, PPSS would require at least one more round of interaction between the committee members and the client. In contrast, the shares in the DZT can be verified non-interactively and without the need to set up an off-chain communication infrastructure.

Distributed Point Functions (DPF). A DPF (and its generalization to function secret sharing) allows a set of parties to secret-share a point function $f \rightarrow (f1,...,fn)$ whose output can be recovered by the output of the local evaluation of each parties, i.e., $f(x) = f1(x) \oplus \cdots \oplus fn(x)$. A DZT can be thought as a threshold version of a DPF with a few important differences: (i) In a DZT the output shares of the parties are publicly verifiable, which is in general not the case for DPFs. Furthermore, (ii) in a DZT the output reconstruction is not restricted to be a linear function. Finally, (iii) DZT supports the evaluation of encrypted queries, whereas in a DPF the inputs are typically public.

4.3 Preliminaries

We denote by $\lambda \in N$ the security parameter. We say that a function $negl(\cdot)$ is negligible if it vanishes faster than any polynomial. Given a set S, we denote by $s \leftarrow\$ S$ the uniform sampling from S. We say that an algorithm is PPT if it can be implemented by a probabilistic machine running in time polynomial in the security parameter.

4.3.1 Bilinear Groups

Let (G1,G2,GT) be an asymmetric bilinear group of prime order p with an efficiently computable pairing $e : G1 \times G2 \rightarrow GT$ and let G be the generator of such a group. We denote by (g1,g2) the generators of the groups G1 and G2, respectively, and by $gT = e(g1,g2)$ the generator of GT. In the following we recall the eXternal Diffie-Hellman (XDH) [9] and the Decisional Bilinear Diffie-Hellman (DBDH) problems over bilinear groups.

Assumption 1 (XDH). Let G be a bilinear group generator. G is XDH-hard if for all PPT distinguishers it holds that the following distributions are computationally indistinguishable

$$(G_1, G_2, G_T, p, g_1, g_2, g_1^x, g_1^y, g_1^{xy}) \approx$$

$$(G_1, G_2, G_T, p, g_1, g_2, g_1^x, g_1^y, g_1^z)$$

where $(G1,G2,GT,p,g1,g2) \leftarrow\$ G(1^\lambda)$ and $(x, y, z) \leftarrow\$ \mathbb{Z}_p^*$.

Assumption 2 (DBDH). Let G be a bilinear group generator. G is DBDH-hard if for all PPT distinguishers it holds that the following distributions are computationally indistinguishable

$$(G_1, G_2, G_T, p, g_1, g_2, g_2^x, g_T^y, g_T^{xy}) \approx$$

$$(G_1, G_2, G_T, p, g_1, g_2, g_2^x, g_T^y, g_T^z)$$

where (G1,G2,GT,p,g1,g2) ←$ G(1^λ) and $(x, y, z) \leftarrow\$ \mathbb{Z}_p^*$.

4.3.2 Non-Interactive Zero-Knowledge

A non-interactive zero-knowledge (NIZK) proof allows a prover to convince a verifier about the validity of a certain statement without revealing anything beyond that. We recall the syntax in the following.

Definition 1 (NIZK). Let L be an NP-language with relation R. A NIZK system for R consists of the following efficient algorithms.

Setup(1λ) : On input the security parameter 1λ, the setup algorithm returns a common reference string crs.

Prove(crs,stmt,wit) : On input the common reference string crs, a statement stmt, and a witness wit, the prover algorithm returns a proof π.

Verify(crs,stmt,π) : On input the common reference string crs, a statement stmt, and a proof π, the verifier algorithm returns a bit {0,1}.

Correctness requires that for all λ ∈ N and all pairs (stmt,wit) ∈ R it holds that

$$\Pr[\text{Verify}(\text{crs},\text{stmt},\text{Prove}(\text{crs},\text{stmt},\text{wit}))] = 1$$

where crs ←$ Setup(1λ). We recall the definition of zero-knowledge in the following.

Definition 2 (Zero-Knowledge). A NIZK system for R is zero-knowledge if there exists a PPT algorithm (Sim0,Sim1) such that for all pairs (stmt,wit) ∈ R and for all PPT distinguishers the following distributions are computationally indistinguishable

$$(\text{crs} \leftarrow \text{Setup}(1^\lambda), \pi \leftarrow \text{Prove}(\text{crs},\text{stmt},\text{wit})) \approx (\text{crs}^*, \pi \leftarrow \text{Sim}_1(\text{crs},\text{stmt},\text{td}))$$

where $(\text{crs}^*, \text{td}) \leftarrow \text{Sim0}(1\lambda)$.

We require that the protocol satisfies the strong notion of simulation extractability.

Definition 3 (Simulation Extractability). A NIZK system for R is simulation-extractable if there exists a PPT algorithm

Ext and a negligible function negl(·) such that for all λ ∈ N and all PPT algorithms A it holds that

$$\Pr\left[\begin{array}{l|l} \mathsf{Verify}(\mathsf{crs},\mathsf{stmt},\pi) & (\mathsf{crs},\mathsf{td}) \leftarrow \mathsf{Sim}_0(1^\lambda) \\ \wedge\ \mathsf{stmt} \notin Q & \pi \leftarrow \mathcal{A}(\mathsf{crs})^{\mathcal{O}(\cdot)} \\ \wedge\ (\mathsf{stmt},\mathsf{wit}) \notin \mathcal{R} & \mathsf{wit} \leftarrow \mathsf{Ext}^{\mathcal{A}}(\mathsf{crs},\mathsf{td},\mathsf{stmt},\pi) \end{array}\right] = \mathsf{negl}(\lambda)$$

where O takes as input a (possibly false) statement stmt and returns Sim1(crs,stmt,td) and we denote by Q the list of queries issued by A.

4.3.3 Secret Sharing

We recall the threshold secret sharing scheme from Shamir over a field F where a randomized algorithm Share takes as input a field element s, a threshold t, and a number of participants n and returns the evaluations of a random (t−1)-degree polynomial at the respective points (1,s1),...,(n,sn). Then any subset S ⊆ {1,...,n} such that |S| = t can recover the secret s by computing

$$s = \sum_{i \in S} \lambda_i \cdot s_i$$

over F, where λi is the i-th Lagrange coefficient.

4.4 Definitions

In the following we present the notion of a distributed zero-tester (DZT) and we provide explicit security and privacy properties. Before delving into the formal definitions, we discuss on a high level our design goals.

4.4.1 Overview

A DZT allows one to share a secret among a selected committee of users in such a way that later on one can authenticate using the knowledge of such a secret, assuming that a large enough portion of committee members is online. Our definition for a DZT is tailored to distributed environments where communication is expensive and an arbitrary subset of parties may decide not to comply with the protocol specifications. Our definitions (and consequently the instantiation that we propose) are guided by the following design principles.

Round Optimality. For our applications of interest, the communication among committee members happens over a blockchain. This makes communication costly in both financial and efficiency terms: The rate of exchanged messages is bounded by the rate at which new blocks appear, which can be in the order of minutes. For this reason we aim to minimize interaction as much as possible. Ideally, an authentication attempt should consist of a single message from the user to all committee members and of a single round of response, where the output of the protocol can be recovered without any further interaction.

Guaranteed Output Delivery. As one cannot trust all committee members, a corrupted user might decide at any point not to respond to any query. We want to guarantee that

authentication protocols can still take place even if an arbitrary subset of committee members (up to some user-defined threshold) goes offline. This property is commonly known as guaranteed output delivery.

Public Verifiability. We require that the outcome of the authentication process is publicly verifiable by any external observer. This feature is needed to allow the smart contract to decide whether the user transaction should take place or not.

Resilience Against Offline Dictionary Attacks. The secrets stored by the user typically come from a low-entropy distribution (e.g., a human-memorable password or answer to some security question). For this reason, it is important the information that an attacker can gather by eavesdropping the communication does not allow it to launch bruteforce attacks where the authentication process is run locally until it succeeds. Clearly the attacker can query the online authentication mechanism, but an unusual number of attempts will trigger a rate-limiting mechanism.

Resilience Against Replay Attacks. In our context authentication is tied to a specific bitstring τ, e.g., we want to make sure that an unusual transaction τ is initiated by the legitimate user. It is therefore important to ensure that previous successful authentications for a string τ cannot be mauled into successful authentications for a different string τ', without the knowledge of the secret.

4.4.2 Syntax and Security Properties

In the following we present the syntax for a DZT protocol. To incorporate the fact that a query is associated with some transaction trans we augment our scheme with tags τ, which we assume to be λ-bit strings. This is without loss of generality since one can set $\tau = H(\text{trans})$, where $H : \{0,1\}* \rightarrow \{0,1\}\lambda$ is a collision resistant hash function.

Definition 4 (DZT). A DZT scheme consists of the following efficient algorithms.

Setup($1\lambda,t,n,\alpha$) : On input the security parameter 1λ, a threshold size t, a committee size n, and a string $\alpha \in \{0,1\}*$, the setup algorithm returns a public key pk and a vector of secret keys (sk1,...,skn).

Query(pk,β,τ) : On input the public key pk, a string $\beta \in \{0,1\}*$, and a tag $\tau \in \{0,1\}^{\lambda}$, the query algorithm returns a query q.

Response(pk,ski,q,τ,i) : On input the public key pk, a secret key ski, a query q, a tag τ, and an index i, the response algorithm returns a partial response pi.

Verdict(pk,p1,...,pt˜,τ) : On input the public key pk, a set of partial responses (p1,...,pt˜), for some $\tilde{t} \leq n$, and a tag τ, the verdict algorithm returns a bit {0,1}.

For correctness, we require that for all $\lambda \in N$, all polynomials $n = n(\lambda)$, all $t \leq n$, all sets $S \subseteq \{1,...,n\}$ of size

$|S| \geq t$, all strings α and τ, all (pk,sk1,...,skn) in the support of Setup(1λ,t,n,α) it holds that

$$\Pr[\text{Verdict}(pk,\{p_i\}_{i\in S},\tau) = 1] = 1$$

where pi ← Response(pk,ski,Query(pk,α,τ),τ,i). Note that correctness is required to hold as long as any set of committee members of size at least t participates in the response to the queries. In other words, the protocol has guaranteed output delivery as long as at most n − t parties are corrupted and may refuse to respond to queries or give an incorrect response.

Next we define the notion of security and we argue why it models the attackers capability in a faithful way. Our definition are inspired by those of Bellare, Pointcheval, and Rogaway [6] imported to our settings. We denote by D set of distinct strings from where the secret is chosen. Our definition captures both the cases where D is small and exponentially large, as long as one can efficiently sample an element from D.

Definition 5 (Security). A DZT is secure if there exists a negligible function negl(·) such that for all λ ∈ N and all PPT algorithms A it holds that

$$\Pr[\mathsf{Exp}_{\mathcal{A}}(1^{\lambda}) = 1] \leq (|Q|+1)/|D| + \text{negl}(\lambda)$$

where the experiment is defined in the following.

$\underline{\text{Exp}_A(1^{\lambda})}$*:*

- $(t,n) \leftarrow A(1^{\lambda})$
- $\alpha \leftarrow_{\$} D$
- $(pk,sk_1,...,sk_n) \leftarrow \text{Setup}(1^{\lambda},t,n,\alpha)$
- $S^* \leftarrow A(pk)$
- $(q^*, \tau^*, p_1^*, \ldots p_t^*) \leftarrow \mathcal{A}^{\mathcal{O}}(\{\mathsf{sk}_i\}_{i\in S^*})$

where $|S*| = t - 1$ and O gives the adversary access to the following interfaces:

1) Accept(*τ*) : *On input a tag τ, the adversary is given q* ← Query(pk,*α,τ*) *and* p_i ← Response(pk,sk_i,*q,τ,i*), *for all i /*∈ *S**.
2) Reject(*τ,f*) : *On input a tag τ and a function f, the adversary is given q* ← Query(pk,*f*(*α*),*τ*) *together with* p_i ← Response(pk,sk_i,*q,τ,i*), *for all i /*∈ *S**. *We require that f is given as a polynomial-size circuit and that it has no fixed point, i.e., there exists no x such that f*(*x*) = *x.*
3) Malicious(*q,τ*) : *On input a query q and a tag τ, the adversary is given* p_i ← Response(pk,sk_i,*q,τ,i*), *for all i /*∈ *S**.

We define Q to be the set of queries to the Malicious interface and Q′ be the set of queries to the Accept interface. The experiment returns 1 if and only if

$$\text{Verdict}^{(\mathsf{pk}, p_1^*, \ldots p_t^*, \tau^*) = 1} \textit{ and } \tau^* \in/ Q'.$$

We now discuss how the experiment defined above models the intuition for the security properties that we want to ensure. First observe that the adversary is allowed to see arbitrarily many accepting and rejecting queries from the honest users without affecting its success probability. This means that no matter how many queries it eavesdrops, its advantage in guessing α should not change. Instead, the Malicious interfaces allows the attacker to guess the value of α exactly once per query and we require that nothing is revealed beyond whether his guess is correct or not. This prevents offline attacks where the attacker locally tests for the correct value of α and returns an accepting query without querying the Malicious interface.

Finally observe that the experiment captures replay attacks: If the adversary was able to maul an honest accepting query into a query for a different tag, it could violate the above definition with a single query to the Accept interface.

$\underline{\mathsf{Setup}}(1^\lambda, t, n, \alpha)$: On input the security parameter 1^λ, a threshold size t, a committee size n, and a string $\alpha \in \mathbb{Z}_p$, the setup algorithm samples a tuple $(x, r, s, \rho) \leftarrow\!\$\ \mathbb{Z}_p^*$ and a t-out-of-n Shamir secret sharing $(x_1, \ldots, x_n) \leftarrow \mathsf{Share}(x, t, n)$. Then it computes $\mathsf{crs} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$. The algorithm sets the public key of the scheme to

$$\mathsf{pk} = (\mathsf{crs}, h, h_1, \ldots, h_n, c_0, c_1, d_0, d_1) = (\mathsf{crs}, g_1^x, g_T^{x_1}, \ldots, g_T^{x_n}, g_1^r, h^r \cdot g_1^{-\rho\alpha}, g_1^s, h^s \cdot g_1^\rho)$$

and the secret keys to $sk_i = x_i$, for all $i \in \{1, \ldots, n\}$.

$\underline{\mathsf{Query}}(\mathsf{pk}, \beta, \tau)$: On input the public key pk, a string $\beta \in \mathbb{Z}_p$, and a tag $\tau \in \{0,1\}^\lambda$, the query algorithm does the following. For all $i \in \{0, \ldots, \log(p)\}$, sample a uniform $r_i \leftarrow\!\$\ \mathbb{Z}_p^*$ and compute

$$(c_{0,i}, c_{1,i}) = \left(d_0^{2^i \cdot \beta_i} \cdot g_1^{r_i}, d_1^{2^i \cdot \beta_i} \cdot h^{r_i}\right)$$

where $\beta = (\beta_1, \ldots, \beta_{\log(p)})$ is parsed in its binary representation. Then compute $\pi_i \leftarrow \mathsf{NIZK.Prove}(\mathsf{crs}, \mathsf{stmt}_i, r_i)$ where

$$\mathsf{stmt}_i = \left\{\exists\ r_i\ s.t.\ (c_{0,i}, c_{1,i}) = (g_1^{r_i}, h^{r_i})\ OR\ (c_{0,i}, c_{1,i}) = \left(d_0^{2^i} \cdot g_1^{r_i}, d_1^{2^i} \cdot h^{r_i}\right) \middle\| \tau \right\}.$$

The algorithm returns $q = (c_{0,0}, c_{1,0}, \pi_0, \ldots, c_{0,\log(p)}, c_{1,\log(p)}, \pi_{\log(p)})$.

$\underline{\mathsf{Response}}(\mathsf{pk}, \mathsf{sk}_i, q, \tau, i)$: On input the public key pk, a secret key sk_i, a query q, a tag τ, and an index i, the response algorithm checks whether for all $i \in \{0, \ldots, \log(p)\}$ it holds that

$$\mathsf{NIZK.Verify}(\mathsf{crs}, \mathsf{stmt}_i, \pi_i) = 1$$

and aborts if this condition is not verified. Then the algorithm evaluates $k \leftarrow H(\mathsf{pk}, q, \tau)$ and computes

$$b_0 = e\left(c_0 \cdot \prod_{i=0}^{\log(p)} c_{0,i}, k\right) \text{ and } b_1 = e\left(c_1 \cdot \prod_{i=0}^{\log(p)} c_{1,i}, k\right).$$

Furthermore, the algorithm computes $b^{(i)} = b_0^{x_i}$ and an equality proof $\tilde{\pi} \leftarrow \mathsf{NIZK.Prove}(\mathsf{crs}, \tilde{\mathsf{stmt}}_i, x_i)$ where

$$\tilde{\mathsf{stmt}}_i = \left\{\exists\ x_i\ s.t.\ b^{(i)} = b_0^{x_i}\ AND\ h_i = g_T^{x_i} \middle\| \tau \right\}.$$

The algorithm returns $p_i = (b^{(i)}, b_1, \tilde{\pi}, \tau, i)$.

$\underline{\mathsf{Verdict}}(\mathsf{pk}, p_1, \ldots, p_{\tilde{t}}, \tau)$: On input the public key pk, a set of partial responses $(p_1, \ldots, p_{\tilde{t}})$, and a tag τ, the verdict algorithm checks whether there exists a set S of responses of size $|S| = t$ such that the corresponding values (b_1, τ) are identical for all responses and

$$\mathsf{NIZK.Verify}(\mathsf{crs}, \tilde{\mathsf{stmt}}_i, \tilde{\pi}_i) = 1$$

for all $i \in S$. If this is the case, the algorithm checks whether

$$\prod_{i \in S} \left(b^{(i)}\right)^{\lambda_i} = b_1$$

and returns 1 if and only if all of the above conditions are satisfied.

Figure 4.1 Our DZT protocol

4.5 Construction of Distributed Zero-Tester

In this section we present our DZT construction, and we show that it satisfies all of the properties of interest. Let (G1,G2,GT,p,g1,g2) be an asymmetric bilinear group of prime order p and let H : {0,1}∗ → G2 be a hash function modeled as a random oracle. We assume the existence of NIZK scheme (NIZK.Setup,NIZK.Prove,NIZK.Verify) for NP. The scheme is described in Figure 1. For completeness, we also present a self-contained version of the underlying threshold homomorphic encryption scheme in Appendix A.

To see why the scheme is correct, note that

$$
\begin{aligned}
&\left(c_0 \cdot \prod_{i=0}^{\log(p)} c_{0,i}, c_1 \cdot \prod_{i=0}^{\log(p)} c_{1,i}\right) \\
&= \left(c_0 \cdot \prod_{i=0}^{\log(p)} d_0^{2^i \cdot \alpha_i} \cdot g_1^{r_i}, c_1 \cdot \prod_{i=0}^{\log(p)} d_1^{2^i \cdot \alpha_i} \cdot h^{r_i}\right) \\
&= \left(c_0 \cdot d_0^{\alpha} \cdot g_1^{\tilde{r}}, c_1 \cdot d_1^{\alpha} \cdot h^{\tilde{r}}\right) \\
&= \left(g_1^{r} \cdot g_1^{s\alpha+\tilde{r}}, h^{r} \cdot g_1^{-\rho\alpha} \cdot g_1^{\rho\alpha} \cdot h^{s\alpha+\tilde{r}}\right) \\
&= \left(g_1^{\psi}, h^{\psi}\right)
\end{aligned}
$$

where $\tilde{r} = \sum_{i=1}^{\log(p)} r_i$ and ψ = r + sα + r̃. Then we have that

$$(b_0, b_1) = (e(g_1^{\psi}, k), e(h^{\psi}, k)) = (g_T^{\psi\kappa}, g_T^{x\psi\kappa})$$

and therefore, for all admissible sets S of size |S| = t we have

$$\prod_{i \in S} b_0^{x_i \lambda_i} = g_T^{x\psi\kappa} = b_1$$

with probability 1. We now show that our DZT scheme satisfies the notion of security.

Theorem 1 (Security). If the XDH and the DBDH problems are hard over (G1,G2,GT,p,g1,g2) then the DZT construction as described in Figure 1 is secure.

Proof. We assume without loss of generality that the adversary outputs a maximally corrupted subset of parties S∗ of size |S| = t − 1. Then we gradually modify the experiment in the following sequence of hybrids.

Hybrid 0: This is the original experiment ExpA(1^{λ}).

Hybrid 1: In this hybrid all NIZK proof issued to the adversary on behalf of honest parties are computed using the simulator instead of the real witness. This modification is computationally indistinguishable by a standard hybrid argument against the zero-knowledge property of the NIZK system.

Hybrid 2: In this hybrid all NIZK proof sent by the adversary are extracted using the Ext algorithm, provided by the simulation extractability property of the NIZK system. If any of the outputs of the extractor does not constitute a valid witness for the corresponding relation, then the experiment aborts. A standard hybrid argument can be used to show that the probability that an abort is triggered is bounded by a negligible function in the security parameter, by the simulation extractability of the NIZK system.

Hybrid 3: In this hybrid we change how the responses of the honest parties are computed for all queries of the adversary. Fix a query of the adversary (to any of the interfaces) and let (b0,b1) be defined as in the Response algorithm. The output of the i-th honest party is identical to the original experiment except for

$$b^{(i)} = \left(\frac{b_1}{\prod_{j \in S^*} \left(b_0^{x_j}\right)^{\lambda_j} \cdot g_T^{\kappa\rho\cdot(\tilde{\alpha}-\alpha)}} \right)^{\lambda_i^-}.$$

where $k = g_2^{\kappa}$, and α˜ depends on the type query that the adversary is asking:

(1) For queries to the Accept interface, $\alpha\tilde{}$ is set to α.

(2) For queries to the Reject interface, $\alpha\tilde{}$ is set to $f(\alpha)$.

(3) For queries to the Malicious interface, the values of $(\beta_0,\ldots,\beta_{\log(p)})$ are read from the extracted NIZK sent by the adversary, and the value of $\alpha\tilde{}$ is set to $\sum_{i=0}^{\log(p)} 2^i \cdot \beta_i$.

