Secure Computation in Decentralized Systems

by

Guy Zyskind

B.Sc., Tel-Aviv University (2012)
S.M., Massachusetts Institute of Technology (2016)

Submitted to the Program in Media Arts and Sciences, School of Architecture and Planning,
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2024

Authored by:	Guy Zyskind Program in Media Arts and Sciences August 06, 2024
Certified by:	Alex "Sandy" Pentland Toshiba Professor of Media Arts and Sciences, Thesis Supervisor
Accepted by:	Joseph Paradiso Academic Head, Program in Media Arts and Sciences

Secure Computation in Decentralized Systems

by

Guy Zyskind

Submitted to the Program in Media Arts and Sciences, School of Architecture and Planning, on August 06, 2024 in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

ABSTRACT

Decentralized systems like Bitcoin and Ethereum are real-world examples of secure distributed systems deployed at scale. Over the past decade, these systems and others have proven to provide a trust-minimized solution for computing. They ensure the correct execution of code (*correctness*), maintain the *integrity* of stored data, and remain consistently available (*availability*). Additionally, they allow any user to interact without the risk of censorship.

However, while decentralized systems guarantee security properties like integrity, correctness, and availability, they do not provide *privacy*. In this regard, they are strictly worse than assuming full trust in a centralized server, since any node in the network must see all data. Furthermore, in many of these open systems (also known as 'permissionless' networks), there are no restrictions on who can operate a node. This means that decentralized systems, and public blockchains in particular, cannot operate on private data, greatly limiting the kinds of use-cases they can support.

This dissertation explores solutions to mitigate the privacy concerns associated with modern decentralized systems, focusing particularly on blockchains. The research employs Secure Multiparty Computation (MPC) techniques to address these issues, demonstrating how MPC, which already shares a similar distributed trust threat model, can enhance privacy in decentralized systems. More specifically, this thesis focuses on the following key areas in decentralized systems:

Access Control Mechanisms and Confidential Smart Contracts: The thesis begins by exploring access control mechanisms on blockchains, and from that builds up to the concept of *confidential smart contracts* – arbitrary programs that execute both *correctly* and *privately.*

Identity Management and Authentication: Building on access control and confidential smart contracts, we examine identity management and authentication within decentralized networks. We develop a highly efficient *Threshold ECDSA* protocol that runs in the server-aided MPC model.

Perhaps more importantly, we revisit the *server-aided MPC* model itself, which sits somewhere between the dishonest and honest-majority MPC paradigms, and show that a confidential smart contract is a real-world realization of the server in this model. We thus theorize that dishonest MPC protocols in general can be practically improved under this model, and argue that because there is a real-world counterpart, this model is realistic.

An Improved Distributed Point Function (DPF) and ORAM: A major theoretical contribution of this work is a novel three-party Distributed Point Function (DPF) construction. This leads to state-of-the-art Oblivious RAM (ORAM) and Distributed ORAM (DORAM) protocols, which are important building blocks in MPC.

Privacy-Preserving Digital Currencies: Using this DPF construction, we revisit the problem of privacy-preserving digital currencies, proposing a solution in the account model. This approach challenges the current consensus that privacy in blockchains requires a UTXO model.

Secure Inference with Private Retrieval: Lastly, the thesis explores how Large Language Models (LLMs) can perform secure inference while retrieving data from private, distributed databases. This method represents a step towards building secure decentralized AI systems that respect user privacy.

Thesis supervisor: Alex "Sandy" Pentland
Title: Toshiba Professor of Media Arts and Sciences

Secure Computation in Decentralized Systems

by

Guy Zyskind

Acknowledgments

Completing this dissertation would not have been possible without the support, guidance, and encouragement of many individuals to whom I am deeply grateful.

First and foremost, I would like to express my sincere gratitude to my advisor, Professor Alex "Sandy" Pentland. His unwavering support, insightful guidance, and invaluable feedback have been instrumental in shaping both my research and my development as a scholar. I am incredibly fortunate to have had the opportunity to learn from and work with such a visionary mentor. Without Sandy, my professional career would have undoubtedly looked very different. He encouraged me to take risks and explore this new and fascinating world of decentralized systems, and he saw the potential in them (and in me) before so many others did.

I would also like to extend my deepest gratitude to my other committee members. First, to Professor Srini Devadas: your belief in my ability to do sound research, especially during moments when my own confidence faltered, has been a cornerstone of my journey. You taught me that successful research is not just about the work itself but about finding the right niche and audience. Your guidance helped me find my path and instilled in me the confidence to pursue it. For that, I will always be deeply thankful.

To Dr. Thomas Hardjono, I am profoundly grateful for the lessons you imparted on the importance of building secure systems that go beyond theoretical research. You showed me that real impact comes from navigating the complex landscape of stakeholders and practical challenges. These lessons have guided me in academia and outside of it.

I am also immensely thankful to my colleagues, collaborators, and friends, who have provided inspiration, motivation, and camaraderie throughout this journey. To Erez, Abdullah, Avishay, Tobin, Rob, Itzik, Lior, Assaf, Yonatan, Tom, Sacha and many others – your support and shared enthusiasm have made this experience both rewarding and enjoyable.

I am truly grateful for my one-of-a-kind family. To my parents, Orna and Yossi, my older brother Nir and my twin sister Einat, your unwavering belief in my abilities and your constant encouragement have been my foundation.

Most importantly, to my beloved wife, Rinat, and the beautiful children we brought into this world, Maya, Alon, and Idan – I owe the deepest gratitude. Your love, patience, and understanding have been my greatest source of strength and happiness. All triumphs and failures pale in comparison to seeing you smile. Thank you for being the heart of my life.

This dissertation is a testament to the collective efforts and encouragement of all these wonderful people. I am truly grateful for your presence in my life.

Contents

Title page 1

Abstract 3

Acknowledgments 7

List of Figures 13

List of Tables 15

1 Introduction 17
- 1.1 Research Questions . . . 19

2 Common Preliminaries 21
- 2.1 Notation . . . 21
- 2.2 Secret Sharing . . . 21
  - 2.2.1 Schoenmakers's Publicly Verifiable *Random* Sharing Scheme . . . 22
- 2.3 Secure Multi-Party Computation (MPC) . . . 23
- 2.4 Encryption and Signature Schemes . . . 24
  - 2.4.1 El-Gamal Encryption Scheme . . . 24
  - 2.4.2 The Paillier Encryption Scheme . . . 24
  - 2.4.3 The ECDSA Scheme and Functionality . . . 24
- 2.5 Zero Knowledge Proof of Knowledge . . . 25
- 2.6 Distributed Point Functions (DPF) . . . 26
  - 2.6.1 Private Information Retrieval (PIR) . . . 27
  - 2.6.2 Private Information Writing (PIW) . . . 29
  - 2.6.3 Oblivious RAM (ORAM) . . . 29
- 2.7 Blockchains . . . 30
  - 2.7.1 Confidential Smart Contract Blockchains . . . 30

3 Decentralized Access-control and Confidential Smart Contracts 31
- 3.1 Decentralized Access-Control . . . 31
  - 3.1.1 Proposed Solution . . . 32
  - 3.1.2 The Network Protocol . . . 33
  - 3.1.3 Building Blocks . . . 33
  - 3.1.4 Blockchain Protocols . . . 35

3.1.5 Privacy and Security Analysis . . . 37
3.1.6 Confidential Smart Contracts: From Storage to Processing . . . 37
3.1.7 Trust and Decision-Making in Blockchains . . . 38
3.2 FHE-Rollups: An EVM-compatiable Confidential Smart Contracts Platform 39
3.2.1 Main contributions . . . 40
3.2.2 Design Goals and Security Model . . . 40
3.2.3 Related Work . . . 41
3.2.4 Layer-2 Rollups . . . 42
3.2.5 Fully Homomorphic Encryption (FHE) . . . 44
3.2.6 System Overview . . . 44
3.2.7 Threshold Services Network (TSN) . . . 45
3.2.8 Security . . . 46
3.2.9 Fraud Proofs . . . 46
3.2.10 Evaluation . . . 47

4 Unstoppable Wallets: Chain-assisted Threshold ECDSA and its Applications 51
4.1 Revisiting the Server-aided Model of MPC . . . 52
4.2 Main contributions . . . 55
4.2.1 Related Work . . . 55
4.3 Threshold ECDSA Protocol . . . 57
4.3.1 Key Generation . . . 58
4.3.2 Signing Protocol . . . 58
4.4 Robust Threshold ECDSA . . . 60
4.5 A Solution for a Single User . . . 62
4.6 Security Proofs . . . 63
4.6.1 Proof of Theorem 4.3.1 . . . 63
4.6.2 Proof of Theorem 4.4.1 . . . 65
4.6.3 Proof of Theorem 4.5.1 . . . 69
4.7 Applications . . . 71
4.7.1 Multisignature Wallet with Policy Checks . . . 71
4.7.2 Wallet Exchange . . . 72
4.8 Implementation and Evaluation . . . 72
4.8.1 Implementation Details . . . 73
4.8.2 Cryptographic Primitives on Chain . . . 73
4.8.3 Performance Evaluation . . . 74
4.8.4 Scalability across n and t . . . 83

5 High-throughput Three-Party DPF and its application to ORAM and Digital Currencies 85
5.0.1 Main Contributions . . . 87
5.0.2 Related Work . . . 88
5.1 MPC Functionalities . . . 90
5.1.1 Overview . . . 90
5.1.2 Access Policy ORAM Functionality . . . 90

5.1.3 Sum-of-Products Functionalities 90
5.1.4 Implementation of Known Functionalities 90
5.2 $(2,3)$-Verifiable DPF 93
5.2.1 Building Block: $(2,2)$-VDPF$^+$ 94
5.2.2 Our $(2,3)$-VDPF Construction 95
5.2.3 Security analysis 96
5.2.4 (2,3)-VDPF with sublinear PRF calls 97
5.3 ORAM with Policy 98
5.3.1 Additional Functionalities 98
5.4 Account-based Digital Currency 99
5.4.1 Registration & Access Control for DPFs 99
5.4.2 The CBDC Protocol 100
5.5 Efficient, Malicious Dot-Product 103
5.5.1 Updatable $(2,3)$-VDPF 104
5.5.2 The Dot-Product Protocol 105
5.6 Implementation, Evaluation and Applications 105
5.6.1 DPFs 105
5.6.2 Account-based Privacy-preserving Cryptocurrency and CBDCs 106
5.6.3 Three-party Distributed ORAM (DORAM) 106

6 Private Retrieval Augmented Generation (PRAG) 119
6.1 Overview 120
6.2 Exact Private Nearest Neighbors 121
6.2.1 Distance calculations 122
6.2.2 Fast secure dot product 122
6.2.3 Relation to private information retrieval 123
6.2.4 Exact top-k for retrieval 123
6.3 Approximate Nearest Neighbors and Inverted Files (IVF) 123
6.3.1 Efficient approximate vector nearest neighbor search in MPC 124
6.3.2 Sublinear Communication Complexity 126
6.3.3 Sacrificing Privacy for Speed in MPC IVF 126
6.4 Implementation and Evaluation 127
6.4.1 Exact Search 127
6.4.2 Approximate Search 127