Observe that such a response is correctly distributed since

$$\left(b^{(i)}\right)^{\lambda_i} \cdot \prod_{j \in S^*} \left(b_0^{x_j}\right)^{\lambda_j} = \frac{b_1}{g_T^{\kappa\rho\cdot(\tilde{\alpha}-\alpha)}} = \frac{h^{\tilde{r}} \cdot g_T^{\kappa\rho\cdot(\tilde{\alpha}-\alpha)}}{g_T^{\kappa\rho\cdot(\tilde{\alpha}-\alpha)}} = h^{\tilde{r}}$$

for some adversarially chosen r˜. It follows that this modification is only syntactical and the view of the adversary is identical to that of the previous hybrid.

Hybrid 4: In this hybrid the shares x_i, for all $i \in S*$, are sampled uniformly from Zp, instead of being computed using the Share algorithm. Then the element hi corresponding to the i-th honest party is computed as

$$h_i = \left(\frac{h}{\prod_{i \in S^*} h_j^{\lambda_j}} \right)^{\lambda_i^-}.$$

Since $|S*| < t$ it follows that the view of the adversary is identical to that induced by the previous hybrid. Hybrid 5: In this hybrid we compute the ciphertexts (c0,c1) and (d0,d1) as encryptions of zero, i.e.,

$$(c_0, c_1) = (g_1^r, h^r) \text{ and } (d_0, d_1) = (g_1^s, h^s)$$

where $(r, s) \leftarrow\$ \mathbb{Z}_p^*$. Indistinguishability follows from two invocations of the XDH assumption.

Hybrid 6: This hybrid is identical to the previous one, except that the responses of the honest parties for all adversarial queries are computed as

$$b^{(i)} = \left(\frac{b_1}{\prod_{j \in S^*} \left(b_0^{x_j}\right)^{\lambda_j} \cdot g_T^{\gamma\cdot(\tilde{\alpha}-\alpha)}} \right)^{\lambda_i^-}.$$

where $\gamma \leftarrow\$ \mathbb{Z}_p^*$ is sampled freshly for each adversarial query and α˜ is defined as before.

Let Q be the set of queries issued by the adversary to the decryption oracle. For all i ∈ {1,...,|Q|} we define the following hybrid distribution

$$\left(g_2^{\kappa}, \ldots, g_2^{\kappa Q}, g_T^{\rho\kappa}, \ldots, g_T^{\rho\kappa_{i-}}, g_T^{\rho\kappa_i}, g_T^{\gamma_{i+}} \ldots, g_T^{\gamma Q}\right).$$

Note that on the one extreme we have that the distribution corresponds to the computation done in in Hybrid 5

$$\left(g_2^{\kappa}, \ldots, g_2^{\kappa Q}, g_T^{\rho\kappa}, \ldots, g_T^{\rho\kappa Q}\right)$$

whereas on the other extreme the distribution is identical to the computation done in Hybrid 6

$$\left(g_2^{\kappa}, \ldots, g_2^{\kappa Q}, g_T^{\gamma}, \ldots, g_T^{\gamma Q}\right).$$

One can easily show that the distance between the i-th and the (i + 1)-th hybrids is negligible by an invocation of the the DBDH assumption: Let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, g_2^x, g_T^y, g_T^z)$ be the tuple taken as input by the distinguisher. The reduction sets gTρ = gTy and answers the queries {1,...,i−1} as specified in Hybrid 5 and the queries {i+1,...,|Q|} as specified in Hybrid 6, programming the random oracle to some value with known discrete logarithm κ. For the i-th query the reduction programs the random oracle to $k = g_2^x$ and computes

$$b^{(i)} = \left(\frac{b_1}{\prod_{j \in S^*} \left(b_0^{x_j}\right)^{\lambda_j} \cdot g_T^{z \cdot (\tilde{\alpha} - \alpha)}}\right)^{\lambda_i^-}.$$

Observe that if z = xy then the view of the adversary is identical to that of the i-th hybrid distribution, whereas if z is uniform in $\mathbb{Z}_p^*$ then the view of the adversary is identical to that of the (i + 1)-th hybrid. This shows that the hybrid distributions must be computationally indistinguishable.

Hybrid 7: In this hybrid we change the response of the honest parties to all adversarial queries. Specifically, we compute

$$b^{(i)} = \left(\frac{b_1}{\prod_{j \in S^*} \left(b_0^{x_j}\right)^{\lambda_j} \cdot g_T^{\gamma \cdot \tilde{b}}}\right)^{\lambda_i^-}.$$

where the bit ˜b is defined as follows:

(1) For queries to the Accept interface, we set $\tilde{b} = 0$.

(2) For queries to the Reject interface, we set $\tilde{b} = 1$.

(3) For queries to the Malicious interface, we set $\tilde{b} = 0$ if and only if $\alpha = \sum_{i=0}^{\log(p)} 2^i \cdot \beta_i$, where $(\beta_0,\ldots,\beta_{\log(p)})$ is extracted from the NIZK produced by the adversary.

First observe that if $\tilde{\alpha} = \alpha$, then the distribution is indentical as $\tilde{b} = \tilde{\alpha} - \alpha = 0$. Furthermore, observe that for all constants $c \neq 0$ and $c' \neq 0$ the following distributions

$$(c, c', c \cdot \gamma) \equiv (c, c', c' \cdot \gamma)$$

where $\gamma \leftarrow\$ \mathbb{Z}_p^*$, are identical. It follows that the view of the adversary is unchanged.

Analysis: We are now in the position of analyzing the success probability of the adversary in Hybrid 7. First observe that the public parameters of the scheme do not contain any information about α. Furthermore, the responses of the queries to the Accept and Reject interfaces are computed independently of α. On the other hand, each query to the Malicious interface reveals a single bit of information, i.e., whether $\tilde{\alpha} = \alpha$, for some adversarially chosen $\tilde{\alpha}$. Let Q be the set of queries to the Malicious interface, we can then bound the probability that the adversary guesses the correct α by $(|Q| + 1)/|D|$.

It remains to analyze the success probability that the adversary has without guessing a correct α, which is equivalent to the probability of producing a false proof for a (fresh) rejecting statement and can be bounded to a negligible value by the simulation extractability of the NIZK system. Thus, the success probability of the adversary against Hybrid 7 is bounded from above by $(|Q|+1)/|D|+\text{negl}(\lambda)$. By the above analysis, the same bound holds for an adversary playing against the original experiment, up to a negligible additive factor in the security parameter.

4.5.1 Efficient Non-Interactive Zero Knowledge

In the following we discuss an efficient instantiation for the NIZK system, which will allow us to scale the efficiency of our system to the regime of practicality. First observe that our NIZK system must support proofs for discrete logarithm equality. This class of NIZKs can be instantiated very efficiently with the classical protocol from Schnorr, which we recall in the following. Let (g,h) be a pair of elements of a group of prime order p and let $(g,\tilde{}\ h\tilde{})$ be two group elements. We want to prove the existence of some $w \in Zp$ such that $g^w = g\tilde{}$ and $h^w = h\tilde{}$. The prover samples a uniform $r \leftarrow\$ Zp$ and computes the commitment (g^r,h^r), then the challenge $c \in Z_p$ is computed by hashing the commitment together with the statement. In our case the statement includes also the string τ, that is concatenated to the group elements and hashed to compute the challenge c. The prover computes $z = wc + r$ and returns $\pi = (g^r,h^r,z)$. The proof is considered valid if $g\tilde{}^c \cdot g^r = g^z$ and $h\tilde{}^c \cdot h^r = h^z$. It is well known [34] that the protocol is zero-knowledge and simulation sound[2] in the random oracle model.

The above protocol can be extended to handle disjunctions (i.e., the OR-composition of proofs) using a standard trick: For the case of two statements, the two proofs are computed in parallel using the same algorithm as above, except that the prover is allowed to choose the challenges c0 and c1 under the constraint that $c_0 + c_1 = c$ where c is computed hashing both statements and the corresponding commitments. The proof is considered valid if both of the resulting proofs correctly verify.

Finally, we remark that Schnorr's NIZKs have a so called transparent setup which is not required to be generated by a trusted party, i.e., the setup consists of sampling the random oracle.

4.5.2 Implementation in Ethereum

Implementing our construction as a smart contract in Ethereum entails a new set of challenges: While in principle bilinear group operations and pairings are efficient enough to be performed on commodity machines, the integration in Ethereum of our DZT requires one to implement them in Solidity, the language of Ethereum smart contracts. Such a language imposes a hard limit on the amount of computation that each smart contract can perform, which makes implementation of cryptographic operations particularly challenging. Fortunately, Solidity has precompiled instructions for the following (bilinear) group operations:

1) Group operations in G_1.
2) Modular exponentiations in G_1.
3) Checks of pairing product equations PPEq : $\mathbb{G}_1^n \times \mathbb{G}_2^n \to \{0,1\}$ of the form

$$e(x_1,y_1)\cdots e(x_n,y_n) = 1^? \quad .$$

However, this semantic turns out to be insufficient to implement our scheme. The reason is that our smart contract (whose computational load consists mainly of calls to the Verdict algorithm in Figure 1) requires us to compute group operations in the target group GT, which are not natively supported by Solidity. One solution to circumvent this issue could be to implement group operations in GT from scratch, without leveraging precompiled instructions. Unfortunately, elliptic curve operations in the target group are notoriously expensive, and this approach causes the smart contract to exceed the maximum amount of computation allowed by the specifications of Ethereum.

To circumvent this issue, we modify our scheme to be compatible with precompiled instructions in Solidity. Our solution is sketched in what follows. Recall that in our scheme each committee member provides the smart contract with a target group element

$$b_{(i)} = e(\bar{c},k)^{sk_i}$$

(along with other information that is irrelevant for this overview). In our modified scheme, each member publishes instead

$$c^{\bar{}sk_i\cdot\psi} \text{ and } k^{1/\psi},$$

2. While our DZT construction requires the stronger notion of simulation extractable NIZK, we settle for simulation soundness in favor of a more efficient scheme, as commonly done in the literature.

$\underline{\mathsf{Setup}}(1^\lambda, t, n, \alpha)$: On input the security parameter 1^λ, a threshold size t, a committee size n, and a string $\alpha \in \mathbb{Z}_p$, the setup algorithm samples a tuple $(x, r, s, \rho) \leftarrow\$ \mathbb{Z}_p^*$ and a t-out-of-n Shamir secret sharing $(x_1, \ldots, x_n) \leftarrow \mathsf{Share}(x, t, n)$. Then it computes $\mathsf{crs} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$. The algorithm sets the public key of the scheme to

$$\mathsf{pk} = (\mathsf{crs}, h, h_1, \ldots, h_n, c_0, c_1, d_0, d_1) = (\mathsf{crs}, g_1^x, g_1^{x_1}, \ldots, g_1^{x_n}, g_1^r, h^r \cdot g_1^{-\rho\alpha}, g_1^s, h^s \cdot g_1^\rho)$$

and the secret keys to $sk_i = x_i$, for all $i \in \{1, \ldots, n\}$.

$\underline{\mathsf{Query}(\mathsf{pk}, \beta, \tau)}$: Same as Figure 1.

$\underline{\mathsf{Response}(\mathsf{pk}, \mathsf{sk}_i, q, \tau, i)}$: On input the public key pk, a secret key sk_i, a query q, a tag τ, and an index i, the response algorithm checks whether for all $i \in \{0, \ldots, \log(p)\}$ it holds that

$$\mathsf{NIZK.Verify}(\mathsf{crs}, \mathsf{stmt}_i, \pi_i) = 1$$

and aborts if this condition is not verified. Then the algorithm evaluates $k \leftarrow H(\mathsf{pk}, q, \tau)$ and computes

$$\tilde{c}_0 = c_0 \cdot \prod_{i=0}^{\log(p)} c_{0,i} \text{ and } \tilde{c}_1 = c_1 \cdot \prod_{i=0}^{\log(p)} c_{1,i}.$$

Furthermore, the algorithm samples some $\psi \leftarrow\$ \mathbb{Z}_p^*$ and sets $c_0^{(i)} = \tilde{c}_0^{\psi \cdot x_i}$ and $k^{(i)} = k^{1/\psi}$. Then it computes an equality proof $\tilde{\pi} \leftarrow \mathsf{NIZK.Prove}(\mathsf{crs}, \tilde{\mathsf{stmt}}_i, x_i)$ where

$$\tilde{\mathsf{stmt}}_i = \left\{ \exists\ x_i \ s.t.\ e\left(c_0^{(i)}, k^{(i)}\right) = e\left(\tilde{c}_0, k\right)^{x_i} \ AND\ h_i = g_1^{x_i} \middle\| \tau \right\}.$$

The algorithm returns $p_i = \left(c_0^{(i)}, \tilde{c}_0, \tilde{c}_1, k^{(i)}, k, \tilde{\pi}, \tau, i\right)$.

$\underline{\mathsf{Verdict}(\mathsf{pk}, p_1, \ldots, p_{\tilde{t}}, \tau)}$: On input the public key pk, a set of partial responses $(p_1, \ldots, p_{\tilde{t}})$, and a tag τ, the verdict algorithm checks whether there exists a set S of responses of size $|S| = t$ such that the corresponding values $(\tilde{c}_0, \tilde{c}_1, k, \tau)$ are identical for all responses and

$$\mathsf{NIZK.Verify}(\mathsf{crs}, \tilde{\mathsf{stmt}}_i, \tilde{\pi}_i) = 1$$

for all $i \in S$. If this is the case, the algorithm checks whether

$$\prod_{i \in S} e\left(\left(c_0^{(i)}\right)^{\lambda_i}, k^{(i)}\right) = e\left(\tilde{c}_1, k\right)$$

and returns 1 if and only if all of the above conditions are satisfied.

Figure 4.2. Ethereum-compatible DZT protocol.

where ψ is a uniformly sampled integer in $\mathbb{Z}_p^*$. Note that we can now compute the Verdict algorithm with a precompiled instruction PPEq (3) and the newly introduced term ψ does not affect the correctness of the scheme, since it cancels out with the pairing. To see why this is the case, observe that

$$e\left(\bar{c}^{\mathsf{sk}_i \cdot \psi}, k^{1/\psi}\right) = e(\bar{c}, k)^{\mathsf{sk}_i} = b^{(i)}.$$

We stress that the presence of the randomness ψ is crucial to avoid mix-and-match attacks. We proceed by presenting our modified scheme more formally.

Construction. We present our modified DZT construction. Let $(G_1, G_2, G_T, p, g_1, g_2)$ be an asymmetric bilinear group of prime order p and let $H : \{0,1\}* \rightarrow G2$ be a hash function modeled as a random oracle. We assume the existence of a NIZK scheme

(NIZK.Setup,NIZK.Prove,NIZK.Verify) for NP. The scheme is described in Figure 2. Correctness follows by a routine calculation.

Note that the modified statement that we need to prove in the response algorithm is again a proof of discrete logarithm equality, except that the first equation involves the comparison of target group elements, which is inefficient. To circumvent this issue, we run the Schnorr NIZK over the source group G1 and we augment the verification with a pairing. More specifically, we let the prover sample a uniform r ←$ Zp and compute the commitment $(\tilde{c}_0^r, g_1^r)$. The challenge c is obtained by hashing the commiment and the statement, then the prover computes z = xi · c + r. The proof consists of $(\tilde{c}_0^r, g_1^r, z)$. The verifier accepts if

$$e\left(\left(c_0^{(i)}\right)^c, k^{(i)}\right) \cdot e\left(\tilde{c}_0^r, k\right) = e\left(\tilde{c}_0^z, k\right)$$

and

$$h_i^c \cdot g_1^r = g_1^z$$

where the challenge c is recomputed locally by the verifier. Since such a protocol is an instance of Schnorr NIZK, zero-knowledge and simulation extractability are immediate.

Analysis. We now analyze the security of our new scheme. One downside of this modification is that its security relies on a new (static) assumption over asymmetric bilinear groups that we introduce in this work, which we refer to as the dual Diffie-Hellman (dDH) assumption. As a positive evidence that the problem is likely to be intractable, we show that the assumption holds in the generic group model.

Assumption 3 (dDH). Let G be a bilinear group generator. G is dDH-hard if for all PPT distinguishers it holds that the following distributions are computationally indistinguishable

$$(G_1, G_2, G_T, p, g_1, g_2, g_{1x}, g_{1y}, g_{1z}, g_{1zy/x}, g_{2v}, g_{2xv}, g_{2w}, g_{2yw}) \approx$$
$$(G_1, G_2, G_T, p, g_1, g_2, g_{1x}, g_{1y}, g_{1z}, g_{1u}, g_{2v}, g_{2xv}, g_{2w}, g_{2yw})$$

where $(G_1, G_2, G_T, p, g_1, g_2) \leftarrow\$ G(1^\lambda)$ and $(u, v, w, x, y, z) \leftarrow\$ \mathbb{Z}_p^*$.

We are now ready to show that the scheme satisfies the security notion for a DZT, assuming the hardness of the XDH and the dDH problems. Due to space constraints, the proof is deferred to Appendix B.

Theorem 2 (Security). If the XDH and the dDH problems are hard over $(G_1, G_2, G_T, p, g_1, g_2)$ then the DZT construction as described in Figure 2 is secure.

4.6 Experimental Evaluations

We implement the DZT-based solution as a smart contract on the Ethereum blockchain. The smart contract is written in Solidity, an Ethereum specific language. Contracts written in Solidity are compiled into bytecode for the Ethereum Virtual Machine (EVM). The code is available. The runtime of programs on the EVM is limited by a unit called gas. Every operation

in the EVM costs gas and there is a hard limit as to how much gas a transaction can consume. This gives a hard limit on how much computation can be done in one transaction. Since the price of Ethereum and gas is fluctuating, all costs displayed in Dollars are calculated with a fixed conversion rate of 1 gas costing 1 GWei and 1 Ether costing 180$. The non-blockchain parts of the implementations use the JSON-RPC API of an Ethereum node to interact with the smart contracts. A pictorial description of the system architecture is given in Figure 3.

4.6.1 Security Policies

The security policy of a smart contract determines when a transaction triggers the usage of a second factor for authentication. In our implementation, the smart contract tracks the total flow of money and the amounts transferred to specific addresses. This allows us to set fine-grained security policies that depends on the history of transactions. We say that a transaction is flagged if the security policy invokes the 2FA protocol for such a transaction. In our prototype, we consider four types of security policies:

Flag every transaction to any new beneficiary.

Flag a transaction if the sum of all coins transferred (since the last flag) is above a predefined threshold.

Flag a transaction if the sum of all funds that are transferred in a specific timeframe (a sliding window) exceeds a predefined threshold.

Flag a transaction if the sum of all coins sent to a specific beneficiary (since the last flag) exceeds a predefined threshold.

This list of policies is non-exclusive and multiple policies can coexist in the same smart contract. If any security policy flags a transaction, the counter for such a policy is reset if and only if the transaction is successfully authenticated.

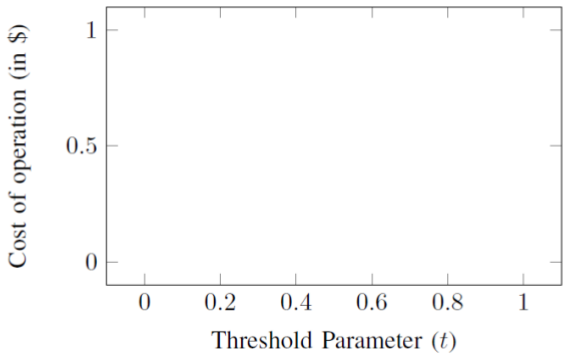


Figure 4.3. Cost of the creation of a DZT group (blue) cost for a client querying the group (green) and cost of submitting and verifying a query (red).

We measured the overhead that the implementation of each policy adds to processing transactions. For policy (1), it cost 0.005$ for a user to set it up. If a transaction is flagged because it is sent to a therefore unknown beneficiary, the user has to pay 0.008$ to now save that the beneficiary is verified for further transactions. For policies (2) and (4), the setup cost is 0.009$ and each transaction costs an additional 0.006$ to sum up the coins transferred. If the threshold is reached, the user has to spend 0.004$ (and 0.006$, respectively) to reset the sum of all coins spent to 0. The cost of policy (3) varies with the size of the sliding window, which we measure for values ranging from 0 to 100 transactions. The added cost does not exceed 0.5$, in a non-optimized implementation.

4.6.2 Performance Evaluation

The steps of the 2FA mechanism are implemented as functions on a smart contract. The smart contract notifies the committee members via Ethereum events if a new protocol step has to be executed, i.e., this happens if a new query is submitted. The contract then runs the Verdict algorithm, which internally verifies the NIZKs and checks whether the conditions dictated by the algorithm are all verified. We stress that the worst-case runtime of the Verdict algorithm is only proportional to t˜ (the number of partial responses fed as input). Recall that the Verdict algorithm consist of two subroutines:

1) Checking the NIZKs: There are at most $t\tilde{}$ NIZKs, so this step can be done in time linear in $t\tilde{}$.
2) Partition the responses in sets with the same value of (b_1, τ): This operation can also be done in time proportional to $t\tilde{}$.

At this point we can check whether there exists a set with more than t elements where all NIZKs verify. Note that, within such a set, any subset of the shares in this set is considered valid (since all the NIZKs verify correctly), so the Verdict algorithm can just pick an arbitrary one (e.g. the lexicographically smallest one).

We use the 256bit version of the Barreto-Naehrig (BN) curves to instantiate the pairings [60]. This curve is especially suited for use in Ethereum, since an elements of its base field fits into the 256bit wide registers of the EVM. We show the costs associated with all operations in Figure 4. For our experiments, we set the threshold size (t) to be equal to the group size (n), since decreasing the value of t only decreases the costs of each algorithm. For the client, the cost of the setup phase grows linearly with the size of the committee. The cost of the queries depends only on the threshold parameter t, i.e., the maximum size of parties that the attacker can corrupt. The cost to verify a query is identical for all members of the committee and we only display the cumulative cost (i.e. the sum of the costs for each committee member), which grows linearly with t. We stress that a lower value of t implies a lower corruption threshold but at the same time an increase in reliability, as only t parties need to be online in order to run protocol.

4.7 A Solution Based on U2F Tokens

The Universal Second Factor (U2F) is a token-based authentication protocol specified by the FIDO Alliance [61]. It is used to enhance the security of standard password-based authentication with a cryptographic secret stored on a hardware token. This protocol is supported by all major browser and by many web services (such as Gmail, Facebook, and Dropbox). The U2F protocol consists of three interacting entities: (1) A hardware token, (2) a client, and (3) a relying party, which guards access to the resources. The U2F protocol specifies two subroutines, a one-time registration and and unbounded number of authentications, which we briefly describe below.

Registration. The registration allows the relying party to bind a physical token to a user account. The relying party is associated with some application identifier (e.g., the url of its website) and begins the registration phase by sending a challenge to the token. The hardware token generates a fresh ECDSA key pair (sk,vk) and sends back the verification key and a key-handle to the relying party. At this point the relying party associates the key handle and vk with the account of the user.

Authentication. Upon each authentication request, the relying party sends the key handle, the identifier, and a challenge to the token. The key handle is used by the token to recover the corresponding signing key, which is used to sign the challenge of the relying party. In addition to the challenge, the token also signs a counter, which is maintained by the token locally and increased upon each authentication run. The relaying party considers the authentication successful if the signature correctly verifies against the verification key vk associated with the user account and if the counter (also included in the signed message) increased from the previous authentication.

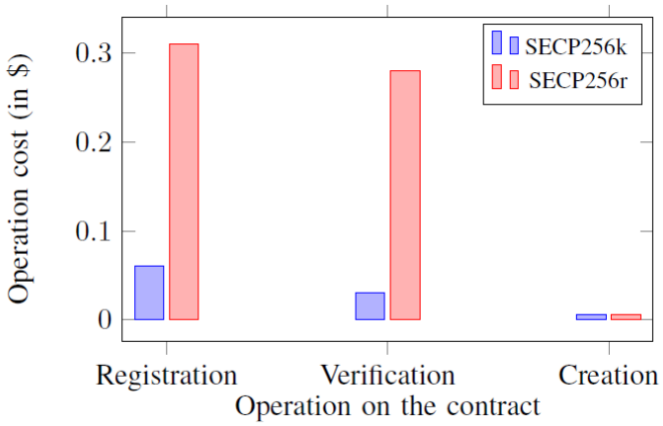


Figure 4.4. Transaction registration, verification and creation costs on curve SECP256k (blue) and curve SECP256r (red).

4.7.1 U2F on Ethereum

In the U2F token-based authentication protocol, the smart contract impersonates the relying party. The authentication protocol for a transaction τ begins with the client querying the smart

contract for a challenge and receives a random nonce and a key handle that is used by the token to specify the correct key to use. The client queries the token for a signature on the nonce and on τ and sends it to the smart contract, which can publicly verify the validity of the signature. A client can use the same token with multiple smart contracts, since U2F tokens allow one to generate an arbitrary number of key pairs, each associated to a unique identifier. In our case, the identifier consists of the address of the smart contract and the chain identity.

Nonce Sampling. A subtlety that needs to be addresses is that Ethereum smart contracts do not support non-deterministic operation and therefore cannot sample truly random nonces. We circumvent this limitation by setting the nonce to $H(\tau)$, where H is a hash function modeled as a random oracle and τ is a unique identifier of the transaction. In Ethereum, a transaction can be uniquely identified by the concatenation of the receiver and sender addresses and the value of a counter, which denotes the number of the transaction from the sending address and can only grow over time.

Implementation. U2F specifies curve SECP256r1 or P-256 as the basis for signature creation and verification. On the other hand, Ethereum uses signatures over the curve SECP256k1 internally and offers an EVM instruction to access the signature verification capabilities that is part of the official standard [62]. To make the two choices compatible, we design the smart contract to allow a user to decide at registration time which curve should be used for verification.

The costs for authenticating transactions are shown in Figure 4.4. All measurements are taken using a simulated token. The use of a soft token does not impact the messages that are exchanged on the blockchain and the interaction between the smart contract and the user client, so this change does not affect the costs. The measurements are taken as the average over 100 transactions. The cost of verifying signatures on the standard curve SECP256r of U2F is significantly higher compared to verification on curve SECP256k: Standard U2F tokens have a one-time registration fee of 0.31$ and a recurring fee of 0.28$ for verifying transaction requests. For the costs over the SECP256k curve, the one-time registration costs 0.07$ and the recurring transaction verification costs 0.03$. The evaluation shows that the usage of U2F in the blockchain setting is feasible in terms of costs. There is a big cost benefit in switching the standard U2F signature curve to one which can be natively verified by a smart contract; however this also requires custom made tokens.

4.8 Conclusions

In this work we proposed a new system to safeguard users' transactions, where a smart contract is entitled to request for a two-factor authentication, in case of exceptional events. We designed and implemented two 2FA protocols in Ethereum. Our performance evaluation shows that augmenting Ethereum with 2FA comes at minimal additional cost.

CHAPTER FIVE

Blockchains Enable Non-Interactive Multiparty Computation

Secure Multiparty Computation (MPC) [19, 63] enables parties to evaluate an arbitrary function in a secure manner, i.e., without revealing anything besides the outcome of the computation. MPC is increasingly important in the modern world and allows people to securely accomplish a number of difficult tasks. Obtaining efficient MPC protocols is thus a relevant problem and it has indeed been extensively studied [19, 63, 64]. One important question is the round complexity of MPC schemes. In the semi-honest case, in 1990, Beaver et al. [65] gave the first constant-round MPC protocol for three or more parties. A number of works [66-68] aiming to analyze and reduce round complexity followed, both in the semi-honest and fully malicious models. In 2016, Garg et al. [64] proved that four rounds are necessary to achieve secure MPC in the fully malicious case in the plain model. Four round MPC protocols have been recently proposed [69-71], resolving the questions of round complexity.

Unfortunately, solutions that achieve even the optimal round complexity are still problematic for many applications since these solutions typically require synchronous communication from the participants. In this work, we rely on blockchains to achieve MPC that does not require participants to be online at the same time or interact with each other.

Such non-interactive solutions advance the state of the art of secure multiparty computation, opening up a whole new realm of possible applications. For example, passive data collection for privacy preserving collaborative machine learning becomes possible. Federated learning is already used to train machine learning models for the keyboards of mobile devices for the purposes of autocorrect and predictive typing [72]. Unfortunately, using off-the-shelf MPC protocols to perform such training securely is not straight-forward. Not all smartphones are online at the same time and it might even be unknown how many devices will end up participating. In contrast, off-the-shelf MPC protocols typically assume that all (honest) participants are indeed online during some time period, and the number of participants is known. This leads us to the following question:

Can we construct a secure MPC protocol which does not require the parties to be online at the same time and guarantees privacy and correctness even if all but one of the parties are fully malicious? Is it possible to design a protocol which requires only a single round of participation from the parties supplying the inputs, and allows the parties to go offline after the first round if they are not interested in learning the output?

Consider such a protocol in the use case outlined above – each smartphone could independently send a single message to a server, and at the end of the collection period the server would obtain the model trained on the submitted inputs, all while preserving the privacy of the gathered inputs.

5.0.1 Our Results

In our work, we provide a solution for MPC which achieves the property that each MPC participant who supplies inputs but does not wish to receive the output can go offline after the first round. The participants are not required to interact with each other. We additionally provide variations of our protocol that offer further desirable properties.

Before we provide the formal theorem statements, we discuss the protocol execution model and the notation.

In our work, we assume the existence of append-only bulletin boards that allow parties to publish data and receive an unforgeable confirmation that the data was published in return. Furthermore, we assume a public key infrastructure (PKI). Finally, we rely on conditional storage and retrieval systems (CSaRs,). Roughly, CSaR systems allow a user to submit a secret along with a release condition. Later, if a (possibly different) user is able to satisfy this release condition, the secret is privately sent to this user. Intuitively, during the process, the secrets cannot be modified and no information is leaked about the secrets. We require that CSaRs are used as ideal functionalities. We note that due to the fact that the existing CSaR system [73] relies on blockchains, and bulletin boards can be realized using blockchains as well [13, 27, 74], relying on bulletin boards in our construction effectively does not add extra assumptions. In the following, for simplicity, we will state that we design our protocols in the blockchain model. Finally, we assume IND-CCA secure public key encryption, and digital signatures.

In our construction, we distinguish between parties who supply inputs (dubbed MPC contributors) and parties who wish to receive outputs (dubbed evaluators).

Our construction is a protocol transforming an MPC scheme π into another scheme π'. The contributors in π' are exactly the participants in π. The evaluators can (but do not have to) be entirely different parties from those who contribute inputs in π.

We are now ready to introduce our first result:

Theorem 1. (Informal) Any MPC protocol π secure against fully-malicious adversaries can be transformed into another MPC protocol π' in the blockchain model that provides security with abort against fully-malicious adversaries and does not require participants to be online at the same time. The MPC contributors are required to participate for only a single round (the evaluators might be required to participate for multiple rounds). The adversary is allowed to corrupt as many MPC contributors in π' as it is allowed to corrupt participants in π. The adversary is allowed to corrupt any number of evaluators.

In addition to this result, we discuss ways to optimize our construction. To this end, we explain why the combined communication and state complexity (where state complexity is the amount of data that parties maintain between the different rounds of the protocol execution) of the underlying MPC protocol is of a particular importance in our construction. Briefly, both the communication and state complexities of the underlying MPC translate directly into the number

of CSaR storage and retrieval requests in our overall construction. We describe a protocol in the plain model which relies on multi-key fully homomorphic encryption (MFHE). Its combined communication and state complexity is independent of the function that we are computing. While optimizing communication complexity has received considerable attention in the community in the past few years, optimizing internal state complexity has been largely overlooked. We believe that this particular problem might be exciting on its own. In our construction which optimizes the combined communication and state complexity, we assume multi-key fully homomorphic encryption, and collision-resistant hash functions. The result that we achieve here is the following:

Theorem 2. (Informal) Let f be an N-party function. Protocol 6 is an MPC protocol computing f in the standard model and secure against fully malicious adversaries corrupting up to $t < N$ parties. Its combined communication and state complexity depends only on the security parameter, number of parties, and input and output sizes. In particular, the combined communication and state complexity is independent of the function f.

Using this MPC protocol in combination with our first construction, under the assumptions that we rely on in our main construction and in the MPC construction with optimized communication and state complexity, we achieve the following:

Corollary 1. (Informal) There exists an MPC protocol π' in the blockchain model which provides security with abort against fully-malicious adversaries and does not require participants to be online at the same time. The MPC contributors are required to participate for a single round (the evaluators might be required to participate for multiple rounds). Furthermore, the number of calls to CSaR in this protocol is independent of the function that is being computed using this MPC protocol 1.

Finally, we achieve an MPC protocol which requires only a single round of participation from MPC contributors with the additional property of guaranteed output delivery, meaning that adversarial parties cannot prevent honest parties from receiving the output. For this, we in particular rely on the underlying protocol having guaranteed output delivery as well (and thus requiring the majority of the MPC contributors to be honest). We rely on the same assumptions (PKI, CSaRs, append-only bulletin boards etc.) as the ones used in our main construction. The formal result that we achieve is the following:

Theorem 3. (Informal) Any MPC protocol π that is secure against fully malicious adversaries and provides guaranteed output delivery can be transformed into another MPC protocol π' in the blockchain model that provides security with guaranteed output delivery against fully malicious adversaries and does not require participants to be online at the same time. The MPC contributors are required to participate for only a single round (the evaluators

[1] A prior version of this paper erroneously stated that the communication complexity (instead of the number of CSaR calls) is independent of the function being computed.

might be required to participate for multiple rounds). The adversary is allowed to corrupt as many MPC contributors in π' as it is allowed to corrupt participants in π. The adversary is allowed to corrupt any number of evaluators.

5.0.2 Technical Overview

In this work, we propose an MPC protocol that does not require participants to be present at the same time. In order to do so, we rely on the following cryptographic building blocks – garbled circuits [19, 75, 76], a primitive which we dub conditional storage and retrieval systems (CSaRs) and bulletin boards with certain properties. Before we introduce the construction idea, we elaborate on each of these primitives.

Roughly, a garbling scheme allows one to "encrypt" (garble) a circuit and its inputs such that when evaluating the garbled circuit only the output is revealed. In particular, no information about the inputs of other parties or intermediate values is revealed by the garbled circuit or during its evaluation. In our construction we use Yao's garbled circuits [19, 76].

In our construction, we rely on bulletin boards which allow parties to publish strings on an append-only log. It must be hard to modify or erase contents from this log. Additionally, we require that parties receive a confirmation ("proof of publish") that the string was published and that other parties can verify this proof. Such bulletin boards have been extensively used in prior works [13, 27, 74] and as pointed out by these works can be realized both from centralized systems such as the Certificate Transparency project and decentralized systems such as proof-of-stake or proof-of-work blockchains.

Finally, we define a primitive which we call conditional storage and retrieval systems (CSaRs). Roughly, this primitive allows for the distributed and secure storage and retrieval of secrets and realizes the following ideal functionality:

- Upon receiving a secret along with a release condition and an identifier, if the identifier was not used before, the secret is stored, and all participants are notified of a valid secret storage request. The release condition is simply an NP statement.
- Upon receiving an (identifier, witness) from a user, the ideal functionality checks whether a secret with this identifier exists and if so, whether the given witness satisfies the release condition of this secret record. If so, the secret is sent to the user who submitted the release request.

While systems that provide a similar primitive has been proposed in the past [73, 77] we provide a clean definition that captures the essence of this functionality. We instantiate the CSaR with eWEB [77], which stands for "Extractable Witness Encryption on a Blockchain". Roughly, it allows users to encode a secret along with a release condition and store the secret on a blockchain. Once a user proves that they are able to satisfy the release condition, blockchain miners jointly and privately release the secret to this user. Along the way, no single party is able to learn any information about the secret.

Our construction. By relying on bulletin boards, Yao's garbled circuits and CSaRs, we are able to transform any secure MPC protocol π into another secure MPC protocol π' that provides security with abort and does not require participants to be online at the same time. At a high level, our idea is as follows: first, each contributor (party who supplies inputs in the protocol) P in the MPC protocol π garbles the next-message function for each round of π. Then, P stores the garbled circuits as well as the garbled keys with a CSaR using carefully designed release conditions. Note that each party P is able to do so individually, without waiting for any information from other parties and can go offline afterwards. Once all contributors have stored their data with the CSaR, one or more "evaluators" (parties who wish to receive the output) interact with the CSaR and use the information stored by the MPC contributors in order to retrieve the garbled circuits and execute the original protocol π. The group of the contributors and the group of evaluators do not need to be the same – in fact, these groups can even be disjoint. The evaluators might change from round to round.

Note that while the high-level overview is simple, there are a number of technical challenges that we must overcome in the actual construction due to its non-interactive nature. For example, since the security of Yao's construction relies on the fact that for each wire only a single key is revealed, we must ensure that each honest garbled circuit is executed only on a single set of inputs. The adversary also must not trick a garbled circuit of some honest party A into thinking that a message broadcast by some party C is message m, and tricking a garbled circuit of another honest party B into thinking that C in fact broadcast message $m' \neq m$. Furthermore, we must ensure that it is hard to execute the protocol "out of order", i.e., an adversary cannot execute some round i prior to round j where $i > j$. Such issues do not come up in the setting where parties are online during the protocol execution and able to witness messages broadcast by other parties.

We solve these issues by utilizing bulletin boards, carefully constructing the release conditions for the garbled circuits and the wire keys and modifying the next-message functions which must be garbled by the contributors.

Note that the next-message functions from round two onward take as inputs messages produced by the garbled circuits in prior rounds. At the time when the MPC contributors are constructing their circuits, the inputs of other parties are not known, and thus it is not possible to predict which wire key (the one corresponding to 0 or the one corresponding to 1) will be needed during the protocol execution. At the same time, one cannot simply make both wire keys public since the security of the garbled circuit crucially relies on the fact that for each wire only a single wire key can be revealed. We solve this problem by storing both wire keys with the CSaR, utilizing bulletin boards, and requiring the evaluators to publish the output of the garbled circuits of each round. Then, (part of) the CSaR release condition for the wire key corresponding to bit b on some wire w of some party's garbled circuit for round i is that the message from round $i - 1$ is published and contains bit b at position w. This way we ensure that while both options for wire *w* are "obtainable", only the wire key for bit b (the one that is needed for the execution) is revealed.

Next, note that in our construction we specifically rely on Yao's garbled circuits. Yao's construction satisfies the so-called "selective" notion of security, which requires the adversary to choose its inputs before it sees the garbled circuit (in contrast to the stronger "adaptive" notion of security which would allow the adversary to choose its inputs after seeing the garbled circuits [38]). We ensure that the selective notion of security is sufficient for our construction by requiring that not only the wire keys, but also the garbled circuits are stored with the CSaR. The release conditions both for the garbled circuit for some round *i* and all its wire keys require a proof that all messages for rounds 1 up to and including round $i-1$ are published by the evaluators. This way, the evaluators are required to "commit" to the inputs before receiving the selectively secure garbled circuits, which achieves the same effect as adaptive garbled circuits.

As outlined above, we must ensure that it is hard for the adversary to trick the garbled circuit produced by some honest party A into accepting inputs from another honest party B that were not produced by B's circuits. We accomplish this by modifying the next-message function of every party A as follows: in addition to every message m that is produced by some party B, the next-message function takes as input a signature σ on m as well and verifies that the signature is correct. If this is not the case for any of the input messages, the next-message function outputs ⊥. Otherwise, the next-message function proceeds as usual and in addition to outputting the resulting message it outputs the signature of party A on this message.

Our end goal is to reduce the security of our construction to the security of the underlying MPC protocol π. While utilizing bulletin boards and introducing signatures is a good step forward, we must be careful when designing the CSaR release conditions. The adversary could sign multiple messages for each corrupted contributor in π, publish these messages on the bulletin board and thus receive multiple keys for some wires. To prevent this, the CSaR release condition must consider only the very first message published for round $i - 1$ on the bulletin board. This way, we ensure that there is only a single instance of the MPC running (only a single wire key is released for each circuit): even if the adversary is able to sign multiple messages on behalf of various MPC contributors, only the very first message published on the bulletin board for a specific round will be used by the CSaR system to release the wire keys for the next round.

The ideas outlined above are the main ideas in our protocol. We now elaborate on a few additional details:

The next-message function of the protocol typically outputs not only the message for the next round, but also the state which is used in the next round. It is assumed that this state is kept private by the party. In our case, the output of the next-message function will be output by the garbled circuit and thus made available to the evaluator. To ensure that the state is kept private, we further modify the next-message function to add an encryption step at the end: the state is encrypted under the public key of the party who is executing this nextmessage function. To ensure that the state can be used by the garbled circuit of the party in the next round, we add a state decryption step at the beginning of the next-message function of that round. Similar to the

public output of the next-message function, we compute a signature on the encryption of the state and verify this signature in the garbled circuit of the next round.

Note that in the construction outlined above, we use some secret information which does not depend on the particular execution but still must be kept private (secret keys of the parties used for the decryption of the state, signing keys used to sign the output of the next-message function etc.). This information is hardcoded in the garbled circuits. We explain how this can be done in a later section.

Finally, note that we require the following property from the underlying protocol π: given a transcript of execution of π, the output of π can be publicly computed. As we note, this property can be easily achieved by slightly modifying the original protocol π.

We provide all protocol details and outline optimizations in a later section and give the formal construction in Protocols 1, 2 and 3. The formal security proof is done by providing a simulator for the construction and proving that an interaction with the simulator in the ideal world is indistinguishable from the interaction with an adversary in the real world.

To summarize, using the construction sketched above we achieve the following result:

Theorem 4. (Informal) Protocols 1, 2 and 3 transform any MPC protocol π secure against fully malicious adversaries into another MPC protocol π' in the blockchain model that provides security with abort against fully-malicious adversaries and does not require participants to be online at the same time. The MPC contributors are required to participate for only a single round (the evaluators might be required to participate for multiple rounds). The adversary is allowed to corrupt as many MPC contributors in π' as it is allowed to corrupt participants in π. The adversary is allowed to corrupt any number of evaluators.

In addition to our main protocol that requires only one message from the MPC contributors and does not require any additional functionality from the CSaR participants apart from the core CSaR functionality itself (storing and releasing secrets), we provide a number of variations that have further desirable properties, such as guaranteed output delivery. We now outline these further contributions.

Improving Efficiency: The efficiency of our construction is strongly tied to the efficiency of the underlying MPC protocol π. Note that in our construction each input wire key of each garbled circuit is stored with the CSaR, and the inputs of the garbled circuits are exactly messages exchanged between the parties as well as the state information passed from previous rounds. Thus, the communication and state complexities translate directly into the number of CSaR store and release operations that the MPC contributors, as well as later the evaluators, must make. In order to reduce the number of CSaR invocations, we describe an MPC protocol which optimizes the combined communication and internal-state complexity. While communication complexity is typically considered to be one of the most important properties of an MPC protocol, state complexity receives relatively little attention. Our main construction

shows that there are indeed use cases where both the communication and the state complexity matter, and we initiate a study of the combined state and communication complexity.

Specifically, we introduce an MPC protocol in which the combined communication and state complexity is independent of the function we are computing. We achieve it in two steps: we start with a protocol secure against semi-malicious adversaries 2 which at the same time has communication and state complexity which is independent of the function that is being computed. Then, we extend it to provide fully malicious security while taking care to retain the attractive communication and state complexity properties in the process.

In more detail, we start with the MPC construction by Brakerski et al. [70] which is based on multi-key fully homomorphic encryption (MFHE) and achieves semi-malicious security. We note that the communication and state complexity of this construction depends only on the security parameters, the number of parties, and the input and output sizes. In particular, note that the construction's combined communication and state complexity is independent of the function we are computing.

Our next step is to extend this construction so that it provides security against malicious adversaries. For this, we propose to use the zero-knowledge protocol proposed by Kilian [36] that relies on probabilistically checkable proofs (PCPs) and allows a party P to prove the correctness of some statement x to the prover V using a witness w. Along the way, we need to make minor adjustments to Kilian's construction because its state complexity is unfortunately too high for our purposes – in particular, in the original construction, the entire PCP string is stored by the prover to be used in later rounds. After making a minor adjustment – recomputing the PCP instead of storing it – to the construction to address this issue, we use this scheme after each round of the construction by Brakerski et al. in order to prove the correct execution of the protocol by the parties. The resulting construction achieves fully malicious security, and its communication and state complexities are still independent of the function that we are computing.