7 Discussion, Conclusions and Future work 131
7.1 Putting it All Together 132

References 135

List of Figures

2.1 The $(2, 2)$-DPF of [20] . . . 28

3.1 Overview of the decentralized platform. . . . 33
3.2 Example of a flow of secure computation in a blockchain network. The top left block (EVote procedure) is the unsecure code, where the arguments marked in (*) are private and stored as shares on the DHT. The network selects a subset of nodes at random to compute a secure version of EVote and broadcasts the results back to the entire network, that stores it on the ledger. . . . 38
3.3 Overview of an FHE Rollup architecture . . . 49
3.4 Smart contract request to decrypt FHE-encrypted data on the rollup . . . 50
3.5 Fraud proof engine for FHE instructions . . . 50
3.6 Excerpt of a Voting Contract. . . . 50

4.1 Communication Model Illustrated . . . 54
4.2 Multisignature Wallet with Policy Checks . . . 81
4.3 Wallet Exchange Application Flow . . . 82
4.4 Gas Used vs. Number of Users (n) . . . 83
4.5 Gas Used vs. Threshold (t) for 15 users . . . 83

5.1 Functionality $\mathcal{F}_{\mathsf{APORAM}}$ . . . 91
5.2 Functionality $\mathcal{F}.\mathsf{Product}$ . . . 91
5.3 Functionality $\mathcal{F}_{\mathsf{SoP}}$ . . . 92
5.4 Protocol for $(2, 2)$-VDPF$^+$ . . . 94
5.6 Protocol for $(2, 3)$-VDPF . . . 98
5.5 Our $(2, 3)$-VDPF construction . . . 109
5.7 Functionality $\mathcal{F}_{\mathsf{AP-ORAM}}$ . . . 110
5.8 Functionality $\mathcal{F}.\mathsf{Product}$ . . . 110
5.9 Functionality $\mathcal{F}_{\mathsf{SoP}}$ . . . 111
5.10 Protocol $\Pi_{\mathsf{CBDC}}.\mathsf{register}$. . . . 111
5.11 Protocol $\Pi_{\mathsf{CBDC}}.\mathsf{read}$. . . . 112
5.12 Protocol $\Pi_{\mathsf{CBDC}}.\mathsf{update}$. . . . 113
5.13 Functionality $\mathcal{F}^{+}_{(2,2)-\mathsf{UVDPF}}$ . . . 113
5.14 Functionality $\mathcal{F}^{+}_{(2,3)-\mathsf{UVDPF}}$ . . . 114
5.15 Protocol $\Pi_{(2,3)-\mathsf{UVDPF}^+}$ . . . 115
5.16 Protocol Π_{Product} . . . 116

5.17 Full domain evaluation of the various DPF constructions, compared to the baseline (2,2) DPF. 116
5.18 Transaction throughput and memory usage. Side-by-side comparison with Solidus. 117
5.19 Latency in processing a request under different channel latency for $N = 2^{20}$. Requests can be parallelized so this is indicative, and bandwidth changes had no effect thanks to the constant communication complexity. 117

6.1 Overview of PRAG architecture using a distributed, secret-shared inverted file index (IVF), for retrieving document token vectors closely matching a privately-generated query vector in LLM-based question answering. 121
6.2 Time taken to retrieve top-k closest vectors in the database for end-to-end MPC exact search across increasing synthetic database sizes. 128
6.3 Information retrieval using IVF improves accuracy with increased n_{probe} (top left) but increases query time as a larger proportion of the index ($\frac{n_{probe}}{n_c}$) must be searched (bottom left). These retrieval approaches (both IVF and exact) scale favorably across multiple servers (right). 129

7.1 An illustrative decentralized medical system combining elements from this thesis to solve privacy 134

List of Tables

3.1 FHE addition times using different VMs for 8-256bit plaintext integers. . . . 47
3.2 FHE multiplication times using different VMs for 8-256bit plaintext integers. 47
3.3 Benchmarking different Solidity Contracts using FHE 48

4.1 Comparison with related work. For protocols that support pre-signing – the number of writes consists of both pre-sign and sign phase, ignoring amortization. 57
4.2 Benchmarks for Multiparty and Two-party ECDSA 74
4.3 Gas cost baselines 75

5.1 Comparison of three-party ORAM protocols using DPFs. Key Size captures client-server communication, Reads and Updates are non-interactive (except for [127], [128]) and focus on overall computation costs. Results are for semi-honest, but we remark that we and [149] provide malicious implementations. 88
5.2 Comparing key sizes (in bytes) of our constructions compared to the baseline (2,2) DPF of [117] and (2,3) DPF of [148]. Since [148] does not provide an implementation, we provide an analytical estimate. 106

Chapter 1

Introduction

The digital era has brought about significant technological advancements, transforming our methods of interaction, transaction, and information exchange. In the last couple of decades in particular, we have seen the rise of decentralized technologies, mainly (but not only) in blockchains, which promise to minimize trust, enhance privacy, and bolster security in digital interactions. Some examples of these systems, which have gained significant traction, include blockchains like Bitcoin [1] and Ethereum [2], as well as storage networks such as IPFS[1]. Thousands of other networks have since emerged and are continuing to gain adoption. Most recently, decentralized AI systems are gaining popularity as well, with networks such as Render[2] allowing anyone to provide compute resources to others.

The main benefits these decentralized systems provide relate to data integrity and the correctness of computations. Specifically, blockchains ensure through consensus protocols, that all parties follow a strict set of programmed rules, and that honest nodes always maintain the correct state of the system[3]. Another way to look at blockchains, from a Secure Multi-Party Computation (MPC) lense, is that a blockchain provides a strong notion of *correctness*, which also guarantees output delivery.

However, while these systems are *correct*, they provide no *privacy*. Thus, solving privacy in decentralized systems, and blockchains more specifically, is the core problem this thesis seeks to solve. Most of the techniques used and developed in this work rely on MPC. This should come at no surprise, since MPC shares the same *distributed trust* mind-set and threat model that decentralized networks do.

This work is motivated by the recognition that privacy must be foundational in the development of decentralized technologies, rather than an afterthought. It must be an integral component, woven into their very architecture, to foster trust, compliance with regulatory standards, and user acceptance. Drawing from my own decade of academic and practical experience in privacy-centric blockchain systems, I aim to address the following privacy and security aspects of decentralized systems in this thesis: (i) user authentication; (ii) access-control; (iii) secure (and private) computation. To maintain a reasonable scope, for each of

[1]https://ipfs.tech

[2]https://render.com

[3]In the case of permissionless blockchains, incentives are put in place to ensure all (rational) actors follow the protocols, and also serve as a sybil resistance mechanism. In permissioned blockchains, the operators' identity is known, so the focus remains only on consensus.

these aspects, I develop cryptographic constructions and applications that demonstrate their applicability. The work itself is split into four main parts.

Firstly, the thesis explores *access control* mechanisms on blockchains, demonstrating how blockchains can effectively implement specific access policies in a trust-minimized manner. This analysis is based on my foundational work in the area [3], which, at the time of this writing, has garnered more than 3,000 citations. This nearly decade-old work also introduced the concept of *confidential smart contracts*. Though not originally termed as such, it has since emerged as a vibrant field of study (e.g. [4]–[11]).

Confidential smart contracts enable the creation of arbitrary programs that operate in a decentralized setting and are designed (under specific security conditions) to maintain both the integrity of the computation and the confidentiality of the inputs. Access control seamlessly integrates with confidential smart contracts because, without the latter, access control is merely about determining which data is accessible to each party. However, with the inclusion of confidential smart contracts, we can also selectively allow parties to gain insights from data without accessing the raw data itself. For an instance, we can facilitate a private election where approved parties can cast their votes, and while the individual votes remain private, the overall tally is disclosed to all. Given their importance, and to provide a more up-to-date perspective of my seminal work, I also explore in the same chapter a new construction for a confidential smart contracts system based on Fully Homomorphic Encryption (FHE).

Building on this, this thesis explores identity management and authentication within decentralized networks. Interestingly, the protocols developed for authentication and key-management leverage the very infrastructure of confidential smart contracts, illustrating a symbiotic relationship between access control and identity verification. More concretely, I explore how to improve the state-of-the-art in terms of custodying private keys, which is one of the largest open problems in decentralized systems[4], by developing a novel Threshold ECDSA protocol.

The Threshold ECDSA protocol we develop operates in the *server-aided MPC* model, which sits somewhere between the dishonest and honest-majority MPC paradigms. One of our insights is that a confidential smart contract is a real-world realization of the server, and we demonstrate this empirically as well through an implementation. We further theorize that many other dishonest MPC protocols can be practically improved under the server-aided model, and argue that because there is a real-world counterpart, this model is realistic.

The third part contains the most significant theoretical contribution of this work – a novel three-party Distributed Point Function (DPF) construction, from which we demonstrate state-of-the-art Oblivious RAM (ORAM) and Distributed ORAM (DORAM) protocols. Both of these are well-studied core problems in the world of MPC: An ORAM hides which items are accessed in a database, where a DORAM is an extension that allows running MPC over RAM programs (as opposed to circuits), which scales-up data-dependent computations (e.g., sorting algorithms).

Expanding on this DPF-based ORAM construction, I define a new functionality called an

[4]To illustrate, in a 2017 study, Chainalysis, the largest data-analysis company in the cryptocurrency space, estimated that 17-23% of all Bitcoins are lost due to poor key-management practices https://www.chainalysis.com/blog/money-supply

ORAM-with-Policy, connecting us back to access control concepts of decentralized systems. From this, I show how to construct a privacy-preserving digital currency system in the account-based model. The central motivation here is to revisit the longstanding problem of private digital currencies, which are only efficient in the Unspent Transaction Output (UTXO) blockchain model (for example, see the work of [12]). My goal is to challenge the current general consensus in academia and industry that privacy in blockchains requires a UTXO-model, which has some significant drawbacks compared to the much simpler account-based model popularized by the likes of Ethereum. To do so, I show an instantiation of a privacy-preserving digital currency system (based on the developed DPF and ORAM-with-policy constructions) that operates in the account model. This construction is simple and uses MPC end-to-end, which can therefore be easily extended to support arbitrary *confidential smart contracts*, and not just simple transactions. The major limitation of this work is a constraint of requiring exactly three-parties to operate the servers, which makes it more suitable for the use case of a Central-Bank Digital Currency (CBDC) as opposed to a permissionless blockchain. However, it is my hope that this rekindles interest in privacy-preserving account-based cryptocurrencies, and opens up a route for future work.

Lastly, and somewhat separate from the rest of the work presented in this thesis, I explore how Large Language Models (LLMs) can do secure inference even when retrieving data from private, secret-shared, databases. Such a method could be the first step in building a secure decentralized AI system that respects user privacy.