We provide the details of the construction and analyse its security and communication/state complexity properties with the formal protocol description in Protocol 6. In this protocol, we assume the existence of an MFHE scheme with circular security and the existence of a collision-resistant hash functions. We are able to achieve the following result which may be of independent interest:

Lemma 1. (Informal) Let f be an N-party function. Protocol 6 is an MPC protocol computing f in the plain (authenticated broadcast) model and secure against fully malicious adversaries corrupting up to $t < N$ parties. Its communication and state complexity depend only

[2] Intuitively, semi-malicious adversaries can be viewed as semi-honest adversaries which are allowed to freely choose their random tapes.

on security parameters, number of parties, and the input and output sizes. In particular, the communication and state complexity of Protocol 6 is independent of the function f.

Using this MPC protocol in combination with our first construction, under the assumptions that we rely on in our main construction and in the MPC construction with optimized communication and state complexity, we achieve the following:

Corollary 2. (Informal) There exists an MPC protocol π' in the blockchain model that has adversarial threshold $t < N$, provides security with abort against fully-malicious adversaries and does not require participants to be online at the same time. Only a single message is required from the MPC contributors (the evaluators might be required to produce multiple messages). Furthermore, the number of calls to CSaR of this protocol is independent of the function that is being computed using this MPC protocol.

Non-Interactive MPC with Guaranteed Output Delivery (GoD). We need to modify our construction in order to provide guaranteed output delivery. In order to achieve GoD, we require the protocol π to have the GoD property as well, and thus the majority of the participants in π (recall that these are exactly the contributors in π') must be honest [3]. While making this change (in addition to a few minor adjustments) would be enough to guarantee GoD in our construction in the setting with only a single evaluator, it is certainly not sufficient when there are multiple evaluators, some of them dishonest. This is due to the following issue: since we must prevent an adversary from executing honest garbled circuits on multiple different inputs, we cannot simply allow each evaluator to execute garbled circuits on the inputs of its choosing. In particular, the CSaR release conditions must ensure that for each wire only a single key is revealed. In our first construction this results in the malicious evaluator being able to prevent an honest evaluator from executing the garbled circuits as intended by submitting an invalid first message for any round. Thus, to ensure guaranteed output delivery while maintaining secrecy, we must ensure that a malicious evaluator posting a wrong message does not prevent an honest evaluator from posting a correct message and using it for the key reveal. In particular, we will ensure that only a correct (for a definition of "correctness" explained below) message can be used for the wire key reveal.

Note that the inputs to the garbled circuits depend on the evaluators' outputs from the previous rounds. Checking the "correctness" of the evaluators' outputs is not entirely straight-forward since an honest execution of a garbled circuit which was submitted by a dishonest party might produce outputs which look incorrect (for example, have invalid signatures). Thus, simply letting the CSaR system check the signatures on the messages supplied by the evaluators might result in an honest evaluator being denied the wire keys for the next round.

---

[3] Note that there is no such restriction on the evaluators in π.

In our GoD construction we overcome this issue largely using the following adjustments:

- all initial messages containing garbled circuits and wire keys are required to be posted before some deadline.
- we use a CSaR with public release (whenever a secret is released, it is released publicly, and the information can be viewed by anyone).
- we ensure that it is possible to distinguish between the case where the evaluator is being dishonest, and the case where the evaluator is being honest, but the contributor in π supplied invalid garbled circuits or keys or did not supply some required piece of information.

We achieve the last point by designing the CSaR release condition in a way that it verifies that the evaluator's output can be explained by the information stored by the contributors in π. In particular, as part of the CSaR's release condition, we require a proof of correct execution for the garbled circuit outputs. The relation that the CSaR system is required to check in this case is roughly as follows: "The execution of the garbled circuit GC on the wire keys $\{k_i\}_{i\in I}$ result in the output provided by E. Here, the garbled circuit GC is the circuit, and $\{k_i\}_{i\in I}$ are the keys for this circuit reconstructed using the values published by the CSaR which are present on the proof of publish supplied by E". Note that due to the switch to the CSaR with public release, the wire keys used for the computation are indeed accessible to the CSaR system after their first release.

Similar to our first construction, we eventually reduce the security of the new protocol to the security of the original protocol. In addition to our first construction however, since the CSaR system is now able to verify messages submitted by the evaluators, honest evaluators are always able to advance in the protocol execution. This insight allows us to ensure that honest evaluators do not need to abort with more than a negligible probability along the way. Thus, if the underlying protocol π achieves guaranteed output delivery, the protocol we propose achieves guaranteed output delivery as well.

The statement about our GoD construction is given below.

Lemma 2. (Informal) Any MPC protocol π which is secure against fully malicious adversaries and provides guaranteed output delivery can be transformed into another MPC protocol π' in the blockchain model that provides security with guaranteed output delivery against fully malicious adversaries and does not require participants to be online at the same time. The MPC contributors are required to participate for only a single round (the evaluators might be required to participate for multiple rounds). The adversary is allowed to corrupt as many MPC contributors in π' as it is allowed to corrupt participants in π. The adversary is allowed to corrupt any number of evaluators.

5.0.3 Related Work

Closest to our work is the line of research that studies non-interactive multiparty computation [78-80], initiated in 1994 by Feige et al. [78], in which a number of parties submit

a single message to a server (evaluator) that, upon receiving all of the messages, computes the output of the function. In their work, Feige et al. allow the messages of the parties to be dependent on some shared randomness that must be unknown to the evaluator. Unfortunately, this means that if the evaluator is colluding with one or more of the participants, the scheme becomes insecure. Overcoming this restriction, Halevi et al. [80] started a line of work on non-interactive collusion-resistant MPC. Their model of computation required parties to interact sequentially with the evaluator (in particular, the order in which the clients connect to the evaluator is known beforehand). Beimel et al. [81] and Halevi et al. [82] subsequently removed the requirement of sequential interaction. Further improving upon these results, the work of Halevi et al. [79] removed the requirement of a complex correlated randomness setup that was present in a number of previous works. Halevi et al. [79] work in a public-key infrastructure (PKI) model in combination with a common random string. As the authors point out, PKI is the minimal possible setup that allows one to achieve the best-possible security in this setting, where the adversary is allowed to corrupt the evaluator and an arbitrary number of parties and learn nothing more than the so-called "residual function", which is the original function restricted to the inputs of the honest parties. In particular, this means that the adversary is allowed to learn the outcome of the original function on every possible choice of adversarial inputs.

Our work differs from the line of work on non-interactive MPC described above in a number of aspects. In contrast to those works, our construction is not susceptible to the adversary learning the residual function – roughly because the adversary must effectively "commit" to its input, and the CSaR system ensures that the adversary only receives a single set of wire keys per honest garbled circuit (the set of wire keys that aligns with the adversarial input). Additionally, in our work the parties do not need to directly communicate with the evaluator. In fact, in our construction that ensures guaranteed output delivery, any party can spontaneously decide to become an evaluator and still receive the result – there is no need to rerun the protocol in this case.

Related to us are also the works on reusable non-interactive secure computation (NISC) [83-87], initiated by Ishai et al. [88]. Intuitively, reusable NISC allows a receiver to publish a reusable encoding of its input x in a way that allows any sender to let the receiver obtain f(x,y) for any f by sending only a single message to the receiver. In our work, we focus on a multi-party case, where a party that does not need the output is not required to wait for other parties to submit their inputs.

Recently, Benhamouda and Lin [89] proposed a model called multiparty reusable Non-Interactive Secure Computation (mrNISC) Market that beautifully extends reusable NISC to the multiparty setting. In this model, parties first commit their inputs to a public bulletin board. Later, the parties can compute a function on-the-fly by sending a public message to an evaluator. An adversary who corrupts a subset of parties learns nothing more about the secret inputs of honest parties than what it can derive from the output of the computation. Importantly, the bulletin board commitments are reusable, and the security guarantee continues to hold even

if there are multiple computation sessions. In their work, Benhamouda and Lin mention that any one-round construction is susceptible to the residual attacks and thus slightly relax the non-interactive requirement in order to solve this problem. Indeed, their construction can be viewed as a 2-round MPC protocol with the possibility to reuse messages of the first round for multiple computations. Our scheme shows that when using blockchains it is indeed possible to provide a construction that requires only a single round of interaction from the parties supplying the input and is nonetheless not susceptible to residual attacks.

Concurrent to our work, Almashaqbeh et al. [89] recently published a manuscript which focuses on designing non-interactive MPC protocols which use blockchains to provide short term security without residual leakage. They focus on the setting where the inputs of all but one of the parties are public. In this setting, designing one-round MPC can be done easily by having all parties send their input to the only party which holds the secret input. This party can then compute the output and distribute it to other parties. The authors are able to extend the setting to the two-party semi-honest private input setting where one round protocols for the party not getting the output can be easily designed as well. While our protocol provides a worst-case security guarantee, they focus on an incentive-based notion of security. While both constructions bypass the residual leakage issue, their security guarantees might degrade with time. The key challenge in their setting is fairness / guaranteed output delivery which they solve using an incentive-based model of security. Hence their work is essentially unrelated to ours.

Finally, recently two works ([90, 91]) appeared which are inspired by blockchains and focus on improving the flexibility of the MPC protocols. Choudhuri et al. [90] proposed the notion of fluid MPC which allows parties to dynamically join and leave the computation. Gentry et al. [91] proposed the YOSO ("You Only Speak Once") model which focuses on stateless parties which can only send a single message. Similar to us, their constructions allow the MPC participants to leave after the first round if they are not interested in learning the output. However, to execute the MPC protocol both Choudhuri et al. and Gentry et al. require a number of committees of different parties which interact with each other, and each committee must provide honest majority. Our protocol preserves privacy of inputs even if there is a single evaluator who is dishonest.

5.1. Preliminaries

In this section we briefly discuss cryptographic building blocks used in our system.

5.1.1 MPC

In our work we consider MPC that allow a set of parties $P = \{P1,...,Pn\}$ to securely compute the output of some function f. We specifically consider MPC protocols in the broadcast model

4, where all parties have access to a broadcast channel and each round consists of parties broadcasting messages to other parties that participate in the protocol. An MPC protocol specifies for each party and each round the so-called next-message function, which defines the computation that is performed by that particular party in that round, as well as the message that the party broadcasts in that round and the state that is passed to the next round. More formally:

Definition 1. Given an interactive broadcast-only d-round MPC protocol, the next-message function for round i of party P_j is the function (mij,sij) ←

$f(x_j, r_j^i, m^i, s_j^{i-1})$, where x_j is Pj's input in the MPC protocol, r_j^i is the local randomness used by party Pj in round i, $m^i = m_1^{i-1} \| m_2^{i-1} \| \ldots \| m_n^{i-1}$ is the concatenation of messages received by each party in round i−1 (note that since we consider a broadcast protocol all parties receive the same message), s_j^i is an auxiliary state information output by P_j in round i ($s_{0j} = \perp$), and m_{ij} is the message output by P_j in round i.

Note: we assume that if a message from round $k < i - 1$ is needed in round i, it is incorporated in all of P_j's state messages from s_j^{k+1} to s_j^{i-1}.

Regarding the security of the MPC protocol, we consider the standard simulationbased notion. In the ideal world parties interact with the ideal functionality FMPC, described in Functionality 1. In the real world, parties engage in the real-world MPC protocol π in the presence of an adversary A, who is allowed to corrupt a set $I \subset [n]$ of parties and may follow an arbitrary polynomial-time strategy. Security of π is defined as follows:

Definition 2. A protocol π is said to securely compute F with abort if for every PPT adversary A in the real world, there exists a PPT adversary S, such that for any set of corrupted parties $I \subset [n]$ with $|I| \leq t$ (where t is the adversarial threshold), every initial input vector $(x_1,...,x_n)$, and every security parameter λ, it holds that

$$\{\text{IDEAL}^{f,S(z),I}(1^\lambda, (x_1, \ldots, x_n))\} =_c \{\texttt{REAL}_{\pi,A(z),I}(1^\lambda, (x_1, \ldots, x_n))\},$$

where $z \in \{0,1\}^*$ is the auxiliary input, IDEALf,S(z),I denotes the output of the interaction of the adversary S(z) (who corrupts parties in I) with the ideal functionality (this output consists of the output of the adversary S(z) as well as the outputs of the honest parties), and REALπ,A(z),I denotes the output of protocol π given the adversary A(z) who corrupts parties in I (this output consists of the output of the adversary A(z) as well as the outputs of the honest parties).

Finally, in our constructions we additionally assume that the underlying protocol π has the property that given the transcript of the protocol execution, the output can be publicly computed (as defined in [92]):

---

[4] Note that we will relax this requirement later, also allowing MPC protocols which use secure point-to-point channels.

Definition 3 (Publicly Recoverable Output). Given a transcript τ of an execution of a protocol π, there exists a function Eval such that the output of the protocol π for all parties is given by $y = \text{Eval}(\tau)$.

In one of our constructions, we consider MPC protocols which provide guaranteed output delivery. In that case the security of protocol π is defined the same way as before, except that the ideal functionality is now FMPC-GoD, described in Functionality 2.

5.1.2 Yao's Grabled Circuits

One of the core building blocks in our construction are Yao's garbled circuits that allow secure two-party computation. In the following, we provide definitions for the garbling process as well as the security of garbling scheme (taken verbatim from [71]):

Definition 4 (Garbling scheme). A garbling scheme for circuits is a tuple of PPT algorithms GC := (Gen,Garble,Eval) such that:

-

$(\{\mathtt{lab}^{w,b}\}_{w\in\mathtt{inp},b\in\{0,1\}}) \leftarrow \mathtt{Gen}(1^\lambda, \mathtt{inp})$: Gen *takes the security parameter* 1^λ *and length of input for the circuit as input and outputs a set of input labels* $\{\mathtt{lab}^{w,b}\}_{w\in\mathtt{inp},b\in\{0,1\}}$.

Functionality 1. $\mathcal{F}_{\text{MPC}}$

1. Let the set of MPC participants be $\mathcal{P} = \{P_1, \ldots, P_n\}$.
2. Let x_i denote the input of the party $P_i \in \mathcal{P}$.
3. The adversary S selects a set $I \subset [n]$ of corrupted parties.
4. Each honest party P_i sends its input $x_i^* = x_i$ to $\mathcal{F}_{\text{MPC}}$. For each corrupted party P_j, the adversary may select any value x_j^* and send it to $\mathcal{F}_{\text{MPC}}$.
5. $\mathcal{F}_{\text{MPC}}$ computes $F(x_1^*, \ldots, x_n^*) = (y_1, \ldots, y_n)$ and sends $\{y_i\}_{i\in I}$ to the adversary.
6. The adversary sends either `abort` or `continue` to $\mathcal{F}_{\text{MPC}}$.
   - If the adversary sent `abort`, $\mathcal{F}_{\text{MPC}}$ sends $\perp$ to each honest party.
   - Otherwise, $\mathcal{F}_{\text{MPC}}$ sends y_i to each honest party P_i.
7. Each honest party P_i outputs the message it received from $\mathcal{F}_{\text{MPC}}$. Each adversarial party can output an arbitrary PPT function of the adversary's view.

Functionality 2. $\mathcal{F}_{\text{MPC-GoD}}$

1. Let the set of MPC participants be $\mathcal{P} = \{P_1, \ldots, P_n\}$.
2. Let x_i denote the input of the party $P_i \in \mathcal{P}$.
3. The adversary S selects a set $I \subset [n]$ of corrupted parties.
4. Each honest party P_i sends its input $x_i^* = x_i$ to $\mathcal{F}_{\text{MPC-GoD}}$. For each corrupted party P_j, the adversary may select any value x_j^* and send it to $\mathcal{F}_{\text{MPC-GoD}}$.
5. $\mathcal{F}_{\text{MPC}}$ computes $F(x_1^*, \ldots, x_n^*) = (y_1, \ldots, y_n)$, substituting each missing value by some default value.
6. $\mathcal{F}_{\text{MPC-GoD}}$ sends y_i to each party P_i.
7. Each honest party P_i outputs the message it received from $\mathcal{F}_{\text{MPC-GoD}}$. Each adversarial party can output an arbitrary PPT function of the adversary's view.

- $\bar{C} \leftarrow$ Garble(C,($\{lab^{w,b}\}_{w\in inp,b\in\{0,1\}}$) : Garble takes as input a circuit
- C : $\{0,1\}^{inp} \rightarrow \{0,1\}^{out}$ and a set of input labels $\{lab^{w,b}\}_{w\in inp,b\in\{0,1\}}$ and outputs the garbled circuit $\bar{C}$.
- y $\leftarrow$ Eval($\bar{C}$, lab^x) : Eval takes as input the garbled circuit $\bar{C}$, input labels labx corresponding to the input x $\in \{0,1\}^{inp}$ and outputs y $\in \{0,1\}^{out}$.

The garbling scheme satisfies the following properties:

1. Correctness: For any circuit C and input x $\in \{0,1\}^{inp}$,

$$\Pr[C(x) = \text{Eval}(C, \bar{} \text{lab}^x)] = 1,$$

where $(\{\text{lab}^{w,b}\}_{w\in inp, b\in\{0,1\}}) \leftarrow \text{Gen}(1^\lambda, inp)$ and $C^{\bar{}} \leftarrow \text{Garble}(C, \{\text{lab}^{w,b}\}_{w\in inp, b\in\{0,1\}})$.

2. Selective Security: There exists a PPT simulator Sim_{GC} such that, for any PPT adversary A, there exists a negligible function $\mu(\cdot)$ such that

$$|Pr[\texttt{Experiment}_{\mathcal{A},\texttt{Sim}_{GC}}(1^\lambda, 0) = 1] - Pr[\texttt{Experiment}_{\mathcal{A},\texttt{Sim}_{GC}}(1^\lambda, 1) = 1]| \leq \mu(\lambda)$$

where the experiment $\text{Experiment}_{A,\text{SimGC}}(1^\lambda, b)$ is defined as follows:

(a) The adversary A specifies the circuit C and an input $x \in \{0,1\}^{inp}$ and gets $C^{\bar{}}$ and $\text{lab}^{\hat{}}$, which are computed as follows: – If b = 0:

- $(\{\text{lab}^{w,b}\}_{w\in inp, b\in\{0,1\}}) \leftarrow \text{Gen}(1^\lambda, inp)$
- $C^{\bar{}} \leftarrow \text{Garble}(C, (\{\text{lab}^{w,b}\}_{w\in inp, b\in\{0,1\}})$

- If b = 1:
- $(C, \bar{} \text{lab}^{\hat{}}) \leftarrow \text{SimGC}(1^\lambda, C(x))$
- The adversary outputs a bit b′, which is the output of the experiment.

We note that Yao's protocol achieves selective security. Very roughly, the security of the party producing the garbled circuit relies on the fact that for each wire of the circuit, only a single garbled key is revealed, and thus the only information the other party gets is the (garbled) output. We refer to the work of Lindell and Pinkas for the details of the construction as well as the security proof .

5.1.3 Append-only Bulletin Boards

In our construction, we rely on public bulletin boards. Specifically, we require that the bulletin boards allows parties to publish arbitrary strings and receive a confirmation (dubbed "proof of publish") that the string was published in return.

Following the approach of Kaptchuk et al, we assume that parties publish their strings as part of a public chain of values, and abstract the bulletin board syntax as follows:

- $(\text{post}, \sigma) \leftarrow \text{Post}(M)$. Intuitively, when a party wishes to post some data M on the public chain, the Post function is called. This call results in post (which consists of M, as well as additional data which identifies this data record on the chain) being appended to the chain. The tuple (post, σ), where σ is the proof of publish, is returned. We assume that a proof of publish is public and can be retrieved for already published posts as well.
- $\{0,1\} \leftarrow \text{Verify}(\text{post}, \sigma)$. The public verification algorithm takes as input a supposedly published record post as well as a proof of publish σ, and verifies that the record post has indeed been published.

Security-wise, we require that the contents of the bulletin board are hard to erase or modify and that the proof of publish is unforgeable. Specifically, we require that up to a negligible probability it is impossible to come up with a pair (post,σ) such that Verify(post,σ) = 1, unless this pair has been generated through a call to the Post algorithm. This property holds even if the adversary is given an oracle that posts arbitrary strings on the bulletin board on the behalf of the adversary.

Such bulletin boards have been extensively investigated in prior works. While specific syntax details of the bulletin board abstraction slightly vary throughout these works, they all ensure that parties are able to post arbitrary strings on an append-only log, and the proof of publish cannot be forged. These works also point out that bulletin boards with the properties described above already exist in practice. They can be realized from centralized systems such as the Certificate Transparency project, and from the decentralized systems such as proof-of-work or proof-of-stake blockchains.

5.1.4 CSaRs

In our work, we rely on what we call conditional storage and retrieval systems (CSaRs) that allow for a secure storage and retrieval of secrets. In more detail, the user who stores the secret with a CSaR specifies a release condition, and the secret is released if and only if this condition is satisfied. While such systems could be realized via a trusted third party, they can also be realized using a set of parties with the guarantee that some sufficiently large subset of these parties is honest. A user can then distribute its secret between the set of parties, and the CSaR's security guarantee ensures that no subset of parties that is smaller than a defined threshold can use its secret shares to gain information about the secret. Recently, multiple independent works appeared that use blockchains to provide such functionality. We provide a clean definition of the core functionality that these works aim to provide (without fixating on blockchains) and outline why the eWEB system satisfies this definition.