1.1 Research Questions

This thesis answers the following research questions:

- **Primary Question.** How can we leverage secure computation to mitigate the privacy concerns associated with decentralized systems, such as blockchains and decentralized AI?
- Is it possible to construct a privacy-preserving digital currency in the account model?
- Is there a secure key management solution for decentralized identity authentication that balances security, robustness, and user-friendliness?
- Is the server-aided MPC model realistic?
- Could LLMs protect user-queries *while* securely searching private databases?

In addition to the above, and as part of the road to answer these questions, we also affirmatively answer the following theoretical question regarding DPFs, which was open before this work:

- Can we build a three-party DPF, secure against a single corruption, which has the same asymptotic key-size and evaluation complexity as two-party DPFs?

Chapter 2

Common Preliminaries

2.1 Notation

We use κ as a computational security parameter. For $x, y \in \{0,1\}^*$ the expression $x||y$ is the concatenation of x and y. Uniformly sampling a random value x from a set X is denoted by $x \leftarrow X$ or $x \overset{\$}{\leftarrow} X$. The result of a probabilistic algorithm A on inputs $x_1, x_2, \ldots$ is written by $x \leftarrow A(x_1, x_2, \ldots)$; in addition, when we want to explicitly mention the randomness used in the algorithm we write $x = A(x_1, x_2, \ldots \ ; \ r)$. Vectors are formatted in bold, e.g., $\boldsymbol{x}$ or in uppercase X, and the i-th index is denoted by $x^{(i)}$. The identity vector $\boldsymbol{e}_i$ has zero in all coordinates except in its i-th coordinate, which equals one.

To represent secret-shared values, we will normally use the common 'in-the-box' notation $[x]$. If the context requires it, we will often describe a specific party's share by a subscript, in which case party i's share will either appear as $[x]_i$ or x_i.

2.2 Secret Sharing

Secret sharing enables a dealer to split a secret x into n pieces or *shares*, such that only a sufficiently large subset of shares can be used to recover the secret. Shamir t-out-of-n secret sharing over the field F (where $t < n \in N$) is defined by a tuple of algorithms $\mathsf{SS}_F = (\mathsf{Share}, \mathsf{Reconstruct})$, where $[x] = ([x]_1, \ldots, [x]_n) = \mathsf{Share}_{t,n}(x; r)$ denotes a sharing of x, and $x = \mathsf{Reconstruct}([x]_{i_1}, \ldots, [x]_{i_{t+1}})$ denotes the reconstruction using $t+1$ shares, which may result with $\perp$ if the shares are inconsistent.

- $[x] = \mathsf{Share}_{t,n}(x; r)$. Given a secret $x \in F$ and a random tape r, pick $a_1, \ldots, a_t \in F$ and output $[x] = \{[x]_1, \ldots, [x]_n\}$, where $[x]_i = P(i)$ and $P(x) = x + a_1 x + a_2 x^2 + \ldots + a_t x^t$.
- $x = \mathsf{Reconstruct}([x]_{i_1}, \ldots, [x]_{i_{t+1}})$. Given $t+1$ shares $[x]_{i_1}, \ldots, [x]_{i_{t+1}}$, where $1 \le i_1 < i_2 < \ldots < i_{t+1} \le n$, interpolate a polynomial P such that $P(i_j) = [x]_{i_j}$ for all $j \in [1, t+1]$ and output $x = P(0)$.

Lagrange interpolation is used in order to get $P(0)$ directly. In our protocol we use Lagrange interpolation to get $P(i)$ also for $i \neq 0$, therefore, we describe below the general case.

Given $t+1$ points $(i_1,[x]_{i_1}),\ldots,(i_{t+1},[x]_{i_{t+1}})$, the polynomial that passes through them is $L(x)=\sum_{j=1}^{t+1}[x]_{i_j}\cdot\ell_j(x)$, where

$$\ell_j(x)=\prod_{\substack{1\le k\le t+1\\ k\ne j}}\frac{x-i_k}{i_j-i_k}.$$

Now, for some value v, we define the coefficient $\lambda_j^v=\ell_j(v)$, then, we have $L(v)=\sum_{j=1}^{t+1}\lambda_j^v\cdot[x]_{i_j}$.

In a typical use-case a dealer calls $[x]=\mathsf{Share}_{t,n}(x;r)$ on its secret x, and send $[x]_i$ to the i-th receiver. In a later point, the receivers want to reconstruct x, so they gather $t+1$ of the shares and run $x=\mathsf{Reconstruct}([x]_{i_1},\ldots,[x]_{i_{t+1}})$. It is a fact that Shamir secret sharing has perfect secrecy, namely, t shares reveal nothing about the secret, whereas $t+1$ shares completely determine it. Shamir secret sharing is not protected from a malicious dealer, that is, the dealer may use a polynomial P of degree higher than t, which may lead to inconsistent reconstruction - different subset of shares reconstruct to different secrets. In addition, Shamir secret sharing is not protected from a malicious receiver, that is, a receiver may contribute a wrong share to make reconstruction output a wrong secret (not the one dealt by the dealer). Verifiable secret sharing schemes solve those issues.

2.2.1 Schoenmakers's Publicly Verifiable *Random* Sharing Scheme

Verifiable secret sharing (VSS) enables a receiver to (1) check in the dealing phase that the share received from the dealer is consistent with a fully determined secret, and (2) check in the reconstruction phase that the shares published by other receivers are correct. Publicly VSS (PVSS) is a more powerful tool that enables a receiver to check consistency not only of its own share, but also all receivers' shares; furthermore, it enables an external party (who is not even a receiver) to check that conditions (1) and (2) hold. While information theoretic schemes for PVSS schemes have been proposed [13], we use Schoenmakers's scheme that is based on the hardness of discrete logarithm, as it is the minimal assumption in our context anyway. Specifically, we use the *special* PVSS version in [14], in which *the secret is random*, which allows using a simpler protocol.[1] Thus, in the following we assume that x is uniformly random from Z_q.

Schoenmakers's PVSS [14] over the group $(\mathbb{G},G,q)$ is parameterized with the receivers' encryption keys, namely, the i-th receiver is associated with El-Gamal key-pair $(\mathsf{ek}_i,\mathsf{dk}_i)$ (see definition in Section 2.4.1). While the scheme supports any encryption scheme, the El-Gamal scheme leads to a very simple implementation and efficient proof. The dealer invokes the zero-knowledge functionality (see definition in Section 2.5) with the relation

$$R_{\mathsf{PVSS},n,t}=\Big\{\big(\{\mathsf{ek}_i,c_i\}_{i=1}^n,\{A_j\}_{j=0}^t\big),\big(\{r_i\}_{i=1}^n,\{a_j\}_{j=0}^t\big)\ \text{s.t.}$$
$$\forall_{i=1}^n: c_i=\mathsf{EG.Enc}\left(\sum_{j=0}^t i^j\cdot a_j, r_i\right)\wedge\forall_{j=1}^t: A_j=a_j\cdot G\Big\}.$$

[1]When the secret x is random it is possible for the dealer to publish $x\cdot G$, whereas in case x is not random the dealer has to publish a Pedersen commitment $x\cdot G+r\cdot H$ where H is another generator of $\mathbb{G}$ for which $\log_G H$ is unknown.

That is, the claim is that c_i is an encryption of $P(i) = \sum_{j=0}^{t} i^j \cdot a_j$ under the i-th public key ek_i, where $P's$ coefficients are $\log_G(A_0), \ldots, \log_G(A_t)$. Note that given A_j's anyone can compute $Q_i = P(i) \cdot G$ by $\sum_{j=0}^{t} i^j \cdot A_j$, thus, interpretting $c_i = (C_{i,1}, C_{i,2})$, this statement is reduced to the statement that $(Q_i, X, C_{i,2} \cdot (C_{i,1})^{-1})$ is a Diffie-Helman tuple, for every i. Indeed, the discrete logs are $x, P(i)$ and $x \cdot P(i)$, respectively. There exists standard NIZK for that statement.

Then, the Schoenmakers's scheme is defined by the tuple of algorithms $\mathsf{PVSS}_{(G,G,q),\{\mathsf{ek}_i\}_i} = (\mathsf{Share}, \mathsf{Reconstruct}, \mathsf{CheckDealer}, \mathsf{CheckShare})$:

- $(\{c_i\}_{i=1}^n, \{A_j\}_{j=0}^t, \pi) \leftarrow \mathsf{Share}_{t,n}(x)$. Set $a_0 = x$ and pick $a_1, \ldots, a_t \in F$ and compute $[x] = \{[x]_1, \ldots, [x]_n\}$, where $[x]_i = P(i)$ and $P(x) = \sum_{j=0}^{j} a_j \cdot x^j$. Then, pick $r_i \leftarrow Z_q^*$, compute $c_i = \mathsf{EG.Enc}_{\mathsf{ek}_i}([x]_i, r_i)$ for every $i \in [1, n]$, and compute $A_j = a_j \cdot G$ for every $j \in [0, t]$. Then, send $(\mathsf{prove}, \mathsf{sid}, \{\mathsf{ek}_i, c_i, r_i\}_i, \{A_j, a_j\})$ to $\mathcal{F}_{\mathsf{zk}}^{R_{\mathsf{PVSS}}}$ to obtain $\pi = (\mathsf{proof}, \mathsf{sid}, \{\mathsf{ek}_i, c_i\}_i, \{A_j\})$ and output $(\{c_i\}_{i=1}^n, \{A_j\}_{j=0}^t, \pi)$.

- $x = \mathsf{Reconstruct}([x]_{i_1}, \ldots, [x]_{i_{t+1}})$. Given $t+1$ shares $[x]_{i_1}, \ldots, [x]_{i_{t+1}}$, where $1 \leq i_1 < i_2 < \ldots < i_{t+1} \leq n$, for which $\mathsf{CheckShare}(\{A_j\}, [x]_{i_k}) = 1$ for all $k \in [1, t+1]$, interpolate a polynomial P such that $P(i_k) = [x]_{i_k}$ for all $k \in [1, t+1]$ and output $x = P(0)$.

- $b \leftarrow \mathsf{CheckDealer}(\{c_i\}_{i=1}^n, \{A_j\}_{j=0}^t \pi)$. Output $b = 1$ iff $\pi = (\mathsf{proof}, \mathsf{sid}, \{\mathsf{ek}_i, c_i\}_i, \{A_j\})$, and $b = 0$ otherwise.

- $b \leftarrow \mathsf{CheckShare}(\{A_j\}_{j=0}^t, [x]_k)$. For $k \in Z_q^*$, output $b = 1$ iff $[x]_k \cdot G = \sum_{j=0}^{t} k^j \cdot A_j$, and $b = 0$ otherwise.

The scheme is a secure publicly verifiable secret sharing if the DDH problem is hard relative to (G, G, q).

2.3 Secure Multi-Party Computation (MPC)

Secure Multiparty Computation (MPC) is a subfield of cryptography aimed at allowing multiple parties to jointly compute a function over their inputs while keeping those inputs private. The outcome of the computation reveals only the specific intended result, and nothing else about the participants' individual inputs is disclosed.

In this thesis, we focus on the variant based on secret-sharing, although other variants based on Garbled Circuits (e.g., [15]) or GMW [16] exist. Our protocols tend to focus on a static active adversary, although some protocols focus on a more niche adversarial model which we introduce ad-hoc in the relevant chapters.