Formally, the ideal CSaR functionality is described in Figure 3. The security of a CSaR system is then defined as follows:

Fig. 3. Ideal CSaR: $\mathsf{Ideal}_{\mathrm{CSaR}}$

1. **SecretStore** Upon receiving an (identifier, release condition, secret) tuple $\tau = (id, F, s)$ from a client P, $\mathsf{Ideal}_{\mathrm{CSaR}}$ checks whether id was already used. If not, $\mathsf{Ideal}_{\mathrm{CSaR}}$ stores τ and notifies all participants that a valid storage request with the identifier id and the release condition F has been received from a client P. Here, the release condition is an NP statement.
2. **SecretRelease** Upon receiving an (identifier, witness) tuple (id, w) from some client C, $\mathsf{Ideal}_{\mathrm{CSaR}}$ checks whether there exists a record with the identifier id. If so, $\mathsf{Ideal}_{\mathrm{CSaR}}$ checks whether $F(w) = true$, where F is the release condition corresponding to the secret with the identifier id. If so, $\mathsf{Ideal}_{\mathrm{CSaR}}$ sends the corresponding secret s to client C.

CSaR Security For any PPT adversary A there exists a PPT simulator S with access to our security model IdealCSaR (described in Ideal CSaR), such that the view of A interacting with S is computationally indistinguishable from the view in the real execution.

5.1.5 MPC in the Presence of Contributors and Evaluators

In the following, we formally define the security of the functionality which we want to achieve. Recall that we consider two sets of parties – MPC contributors who supply inputs and MPC evaluators who wish to obtain the output.

We consider the simulation-based notion of security. In the ideal world, parties interact with the ideal functionality Feval-MPC, described in Figure 4. Note that the difference to the standard ideal functionality for MPC with abort (described in Figure 1) is that we distinguish between contributors and evaluators.

In the real world, parties execute the protocol π in the presence of an adversary A. The adversary A is allowed to corrupt a set of contributors $I \subset [n]$ as well as a set of evaluators $I' \subset [n']$. A is allowed to send messages in place of corrupted parties and can follow an arbitrary polynomial-time strategy. Security of π is defined as follows:

Definition 5. A protocol π is said to securely compute F with abort in the presence of contributors and evaluators if for every PPT adversary A in the real world, there exists a PPT adversary S, such that for any set of corrupted evaluators $I' \subset [n']$, any set of contributors $I \subset [n]$ with $|I| \leq t$ (where t is the adversarial threshold), every initial input vector (x1,...,xn), and every security parameter λ, it holds that

$$\{\mathtt{IDEAL}_{f,S(z),I}(1^\lambda, (x_1, \ldots, x_n))\} =_c \{\mathtt{REAL}_{\pi,A(z),I}(1^\lambda, (x_1, \ldots, x_n))\},$$

where $z \in \{0,1\}*$ is the auxiliary input, $\mathrm{IDEAL}_{f,S(z),I}$ denotes the output of the interaction of the adversary S(z) (who corrupts parties in I) with the ideal functionality Feval-MPC (this output consists of the output of the adversary S(z) as well as the outputs of the honest parties), and $\mathrm{REAL}_{\pi,A(z),I}$ denotes the output of the interaction between the adversary A(z) who

corrupts parties in I and the honest parties in the protocol π (this output consists of the output of the adversary A(z) as well as the outputs of the honest parties).

In one of our constructions, we consider MPC protocols which provide guaranteed output delivery. In that case the security of protocol π is defined the same way as before, except that the ideal functionality is now Feval-MPC-GoD, described in Functionality 5.

5.1.6 Multi-Key FHE with Distributed Setup

Our construction of an MPC scheme which combined communication and state complexity is independent of the function being computed is based on the MPC protocol of Brakerski et al.

Functionality 4. $\mathcal{F}_{\text{eval-MPC}}$

1. We distinguish between the set of MPC contributors $\mathcal{P} = \{P_1, \ldots, P_n\}$ and the set of evaluators $\mathcal{E} = \{E_1, \ldots, E_{n'}\}$. These sets can be, but do not need to be disjoint.
2. Let x_i denote the input of the party $P_i \in \mathcal{P}$.
3. The adversary S selects a set of contributors $I \subset [n]$ to corrupt.
4. The adversary S selects a set of evaluators $I' \subset [n']$ to corrupt.
5. Each honest party P_i sends its input $x_i^* = x_i$ to $\mathcal{F}_{\text{eval-MPC}}$. For each corrupted party P_j, the adversary may select any value x_j^* and send it to $\mathcal{F}_{\text{eval-MPC}}$.
6. $\mathcal{F}_{\text{eval-MPC}}$ computes $F(x_1^*, \ldots x_n^*)$ and sends $F(x_1^*, \ldots x_n^*)$ to the adversary.
7. The adversary sends either `abort` or `continue` to $\mathcal{F}_{\text{eval-MPC}}$.
   - If the adversary send `abort`, $\mathcal{F}_{\text{eval-MPC}}$ sends $\perp$ to all honest evaluators.
   - Otherwise, $\mathcal{F}_{\text{eval-MPC}}$ sends $F(x_1^*, \ldots x_n^*)$ to each honest evaluator.
8. Each honest evaluator outputs the message it received from $\mathcal{F}_{\text{eval-MPC}}$. Each adversarial party can output an arbitrary PPT function of the adversary's view.

Functionality 5. $\mathcal{F}_{\text{eval-MPC-GoD}}$

1. We distinguish between the set of MPC contributors $\mathcal{P} = \{P_1, \ldots, P_n\}$ and the set of evaluators $\mathcal{E} = \{E_1, \ldots, E_{n'}\}$. These sets can be, but do not need to be disjoint.
2. Let x_i denote the input of the party $P_i \in \mathcal{P}$.
3. The adversary S selects a set of contributors $I \subset [n]$ to corrupt.
4. The adversary S selects a set of evaluators $I' \subset [n']$ to corrupt.
5. Each honest party P_i sends its input $x_i^* = x_i$ to $\mathcal{F}_{\text{eval-MPC}}$. For each corrupted party P_j, the adversary may select any value x_j^* and send it to $\mathcal{F}_{\text{eval-MPC}}$.
6. $\mathcal{F}_{\text{eval-MPC}}$ computes $F(x_1^*, \ldots x_n^*)$ and sends it to the adversary as well as each honest evaluator.
7. Each honest evaluator outputs the message it received from $\mathcal{F}_{\text{eval-MPC-GoD}}$. Each adversarial party can output an arbitrary PPT function of the adversary's view.

[15], which in turn utilizes multi-key fully homomorphic encryption scheme with distributed setup. In the following, we formally define this primitive (in large parts taken verbatim from Brakerski et al. [15]).

Definition 6 (Multi-key homomorphic encryption scheme). A multi-key homomorphic encryption scheme with distributed setup consists of five procedures, MFHE = (MFHE.DistSetup,MFHE.Keygen,MFHE.Encrypt,MFHE.Decrypt,MFHE.Eval) :

- Setup $\mathsf{params}_i \leftarrow \mathsf{MFHE.DistSetup}(1^\kappa, 1^N, i)$: On input the security parameter κ and number of users N, outputs the system parameters for the i-th player params_i . Let $\mathsf{params} = \{\mathsf{params}_i\}_{i\in[N]}$.
- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{MFHE.Keygen}(\mathsf{params}, \mathsf{i})$: On input params and entry number i the key generation algorithm outputs a public/secret key pair $(\mathsf{pk}, \mathsf{sk})$.
- $c \leftarrow \mathsf{MFHE.Encrypt}(\mathsf{pk},x)$: On input pk and a plaintext message $x \in \{0,1\}^*$ output a "fresh ciphertext" c. (We assume for convenience that the ciphertext includes also the respective public key.)
- $\hat{c} := \mathsf{MFHE.Eval}(\mathsf{params};\mathcal{C}; (c_1, \ldots, c_l))$: On input a (description of a) Boolean circuit $\mathcal{C}$ and a sequence of fresh ciphertexts $(c_1, \ldots, c_l)$, output an "evaluated ciphertext" $\hat{c}$. (Here we assume that the evaluated ciphertext includes also all the public keys from the c_i's.)
- $x := \mathsf{MFHE.Decrypt}((\mathsf{sk}_1, \ldots, \mathsf{sk}_\mathsf{N}), \hat{\mathsf{c}})$: On input an evaluated ciphertext $\hat{c}$ (with N public keys) and the corresponding N secret keys $(\mathsf{sk}_1, \ldots, \mathsf{sk}_\mathsf{N})$, output the message $x \in \{0,1\}^*$.

The scheme is correct if for every circuit C on N inputs and any input sequence $x_1,...,x_N$ for C, we set paramsi ← MFHE.DistSetup$(1^\kappa, 1^N, i)$, params = {paramsi}i∈[N], and then generate N key-pairs and N ciphertexts (pki,ski) ←

MFHE.Keygen(params) and ci ← MFHE.Encrypt(pki,xi), then we get MFHE.Decrypt((sk1,...,skN),MFHE.Eval(params;C;(c1,...,cN))) = C(x_1,...,x_N)

except with negligible probability (in κ) taken over the randomness of all these algorithms.

In the work of Brakerski et al. the following two properties are needed of the multi-key FHE schemes: first, the decryption procedure consists of a "local" partial-decryption procedure evi ← MFHE.PartDec(c,ˆ ski) that only takes one of the secret keys and outputs a partial decryption share, and a public combination procedure that takes these partial shares and outputs the plaintext, x ← MFHE.FinDec(ev1,...,evN,cˆ). Another property that is needed is the ability to simulate the decryption shares. Specifically, there exists a PPT simulator ST, that gets for input:

- the evaluated ciphertext ˆc,
- the output plaintext x := MFHE.Decrypt((sk1,←,skN),cˆ),
- a subset I ⊂ [N], and all secret keys except the one for I, $\{\mathrm{skj}\}_{j\in[N]\setminus I}$.

- The simulator produces as output simulated partial evaluation decryption shares:

$\{ev\tilde{}i\}_{i\in I} \leftarrow S^T(x,c,I,\hat{} \{skj\}_{j\in[N]\setminus I})$. We want the simulated shares to be statistically close to the shares produced by the local partial decryption procedures using the keys [93]i∈I, even conditioned on all the inputs of S^T. A scheme is simulatable if it has local decryption and a simulator as described here.

Brakerski et al. require that semantic security for the i-th party holds even when all $\{paramsj\}j_{\in[N]\setminus i}$ are generated adversarially and possibly depending on paramsi.

They consider a rushing adversary that chooses N and i ∈ [N], then it sees paramsi and produces paramsj for all j ∈ [N]\{i}. After this setup, the adversary is engaged in the usual semantic-security game, where it is given the public key, chooses two messages and is given the encryption of one of them, and it needs to guess which one was encrypted.

Simulatability of the decryption shares is defined as before, but now the evaluated ciphertext is produced by the honest party interacting with the same rushing adversary (and statistical closeness holds even conditioned on everything that the adversary sees).

5.2 Our Non-Interactive MPC Construction

We now present our first construction - given an MPC protocol π, we use Yao's garbled circuits as well as a CSaR to transform it into an MPC protocol π′ that does not require parties to be online at the same time and only requires a single message from the contributors in π. The contributors in π do not need to interact with each other. First, we briefly outline the assumptions we make and define the adversarial model.

Assumptions. We assume a public-key infrastructure and the existence of a CSaR. To distinguish between concurrent executions of the protocol, we give each computation a unique identifier id, and we assume that the evaluators know the public keys of the parties eligible to contribute in the protocol π. We assume the existence of a bulletin board modeled as an append-only log that provides a proof of publish which cannot be (efficiently) forged. Such bulletin boards can be implemented in practice via a blockchain. Finally,we assume IND-CCA secure public key encryption.

For the ease of presentation, we assume the following about the MPC protocol π: (a) it is in a broadcast model, and (b) it has a single output which is made public to all participants in the last round [5].

[5] Note that these are not real limitations: if a protocol has several outputs, some of which cannot be made public, the MPC functionality broadcasts the encryption of a party's output under that party's public key. Additionally, later in this section we discuss how protocols with point-to-point channels can be supported in the broadcast model.

Adversary model. We consider a computationally bounded, fully malicious, static adversary A. Once an adversary corrupts a party it remains corrupted: the adversary is not allowed to adaptively corrupt previously honest parties.

5.2.1 Construction Overview

Intuitively, there are two main steps in the protocol. In the first step, the parties (dubbed "contributors") prepare the garbled circuits (and keys) and store these with the CSaR. In the second step, one or more parties (we dub them "evaluators") use the garbled circuits to execute the original protocol π.

Step 1. Preparing Garbled Circuits and Keys. Each party P_j that wishes to participate (contribute inputs) in π starts by garbling the slightly modified nextmessage functions of each round of π. Typically, the next-message function takes as input some subset of the following: the secret input of the party, local randomness of the party for that particular round, the messages received in the previous rounds, some secret state passed along from the previous round. The output consists of the message that is broadcast as well as the state that is passed to the next round. We make the following modifications: in each round i, instead of the state s_j^i that is passed to the next round, the function outputs the encryption c_j^i of the state as well as a signature sigprji over this encryption. Additionally, the modified next-message function outputs the public message m_j^i that is supposed to be broadcast by P_j in this round, as well as the signature sigpubij over this message. The secret key as well as the signature key of P_j are hard-coded in the circuit (we explain how it can be done later in this section). Prior to executing the original next-message function, the modified function decrypts the state using the hard-coded secret key of P_j and verifies the signatures on each public message as well as the signature on the state passed in from previous round. Intuitively, these modifications are due to the following reasons:

- The state of the party is passed in an encrypted state because the state information is assumed to be private in the original MPC construction.
- The parties need to sign their messages (and verify signatures on the messages passed as inputs) since we must prevent the adversary from tricking an honest party into acceptance of a message that is supposedly generated by another honest party, but in reality is mauled by the adversary.

Once the garbled circuits are prepared, P_j stores the garbled circuits with CSaR. Note that the next-round functions in particular take messages produced by other parties as inputs. Thus, there is no way for the party to know at the time the garbled circuits are constructed, whether the key corresponding to bit 0 or the key corresponding to bit 1 will be chosen for some wire w. To allow an evaluator to execute the garbled circuits anyway, P_j additionally stores both wire keys for each input wire with CSaR, each with a separate CSaR request. This needs to be done for every single round, since in any particular round the inputs will depend on the messages produced by the garbled circuits of other parties in the previous round.

Intuitively, in order to be able to reduce the security of this protocol to the security of the original MPC protocol, we need to ensure not only that the adversary is not able to maul messages of the honest parties and see the parties' private information, but also that the protocol is executed in order and there is only a single instance of the protocol running. This is ensured by carefully constructing conditions that must be met in order to release the garbled circuits and wire keys. In order to release a garbled circuit for some round i, a party needs to provide a proof that the execution of the protocol up to and including round i − 1 is finalized. In order to release a wire key corresponding to bit b on a wire corresponding to position p of the input to some garbled circuit, a party needs to additionally provide a proof that the input bit to position p in this circuit is indeed bit b. In the following, we first explain how the protocol is executed, and then explain how exactly the release conditions look like.

Step 2. Executing π. Once all required information is stored, an evaluator E can execute the original MPC protocol π. It is not required that E is one of the parties participating in the protocol π and in fact, there can be multiple evaluators (for simplicity, we refer to all of them as "E"). E executes the garbled circuits round-by-round. Once E has executed all garbled circuits for a certain round, E publishes the concatenation of the outputs of these circuits on a bulletin board. Then, E uses the proof of publishing of this message in order to release the garbled circuits as well as the wire keys of the next round.

First round optimization. Note that the message broadcast by the parties in the first round of the protocol π does not require any information from the other participants in the MPC protocol. Thus, instead of storing the garbled circuits for the first round, we let the parties publish their first message (and the signature on it) directly. The secret state that needs to be passed to the second round is hard-coded in the garbled circuit of the second round.

Release conditions. As described above, after the execution of all garbled circuits of the certain round, the evaluator is tasked with publishing the (concatenation of the) outputs of these circuits. This published message servers as a commitment to the evaluator's execution of this round, and this is what is needed to release the gabled circuits of the next round. We additionally require that the length of each published message is the same as expected by the protocol (corresponds to the number of input wires), and the correct length requirement holds for every part of this message (i.e., the public message, the signature over it, the state, and the signature over the state for each contributing party). In order to ensure that there is only a single evaluation of the original MPC running, only the very first published message that is of a correct form (i.e., satisfies the length requirements) can be used as the witness to release garbled circuits and keys of a certain round. We call such messages authoritative messages. Formally, the authoritative message of round d > 1 is a published message that satisfies the following conditions:

- Message is of the form (id,d,m), where m is of the form ($m_1^d \| \cdots \| m_n^d \|$
- $sigpub_1^d \| \cdots \| sigpub_n^d \| c_1^d \| \cdots \| c_n^d \| sigpr_1^d \| \cdots \| sigpr_n^d$). This corresponds to

- the concatenated output of the garbled circuits of round d: public messages followed by signatures over each public message, and encryptions of state followed by signatures over each ciphertext.
- each mdj, cdj, sigpubdj, sigprjd has correct length.
- This is the first published message that satisfies the requirements above.

Due to our first round optimization the authoritative message of the first round is slightly different. In particular, there are up to n authoritative messages for the first round – one for each contributing party. Formally, an authoritative message of round d = 1 from party Pk is a published message that satisfies the following conditions:

- Message is of the form (id,1,k,m1k,sigpub1k).
- m1k and sigpub1k both have correct length.
- This is the first published message that satisfies the requirements above.

In terms of authoritative messages, the release conditions can be now defined as follows: in order to release the garbled circuits for round i, we require that all authoritative messages for rounds 1 up to and including round $i - 1$ are published. In order to release the wire key for some bit b of an input wire w of a garbled circuit the authoritative message of the previous round must contain bit b at the same position w.

Identifying secrets In order for the evaluator to know the identifiers of the secrets it must request from CSaR, we require that upon storing the secrets (i.e., garbled circuits and wire keys), the contributors choose their CSaR secret identifiers (appending their own party identifier to the secret in order to ensure that it has not been used before) and publish those identifiers on the bulletin board (we assume messages can't be posted or stored by a party pretending to be another party). For readability purposes, further we exclude this detail from the construction description.

Removing point-to-point channels. While in our construction we assume that the original MPC protocol is in a broadcast model, it is very common for MPC protocols to assume secure point-to-point channels. We can handle such protocols as well since an MPC protocol that assumes point-to-point channels can be easily converted to a protocol in a broadcast model. A generic transformation is outlined in the eWEB paper (Protocols 1 and 2 in [28]), it requires using a protocol to "package" a message that must be sent and another protocol to "unpack" a message received by a party. Intuitively, these protocols rely on authenticated communication channels (which can be realized via signatures). The packaging is done via appending the id of the sender to the message and IND-CCA encrypting the resulting string. The unpacking is done via decrypting and verifying that the party id specified in the message corresponds to the id of the party who sent this message via the authenticated communication channel.

Hardcoding secret inputs. As mentioned above, some of the information used in the modified next-message function (such as the secrets of the parties, their secret keys etc.) is hardcoded in the circuit. Say the hardcoded input wire is w, and its value is (bit) b. Then, the party preparing the garbled circuit that uses w does so as follows: whenever one of the inputs to a gate is w, the party removes the wire corresponding to w from the circuit and computes the values in the ciphertexts using bit b only (instead of computing the output both for w = 0 and w = 1). We give an example for the computation of the AND-Gate in Figure 6. For security purposes, it is important that we do not perform any circuit optimizations based on the value of w.

x	w	out
0	0	K_0
0	1	K_0
1	0	K_0
1	1	K_1

x	out
0	K_0
1	K_0

Fig.5.1. On the left, we show the computation of the AND-gate in Yao's construction. Given the garbled keys of x and w, depending on whether they correspond to zero or one, the doubly-encrypted ciphertext contains K_0 or K_1. On the right, we show the computation for the AND-gate if w = 0. In this case, both ciphertexts contain K_0.

Notation. In the following, we denote party Pj's public and secret encryption key pair as (pkj,skj). We denote party Pj's signature and verification keys as sigkj and verkj. By mij we denote messages that are generated by the party Pj in the i-th round.

Further Details. Note that eWEB, the construction that we use as the instantiation of the CSaR, assumes a CRS. This requirement can be removed in our case by simply allowing each participant in the protocol π to prepare the CRS on its own. From a security standpoint, this is unproblematic – we only wish to protect the secrets of honest clients, and if a client is honest, it will generate the CRS honestly as well 6.

Additionally, we note that in eWEB the party storing the secret is required to send multiple messages. In order to ensure that in our MPC protocol a single message from the MPC participant is sufficient and the parties can go offline after sending this message, we slightly modify the eWEB construction. Roughly, in eWEB miners are tasked with jointly preparing a random value r s.t. each miner knows a share of r. The user then publishes the value s+r (where s denotes the secret to be stored), and the miners compute their shares of s by subtracting their shares of r from s + r. Along the way, the commitments to the sharing of s are made public. We modify it as follows: the user simply publishes the commitments to the sharing of s and sends

[6] Note that this change reduces the efficiency of the eWEB system – instead of batching secrets from different clients, only secrets from a single client can be processed together now.

shares of s (along with the witnesses) to the miners who then verify the correctness of the shares and witnesses.

Finally, note that we require that the original protocol π has the publicly recoverable output property (see Definition 3). For security with abort, this property can be easily achieved as follows: first, all parties broadcast the output. Then, if all parties broadcasted the same value, this value is taken as the output. Otherwise, protocol is considered to be aborted. In the following, for simplicity we assume that protocol π has the publicly recoverable output property and Eval denotes the algorithm used to retrieve the output from the transcript.

The full construction is given in Protocols 1 and 2 (preparation of the garbled circuits and keys), as well as Protocol 3 (execution phase).