We prove security using the commonly used simulation paradigm [17], in which we show by simulation that our real world protocol and adversary executions can be simulated by a simulator in the ideal world.

2.4 Encryption and Signature Schemes

2.4.1 El-Gamal Encryption Scheme

The El-Gamal encryption scheme [18] over group $(\mathbb{G}, G, q)$ is defined by $\mathsf{EG} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$:

- $(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{Gen}()$. Pick $x \leftarrow Z_q^*$ and output (Y, x) where $Y = x \cdot G$ (i.e., $pk = Y$ and $\mathsf{dk} = x$).
- $C = \mathsf{Enc}_{\mathsf{ek}}(m, r)$. For a uniformly random $r \in Z_q$ and arbitrary $m \in Z_q^*$, output $C = (C_1, C_2) = (r \cdot G, (r \cdot Y) \cdot m)$.
- $m = \mathsf{Dec}_{\mathsf{dk}}(C)$. For $C_1, C_2 \in \mathbb{G}$, interpret $c = (C_1, C_2)$ and $x = \mathsf{dk}$, and output $C_2 \cdot (x \cdot C_1)^{-1}$.

The scheme is proven to be CPA-secure under the assumption that the decisional Diffie-Helman is hard relative to $(\mathbb{G}, G, q)$.

2.4.2 The Paillier Encryption Scheme

The Paillier encryption scheme [19] is defined by the tuple of algorithms $\mathsf{Paillier} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ described below.

- $\mathsf{Gen}(1^\kappa, q)$. Given a security parameter 1^κ and a prime q, sample $\mathsf{poly}(\kappa)$-bit primes p_1 and p_2 and output $(N; (p_1, p_2))$ where $N = p_1 \cdot p_2$ is the public encryption key and $sk = (p_1, p_2)$ is the secret key. Define $\mathcal{P} = (Z_q, +)$, $\mathcal{R} = (Z_N^*, \cdot)$ and $\mathcal{C} = Z_{N^2}^*$.
- $\mathsf{Enc}(pk, x; \eta)$. Given the public key N, a message $x \in Z_q$ and randomness $\eta \in Z_N^*$, output
$$\mathsf{ct} = \left[(1+N)^x \cdot \eta^N \mod N^2\right].$$
- $\mathsf{Dec}(sk, \mathsf{ct})$. Given the secret key (p_1, p_2) and a ciphertext ct, compute $N = p_1 \cdot p_2$ and output
$$\mathsf{pt} = \left[\frac{[\mathsf{ct}^{\phi(N)} \mod N^2] - 1}{N} \cdot \phi(N)^{-1} \mod N\right] \mod q.$$

2.4.3 The ECDSA Scheme and Functionality

The ECDSA scheme is defined by the following algorithms (the group $\mathbb{G}, G, q$ is an implicit parameter in the algorithms):

- $\mathsf{Gen}()$. Choose $x \leftarrow Z_q^*$ and compute $X = x \cdot G$. Output x as the signing key and X as the verification key.
- $\mathsf{Sign}(x, M)$. For a message $M \in \{0,1\}^*$, choose $k \leftarrow Z_q^*$ and compute $r = (k \cdot G).x \mod q$ and $s = k^{-1}(m + rx) \mod q$, where $m = H_q(M)$ and $H_q : \{0,1\}^* \to Z_q$ is modeled as a random oracle. Output the signature (r, s).

- $\mathsf{Verify}(X, M, (r, s))$. For a message $M \in \{0,1\}^*$, compute $m = H_q(M)$ and output 1 iff $(ms^{-1} \cdot G + rs^{-1} \cdot X).x \mod q = r$, otherwise output 0.

Indeed, if (r, s) is computed correctly on M, then $ms^{-1} \cdot G + rs^{-1} \cdot X = ms^{-1} \cdot G + rxs^{-1} \cdot G = (m + rx)s^{-1} \cdot G = (m + rx)\left(k^{-1}(m + rx)\right)^{-1} \cdot G = (m + rx)k(m + rx)^{-1} \cdot G = k \cdot G = R$ and so, projection to the x coordinate results with $R.x = r$.

The (Threshold) ECDSA functionality (Functionality 1) supports two interfaces, the key-generation interface is called once, followed by many, calls to the sign interface. Since we are interested in both robust and non-robust version of this functionality, the robust version should have the gray text omitted, as in that functionality the adversary does not get to decide on whether to forward outputs to the parties or not.

FUNCTIONALITY 1. (*The ECDSA Functionality: $\mathcal{F}_{ECDSA}$*)
The functionality is parameterized with the ECDSA group description $(\mathbb{G}, G, q)$ as well as a threshold parameter t, with $1 \leq t < n$. The functionality works with parties $P_1, \ldots, P_n$, P_c, and an adversary $\mathcal{S}$ as follows.

- Upon receiving (keygen) from all parties:
  1. Generate an ECDSA key-pair (X, x) by choosing a random $x \leftarrow Z_q^*$ and computing $X = x \cdot G$.
  2. Choose a hash function $H_q : \{0,1\} \rightarrow \{0,1\}^{\lfloor \log q \rfloor}$.
  3. If received $(\mathsf{keygen}, \mathsf{abort})$ from $\mathcal{S}$ then output $\perp$ and halt; otherwise, if received $(\mathsf{keygen}, \mathsf{continue})$ then continue.
  4. Store (H_q, x), output X to all parties, and ignore future calls to keygen.

- Upon receiving $(\mathsf{sign}, \mathsf{sid}, M)$ from P_c and $t + 1$ parties out of $\{P_1, \ldots, P_n\}$, if keygen was already called and sid was not already used:
  1. Choose a random $k \in Z_q^*$, compute $R \leftarrow k \cdot G$ and let $r = R.x \mod q$; then send R to all parties.
  2. Let $m = H_q(M)$. Compute $s \leftarrow k^{-1}(m + rx) \mod q$.
  3. If received $(\mathsf{sign}, \mathsf{sid}, \mathsf{abort})$ from $\mathcal{S}$ then output $\perp$ and halt; otherwise, if received $(\mathsf{sign}, \mathsf{sid}, \mathsf{continue})$ then continue.
  4. Send (r, s) to all parties.

Similarly, the two-party variant of the functionality is presented in Functionality 2.

2.5 Zero Knowledge Proof of Knowledge

For an NP-relation R, we use the $\mathcal{F}_{\mathsf{zk}}^R$ functionality (Functionality 3 below). The protocols we use to realize $\mathcal{F}_{\mathsf{zk}}^R$ are public coin, therefore they can be instantiated with a non-interactive

FUNCTIONALITY 2. (*2P ECDSA Functionality:* $\mathcal{F}_{ECDSA}$)
The functionality is parameterized with the ECDSA group description (G, G, q) and works with parties P_u, P_c, and an adversary $\mathcal{S}$ as follows.

- Upon receiving (keygen) from P_u:
  1. Generate an ECDSA key-pair (X, x) by choosing a random $x \leftarrow Z_q^*$ and computing $X = x \cdot G$.
  2. Choose a hash function $H_q : \{0,1\} \to \{0,1\}^{\lfloor \log q \rfloor}$.
     (a) Store (H_q, x).
     (b) Output X to P_u and P_c.
     (c) Ignore future calls to keygen.

- Upon receiving $(\mathsf{sign}, \mathsf{sid}, M)$ from P_u, if keygen was already called and sid was not already used:
  1. Choose a random $k \in Z_q^*$
  2. Compute $R \leftarrow k \cdot G$ and let $r = R.x \mod q$; then send R to P_u and P_c.
  3. Let $m = H_q(M)$. Compute $s \leftarrow k^{-1}(m + rx) \mod q$.
  4. Send (r, s) to P_u and $\mathcal{S}$.

version in the random oracle model via the Fiat-Shamir transform.

FUNCTIONALITY 3. (*The ZKPoK Functionality:* $\mathcal{F}_{\mathsf{zk}}^{R}$)
The functionality works with a prover $\mathcal{P}$ and verifiers $\vec{\mathcal{V}}$.

- Upon receiving $(\mathsf{prove}, \mathsf{sid}, x, w)$ from $\mathcal{P}$, if $(x, w) \in R$ and sid has never been used before, send $(\mathsf{proof}, \mathsf{sid}, x)$ to $\vec{\mathcal{V}}$.

2.6 Distributed Point Functions (DPF)

We start off by presenting a definition for DPFs.

Definition 2.6.1. *Let $\alpha \in \{1, \ldots, n\}$ and $\beta \in F$, a $(2,2)$-Distributed Point Function (DPF), denoted $F_{\alpha,\beta}^{(2,2)}$ is defined by algorithms:*

- $(f_0, f_1) \leftarrow \mathsf{DPF.Gen}(1^\kappa, \alpha, \beta)$.
- $y_b = \mathsf{DPF.Eval}(b, f_b, x)$, *where* $b \in \{0,1\}$.

Correctness. It must hold that $\mathsf{DPF.Eval}(0, f_0, \alpha) + \mathsf{Eval}(1, f_1, \alpha) = \beta$ *and* $\mathsf{DPF.Eval}(0, f_0, x) + \mathsf{DPF.Eval}(1, f_1, x) = 0$ *for all* $x \neq \alpha$.

Privacy. For every $b \in \{0,1\}$ there exists a simulator $\mathcal{S}$ such that

$$f_b \overset{c}{\equiv} \mathcal{S}(1^\kappa, b, n)$$

where $(f_0, f_1) \leftarrow \mathsf{DPF.Gen}(1^\kappa, \alpha, \beta)$ and the distribution is over the coin tosses of algorithms $\mathsf{DPF.Gen}$ and $\mathcal{S}$.

Evaluation can also be defined on a vector $\boldsymbol{x} = (x_1, \ldots, x_n)$, by:

$$\begin{aligned}\boldsymbol{y}_b &= \mathsf{DPF.Eval}(b, f_b, \boldsymbol{x})\\ &= (\mathsf{DPF.Eval}(b, f_b, x_1), \ldots, \mathsf{DPF.Eval}(b, f_b, x_n)).\end{aligned}$$

By the correctness of the DPF, it holds that $\boldsymbol{y} = \boldsymbol{y}_0 + \boldsymbol{y}_1 = \boldsymbol{e}_\alpha \cdot \beta$. When $\boldsymbol{x}$ covers the entire domain of α this is called a *full domain evaluation*. The computational complexity of a full domain evaluation is better than an individual evaluation on each $x_i \in \boldsymbol{x}$ in isolation.

In Figure 2.1 we give the $(2,2)$-DPF by Boyle et al. [20], which is the most efficient one - it has keys (f_0 and f_1) of length $O(\log n)$, where n is the size of the domain and a full domain evaluation incurs $O(\kappa \cdot n)$ computation. We note that the scheme in [20] describes the DPF over a general group $\mathcal{G}$ whereas in the constructions presented in this thesis we use either the binary field F_{2^l} where $l = \kappa$ or a prime field F_q defined with the prime $q \approx 2^\kappa$. For simplicity we denote the prime field by F, leaving the prime parameter implicit. Addition of $x, y \in F_{2^l}$ is computed by $x \oplus y$ and addition of $x, y \in F$ is computed by $x + y \mod q$.

We continue with a definition of *verifiability*, which enables the key holders to verify the validity. Formally:

Definition 2.6.2. *A verifiable DPF (VDPF) is a DPF (as per Definition 2.6.1) with the following additional procedures:*

- $\pi_b = \mathsf{VDPF.Prove}(b, f_b)$, *where* $b \in \{0,1\}$.
- $\{\texttt{accept,reject}\} \leftarrow \mathsf{VDPF.Verify}(\pi_0, \pi_1)$

Correctness is the same as in Definition 2.6.1; Privacy is enhanced to include the proof generated by the other key, that is, there exists a simulator $\mathcal{S}$ such that

$$(f_b, \pi_{1-b}) \overset{c}{\equiv} \mathcal{S}(1^\kappa, b, n)$$

where $(f_0, f_1) \leftarrow \mathsf{VDPF.Gen}(1^\kappa, \alpha, \beta)$ and $\pi_b = \mathsf{VDPF.Prove}(b, f_b)$, where $b \in \{0,1\}$, and the distribution is over the coin tosses of algorithms $\mathsf{VDPF.Gen}$ and $\mathcal{S}$.

Verifiability. This ensures that the two keys are well formed*, that is: Let $\boldsymbol{y}_b = \mathsf{VDPF.Eval}(b, f_b, \boldsymbol{x})$, then $\texttt{accept} = \mathsf{VDPF.Verify}(\pi_0, \pi_1)$ if and only if $\boldsymbol{y}_0 + \boldsymbol{y}_1 = \boldsymbol{e}_{\alpha'} \cdot \beta'$ for some α', β'.*

2.6.1 Private Information Retrieval (PIR)

Private Information Retrieval (PIR) refers to privately reading an element i at a database (a vector) D, without the server learning i (the access pattern). There are single-server (e.g., **simplepir**) and multi-server PIR schemes [21]. In this work we are focused on the

```
**Parameters:** Let $G : \{0,1\}^\kappa \to \{0,1\}^{2\kappa+2}$ be a PRG.
DPF.Gen$(1^\kappa, \alpha, \beta)$

1. Sample $s_0^0 \leftarrow \{0,1\}^\kappa$ and $s_1^0 \leftarrow \{0,1\}^\kappa$. Set $t_0^0 = 0$ and $t_1^0 = 1$.
2. Let $\alpha_1, \ldots, \alpha_n$ be the bits of $\alpha$.
3. For $i$ from 1 to $n$ do:
    (a) $(s_b^L, t_b^L, s_b^R, t_b^R) \leftarrow G(s_b^{(i-1)})$ for $b \in \{0,1\}$.
    (b) $\alpha_i = 0\text{Diff} \leftarrow L; \text{Same} \leftarrow R\text{Diff} \leftarrow R; \text{Same} \leftarrow L$
    (c) $s_{cw} \leftarrow s_0^{Same} \oplus s_1^{Same}$
    (d) $t_{cw}^L \leftarrow t_0^L \oplus t_1^L \oplus \alpha_i \oplus 1$
    (e) $t_{cw}^R \leftarrow t_0^R \oplus t_1^R \oplus \alpha_i$
    (f) $cw_i \leftarrow s_{cw}||t_{cw}^L||t_{cw}^R$
    (g) $s_b^{(i)} \leftarrow s_b^{Diff} \oplus t_b^{(i-1)} \cdot s_{cw}$ for $b \in \{0,1\}$.
    (h) $t_b^{(i)} \leftarrow t_b^{Diff} \oplus t_b^{(i-1)} \cdot t_{cw}^{Diff}$ for $b \in \{0,1\}$.
4. Compute output correction word: $ocw \leftarrow \beta \oplus s_0^{(n)} \oplus s_1^{(n)}$.
5. Set $f_b \leftarrow (s_b^0, \{cw_i\}_{i=1}^n, ocw)$ for $b \in \{0,1\}$.
6. Output: $(f_0, f_1)$.

DPF.Eval$(b, f_b, x)$

1. Parse the DPF key $(s_0, \{cw_j\}_{j=1}^n, ocw) \leftarrow f_b$.
2. Let $s \leftarrow s_0$ and $t \leftarrow b$.
3. Let $x'_1 = (x_i), \ldots, x'_n = (x_i)$ be the bits of $x_i$.
4. For $j$ from 1 to $n$ do:
    (a) Parse: $(s_{cw}, t_{cw}^L, t_{cw}^R) \leftarrow cw_j$
    (b) $\tau \leftarrow G(s) \oplus (t \cdot [s_{cw}||t_{cw}^L||s_{cw}||t_{cw}^R])$
    (c) Parse: $(s^L, t^L, s^R, t^R) \leftarrow \tau$
    (d) $x'_j = 0(s,t) \leftarrow (s^L, t^L)(s,t) \leftarrow (s^R, t^R)$
5. $t = 0y \leftarrow sy \leftarrow s \oplus ocw$
6. Output: $y$
```

Figure 2.1: The $(2,2)$-DPF of [20]

latter. DPFs have gained a lot of popularity due to their efficiency in performing PIR in the multi-server setting.

A well-established two-party PIR protocol assumes that a public DB D is replicated across two servers. A client may then privately read the α-th database entry by sharing a point function $f_{\alpha,1}$, and sending the shares f_0 and f_1 to the PIR servers. Then, the i-th server computes $d_i = \sum_x f_i(x) \cdot D[x]$ and hands d_i back to the client. Finally, the client can reconstruct the result by computing $d = d_0 + d_1$. Note that evaluating $f(x)$ over the entire domain $|D|$ creates a shared one-hot-vector, so d correctly encodes $D[\alpha]$.

Note that this procedure does not work when the database itself is secret shared, as I describe in more detail below.

2.6.2 Private Information Writing (PIW)

A related problem involves Private Information Writing (PIW). In this case, we wish to privately write a value β at $D[\alpha]$, such that the server(s) do not learn either α or β. Clearly, in the two-server setting, this only makes sense when the database is secret shared, i.e., there are two databases D_0 and D_1 s.t. $D[x] = D_0[x] + D_1[1]$. Similar to PIR, a DPF can be used in order to privately *update* an entry in that database. Specifically, a client sends the shares f_0 and f_1 for some point function $f_{\alpha,\beta}$, and then the i-th server adds $f_i(x)$ to the value at $D[x]$, for all x in the domain. This way, the client can privately add β to the α-th entry of the shared database.

2.6.3 Oblivious RAM (ORAM)

Observe that in the PIR setting, the database is replicated (and therefore public), and in the PIW setting, the database has to be secret-shared. Therefore, supporting both private information retrieval and writing operations, is more challenging. If we assume that the database is secret-shared, then now the PIR protocol will require a dot product between two vectors, which requires linear communication in the size of the database (as opposed to being non-interactive). Supporting both PIR and PIW together in a single database are crucial, as they form the basis for Oblivious RAM (ORAM).

ORAM, introduced by Goldreich and Ostrovsky [22] and followed by many works ([23]–[26] to name a few), is a cryptographic protocol between a client and a server. Such a protocol enables the client to upload its database to a server and access specific records in that database while being protected from the server. Beside the usual confidentiality guarantees (which are typically enabled by encryption schemes) an ORAM guarantees the user that nothing about the *access pattern* is being leaked to the server; that is, the server learns nothing about the entries being accessed or the actions being applied to those entries (either read or update). As shown time and again (e.g., [27]), such access pattern leakage may be devastating for a system that strives to preserve the user's privacy, and may cause even to breach confidentiality. Over the years, there were protocols that proposed to distribute the server's role into multiple (distrustful) entities in order to improve efficiency and to support more than one client (where sections of the database may be accessed by many clients). Such solutions are known as *multi-client multi-server ORAM*.

2.7 Blockchains

While there are several ways to define a Blockchain, the most apt definition for our needs is as a decentralized system that simulates a trusted third party, which can run arbitrary programs (known as *smart contracts*), while ensuring both *correctness* and *availability*. It also records everything in a persistent ledger, and maintains its *integrity*. In other words, a blockchain serves as a trusted, always online, general-purpose computing resource which clients can interact with. While many blockchains exist today, two of the primary examples are Bitcoin [1], which serves mostly as a single-purpose blockchain for facilitating financial transactions between parties, and Ethereum [2], which can execute, in a trusted-manner, arbitrary programs (i.e., smart contracts).

2.7.1 Confidential Smart Contract Blockchains

As mentioned, smart contracts are a fundamental component of many blockchain platforms, enabling users to automate the execution of agreements and facilitate trustless interactions between parties. These self-executing, deterministic programs provide correctness and availability by ensuring that the code executes exactly as programmed without downtime, censorship, fraud, or third-party interference. However, traditional smart contracts do not inherently provide privacy, as their logic and data are visible to all network participants.

To address this issue, starting from their introduction in the work presented in the first part of Chapter 3, published almost a decade ago, confidential smart contract blockchains were developed (e.g., [4], [5], [9]–[11]).

This means that these blockchains inherently hide sensitive input data fed into contracts, persistent state data, and depending on the use case, hide the output as well, even from the nodes operating the chain. Confidential smart contract-enabled blockchains may employ different privacy-preserving techniques, such as secret-sharing MPC (e.g., [4]), TEEs (e.g., [5]), or even HE ([10], [11]). To date, privacy-preserving blockchains that have been deployed in production leverage Trusted Execution Environments (TEEs). Attesting to their usefulness in practice, a recent survey paper has identified and examined 17 such blockchains [5]. In their paper, the authors review different design choices regarding how these blockchains are built in practice. They identify systems where the contract execution happens on-chain (e.g., [28], [29]), or off-chain (e.g., [9], [30]), in a permissioned setting, or a permissionless one. These design choices show that different trade-offs exists in terms of the guarantees these systems provide (for example, in terms of liveness, correctness and privacy).

Chapter 3

Decentralized Access-control and Confidential Smart Contracts

In this chapter, I cover two important aspects in the privacy of blockchains – *decentralized access control* and *confidential smart contracts*.

Both of these concepts were first introduced almost a decade ago in a short paper we published [3], which became a foundational work with over 3,000 citations at the time of this writing. It is therefore useful to present it in full here, even though the technical details themselves are somewhat dated at this point.

As a follow-up to this work, my Master's thesis [31] and related paper [4] described the first *confidential smart contracts* system, along with the seminal work of [7]. While these works presented all kinds of novel ideas, they were incomplete in both definition and implementation. For example, they suffered from high latency and were not integrated into a proper smart-contract language like Solidity (because Ethereum was in its early days back then).

Therefore, in this chapter, I will also introduce a modern variant of a *confidential smart contract* system - in particular, one that is based on (threshold) Fully Homomorphic Encryption (FHE). While FHE is a heavier cryptographic primitive, unlike the work in [4], it does not require the computing parties to communicate (beyond decryption). This, in practice, is much preferable in a high-latency decentralized system like a blockchain. To somewhat counteract the inefficiency associated with FHE, this work presents an FHE-Rollup variant.