Protocol 1 NON-INTERACTIVE MPC – *CircuitPreparationPhase*

1. P_j computes the output (m_j^1, s_j^1) of the first round of π. P_j computes the signature $sigpub_j^1$ on the message $(id, 1, j, m_j^1)$ using its signing key $sigk_j$. P_j posts $\mathtt{M}_\mathtt{j}^1 = (id, 1, j, m_j^1, sigpub_j^1)$ on the bulletin board.
2. P_j produces Yao's garbled circuits $\{GC_j^i\}$ for each round $i > 1$ based on the circuit C_j^i of the next message function f_j^i of the original MPC protocol π:

Protocol 2 NON-INTERACTIVE MPC – *KeyStoragePhase*

1. Securely store input wire keys $(\{\mathtt{lab}_j^{w,b,2}\}_{w \in \mathtt{inp}_\mathtt{j}^2, b \in \{0,1\}})$ for the circuit of the second round using CSaR. For each party P_k whose first round message is needed for the computation, the witness required to decrypt the wire key corresponding to the i-th bit of the input being 0 (resp. 1) is a **valid proof of publishing** of the following:
   (a) All of the authoritative messages of the first round.
   (b) i-th bit of the authoritative message of round 1 of Party P_k is 0 (resp. 1).
2. Securely store input wire keys $(\{\mathtt{lab}_j^{w,b,d}\}_{w \in \mathtt{inp}_\mathtt{j}^\mathtt{d}, b \in \{0,1\}})$ for the circuit of the d-th $(d \geq 3)$ round using CSaR. The witness needed to decrypt the wire key corresponding to the i-th bit of the input being 0 (resp. 1) is a **valid proof of publishing** of the following:
   (a) All of the authoritative messages of the first $d-1$ rounds.
   (b) i-th bit of the authoritative message of round $d-1$ is 0 (resp. 1).

(d) Compute $s_j^{i-1} = Dec_{sk_j}(c_j^{i-1})$.
(e) Obtain (m_j^i, s_j^i) by executing $f_j^i(x_j, r_j^i, m^i, s_j^{i-1})$, where $m^i = m_1^{i-1} \| \cdots \| m_n^{i-1}$.
(f) Compute the signature $sigpub_j^i$ on the public message (id, i, j, m_j^i) using the signing key $sigk_j$.
(g) Compute the encryption of the state $c_j^i = Enc_{pk_j}(s_j^i)$.
(h) Compute the signature $sigpr_j^i$ on the tuple (id, i, j, c_j^i) including the encryption of state using the signing key $sigk_j$.
(i) Output $(m_j^i, sigpub_j^i, c_j^i, sigpr_j^i)$.

3. P_j securely stores garbled circuits GC_j^i for all rounds $i > 1$ using a CSaR. The witness needed to release the garbled circuit of round i is a valid proof of publishing of all authoritative messages from round 1 and up to and including round $i-1$.

Security Analysis Intuitively, correctness of the construction as well as the secrecy of the honest parties' inputs follow from the correctness as well as security properties of the underlying cryptographic primitives as well as the original protocol π. We formally show security by providing a simulator in the ideal model and showing that no PPT adversary can distinguish between interaction with the simulator and the interaction with the honest parties. Intuitively, we rely on the security of the cryptographic primitives used in our construction to show that the adversary is not able to use a garbled circuit from an honest party in a "wrong" way. In

particular, the adversary cannot trick an honestly produced garbled circuit into accepting wrong inputs from other honest parties i.e., inputs that were not produced using the garbled circuits or published (for the first message) by those parties directly, or claim that a required message from some honest party is missing. Additionally, there is no way for the adversary to execute honest garbled circuits for the same round on inconsistent inputs (or execute a single honest garbled circuit multiple times on a different inputs) since only the authoritative message published for a single round is considered valid. We then rely on the security of the original protocol π.

5.3. Optimizations

Our next goal is to minimize the number of CSaR invocations in our construction. For this, we will focus on our main construction (Protocols 1, 2 and 3), but the optimizations are applicable to our guaranteed output delivery construction (which will be introduced later) as well.

Protocol 3 NON-INTERACTIVE MPC – *ExecutionPhase*

1. The evaluator E uses messages $(id, 1, z, m_z^1, sigpub_z^1)$ posted on the bulletin board by each party P_z as the proof of publishing to get the garbled circuits (and keys) for the second round stored in CSaR by each participant in π. Then, E computes the outputs $(m_j^2, sigpub_j^2, c_j^2, sigpr_j^2)$ of the second round by executing the garbled circuits.
2. If an authoritative message of the second round was not published on the bulletin board yet, set $m = (m_1^2 \| \cdots \| m_n^2 \| sigpub_1^2 \| \cdots \| sigpub_n^2 \| c_1^2 \| \cdots \| c_n^2 \| sigpr_1^2 \| \cdots \| sigpr_n^2)$, publish $\mathtt{M}^2 = (id, 2, m)$:

   $$(\mathtt{post}^2, \sigma^2) \leftarrow \mathtt{Post}(\mathtt{M}^2)$$

   and use the proof of publish σ^2 as the witness to decrypt the wire keys and the garbled circuits of the next round. If an authoritative message $(id, 2, m')$ was published on the bulletin board, use its proof of publishing as the witness if $m' = m$. Otherwise, stop the execution and output $\perp$.
3. In each following round $d \geq 3$, E executes each garbled circuit published by party P_z for round $d-1$. Then, E concatenates the outputs and checks if there is a message on the bulletin board for this round. If there is no such message, E posts the computed output $\mathtt{M}^{\mathtt{d}} = (id, d, m_1^{d-1} \| \cdots \| m_n^{d-1} \| sigpub_1^{d-1} \| \cdots \| sigpub_n^{d-1} \| c_1^{d-1} \| \cdots \| c_n^{d-1} \| sigpr_1^{d-1} \| \cdots \| sigpr_n^{d-1})$:

   $$(\mathtt{post}^{\mathtt{d}}, \sigma^d) \leftarrow \mathtt{Post}(\mathtt{M}^{\mathtt{d}})$$

   and uses the proof of publishing σ^d as witness to obtain input keys and garbled circuits of the next round. Otherwise, if a message for this round is already published and is the same as the one computed by E, E uses the proof of publishing of this message as the witness. If it is not the same message as the one computed by E, E aborts the execution.
4. Let τ denote the resulting transcript of execution of π. E outputs $\mathsf{Eval}(\tau)$ as the result.

Let n denote the number of parties participating in the original MPC protocol π, nrounds denote the number of rounds in$^{\pi}$, $n^i_{wires,j}$denote the number of input wires of a garbled circuit of the next-message function for round i of party Pj. Then, the number of CSaR secret store operations is upper bounded by:g

The term n ∗ (nrounds − 1) is due to the fact that each party needs to store a garbled circuit for each round, except for the very first one. The term $\sum_{i=2}^{n_{rounds}} \sum_{i=1}^{n} 2 * n^i_{wires,j}$ is added because each party also needs to store two wire keys for each input wire of each garbled circuit it publishes.

The number of CSaR secret release operations for each evaluator is upper bounded by:

$$N_{release} = n * (n_{rounds} - 1) + \sum_{i=2}^{n_{rounds}} \sum_{j=1}^{n} n^i_{wires,j}$$

This is because the evaluator needs all of the garbled circuits, as well as a single wire key for each input wire of each garbled circuit, to perform the computation.

Note that the dominant factor in both of the equations is $\sum_{i=2}^{n_{rounds}} \sum_{j=1}^{n} n^i_{wires,j}$.

This term is precisely the combined communication and (encrypted) state complexity of the original MPC protocol π, minus the messages of the first round and plus the signatures on the public messages and the state. Thus, in order to minimize the number of CSaR invocations, we must first and foremost optimize the combined communication and state complexity of the original MPC scheme. We discuss a possible way to do this in the next section.

5.4. Optimizing Communication and State Complexity in MPC

Our goal in this section is to design an MPC protocol in the plain model such that its combined communication and state complexity is independent of the function that it is computing. While a number of works have focused on optimizing communication complexity, we are not aware of any construction optimizing both the communication and state complexity.

We achieve it in two steps, starting with a protocol secure against semi-malicious adversaries. Semi-malicious security, introduced by Asharov et al, intuitively means that the adversary must follow the protocol, but can choose its random coins in an arbitrary way. The adversary is assumed to have a special witnesstape and is required to write a pair of input and randomness (x,r) that explains its behavior. We specifically start with a semi-malicious MPC protocol that has attractive communication and state complexity (i.e., independent of the function being computed). Then, we extend it so that the resulting construction is secure against not only semi-malicious, but also fully malicious adversaries.

5.4.1. Step 1: MPC with semi-malicious security

Our starting point is the solution proposed in the work of Brakerski et al. [15] based on multi-key fully homomorphic encryption (MFHE) that achieves semimalicious security 7. The construction is for deterministic functionalities where all the parties receive the same output,

---

[7] Their scheme is secure when exactly all but one parties are corrupted. To transform it into a scheme that is secure against any number of corruptions, Brakerski et al. suggest to extend it by a protocol proposed by Mukherjee and Wichs that relies on a so-called extended function. For simplicity, we skip this technical detail in our protocol. We note, however, that the additional communication and state complexity incurred due to the transformation depend only on the security parameter, as well as the parties' input and output sizes.

however it can be easily extended using standard techniques to randomized functionalities with individual outputs for different parties. For technical details behind the construction and the security proof we refer to Brakerski et al.

We note that while Brakerski et al. do not explicitly explain how to handle circuits of arbitrary depth, the bootstrapping approach outlined by Mukherjee and Wichs can be used here. Informally, the bootstrapping is done as follows: each party encrypts their secret key bit-by-bit using their public key and broadcasts the resulting ciphertext. These ciphertexts are used to evaluate the decryption circuit, thus reducing the noise. To do so, the parameters of the MFHE scheme must be set in a way that allows it to handle the evaluation of the decryption circuit. We assume circular security that ensures that it is secure to encrypt a secret key under its corresponding public key and refer to Mukherjee and Wichs for details.

To summarize, the construction in Protocol 4 is an MPC protocol secure against semi-malicious adversaries and can handle functions of arbitrary depth [8].

The communication complexity in Protocol 4 depends only on the security parameters, the number of parties, and input and output sizes. Note that for a party Pk the state that is passed between the rounds in Protocol 4 consists of the following data:

- params_k (passed from round one to round two and round three)
- params, $(\text{pk}_k, \text{sk}_k)$, $\{c_{k,j}\}_{j\in[l_{in}]}$, $\{\tilde{c}_{k,j}\}_{j\in[l_{key}]}$ (passed from round two to round three)
- $\{ev_{k,j}\}_{j\in l_{out}}$ (passed from round three to round four)

Note that this data depends only on security parameters, number of parties, and input and output sizes. Thus, the communication and state complexity of the semi-malicious protocol does not depend on the circuit we are computing.

5.4.2. Step 2: MPC with fully malicious security

In order to protect from fully malicious adversaries, we extend the construction above with the zero-knowledge protocol proposed by Kilian. In the following, we first elaborate on Kilian's protocol and some changes we need to make to it in order to keep the combined communication and state complexity low. Then, we elaborate on how Kilian's protocol is used in the overall MPC construction.

Kilian's zero-knowledge protocol Kilian's construction [36] relies on probabilistically checkable proofs (PCPs) and allows a party P to prove the correctness of some statement x using a witness w to the prover V . We specifically chose Kilian's construction because of its

[8] Again, this construction is secure against exactly $N-1$ corruptions (where N is the total number of parties). When used with the extended function transformation by Mukherjee and Wichs (which we skip here for readability purposes), the construction becomes secure against arbitrary many corruptions.

attractive communication and state complexities. Note that we make a minor change to Kilian's construction (Protocol 5) – instead of storing the PCP string that was computed in round two to use it in round four (as is done in the Kilian's original scheme), P recomputes the string (using the same randomness) in round four. Clearly, this changes nothing in terms of correctness and security. However, it allows us to drastically cut the state complexity of Kilian's original construction since the storage of the PCP becomes unnecessary.

Full construction The MPC construction secure against fully malicious adversaries is effectively the same as the semi-malicious one, except that additionally the parties commit to their input and randomness in the semi-malicious protocol and prove (using any zero-knowledge argument of knowledge, denoted by ZKAoK Protocol 4 Optimizing MPC

---

1. Let P_k be the party executing this protocol.
2. Run $\text{params}_k \leftarrow \text{MFHE.DistSetup}(1^\kappa, 1^N, k)$. Broadcast params_k.
3. Set $\text{params} = (\text{params}_1,...,\text{params}_N)$, and do the following:
   - Generate a key-pair $(pk_k,sk_k) \leftarrow \text{MFHE.Keygen}(\text{params},k)$
   - Let l_{in} denote the length of the party's input. Let $x_k[j]$ denote the j-th bit of P_k's input x_k. Let l_{key} denote the length of the party's secret key. – Encrypt the input bit-by-bit:

$$\{c_{k,j} \leftarrow \text{MFHE.Encrypt}(pk_k,x_k[j])_{j\in[lin]}$$

   - Encrypt the secret key bit-by-bit:

$$\{\tilde{c}_{k,j} \leftarrow \text{MFHE.Encrypt}(pk_k,sk_k[j])_{j\in[lkey]}$$

   - Broadcast the public key and the ciphertexts $(pk_k,\{c_{k,j}\}_{j\in[lin]},\{\tilde{c}_{k,j}\}_{j\in[lkey]})$
4. On receiving values $\{pk_i,c_{i,j}\}_{i\in[N]\setminus\{k\},j\in[lin]}$ execute the following steps:
   - Let f_j be the boolean function for j-th bit of the output of f. Let l_{out} denote the length of the output of f.
   - Run the evaluation algorithm to generate the evaluated ciphertext bit-by-bit:

$$\{c_j \leftarrow \text{MFHE.Eval}(\text{params},f_j,(c_{1,1},...,c_{N,lin}))\}_{j\in[lout]},$$

   while performing a bootstrapping (using the previously broadcasted encryptions of the secret keys) whenever needed. – Compute the partial decryption for all $j \in [l_{out}]$:

$$ev_{k,j} \leftarrow \text{MFHE.PartDec}(sk_k,c_j)$$

   - Broadcasts the values $\{ev_{k,j}\}_{j\in lout}$
5. On receiving all the values $\{ev_{i,j}\}_{i\in[N],j\in[lout]}$ run the final decryption to obtain the j-th output bit: $\{y_j \leftarrow \text{MFHE.FinDec}(ev_{1,j},...,ev_{N,j},c_j)\}_{j\in[lout]}$. Output $y = y1 ...yl_{out}$.

in the following) that they know the opening to the commitment. Kilian's construction is executed by each party Pk after each of the first three rounds of Protocol 4. In more detail:

We assume that there exists some ordering of parties participating in Protocol 4. Following the approach outlined by Asharov et al., in each round d of Protocol 4 we use Kilian's construction as follows:

For each pair of parties (P_i,P_j), Pi acts as a prover to the verifier Pj in order to prove the statement

$$\text{NextMessage}_d(x_i, r_i, [64]^d_{k=1}) = m_i^d, com(x_i || r_i, r_i') = c_i$$

Here, NextMessage is the function executed by Pi in this round according to Protocol 4, xi is the secret input of Pi, ri is the randomness used by Pi in the semi-malicious construction, $\{m_k\}_{k=1}^d$ are (concatenations of) the messages broadcast by all parties participating in Protocol 4 in rounds 1 to d, mdi is the

Protocol 5 Optimizing MPC - Kilian's construction

1. Verifier V chooses a collision-resistant hash function h and sends its description to the prover P.
2. Prover P uses the PCP prover P' to construct a PCP string $\psi \leftarrow P(x,w)$. Denote by r_p the randomness used by the prover in the generation of ψ. P computes the root of the Merkle tree (using the hash function h) on ψ, and sends the commitment to the Merkle tree root to the verifier V.
3. V chooses a randomness r_v and sends it to P.
4. P recomputes the PCP string $\psi \leftarrow P(x,w)$ using the randomness r_p and sends PCP answers to the set of queries generated according to the PCP verifier V' (executed on randomness r_v) to V.
5. V checks the validity of the answers, and accepts if all answers are valid and consistent with the previously received Merkle tree root. Otherwise, V outputs $\perp$.

message broadcast by Pi in round d, and ci is the commitment broadcast by P_i in the first round (com(x,r) denotes a perfectly binding, computationally hiding commitment to value x using randomness r). If a check fails, P_j broadcasts $\perp$ and aborts. These proofs are done sequentially (starting a new one only after the previous is fully finished), following the ordering of the (pairs of) parties. If at least one party has broadcasted $\perp$, all parties abort.

5.4.3 Properties of the resulting MPC construction

We now discuss the properties of the scheme constructed above. Specifically, we show the following:

Theorem 5. Let f be an N-party function. Protocol 6 is an MPC protocol computing f in the plain (authenticated broadcast) model which is secure against fully malicious adversaries corrupting up to t < N parties. Its communication and state complexity depend only on security parameters, number of parties, and input and output sizes. In particular, the complexity is independent of the function f.

Security We outline why this construction is secure. Intuitively, in order to prove security we construct the simulator S as follows: S commits to 0 for each honest party, and uses a zero-knowledge argument of knowledge simulator to prove that it knows the opening to the commitment. Then, S uses an extractor Ext of the argument of knowledge construction to retrieve the input and randomness $x_i,\ r_i,\ r_i'$ of each corrupted party Pi's valid proof. Then, in each round S uses the simulator Ssm of the semi-malicious scheme to retrieve the honest parties' messages, while forwarding messages broadcasted by any adversarial party Pi to

Ssm (aborting whenever NextMessage$_d(x_i, r_i, \{m^k\}_{k=1}^d, s_i^{d-1}) \neq m_i^d$ but the proof supplied by the adversary goes through, and writing witnesses (xi,ri) extracted by Ext on the witness tape of Pi otherwise). S uses the zero-knowledge simulator Szk of Kilian's protocol to simulate proofs on behalf of the honest parties. S Protocol 6 Optimizing MPC - handling fully malicious adversaries

---

1. Let P_z denote the party executing this protocol.
2. Let NextMessage$_d(\cdot)$ denote the next message function of Protocol 4.
3. Compute and broadcast$c_z = com(x_z||r_z, r_z').$
4. Sequentially, for each ordered pair of parties (P_i,P_j):
   (a) If $P_i = P_z$: Act as a prover in a ZKAoK to prove knowledge of$x_z||r_z,\ r_z'$ such that $c_z = com(x_z\|r_z, r_z')$.
   (b) If $P_j = P_z$: act as verifier in a ZKAoK to check knowledge of $x_i\|r_i,\ r_i'$ such that $c_i = com(x_i\|r_i, r_i')$. If this check fails, broadcast ⊥.
5. If any party party broadcast ⊥, abort.
6. For each round d = 1,...,3
   (a) Let$m^d = m_1^{d-1}, \ldots, m_n^{d-1}$.
   (b) Compute NextMessage$_d(x_z, r_z, [64]^d_{k=1}) = m^d_z$.
   (c) Broadcastm_z^d.
   (d) Sequentially, for each ordered pair of parties (P_i,P_j):
   i. If $P_i = P_z$, P_z acts as a Prover in Protocol 5 and uses the witness

$(x_z, r_z, c^{d_z^{-1}}, r_z')$ to prove that the following holds:

$$\text{NextMessage}_d(x_z, r_z, \{m^k\}_{k=1}^d) = m_z^d, com(x_z||r_z, r_z') = c_z$$

ii. If $P_j = P_z$, P_z acts as a Verifier in Protocol 5 to verify that there exist $(x_i, r_i, c^{d_i^{-1}}, r_i')$ such that the following holds:

$$\text{NextMessage}_d(x_i, r_i, \{m^k\}_{k=1}^d) = m_i^d, com(x_i||r_i, r_i') = c_i$$

If this verification check fails, broadcast ⊥ and abort. (e) If any party party broadcast ⊥, abort.

7. Output $\text{NextMessage}_4(x_z, r_z, [64]^4_{k=1}, c^3_z) = m^d_z$.

---

honestly checks the proofs submitted by the adversary, aborting (according to the protocol) whenever a proof is invalid.

Communication and State Complexity Analysis As we mentioned above, the communication complexity of Protocol 4 depends only on security parameters, number of parties, and input and output sizes. In particular, the communication and state complexity of the semi-malicious protocol does not depend on the circuit we are computing.

The communication complexity of Kilian's protocol depends on the security parameter as well as the length of the statement. In our case, the statement consists of the messages sent by the parties participating in the semi-malicious MPC protocol in the previous round as well as the message output by the party in the current round. Since the communication complexity of the semi-malicious MPC protocol is independent of the function being computed, the communication complexity of the overall construction is also independent of the function being computed. As for the state complexity, recall that we made a minor change to Kilian's original protocol – instead of storing the PCP, the prover simply recomputes (using the same randomness) it whenever it is needed. Due to this simple modification the PCP string does not contribute to the state complexity. The only other things contributing to the state complexity is the hash function h and the randomness rv, both independent of the function being computed by the MPC[9].

The combined communication and state complexity added due to the broadcasted commitments as well as ZKAoK proofs about these commitments also depends only on security parameters, number of parties, and input and output sizes.

[9] Additionally, they can be chosen by *V* independently of any messages from *P*, and thus they can be hardcoded in the garbled circuits and do not add to the state complexity of the non-interactive construction.

Thus, we have shown that the communication and state complexity of our construction in **Protocol 6** is independent of the function the MPC protocol is tasked with computing.

Integrating communication and state optimized MPC As we showed in previous section, the number of CSaR secret store operations in our non-interactive MPC construction (Protocols 1, 2 and 3) is upper bounded by:

nrounds n

Nstore = n ∗ (nrounds − 1) + X X2 ∗ niwires,j

i=2 j=1

The number of CSaR secret release operations for each evaluator is upper bounded by:

nrounds n

Nrelease = n ∗ (nrounds − 1) + X Xniwires,j

i=2 j=1

As we pointed out in previous section, the term $\sum_{i=2}^{n_{rounds}} \sum_{j=1}^{n} n_{wires,j}^{i}$ is precisely the combined communication and (encrypted) state complexity of the underlying MPC protocol π, minus the messages of the first round and plus signatures on the public messages and the state. Thus, when using Protocol 6 as the underlying protocol π in our main non-interactive MPC construction (Protocols 1, 2 and 3), we obtain a construction which number of CSaR store and release operations depends only on the number of rounds in π, security parameters, number of parties, and input and output sizes. All of these parameters are independent of the function that π is tasked with computing. Thus, we get the following result:

Corollary 3. There exists an MPC protocol π′ in the blockchain model that has adversarial threshold t < N, provides security with abort against fully-malicious adversaries and does not require participants to be online at the same time. Only a single message is required from the MPC contributors (the evaluators might be required to produce multiple messages). Furthermore, the number of calls to CSaR of this protocol is independent of the function that is being computed using this MPC protocol.

5.5. Guaranteed Output Delivery

In this section, we provide an extension of our main construction that ensures guaranteed output delivery, meaning that the corrupted parties cannot prevent honest parties from receiving their output.