Finally, in the next chapter, we will build on the existence of *confidential smart contracts* and show how they can be used to improve user authentication, and MPC more generally.

3.1 Decentralized Access-Control

In this work, we discuss how we can leverage the availability and trust-minimized nature of a blockchain-system as an access-control mechanism. We are motivated by the privacy challenges users face when using third-party services. We focus specifically on mobile platforms, where services deploy applications for users to install. These applications constantly collect high-resolution personal data of which the user has no specific knowledge or control. In our analysis, we assume that the services are *honest-but-curious* (i.e., they follow the protocol), but they may not always be available or easily allow data mobility. Note that the same system could be used for other data-privacy concerns, such as patients sharing their medical data for scientific research, while having the means to monitor how it is used and the ability to instantly opt-out. In light of this, our system protects against the following common privacy issues:

Data Ownership. Our framework focuses on ensuring that users own and control their personal data. As such, the system recognizes the users as the owners of the data and the services as guests with delegated permissions.

Data Transparency and Auditability. Each user has complete transparency over what data is being collected about her and how they are accessed.

Fine-grained Access Control. One major concern with mobile applications is that users are required to grant a set of permissions upon sign-up. These permissions are granted indefinitely and the only way to alter the agreement is by opting-out. Instead, in our framework, at any given time the user may alter the set of permissions and revoke access to previously collected data. One application of this mechanism would be to improve the existing permissions dialog in mobile applications. While the user-interface is likely to remain the same, the access-control policies would be securely stored on a blockchain, where only the user is allowed to change them.

3.1.1 Proposed Solution

We begin with an overview of our system. As illustrated in Figure 3.1, the three entities comprising our system are mobile phone *users*, interested in downloading and using applications; *services*, the providers of such applications who require processing personal data for operational and business-related reasons (e.g., targeted ads, personalized service); and *nodes*, entities entrusted with maintaining the blockchain and a distributed private key-value data store in return for incentives. Note that while users in the system normally remain (pseudo) anonymous, we could store service profiles on the blockchain and verify their identity.

The system itself is designed as follows. The blockchain accepts two new types of transactions: T_{access}, used for access control management; and T_{data}, for data storage and retrieval. These network operations could be easily integrated into a mobile software development kit (SDK) that services can use in their development process.

To illustrate, consider the following example: a user installs an application that uses our platform for preserving her privacy. As the user signs up for the first time, a new shared $(user, service)$ identity is generated and sent, along with the associated permissions, to the blockchain in a T_{access} transaction. Data collected on the phone (e.g., sensor data such as location) is encrypted using a shared encryption key and sent to the blockchain in a T_{data} transaction, which subsequently routes it to an off-blockchain key-value store, while retaining only a pointer to the data on the public ledger (the pointer is the SHA-256 hash of the data).

Both the service and the user can now query the data using a T_{data} transaction with the pointer (key) associated to it. The blockchain then verifies that the digital signature belongs to either the user or the service. For the service, its permissions to access the data are checked as well. Finally, the user can change the permissions granted to a service at any time by issuing a T_{access} transaction with a new set of permissions, including revoking access to previously stored data. Developing a web-based (or mobile) dashboard that allows an overview of one's data and the ability to change permissions is fairly trivial and is similar to developing centralized-wallets, such as Coinbase for Bitcoin[1].

The off-blockchain key-value store is an implementation of Kademilia [32], a distributed hashtable (or *DHT*), with added persistence using LevelDB[2] and an interface to the blockchain. The DHT is maintained by a network of nodes (possibly disjoint from the blockchain network), who fulfill approved read/write transactions. Data are sufficiently randomized across the nodes and replicated to ensure high availability. It is instructive to note that alternative off-blockchain solutions could be considered for storage. For example, a centralized cloud might be used to store the data. While this requires some amount of trust in a third-party, it has some advantages in terms of scalability and ease of deployment.

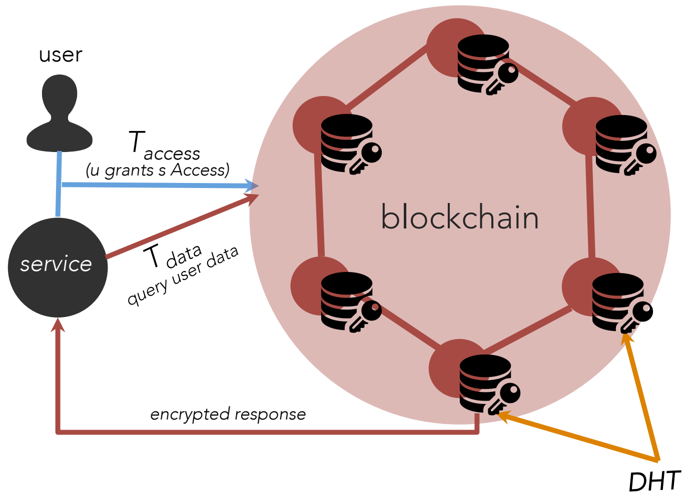


Figure 3.1: Overview of the decentralized platform.

3.1.2 The Network Protocol

We now describe in detail the underlying protocol used in the system. We utilize standard cryptographic building blocks in our platform: a symmetric encryption scheme defined by the 3-tuple $(\mathcal{G}_{enc}, \mathcal{E}_{enc}, \mathcal{D}_{enc})$ – the key generator, encryption and decryption algorithms respectively; a digital signature scheme (*DSS*) described by the 3-tuple $(\mathcal{G}_{sig}, \mathcal{S}_{sig}, \mathcal{V}_{sig})$ – the key generation, signature and verification algorithms respectively, implemented using ECDSA with *secp256k1* curve [33]; and a cryptographic hash function $\mathcal{H}$, instantiated by a *SHA-256* [34] implementation.

3.1.3 Building Blocks

We now briefly introduce relevant building blocks that are used throughout this chapter. We assume some familiarity with Bitcoin [1] and blockchains.

[1]Coinbase bitcoin wallet, http://www.coinbase.com

[2]LevelDB, http://github.com/google/leveldb

Identities

Blockchains utilize a pseudo-identity mechanism. Essentially a public-key, every user can generate as many such pseudo-identities as she desires in order to increase privacy. We now introduce *compound identities*, an extension of this model used in our system. A compound identity is a shared identity for two or more parties, where some parties (at least one) own the identity (*owners*), and the rest have restricted access to it (*guests*). Protocol 1 illustrates the implementation for a single owner (the user) and a single guest (the service). As illustrated, the identity is comprised of signing key-pairs for the owner and guest, as well as a symmetric key used to encrypt (and decrypt) the data, so that the data is protected from all other players in the system. Formally, a compound identity is externally (as seen by the network) observed by the 2-tuple:

$$Compound_{u,s}^{(public)} = (pk_{sig}^{u,s}, pk_{sig}^{s,u}) \tag{3.1}$$

Similarly, the entire identity (including the private keys) is the following 5-tuple:

$$Compound_{u,s} = (pk_{sig}^{u,s}, sk_{sig}^{u,s}, pk_{sig}^{s,u}, sk_{sig}^{s,u}, sk_{enc}^{u,s}) \tag{3.2}$$

Protocol 1 Generating a compound identity

```
 1: procedure CompoundIdentity($(u, s)$)
 2:     $u\ and\ s\ form\ a\ secure\ channel$
 3:     $u$ executes:
 4:         $(pk_{sig}^{u,s}, sk_{sig}^{u,s}) \leftarrow \mathcal{G}_{sig}()$
 5:         $sk_{enc}^{u,s} \leftarrow \mathcal{G}_{enc}()$
 6:         $u\ shares\ sk_{enc}^{u,s}, pk_{sig}^{u,s}\ with\ s$
 7:     $s$ executes:
 8:         $(pk_{sig}^{s,u}, sk_{sig}^{s,u}) \leftarrow \mathcal{G}_{sig}()$
 9:         $s\ shares\ pk_{sig}^{s,u}\ with\ s$
10:     // Both $u$ and $s$ have $sk_{enc}^{u,s}, pk_{sig}^{u,s}, pk_{sig}^{s,u}$
11:     return $pk_{sig}^{u,s}, pk_{sig}^{s,u}, sk_{enc}^{u,s}$
12: end procedure
```

Blockchain Memory

We let L be the blockchain memory space, represented as the hastable $L : \{0,1\}^{256} \rightarrow \{0,1\}^N$, where $N >> 256$ and can store sufficiently-large documents. We assume this memory to be tamper-proof under the same adversarial model used in Bitcoin and other blockchains. To intuitively explain why such a trusted data-store can be implemented on any blockchain (including Bitcoin), consider the following simplified, albeit inefficient, implementation: A blockchain is a sequence of timestamped transactions, where each transaction includes a variable number of output addresses (each address is a 160-bit number). L could then be implemented as follows – the first two outputs in a transaction encode the 256-bit memory address pointer, as well as some auxiliary meta-data. The rest of the outputs construct the serialized document. When looking up $L[k]$, only the most recent transaction is returned, which allows update and delete operations in addition to inserts.

Policy

A set of permissions a user u grants service s, denoted by $POLICY_{u,s}$. For example, if u installs a mobile application requiring access to the user's location and contacts, then $POLICY_{u,s} = \{location, contacts\}$. It is instructive to note that any type of data could be stored safely this way, assuming the service will not subvert the protocol and label the data incorrectly. Safeguards to partially prevent this could be introduced to the mobile SDK, but in any case, the user could easily detect a service that cheats, as all changes are visible to her.

Auxiliary Functions

$Parse(x)$ de-seralizes the message sent to a transaction, which contains the arguments; $CheckPolicy(pk^k_{sig}, x_p)$, illustrated in Protocol 2, verifies that the originator has the appropriate permissions.

Protocol 2 Permissions check against the blockchain

```
1: procedure CheckPolicy($pk^k_{sig}, x_p$)
2:     $s \leftarrow 0$
3:     $a_{policy} = \mathcal{H}(pk^k_{sig})$
4:     if $L[a_{policy}] \neq \emptyset$ then
5:         $pk^{u,s}_{sig}, pk^{s,u}_{sig}, POLICY_{u,s} \leftarrow Parse(L[a_{policy}])$
6:         if $pk^k_{sig} = pk^{u,s}_{sig}$ or
7: $(pk^k_{sig} = pk^{s,u}_{sig}$ & $x_p \in POLICY_{u,s})$ then
8:             $s \leftarrow 1$
9:         end if
10:    end if
11:    return $s$
12: end procedure
```

3.1.4 Blockchain Protocols

Here we provide a detailed description of the core protocols executed on the blockchain. Protocol 3 is executed by nodes in the network when a T_{access} transaction is received, and similarly, Protocol 4 is executed for T_{data} transactions.

As mentioned earlier, T_{access} transactions allow users to change the set of permissions granted to a service, by sending a $POLICY_{u,s}$ set. Sending the empty set revokes all access-rights previously granted. Sending a T_{access} transaction with a new compound identity for the first time is interpreted as a user signing up to a service.

Similarly, T_{data} transactions govern read/write operations. With the help of $CheckPolicy$, only the user (always) or the service (if allowed) can access the data. Note that in lines 9 and 16 of Protocol 4 we used shorthand notation for accessing the DHT like a normal hashtable. In practice, these instructions result in an off-blockchain network message (either read or write) that is sent to the DHT.

Protocol 3 Access Control Protocol

```
 1: procedure HandleAccessTX($pk^{k}_{sig}, m$)
 2:     $s \leftarrow 0$
 3:     $pk^{u,s}_{sig}, pk^{s,u}_{sig}, POLICY_{u,s} = Parse(m)$
 4:     if $pk^{k}_{sig} = pk^{u,s}_{sig}$ then
 5:         $L[\mathcal{H}(pk^{k}_{sig})] = m$
 6:         $s \leftarrow 1$
 7:     end if
 8:     return $s$
 9: end procedure
```

Protocol 4 Storing or Loading Data

```
 1: procedure HandleDataTX($pk^{k}_{sig}, m$)
 2:     $c, x_p, rw = Parse(m)$
 3:     if $CheckPolicy(pk^{k}_{sig}, x_p) = \texttt{True}$ then
 4:         $pk^{u,s}_{sig}, pk^{s,u}_{sig}, POLICY_{u,s} \leftarrow Parse(L[\mathcal{H}(pk^{u,s}_{sig})])$
 5:         $a_{x_p} = \mathcal{H}(pk^{u,s}_{sig} \parallel x_p)$
 6:         if $rw = 0$ then                                  ▷ rw=0 for write, 1 for read
 7:             $h_c = \mathcal{H}(c)$
 8:             $L[a_{x_p}] \leftarrow L[a_{x_p}] \cup h_c$
 9:             (DHT) $ds[h_c] \leftarrow c$
10:             return $h_c$
11:         else if $c \in L[a_{x_p}]$ then
12:             (DHT) return $ds[h_c]$
13:         end if
14:     end if
15:     return $\emptyset$
16: end procedure
```

3.1.5 Privacy and Security Analysis

We rely on the blockchain being tamper-proof, an assumption that requires a sufficiently large network of untrusted peers. In addition, we assume that the user manages her keys in a secure manner. Chapter 4 tackles this problem specifically. We now show how our system protects against adversaries compromising nodes in the system. Currently, we are less concerned about malicious services that change the protocol or record previously read data, as they are likely to be reputable, but we provide a possible solution for such behavior in section 3.1.6.

Given this model, only the user has control over her data. The decentralized nature of the blockchain combined with digitally-signed transactions ensure that an adversary cannot pose as the user, or corrupt the network, as that would imply the adversary forged a digital-signature, or gained control over the majority of the network's resources. Similarly, an adversary cannot learn anything from the public ledger, as only hashed pointers are stored on it.

An adversary controlling one or more DHT nodes cannot learn anything about the raw data, as it is encrypted with keys that none of the nodes posses. Note that while data integrity is not ensured in each node, since a single node can tamper with its local copy or act in a byzantine way, we can still in practice minimize the risk with sufficient distribution and replication of the data.

Finally, generating a new compound identity for each user-service pair guarantees that only a small fraction of the data is compromised in the event of an adversary obtaining both the signing and encryption keys. If the adversary obtains only one of the keys, then the data is still safe. Note that in practice we could further split the identities to limit the exposure of a single compromised compound identity. For example, we can generate new keys for every hundred records stored.

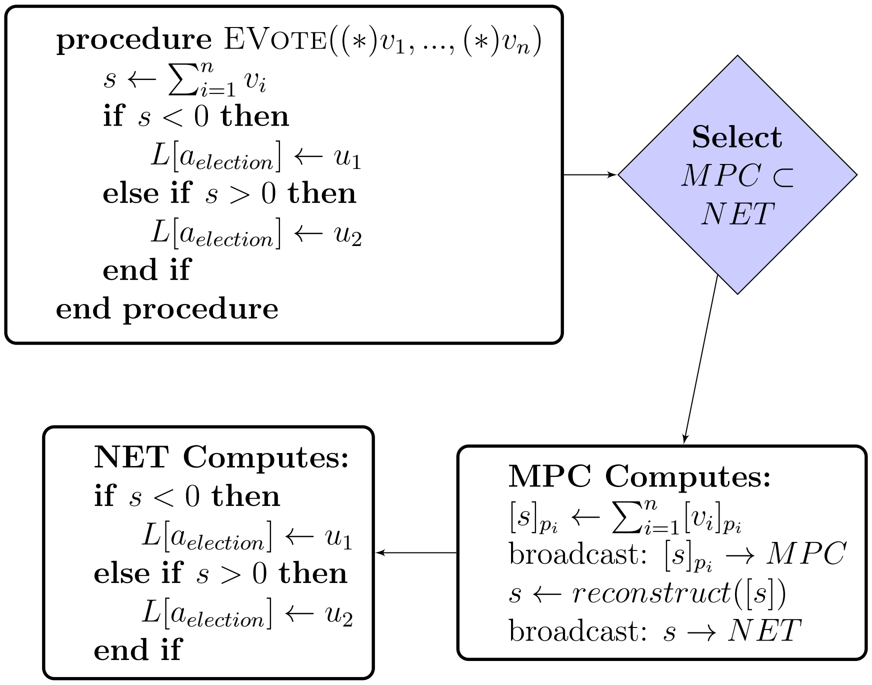
3.1.6 Confidential Smart Contracts: From Storage to Processing

One of the major contributions of this work is demonstrating how to overcome the public nature of the blockchain. So far, our analysis focused on storing pointers to encrypted data. While this approach is suitable for storage and random retrieval, it is not very efficient for processing data. More importantly, once a service queries a piece of raw data, it could store it for future analysis.

A better approach might be to never let a service observe the raw data, but instead, to allow it to run computations directly on the network, in a privacy-preserving manner, and obtain the final results. While this term was not present at the time of publication, over the years since this work was published, this idea of allowing arbitrary programs to run over a blockchain while guaranteeing both the correctness of the computation and the confidentiality of the underlying data became known as *confidential smart contracts*. Below is a sketch of a solution using secret-sharing based MPC, which was further elaborated in my subsequent work [4], [31].

A sketch of a Confidential Smart Contracts system. First, users can split data into shares (e.g., using Shamir's Secret Sharing [35]), rather than encrypting them, we could then use secure Multi-party Computation (*MPC*) to securely evaluate any function [36].

In Figure 3.2, we illustrate how MPC might work with blockchains and specifically in our framework. Consider a simple example in which a city holds an election and wishes to allow online secret voting. It develops a mobile application for voting which makes use of our system, now augmented with the proposed MPC capabilities. After the online elections take place, the city subsequently submits their back-end code to aggregate the results. The network selects a subset of nodes at random and an interpreter transforms the code into a secure MPC protocol. Finally, the results are stored on the public ledger, where they are safe against tampering. As a result, no one learns what the individual votes were, but everyone can see the results of the elections.



```
procedure EVOTE($(*)v_1, ..., (*)v_n$)
    $s \leftarrow \sum_{i=1}^{n} v_i$
    if $s < 0$ then
        $L[a_{election}] \leftarrow u_1$
    else if $s > 0$ then
        $L[a_{election}] \leftarrow u_2$
    end if
end procedure
```

```
MPC Computes:
$[s]_{p_i} \leftarrow \sum_{i=1}^{n} [v_i]_{p_i}$
broadcast: $[s]_{p_i} \rightarrow MPC$
$s \leftarrow reconstruct([s])$
broadcast: $s \rightarrow NET$
```

```
NET Computes:
if $s < 0$ then
    $L[a_{election}] \leftarrow u_1$
else if $s > 0$ then
    $L[a_{election}] \leftarrow u_2$
end if
```

Figure 3.2: Example of a flow of secure computation in a blockchain network. The top left block (EVote procedure) is the unsecure code, where the arguments marked in (*) are private and stored as shares on the DHT. The network selects a subset of nodes at random to compute a secure version of EVote and broadcasts the results back to the entire network, that stores it on the ledger.

In the next Section (3.2), I will discuss our recent work on building a modern take on a confidential smart contracts platform that uses threshold FHE.

3.1.7 Trust and Decision-Making in Blockchains

Bitcoin, or blockchains in general, assumes all nodes are equally untrusted and that their proportion in the collective decision-making process is solely based on their computational resources (known as the Proof-of-work algorithm) [1]. In other words – for every node n, $trust_n \propto resources(n)$ (probabilistically) decides the node's weight in votes. This leads to adverse effects, most notably vulnerability to sybil attacks, excessive energy consumption and high-latency.

Intuitively, Proof-of-Work reasons that nodes which pour significant resources into the system are less likely to cheat. Using similar reasoning we could define a new dynamic measure of trust that is based on node behavior, such that good actors that follow the protocol are rewarded. Specifically, we could set the trust of each node as the expected value of it behaving well in the future. Equivalently, since we are dealing with a binary random variable, the expected value is simply the probability p. A simple way to approximate this probability is by counting the number of good and bad actions a node takes, then using the sigmoid function to squash it into a probability. In practice, every block i we should re-evaluate the trust score of every node as –

$$trust_n^{(i)} = \frac{1}{1 + e^{-\alpha(\#good - \#bad)}}, \tag{3.3}$$

where α is simply the step size.

With this measure, the network could give more weight to *trusted* nodes and compute blocks more efficiently. Since it takes time to earn trust in the system, it should be resistant to sybil attacks. This mechanism could potentially attract other types of attacks, such as nodes increasing their reputation just to act maliciously at a later time. This might be mitigated by randomly selecting several nodes, weighted by their trust, to vote on each block, then taking the *equally-weighted* majority vote. This should prevent single actors from having too much influence, regardless of their trust-level.

3.2 FHE-Rollups: An EVM-compatiable Confidential Smart Contracts Platform

As mentioned earlier, in recent years, researchers and practitioners have employed several privacy-preserving technologies to solve the problem of confidentiality on the blockchain, in a body of work that became known as *confidential smart contracts* (e.g., [3], [4], [8], [9], [11], [28]). Of all of these techniques, Fully Homomorphic Encryption (FHE) is perhaps the most ambitious, as it allows to directly compute over encrypted data without decrypting it.

FHE has improved by leaps and bounds since Gentry presented the first construct almost a decade and a half ago [37]. Still, FHE requires significant computational overhead compared to computing in plaintext, making it impractical for execution at the layer-1 (L1) level where every node is required to replicate the entire computation, which is the approach that state-of-the-art FHE-based confidential smart contracts frameworks are taking [11], [38].

Inspired by the recent movement towards layer-2 solutions in the Ethereum ecosystem [39]–[41], we present the first architecture of an FHE-based rollup. We argue that while for plaintext computation rollups are a needed solution, in the context of FHE, where the computational overhead is orders of magnitude higher, they are a necessity.

In a rollup architecture, smart contract execution (the heavy-duty part of validating blocks) is separated from verifying the execution and reaching consensus. This ensures that only a single node (or a small number of nodes) is actually doing the computational heavy lifting, without impairing security. Furthermore, this node can be vertically (and horizontally) scaled as needed, including utilizing more expensive specialized hardware (GPUs, ASICs).