In order to provide guaranteed output delivery, the first step is to build upon an MPC protocol π that also has this property. However, note that this change by itself is not sufficient – a malicious evaluator could still disrupt the execution of our original construction by simply

providing an authoritative message that contains an invalid signature and thus forcing honest garbled circuits to abort. It is clear that we cannot simply accept such invalid signatures. Thus, further modifications are required. In general, compared to our main protocol we make the following changes:

- The original MPC protocol must have the guaranteed output delivery property.
- We introduce a deadline by which all initial messages must be posted. In the following, we denote this deadline by τ.
- Signatures on the messages are verified not by the garbled circuits, but rather by the CSaR parties as part of the CSaR request. The signature is computed on the whole message, rather than separately for the public and state parts of the next-message function's output.
- We use CSaR with public release, which is similar to CSaR, but instead of privately releasing secret shares to the user, the parties release the shares publicly (e.g., by posting them on the bulletin board).
- As a part of the release condition, the garbled circuits and wire keys of the current round (that were previously published on the bullet board) are used to check whether the message submitted by the evaluator is indeed the output of the garbled circuit in question. Only if this is the case (i.e., the evaluator acted honestly) is the evaluator allowed to receive the next wire keys. The evaluator uses a proof of publishing of the garbled circuits and the wire keys released by the CSaR to prove the correctness of the computation. Roughly the following statement is checked: "The execution of the garbled circuit GC on the wire keys $\{k_i\}_{i\in I}$ results in the output provided by E. Here, the garbled circuit GC is the circuit, and $\{k_i\}_{i\in I}$ are the keys for this circuit reconstructed using the published values of the CSaR present on the proof of publish supplied by E".
- If a message from the first round was not published, or a garbled circuit or wire key from some party was not stored with CSaR, the evaluator needs to prove that with respect to the genesis block, by deadline τ indeed no such message was stored. We call such proof a "proof of missing message".
- In the cases described in the last two points, the CSaR releases default wire keys (encoding "$\perp$") for each garbled circuit that is supposed to use the missing message.

In order to allow for an easy verification of the evaluator's claims of invalid garbled circuits, we use CSaR with public release (CSaR-PR, see Figure 7), which is the same as CSaR, except that the witness is supplied by the client that wishes to receive the secrets publicly, and the secrets (garbled circuits and wire keys in our case) are released publicly as well (as long as the release condition is satisfied). Such CSaR-PR can be instantiated with the PublicWitness construction presented in the eWEB work. For simplicity, in the following we assume that the public release of the computation result is permitted. If the application requires that only a certain party obtains the function result, it can be easily supported by changing the output of the function that is being computed to the encryption of this output under that party's public key.

The definition of the authoritative message for this construction is a bit different from the definition in our main construction to account for the fact that the signatures and proofs of

execution are checked by the CSaR parties. Formally, the authoritative message of round d > 1 is a published message that satisfies the following conditions:

- Message is of the form (*id*,*d*,*m*), where *m* is of the form ($m_1^d \parallel \cdots \parallel m_n^d \parallel c_1^d \parallel \cdots \parallel c_n^d \parallel sig_1^d \parallel \cdots \parallel sig_n^d \parallel \mathcal{P}$), where P is some additional proof data, as explained below.
- each m^d_j, c^d_j, sig_j^d has correct length, and each sig_j^d is a valid signature of P_d on the tuple (id,d,j,m^d_j,c^d_j), and P contains a proof that for each contributor P_d the output of P_d's garbled circuit for that round is indeed what the evaluator claims this output to be[10]. The following exceptions are allowed:
  1. if a garbled circuit or wire key needed for the evaluation of that garbled circuit from some party P_j is missing and the corresponding message part could not be computed, the evaluator must prove that P_j failed to post the garbled circuit or wire key and the deadline τ has passed. Recall that in our main construction we require CSaR secret identifiers to be published on the bulletin board (in order for the evaluator to know what secrets it must request from the CSaR). If P_j failed to post the secret identifier, "proof of missing message" is used to prove that this message does not exist. If P_j posted this identifier, but the corresponding message is not stored with CSaR, CSaR publicly returned $\perp$ upon evaluator's request to retrieve this message and the proof of this publication is used to prove that the message was not stored. In both cases, wire keys for the default value $\perp$ are released by the CSaR participants as wire keys corresponding to the output of the missing circuit.
  2. If a m^d_j, c^d_j, or sig^d_j has incorrect length, or sig^d_j is not a valid signature of P_d on the tuple (id,d,j,m^d_j,c^d_j), but the evaluator proved that it is indeed the output of P_d's garbled circuit, this still counts as an authoritative message. In this case, wire keys for the default value $\perp$ are released by the CSaR participants as wire keys corresponding to m^d_j and c^d_j.
- The deadline τ has passed at the time of posting.
- This is the first published message that satisfies the requirements above.

Same as in our main construction, there are up to n authoritative messages for the first round – one for each contributing party. Formally, an authoritative message of round d = 1 from party Pk is a published message that satisfies the following conditions:

- Message is of the form (id,1,k,m^1_k,sig_k^1).
- sig_k^1 is a P_k's correct signature over m^1_k.

[10] The "proof" simply consists of the whole bulletin board. CSaR retrieves the garbled circuit of P_j and the corresponding wire keys that were published by CSaR on the bulletin board, executes the garbled circuit and checks whether the output is consistent with the message posted by the evaluator.

- m^1_k has correct length.
- The deadline τ has not passed at the time of posting.
- This is the first published message that satisfies the requirements above.

If a required authoritative first message from some party Pj is missing, the evaluator must prove that Pj failed to post this message and the deadline τ has passed ("proof of missing message"). In this case, wire keys for the default value ⊥ are released by the CSaR participants as wire keys corresponding to that message.

Finally, note that same as in our main construction, we require that the original protocol π has the publicly recoverable output property, now with the additional guarantee of output delivery. The publicly recoverable output property with guaranteed output delivery can be easily achieved as follows in a protocol which has guaranteed output delivery: first, all parties broadcast the output. Then, the value that was broadcasted by more than half of the parties is taken as the output. Note that if π has guaranteed output delivery, each honest participant in π is guaranteed to be able to correctly compute the honest output. Given honest majority among the participants (which we assume in order for π to provide the guaranteed output delivery anyway), the protocol outlined above results in a correct output. In the following, for simplicity we assume that protocol π has the publicly recoverable output property with guaranteed otput delivery and Eval denotes the algorithm used to retrieve the output from the transcript.

The full construction is given in Protocols 7 and 8 (preparation of the garbled circuits and keys), as well as Protocol 9 (execution phase). Just as in our main construction, we show security by providing a simulator that does not have access to the honest parties' secrets and showing that no PPT adversary is able to distinguish the interaction with the simulator from the interaction with the honest parties. However, this time we additionally prove that the guaranteed output delivery property holds for our construction.

Protocol 7

---

1. P_j computes the output (m^1_j, s^1_j) of the first round of the MPC protocol for F. P_j computes the signature sig_j^1 on the tuple $(id,1,j,m^1_j)$ using its signing key $sigk_j$. P_j posts $(id,1,j,m^1_j,sig_j^1)$ on the bulletin board.
2. P_j produces Yao garbled circuit $\{GC_j^i\}$ for each round $i > 1$ based on the circuit C_j^i of the next-message function f^i of the original MPC protocol π:

$$(\{\text{lab}^{w,b,i}_j\}_{w\in \text{inp}}\text{ij},b\in\{0,1\}) \leftarrow \text{Gen}(1^\lambda,\text{inp}^\text{i}_\text{j})$$

$$GC_j^i \leftarrow \text{Garble}^{(C^i_j,\,(\{\mathtt{lab}^{w,b,i}_j\}_{w\in\mathtt{inp}^\mathtt{i}_\mathtt{j},b\in\{0,1\}})}$$

Here, $\text{inp}^\text{i}_\text{j}$ is the length of the input to the circuit C_j^i. This circuit takes as input messages $\{m_k^{i-1}\}_{k=1}^n$ published by the parties in the previous round, and the encryption c_j^{i-1} of the secret state passed by P_j from the previous round. All of P_j's keys, input and randomness are hardcoded in the circuit.

The verification and public keys of other contributors are also hardcoded in the circuit. For the circuit of the second round, the secret state passed from the first round is hardcoded in the circuit as well. The circuit decrypts the secret state and executes the next message function of the current round:

(a) Compute $sij-1 = Decskj(cij-1)$.

(b) Obtain (m^i_j, s^i_j) by executing $\tilde{f}(x_j, r^i_j, m^i, s^{i-1}_j)$, where $m^i = m^{i-1}_1 \| \cdots \| m^{i-1}_n$.

(c) Compute the encryption of the state $c^i_j = Enc_{pkj}(s^i_j)$.

(d) Compute the signature sig^i_j on the tuple (id,i,j,m^i_j,c^i_j) using the signing key $sigk_j$.

(e) Output (m^i_j, c^i_j, sig^i_j).

3. P_j securely stores garbled circuits $\{GC^i_j\}$ for all rounds $i > 1$ using CSaR-PR. The witness needed to decrypt the ciphertext of some round i is a valid proof of publishing of all authoritative messages of round 1 and up to (and including) round $i - 1$. If τ was reached and some party did not post its authoritative message of the first round, the witness does not need to include a proof of publishing of the message computed by the garbled circuits of this party. Instead, the witness needs to include a proof of missing message by the deadline τ.

Research Corporation (SRC) program sponsored by DARPA. Vipul Goyal and Yifan Song were supported by the NSF award 1916939, DARPA SIEVE program, a Cylab Presidential Fellowship, a gift from Ripple, a DoE NETL award, a JP Morgan Faculty Fellowship, a PNC center for financial services innovation award, and a Cylab seed funding award.

Protocol 8 Non-Interactive MPC with GoD − *KeyPreparationPhase*

1. Securely store input wire keys ( ) for the circuit of the second round using CSaR-PR. For each party P_k whose first round message m^1_k is needed for the computation, the witness required to decrypt the wire key corresponding to the i-th bit of the input m^1_k being 0 (resp. 1) is a **valid proof of publishing** with respect to the genesis block of the following:

   (a) Each authoritative message of the first round is published. If a message is missing, the witness needs to include a proof of missing message by deadline τ instead of that message. For each missing message that is needed in the computation, wire keys for the default value $\perp$ are released. (b) i-th bit of m^1_k is 0 (resp. 1).

2. Securely store input wire keys $(\{\mathtt{lab}^{w,b,d}_j\}_{w \in \mathtt{inp}^d_j, b \in \{0,1\}})$ for the circuit of the d-th

   ($d \geq 3$) round using CSaR-PR. Say a message m^{d_j-1} (resp., c^{d_j-1}) is needed for the computation. The witness needed to decrypt the wire key corresponding to the i-th bit of m^{d_j-1} (resp., c^{d_j-1}) being 0 (resp. 1) is a **valid proof of publishing** with respect to the genesis block of the following:

   (a) All authoritative messages of round 1 up to and including round $d - 1$ are published (subject to the constraint that τ is reached and some party did not post its authoritative message of the first

round). Recall that an authoritative message is defined in a way that allows for missing or invalid partial messages (given a valid execution proof from the evaluator) – in those cases, for each missing message that is needed in the computation, wire keys for the default value $\perp$ are released.

(b) i-th bit of m^{d_j-1} (resp., c^{d_j-1}) is 0 (resp. 1).

Protocol 9 Non-Interactive MPC with GoD − *ExecutionPhase*

1. Wait until either deadline τ has passed.
2. The evaluator E uses messages (id,1,z,$m^1{}_z$,$sigpub^1{}_z$) posted on the bulletin board by each party P_z as the proof of publishing to get the garbled circuits (and keys) for the second round stored in CSaR by each participant in π. Then, E computes the outputs ($m^2{}_j$,$sigpub^2{}_j$,$c^2{}_j$,$sigpr_j{}^2$) of the second round by executing the garbled circuits. If for a party P_j any part of the information required to compute the output is missing, output $\perp$ is used in the following.
3. Check whether an authoritative message was published for round 2. If yes, check if this message is consistent with own output and if so, simply use its proof of publish as the witness to decrypt the wire keys of the next round. If the message is not consistent, abort. If the authoritative message is not published yet, publish $(id, 2, m_1^2 \| \cdots \| m_n^2 \| c_1^2 \| \cdots \| c_n^2 \| sig_1^2 \| \cdots \| sig_n^2)$ (appending the proof of execution, as well as proofs of of missing/invalid messages if necessary) and use the proof of publish as the witness.
4. In each following round $d \geq 3$, E executes each garbled circuit published by party P_z for round $d-1$. Then, E checks whether the authoritative message was published for that round and whether this message is consistent with own output and if so, simply uses its proof of publish as the witness to decrypt the wire keys of the next round. If the message is not consistent, E aborts. If the authoritative message is not published yet, E publishes the concatenated output of the garbled circuits along with the proof of execution. In any case, E uses the proof of publish of the authoritative message to release the wire keys and the garbled circuits of the next round.
5. Whenever any needed wire key and/or garbled circuit was missing, E additionally supplies a proof of missing message to decrypt the default wire keys of the next round.
6. Let τ' denote the resulting transcript of execution of π. E outputs Eval(τ') as the result.

CHAPTER SIX

Blockchain Protocol for Sensor Network

A critical and required property of the *Smart Private Ledger* is the ability to privately store and retrieve secret data with access control. This ensures that an entity is able to store classified data over the blockchain such that only a large set of nodes collaborating together can retrieve it. Suppose a 'dealer' would like to store classified data M on the blockchain. First, they generate a secret key K and encrypt M under K using a secret key encryption scheme (where the same key is used for encryption and decryption). This ciphertext is then publicly posted on the blockchain. An adversarial party is not able to access the classified data as the secret key (that only the dealer has access to) is required to decrypt the publicly available ciphertext.

The next step is to securely distribute the shares of secret key K to the nodes on the blockchain network in a manner such that only a pre-defined large set of nodes collaborating together can recover K. This ensures that even if a small number of nodes are compromised by an adversary the system remains secure and the adversary is not able to recover the secret key to decrypt the publicly available ciphertext. To securely generate and distribute the shares of the secret key to each miner on the network, the Shamir Secret Sharing scheme is used and described below.

Assume that there are N participants in the protocol, which in our case are the miners in the blockchain network. Given a secret S (e.g., a specific value of strain or temperature), which in our case is a 256-bit integer, the secret sharing protocol divides S into N shares, giving each participant its unique share. With each participant given their unique share, knowledge of a predefined number $K$of the Nshares is required to recover the secret S. This is denoted as a (K, N) threshold scheme.

Mathematically, we implemented a (K, N) threshold Shamir Secret Sharing scheme using polynomial interpolation. As stated before, let the secret be S. We then construct a random polynomial $f(x) = S + S_1 x + S_2 x^2 + \ldots + S_{K-1} x^{K-1}$, where the secret is the constant term. A share is defined as a tuple $(i, f(i))$ for some $i \in \mathbb{Z}$. Note that the degree of the polynomial is $K - 1$; a known result that requires $k + 1$ points to uniquely recover a k-degree polynomial and requires K out of the N shares to recover the secret S (see Figure 4). Furthermore, each participant is given a unique share $(i, f(i))$, which is done by assigning a unique i to each participant.

Given K shares, polynomial interpolation (Lagrange interpolation) is used to recover the constructed polynomial f. Let the K shares be $(x_1, y_1), \ldots, (x_K, y_K)$. Define the Lagrange basis functions f_j

$$f_j(x) = \prod_{m=1,\, m \neq j}^{m=K} y_j \frac{x - x_m}{x_j - x_m} \quad (1)$$

Then, the originally generated polynomial f is

$$f(x) = \sum_{j=1}^{K} f_j(x) \quad (2)$$

Now that we are able to uniquely interpolate the polynomial f with the K shares, the secret is recovered as the constant term in f. Note that the dealer uses the K threshold to generate the random polynomial, which is then used to create the secret shares.

The security of this scheme is dependent on the random generation of the sharing polynomial and that the polynomial is generated uniquely each time secret shares are created. Furthermore, the result that K shares uniquely generate a polynomial in a finite field ensures that if an adversary had access to fewer than K shares (assume $K - 1$ shares), all viable values of the secret are possible and equally likely to be the constant terms in the interpolated polynomial. This therefore provides the adversary with no additional information regarding the secret!

The dealer generates these shares using Shamir's Secret Sharing scheme and distributes them to corresponding miners by encrypting each miner's share with their corresponding public key before posting all the encrypted shares on the blockchain. Only the assigned miner is able to access their share, as their corresponding private key is required to decrypt their publicly posted share.

At a future stage, the dealer posts a release condition on the blockchain (signed with the dealer's corresponding digital signature), indicating the miners to release their secret shares. In this release condition, the dealer also specifies a public key of the person who requests for reconstruction (person who the dealer would like the private data to be available to). All miners post their secret shares encrypted under this specified public key. This protocol ensures that only the specified user is able to perform the reconstruction as they are able to decrypt the shares with their associated private key.

Note that in the reconstruction step, it is possible for some of the nodes to be actively corrupted and release incorrect shares. To solve this problem, each miner signs their released share with a unique digital signature (signed using each miner's private key and verified using their corresponding public key). Once these decrypted shares are publicly posted on the blockchain, each miner (handling a node of the SCADA system of the blockchain) can reconstruct the original polynomial randomly generated by the dealer in the Shamir Secret Sharing protocol, hence gaining access to the secret key. This secret key can then be used to decrypt the ciphertext (posted publicly on the blockchain), giving each miner access to the classified data M.

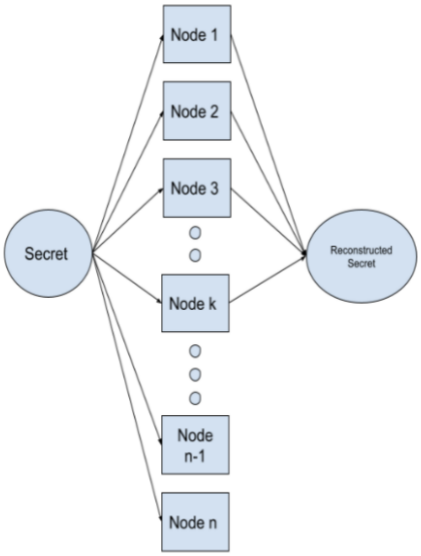


Figure 6.1: Image of transmission module connected with temperature sensor

6.1 Access Control

In addition to developing a *Smart Private Ledger* which facilitates secure data storage and retrieval, we also enable access control for sensor data in the smart ledger. For example, the main SCADA system may have a higher level of access than other nodes in the blockchain, which could be given lower access levels. To implement this, we define a given number of secrecy levels (say 1-5) which can be easily modified depending on the use case.

Then, depending on the access priority of each node in the network, assign a secrecy level to each of the miner public keys. Therefore, each of the secrecy levels now has a corresponding list of associated public keys. The dealer (e.g. who controls the SCADA system or is in charge of deciding when data should be released to nodes on the blockchain) is then able to assign a secrecy level to each piece of sensor data. If a user would like to request for secret key

reconstruction to decrypt data with secrecy level n (3 for example), the user would need to post a statement on the blockchain signed with their public key associated with secrecy level n. This ensures *attribute-based* access control, where the data owner is able to predefine a set of attributes that a user must have to access the data.

6.2 Security and Availability Guarantees of Protocol

We show security guarantees of our protocol if a node on the blockchain network is compromised. In order for an adversary to recover the secret key that the private data is encrypted with, they would have to potentially collude with a significant number of the miners (depending on the threshold set by the dealer in the Shamir Secret Sharing scheme when secret key shares are generated). A cyberattack requiring the collaboration of multiple (potentially a majority) miners is extremely difficult to launch in practice without going undetected before it takes place, as honest miners can report unusual behavior if occurring in a significant portion of the blockchain network.

We now show that our protocol guarantees availability of data, that all authorized users on the blockchain with appropriate access thresholds (defined in the previous section) can access the private data when needed if they meet the threshold of reconstruction set by the dealer. A common way an adversary may compromise availability is by launching a DoS (Denial of Service) attack on some node in the blockchain. However, due to the distributed, public nature of the blockchain network, unless the adversary launches such an attack over sufficiently many nodes in the network, availability is guaranteed for all active miners in the network.

Furthermore, even if an adversary obtains the keys of a miner and posts a reconstruction request on the blockchain network, as this request is publicly posted and accessible to everyone on the network, the compromised miner can then immediately post a message on the network to stop reconstruction before the private data is leaked to the adversary.

6.3 Implementation

A schematic of the Smart Private Ledger is shown in Figure 6.1 and 6.2. For the purpose of this project, we use Ethereum as the blockchain technology due to its capability of performing computations over stored data and smart contract deployment. Furthermore, in the implementation of the Smart Private Ledger, smart contracts are used for the dealer to post i) ciphertext (encrypted data file), ii) encrypted secret key shares and iii) release condition on the blockchain and for the miners to post i) decrypted shares (after the release condition is posted) and ii) reconstruction requests. This design using smart contracts ensures that the Smart Private Ledger is easily usable and deployable.

6.4 Simulation Results

We ran simulations of our Smart Private Ledger blockchain described in the previous sections in order to measure performance of each stage of the system. As the moving variables in

the system (which can be changed) are the data file sizes and number of miners on the network, it naturally follows to measure how system performance scales with these variables.

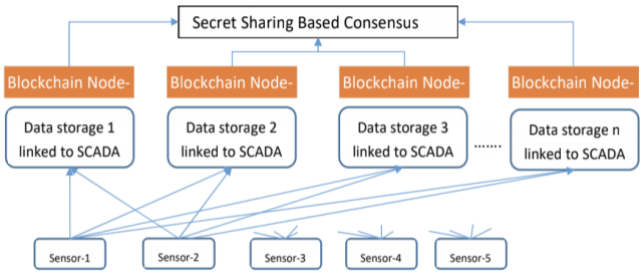


Figure 6.2: Blockchain network and data storage mechanism with access control

In order to determine system performance for increasing file sizes, we measured the time taken for the program to run (from secret generation to termination) on sensor data (stored in XLS files) of various file sizes (20, 40, 60, 80, 100 megabytes). The results for these simulations are shown in Figure 6.3. The file sizes were chosen to reflect large spreadsheet files which could reflect sensor data storage. In addition, we also measured how system performance changes as the number of miners (measured for n= 4, 8, 16, 32, 64 where k=n/2) is scaled (Figure 6.3). Note that we take half the total number of miners as the number of miners required for secret reconstruction in each measurement. The two primary components of the system which can be affected by the changing number of miners (secret generation and reconstruction) were measured separately to provide a more accurate representation of how changing the number of miners affects the relevant parts of our system.

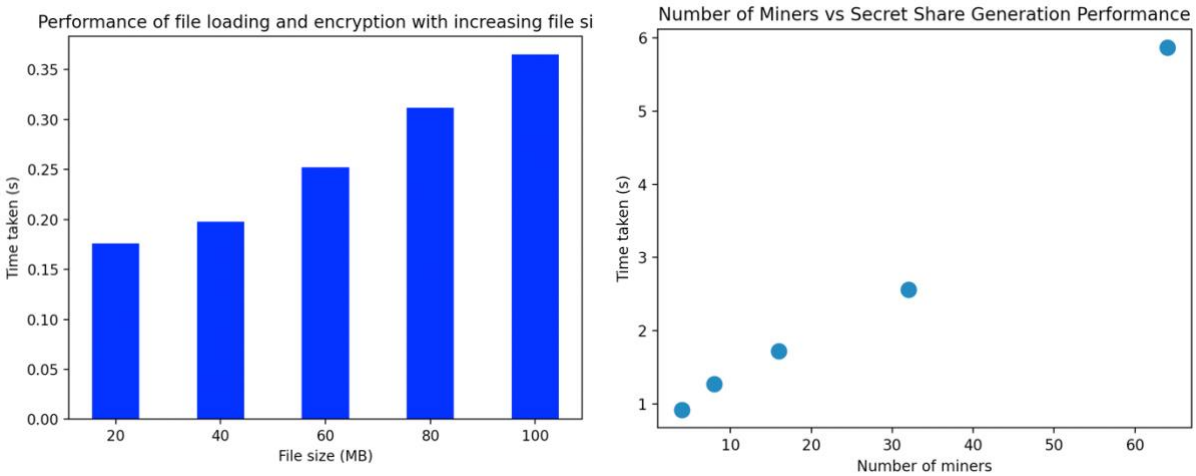


Figure 6.3: Measuring how system performance of file loading and file encryption steps scales with file size (left) and measuring how system performance of secret share generation and encryption scales with the number of miners in the network (right).

The data in Figures 6.3 left and right are obtained by running our simulations for 10 iterations and by taking an average of the time taken across the 10 iterations. It appears from

these results that run times scale linearly with number of miners and file sizes. However, external factors such as the specific machine programs are run on could impact exact run times. Furthermore, we note that the most time-consuming operations for the dealer are loading the file into the program and generating the secret shares, both of which take place once for a single program instance.

6.5 Simulated Cyberattacks

Our Smart Private Ledger system is also secure against common cyber-attacks such as the Ukraine power grid attack in 2015 and Colonial Pipeline cyberattack in 2021. The cyberattack method used in most cases is a form of SCADA Hijack where attackers gain access to the Industrial Control System (ICS) network through individual workstations. Attackers typically attempt to install malicious software on individual internet connected machines through phishing attempts.

However, with the Smart Private Ledger, it is much harder for any adversary to compromise the entire network through attacking an individual machine. For example, in the Ukraine power grid attack, spear-phishing emails were sent to individual machines to obtain personal information and credentials required to take control of the SCADA systems. In a generic cyberattack of this form, once personal credentials are obtained, compromising a small number of nodes allows the adversary to use malware such as BlackEnergy (used in the Ukraine power grid attack in 2015) to execute denial-of-service attacks and gain control of the entire system.

Our scheme significantly mitigates the possibility of such attacks as adversaries are unable to access privately stored information on the network by compromising a single node through techniques such as phishing attacks. This is due to the secret sharing mechanism in our scheme, the adversary would need to compromise a significant predetermined number (unknown to the adversary) of nodes in the network (each potentially running a different architecture and operating system) over the internet.

This process is unlikely to go undetected before preventative measures can be taken. Therefore, the Smart Private Ledger significantly reduces the possibility of a cyberattack due to a single point of compromise in the network, which is currently the most common technique used when hijacking SCADA systems.

CHAPTER SEVEN

Conclusions and Impact

A novel private blockchain protocol, *Smart Private Ledger*, with hierarchical access control is designed and implemented in this work which provides cybersecurity for machine-to-machine interactions, infrastructure for secure data logging for sensors, and decentralized data storage and retrieval. A lab-scale sensor network consisting of strain and temperature sensors which simulates encrypted information flows in a typical power plant is developed to test this protocol. A single transmitter receives data from multiple type of sensors and securely transmits it to a base station which acts as a blockchain node.

Our protocol addresses security concerns for cyberattacks on distributed sensor networks that are vulnerable to threats from a given machine in the network (i.e., similar to the recent cyberattack on the Colonial Pipeline). This is achieved by integrating the *Smart Private Ledger* into SCADA systems of the sensor networks that ensures strong security guarantees on sensor data using fundamental ideas from secret sharing and cryptographic digital signature schemes. The use of ordinary laptops and desktops as the nodes of the *Smart Private Ledger* leads to a blockchain network with minimal cost, lowering the barrier to access this important security technology for underserved areas/regions. This work thus establishes *Smart Private Ledger* as the next generation of blockchain technology for cybersecure sensor network compatible with existing SCADA systems and easily deployable for decentralized data storage and retrieval required for various applications. As a next step, we recommend implementation of this network in security-critical public infrastructure such as power plants and power grids.

The project also achieved significant success in terms of student training. The students received training in disparate areas of theoretical computer science and sensor networks. One of the students has joined Virginia Tech as a tenure-track assistant professor. Other students joined companies such as Google Inc., Simple Origin Inc., etc.

REFERENCES

1. Analytica, O., *US pipeline hack signals critical infrastructure risks.* Emerald Expert Briefings, 2021(oxan-es).
2. Case, D.U., *Analysis of the cyber attack on the Ukrainian power grid.* Electricity Information Sharing and Analysis Center (E-ISAC), 2016. **388**: p. 1-29.
3. Greenberg, A., *The untold story of NotPetya, the most devastating cyberattack in history.* Wired, August, 2018. **22**.
4. Liang, G., et al., *The 2015 ukraine blackout: Implications for false data injection attacks.* IEEE transactions on power systems, 2016. **32**(4): p. 3317-3318.
5. Buterin, V., *Privacy on the Blockchain.* Ethereum Blog, 2016.
6. Thomas, L., et al., *Automation of the supplier role in the GB power system using blockchain based smart contracts.* 2017.
7. Aitzhan, N.Z. 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, 2016. **15**(5): p. 840-852.
8. Mengelkamp, E., et al., *Designing microgrid energy markets: A case study: The Brooklyn Microgrid.* Applied energy, 2018. **210**: p. 870-880.
9. Horta, J., D. Kofman, and D. Menga, *Novel paradigms for advanced distribution grid energy management.* arXiv preprint arXiv:1712.05841, 2017.
10. Goranović, A., et al. *Blockchain applications in microgrids an overview of current projects and concepts*. in *IECON 2017-43rd Annual Conference of the IEEE Industrial Electronics Society*. 2017. IEEE.
11. Liu, J., et al., *How to build time-lock encryption.* Designs, Codes and Cryptography, 2018. **86**: p. 2549-2586.
12. Chiesa, A., et al. *Marlin: Preprocessing zkSNARKs with universal and updatable SRS*. in *Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part I 39*. 2020. Springer.
13. Goyal, R. and V. Goyal. *Overcoming cryptographic impossibility results using blockchains*. in *Theory of Cryptography: 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15, 2017, Proceedings, Part I 15*. 2017. Springer.
14. Garg, S., et al. *Witness encryption and its applications*. in *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*. 2013.
15. Applebaum, B., et al., *Encoding functions with constant online rate, or how to compress garbled circuit keys.* SIAM Journal on Computing, 2015. **44**(2): p. 433-466.
16. Garg, S., et al., *On the implausibility of differing-inputs obfuscation and extractable witness encryption with auxiliary input.* Algorithmica, 2017. **79**: p. 1353-1373.
17. Beuchat, J.-L., et al., *High-Speed Software Implementation of the Optimal Ate Pairing over Barreto-Naehrig Curves.* Pairing, 2010. **6487**: p. 21-39.

18. Schnorr, C.-P. *Efficient identification and signatures for smart cards*. in *Advances in Cryptology—CRYPTO'89 Proceedings 9*. 1990. Springer.
19. Yao, A.C. *Protocols for secure computations*. in *23rd annual symposium on foundations of computer science (sfcs 1982)*. 1982. IEEE.
20. Barak, B., et al. *On the (im) possibility of obfuscating programs*. in *Advances in Cryptology—CRYPTO 2001: 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19–23, 2001 Proceedings*. 2001. Springer.
21. Damgård, I. and J.B. Nielsen. *Scalable and unconditionally secure multiparty computation*. in *Advances in Cryptology-CRYPTO 2007: 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007. Proceedings 27*. 2007. Springer.
22. Maram, S.K.D., et al. *CHURP: dynamic-committee proactive secret sharing*. in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2019.
23. Ostrovsky, R. and M. Yung. *How to withstand mobile virus attacks*. in *Proceedings of the tenth annual ACM symposium on Principles of distributed computing*. 1991.
24. Baron, J., et al. *Communication-optimal proactive secret sharing for dynamic groups*. in *Applied Cryptography and Network Security: 13th International Conference, ACNS 2015, New York, NY, USA, June 2-5, 2015, Revised Selected Papers*. 2016. Springer.
25. Shamir, A., *How to share a secret. Commun. ACM.* 1979.
26. Kate, A., G.M. Zaverucha, and I. Goldberg. *Constant-size commitments to polynomials and their applications*. in *Advances in Cryptology-ASIACRYPT 2010: 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings 16*. 2010. Springer.
27. Choudhuri, A.R., et al. *Fairness in an unfair world: Fair multiparty computation from public bulletin boards*. in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2017.
28. Kokoris-Kogias, E., et al., *Verifiable management of private data under byzantine failures.* IACR ePrint, 2018. **209**: p. 2018.
29. Gilad, Y., et al. *Algorand: Scaling byzantine agreements for cryptocurrencies*. in *Proceedings of the 26th symposium on operating systems principles*. 2017.
30. Androulaki, E., et al. *Hyperledger fabric: a distributed operating system for permissioned blockchains*. in *Proceedings of the thirteenth EuroSys conference*. 2018.
31. Pass, R. and E. Shi. *Fruitchains: A fair blockchain*. in *Proceedings of the ACM symposium on principles of distributed computing*. 2017.
32. Abdolmaleki, B., S. Ramacher, and D. Slamanig. *Lift-and-shift: obtaining simulation extractable subversion and updatable SNARKs generically*. in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 2020.
33. Kosba, A., et al., *C $\emptyset $ C $\emptyset $: A Framework for Building Composable Zero-Knowledge Proofs.* Cryptology ePrint Archive, 2015.

34. Rivest, R.L., A. Shamir, and D.A. Wagner, *Time-lock puzzles and timed-release crypto.* 1996.
35. Galil, Z., S. Haber, and M. Yung. *Cryptographic computation: Secure fault-tolerant protocols and the public-key model*. in *Advances in Cryptology—CRYPTO'87: Proceedings 7*. 1988. Springer.
36. Cleve, R. *Limits on the security of coin flips when half the processors are faulty*. in *Proceedings of the eighteenth annual ACM symposium on Theory of computing*. 1986.
37. Blum, M. and S. Micali, *How to generate cryptographically strong sequences of pseudo random bits*, in *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*. 2019. p. 227-240.
38. Bellare, M., V.T. Hoang, and P. Rogaway. *Adaptively secure garbling with applications to one-time programs and secure outsourcing*. in *Advances in Cryptology–ASIACRYPT 2012: 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings 18*. 2012. Springer.
39. Durnoga, K., et al. *One-time programs with limited memory*. in *Information Security and Cryptology: 9th International Conference, Inscrypt 2013, Guangzhou, China, November 27-30, 2013, Revised Selected Papers*. 2014. Springer.
40. Faust, S., et al. *On the Non-malleability of the Fiat-Shamir Transform*. in *INDOCRYPT*. 2012. Springer.
41. Pedersen, T.P. *Non-interactive and information-theoretic secure verifiable secret sharing*. in *Advances in Cryptology—CRYPTO'91: Proceedings*. 2001. Springer.
42. Decker, C. and R. Wattenhofer. *Bitcoin transaction malleability and MtGox*. in *Computer Security-ESORICS 2014: 19th European Symposium on Research in Computer Security, Wroclaw, Poland, September 7-11, 2014. Proceedings, Part II 19*. 2014. Springer.
43. Mehar, M.I., et al., *Understanding a revolutionary and flawed grand experiment in blockchain: the DAO attack.* Journal of Cases on Information Technology (JCIT), 2019. **21**(1): p. 19-32.
44. Gentry, C. *Fully homomorphic encryption using ideal lattices*. in *Proceedings of the forty-first annual ACM symposium on Theory of computing*. 2009.
45. Boneh, D., et al. *Threshold cryptosystems from threshold fully homomorphic encryption*. in *Advances in Cryptology–CRYPTO 2018: 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2018, Proceedings, Part I 38*. 2018. Springer.
46. ElGamal, T., *A public key cryptosystem and a signature scheme based on discrete logarithms.* IEEE transactions on information theory, 1985. **31**(4): p. 469-472.
47. Shamir, A., *How to share a secret.* Communications of the ACM, 1979. **22**(11): p. 612-613.
48. Blum, M., P. Feldman, and S. Micali, *Non-interactive zero-knowledge and its applications*, in *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*. 2019. p. 329-349.

49. Doerner, J., et al. *Threshold ECDSA from ECDSA assumptions: The multiparty case*. in *2019 IEEE Symposium on Security and Privacy (SP)*. 2019. IEEE.
50. Gennaro, R. and S. Goldfeder. *Fast multiparty threshold ECDSA with fast trustless setup*. in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 2018.
51. Lindell, Y. and A. Nof. *Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody*. in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 2018.
52. Schnorr, C.-P., *Efficient signature generation by smart cards.* Journal of cryptology, 1991. **4**: p. 161-174.
53. Doerner, J., et al. *Secure two-party threshold ECDSA from ECDSA assumptions*. in *2018 IEEE Symposium on Security and Privacy (SP)*. 2018. IEEE.
54. Goldfeder, S., et al., *Securing Bitcoin wallets via a new DSA/ECDSA threshold signature scheme*, in *et al.* 2015.
55. Lindell, Y. *Fast secure two-party ECDSA signing*. in *Advances in Cryptology–CRYPTO 2017: 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20–24, 2017, Proceedings, Part II 37*. 2017. Springer.
56. Gennaro, R., S. Goldfeder, and A. Narayanan. *Threshold-optimal DSA/ECDSA signatures and an application to bitcoin wallet security*. in *Applied Cryptography and Network Security: 14th International Conference, ACNS 2016, Guildford, UK, June 19-22, 2016. Proceedings 14*. 2016. Springer.
57. Di Raimondo, M. and R. Gennaro. *Provably secure threshold password-authenticated key exchange*. in *Advances in Cryptology—EUROCRYPT 2003: International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4–8, 2003 Proceedings 22*. 2003. Springer.
58. MacKenzie, P., T. Shrimpton, and M. Jakobsson. *Threshold password-authenticated key exchange*. in *Advances in Cryptology—CRYPTO 2002: 22nd Annual International Cryptology Conference Santa Barbara, California, USA, August 18–22, 2002 Proceedings 22*. 2002. Springer.
59. Canetti, R. and S. Goldwasser. *An efficient threshold public key cryptosystem secure against adaptive chosen ciphertext attack*. in *Advances in Cryptology—EUROCRYPT'99: International Conference on the Theory and Application of Cryptographic Techniques Prague, Czech Republic, May 2–6, 1999 Proceedings 18*. 1999. Springer.
60. Barreto, P.S. and M. Naehrig. *Pairing-friendly elliptic curves of prime order*. in *Selected Areas in Cryptography: 12th International Workshop, SAC 2005, Kingston, ON, Canada, August 11-12, 2005, Revised Selected Papers 12*. 2006. Springer.
61. Srinivas, S., et al., *Universal 2nd factor (U2F) overview.* FIDO Alliance Proposed Standard, 2015. **15**.
62. Wood, G., *Ethereum: A secure decentralised generalised transaction ledger.* Ethereum project yellow paper, 2014. **151**(2014): p. 1-32.

63. Micali, S., O. Goldreich, and A. Wigderson. *How to play any mental game*. in *Proceedings of the Nineteenth ACM Symp. on Theory of Computing, STOC*. 1987. ACM New York, NY, USA.
64. Garg, S., et al. *The exact round complexity of secure computation*. in *Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35*. 2016. Springer.
65. Beaver, D., S. Micali, and P. Rogaway. *The round complexity of secure protocols*. in *Proceedings of the twenty-second annual ACM symposium on Theory of computing*. 1990.
66. Goyal, V. *Constant round non-malleable protocols using one way functions*. in *Proceedings of the forty-third annual ACM symposium on Theory of computing*. 2011.
67. Katz, J., R. Ostrovsky, and A. Smith. *Round efficiency of multi-party computation with a dishonest majority*. in *Advances in Cryptology—EUROCRYPT 2003: International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4–8, 2003 Proceedings 22*. 2003. Springer.
68. Pass, R. *Bounded-concurrent secure multi-party computation with a dishonest majority*. in *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*. 2004.
69. Badrinarayanan, S., et al. *Promise zero knowledge and its applications to round optimal MPC*. in *Advances in Cryptology–CRYPTO 2018: 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2018, Proceedings, Part II*. 2018. Springer.
70. Brakerski, Z., S. Halevi, and A. Polychroniadou. *Four round secure computation without setup*. in *Theory of Cryptography: 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15, 2017, Proceedings, Part I*. 2017. Springer.
71. Rai Choudhuri, A., et al. *Round optimal secure multiparty computation from minimal assumptions*. in *Theory of Cryptography: 18th International Conference, TCC 2020, Durham, NC, USA, November 16–19, 2020, Proceedings, Part II 18*. 2020. Springer.
72. McMahan, B. and D. Ramage, *Google AI blog: Federated learning: Collaborative machine learning without centralized training data.* URL https://ai. googleblog. com/2017/04/federated-learning-collaborative. html, 2017.
73. Goyal, V., et al. *Storing and retrieving secrets on a blockchain*. in *Public-Key Cryptography–PKC 2022: 25th IACR International Conference on Practice and Theory of Public-Key Cryptography, Virtual Event, March 8–11, 2022, Proceedings, Part I*. 2022. Springer.
74. Kaptchuk, G., I. Miers, and M. Green, *Giving state to the stateless: Augmenting trustworthy computation with ledgers.* Cryptology ePrint Archive, 2017.
75. Bellare, M., V.T. Hoang, and P. Rogaway. *Foundations of garbled circuits*. in *Proceedings of the 2012 ACM conference on Computer and communications security*. 2012.

76. Yao, A.C.-C. *How to generate and exchange secrets*. in *27th annual symposium on foundations of computer science (Sfcs 1986)*. 1986. IEEE.
77. Benhamouda, F., et al. *Can a public blockchain keep a secret?* in *Theory of Cryptography: 18th International Conference, TCC 2020, Durham, NC, USA, November 16–19, 2020, Proceedings, Part I 18*. 2020. Springer.
78. Feige, U., J. Killian, and M. Naor. *A minimal model for secure computation*. in *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*. 1994.
79. Halevi, S., et al. *Non-interactive multiparty computation without correlated randomness*. in *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part III*. 2017. Springer.
80. Halevi, S., Y. Lindell, and B. Pinkas. *Secure computation on the web: Computing without simultaneous interaction*. in *Advances in Cryptology–CRYPTO 2011: 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings 31*. 2011. Springer.
81. Beimel, A., et al. *Non-interactive secure multiparty computation*. in *Advances in Cryptology–CRYPTO 2014: 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II 34*. 2014. Springer.
82. Halevi, S., et al. *Secure multiparty computation with general interaction patterns*. in *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*. 2016.
83. Badrinarayanan, S., et al. *Non-interactive secure computation from one-way functions*. in *Advances in Cryptology–ASIACRYPT 2018: 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2–6, 2018, Proceedings, Part III*. 2018. Springer.
84. Canetti, R., A. Jain, and A. Scafuro. *Practical UC security with a global random oracle*. in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. 2014.
85. Chase, M., et al. *Reusable non-interactive secure computation*. in *Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part III 39*. 2019. Springer.
86. Afshar, A., et al. *Non-interactive secure computation based on cut-and-choose*. in *Advances in Cryptology–EUROCRYPT 2014: 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings 33*. 2014. Springer.
87. Badrinarayanan, S., et al. *Two-message witness indistinguishability and secure computation in the plain model from new assumptions*. in *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part III*. 2017. Springer.

88. Ishai, Y., et al. *Efficient non-interactive secure computatiosn*. in *Advances in Cryptology–EUROCRYPT 2011: 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings 30*. 2011. Springer.
89. Benhamouda, F. and H. Lin. *Mr NISC: multiparty reusable non-interactive secure computation*. in *Theory of Cryptography: 18th International Conference, TCC 2020, Durham, NC, USA, November 16–19, 2020, Proceedings, Part II 18*. 2020. Springer.
90. Choudhuri, A.R., et al. *Fluid MPC: secure multiparty computation with dynamic participants*. in *Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part II 41*. 2021. Springer.
91. Gentry, C., et al. *YOSO: You Only Speak Once: Secure MPC with Stateless Ephemeral Roles*. in *Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part II*. 2021. Springer.
92. Jain, A., N. Manohar, and A. Sahai. *Combiners for functional encryption, unconditionally*. in *Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part I 39*. 2020. Springer.
93. Lowengrub, J. and L. Truskinovsky, *Quasi–incompressible Cahn–Hilliard fluids and topological transitions.* Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences, 1998. **454**(1978): p. 2617-2654.