The latter is common with zero-knowledge (zk) based rollups [3], which like FHE also leverages computationally-intensive cryptography, and can be leveraged in much the same way for FHE computations.

However, in our design, we take an optimistic rollup approach as opposed to a zk-rollup approach, allowing us to avoid the orders of magnitude penalty incurred by state-of-the-art verifiable FHE techniques [42]. In fact, our framework can be seen as a cryptoeconomic solution to solve the same problem of verifiability in FHE.

3.2.1 Main contributions

In this part, we make the following main contributions:

- We introduce the first layer-2 *confidential smart contracts* platform, enabling greater efficiency and scalability.
- We demonstrate through a proof-of-concept implementation, that an optimistic FHE rollup can be built on top of Ethereum, without making any changes to the base layer. While our work extends beyond Ethereum and EVM chains, showing that this is possible on Ethereum *today* implies that the most used smart-contract ecosystem can be augmented with confidential smart contracts.
- We implement and benchmark three types of confidential smart contracts in Solidity: (i) a confidential ERC-20 contract; (ii) a sealed-bid auction contract; (iii) and a private voting contract. All of these examples can only operate in a blockchain with confidentiality. We also demonstrate empirically that our solution is concretely efficient and practical.
- Outside the context of blockchains, our solution can be seen as a more efficient (cryptoeconomic) solution to the problem of verifiable FHE.

3.2.2 Design Goals and Security Model

Design Goals

Our system is built with the following objectives in mind:

- *Correctness and Availability.* A smart contract is executed correctly and with guaranteed output.
- *Input Confidentiality.* Nothing is learned about users' inputs during the execution of a smart contract. Since computations are reactive (i.e., stateful), the state can be considered as another private input.
- *Selective Output Confidentiality.* Smart contract outputs can be re-encrypted and selectively shared with the querying user, or with another with proper permissions. No one else needs to learn anything about the output. Outputs can also be made public.

[3]e.g., https://www.ingonyama.com, https://www.risczero.com

- *Efficiency.* Execution efficiency is proportional to the computation complexity of the underlying FHE execution. This captures the rollup's efficiency property, which denotes that there is no need for consensus or replicating the computation.

In terms of confidentiality, we note that we do not try to hide the following: (i) identity of the user initiating a transaction (e.g., for executing a smart contract); (ii) The smart contract being called, and the method being called. In other words, there is no *circuit privacy.*

Security Model

We assume our Rollup is built on-top of a layer-1 that provides the usual properties of a blockchain (i.e., correctness and availability).

For our Threshold Services Network (see Section 3.2.7), we assume nodes share pairwise secure channels and a broadcast channel (for the latter, the Layer-1 can be used directly). We also assume an honest majority between the nodes, as required by the underlying decryption protocol we use. We note that our architecture generalizes to threshold decryption protocols that support a dishonest majority, and that this assumption is not a hard requirement.

Finally, like other optimistic rollups, we assume at least a single honest validator (also known as a verifier) exists.

3.2.3 Related Work

Our work builds upon the existing body of research and development of *confidential smart contract* platforms. Unlike all other works, to the best of our knowledge this is the first work that describes a solution that operates fully and natively as a layer-2. More specifically, other confidential smart contract platforms usually differ by the kinds of privacy-preserving technologies they use:

- **Trusted Execution Environment (TEEs) Based**. Currently, the only confidential smart contracts networks in production are using TEEs (or secure enclaves) [5], [9], [28], [30]. These networks simulate secure computation by allowing users to encrypt their transactions with keys held inside of a secure enclave. Transactions then get decrypted and executed inside of the enclave, which ensures confidentiality as long as we can trust the security of the TEE. While TEEs are by far the most efficient solution, they are susceptible to side-channel attacks and other vulnerabilities (e.g., [43]–[45]).

- **Secure Multiparty Computation (MPC) Based**. Since our work relies on Threshold FHE, we share a similar threat model with these works (e.g., [3], [4], [8], [46]–[48]). However, for the purpose of presentation, we separate these from FHE-based solutions, as they often rely on linear secret-sharing [49], [50] and garbled circuits techniques [51]. The main drawback in these techniques compared to our work is that MPC protocols need to communicate data proportional to the circuit size in order to evaluate it. In the case of secret-sharing-based MPC, all parties also need to sequentially communicate with all other parties any time they evaluate a multiplication gate. In the context of public blockchains, this is impractical, as the latency is quite high and the bandwidth is limited. Furthermore, as these systems require multiple interacting nodes for every contract execution, they are not amenable to a rollup architecture such as the one we are proposing.

- **Zero-knowledge (ZK)-based**. Different works have considered confidential smart contracts using different ZK schemes. However, since ZK techniques are more suitable for verifiable computation (e.g., [52]), their utility for confidential smart contracts is limited. To overcome this, Hawk [7] suggested having a data-manager – an off-chain party that is tasked with collecting inputs from different clients and is trusted with seeing everyone's data. Alternatively, other platforms impose limitations on developers [53]–[56].

- **FHE-based**. In the last couple of years, as a result of significant FHE performance improvements, HE and FHE based solutions have started to emerge [10], [11], [38], [57], [58]. These platforms are closest to our work, but none of them adopt a rollup architecture, which limits their scalability in practice. Some of these works adopt a Threshold FHE structure as we do (e.g., [38]), whereas others such as [10] allow only limited functionality, and without a shared state.

3.2.4 Layer-2 Rollups

Rollups are a scaling solution designed to alleviate the congestion on the primary layer-1 chain, in particular Ethereum. With Ethereum's growing user base, the need for scaling solutions has become paramount. The primary goal of scalability is to enhance transaction speed and throughput without compromising on decentralization or security. Rollups execute transactions outside the base layer, posting back state-updates alongside proofs of correct execution. Ultimately, the layer-1 reaches consensus on these state updates, but it does so without re-executing the transactions on the base layer. In other words, transactions on the rollup are secured by layer-1's inherent security. There are two main types of rollups, which differ by how proofs are created and verified:

- **Optimistic Rollups**. These assume transactions are valid unless challenged. They move computation off-chain but post transaction data to the layer-1 (or another data availability network), allowing anyone to re-run the transactions off-chain and verify for themselves that the execution is correct. If any verifier detects malicious behavior, they can submit a fraud-proof on-chain, in which case the layer-1 acts as the final arbiter. For this reason, Optimistic Rollups require a dispute period (often of a few days) [40].

- **ZK Rollups**. These rollups similarly execute contracts off-chain and submit validity proofs back when sending a state-update. Validity proofs are constructed using advanced cryptographic techniques known as (succinct) ZK Proofs. They can be efficiently verified on-chain directly, without posting the full transaction data or having a dispute period (e.g., [59]).

Optimistic and ZK Rollups have inherently different trade-offs. Optimistic rollups suffer from a dispute-period delay, making finality longer. They also require posting the transactions themselves on-chain, which negates some of the scalability benefits [4].

On the other hand, ZK Rollups require significant computation power (and time) to produce a proof, especially the closer you try to get to native EVM [60]. A related downside

[4]This is meant to be mitigated to an extent with EIP-4844 and Danksharding.

is that these rollups are much more complicated to build, resulting in large amounts of code. The likelihood of critical vulnerabilities with zkEVMs is therefore much higher, at least until they have been battle tested enough over time.

Optimistic Rollups in More Detail

Optimistic rollups bundle multiple off-chain transactions and submit them to the L1 chain, reducing costs for users. They are termed "optimistic" because they assume transactions are valid unless proven otherwise. If a transaction is challenged, a fraud proof is computed. If proven fraudulent, penalties are applied. Today, optimistic rollups operate atop Ethereum, managed by Ethereum-based smart contracts. They process transactions off-chain but post data batches to an on-chain rollup contract. Ethereum ensures the correctness of rollup computations and handles data availability, making rollups more secure than standalone off-chain solutions or side-chains. They also create an inherent economic-security alignment between the two layers, as the layer-2 receives security while paying for incurred fees at the layer-1 level.

From an architectural perspective, optimistic rollups consist of the following:

- *Transaction Execution.* Users send transactions to operators or validators, who aggregate and compress them for the layer-1.
- *Submitting to the layer-1.* Operators bundle transactions and send them to the layer-1 using calldata.
- *State Commitments.* The rollup's state is represented as a Merkle tree. Operators submit old and new state roots, ensuring the chain's integrity.
- *Fraud Proofs and Disputes.* These allow anyone to challenge a transaction's validity. If a challenge is valid (arbitrated by the layer-1), the fraudulent party is penalized.

Fraud Proofs play a vital role in optimistic rollups, and they are at the root of how we ensure that a rollup publishes a correct state update. Even in the face of malicious nodes trying to delay or tamper with transactions, the chain's integrity is preserved as long as there's a single honest node that observes state updates and checks that they are correct. In the context of rollups built on Ethereum, because the actual data is posted onto Ethereum, anyone in the world can act as a verifier. Once an honest verifier detects an incorrect state update (e.g., by including a tampered-with transaction), it can submit a dispute, which initiates the fraud proof game: a multi-round interactive protocol. Here, the asserter (the node that produced the state update) and challenger (the verifier who issued a dispute) follow a protocol overseen by a layer-1 verifier contract to ascertain the honest party. The protocol proceeds recursively, each time dividing the computation into two equal parts. The challenger chooses one part to challenge each time. This process, termed the *bisection protocol*, persists until only one computational step is in question. Once the interactive protocol narrows down to a single instruction, it is the layer-1 contract's turn to resolve the dispute by evaluating the instruction result as well as both the asserter's and the challenger's claims and their respective results to determine which one is correct.

3.2.5 Fully Homomorphic Encryption (FHE)

Fully Homomorphic Encryption (FHE) enables computations on ciphertexts that, when decrypted, match the results of those operations as if they were performed on the plaintext directly.

In practice, ever since the original scheme by Gentry [37], several FHE schemes have been developed, which are based on the original Learning With Errors (LWE) hardness problem, or related algebraic constructs such as its ring variant [61]. Our implementation utilizes the Torus FHE (TFHE) [62] scheme, but we note that our construct itself does not really matter for the purpose of constructing an FHE-rollup, we describe FHE more generally below, in a black-box manner.

A generic FHE scheme can be denoted by the tuple of algorithms $\mathsf{FHE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec},)$, as follows:

- $\mathsf{Gen}(1^\kappa)$. Given a security parameter 1^κ, the algorithm outputs a pair of keys (pk, sk) where pk is the public encryption key and sk is the secret decryption key. Define domains $\mathcal{P}$ for plaintexts, $\mathcal{R}$ for randomness, and $\mathcal{C}$ for ciphertexts, as well as the set of permissible functions $\mathcal{F}$.

- $\mathsf{Enc}(pk, m; r)$. Given the public key pk, a message $m \in \mathcal{P}$ and randomness $r \in \mathcal{R}$, the encryption algorithm produces a ciphertext $c \in \mathcal{C}$ such that

$$c = \mathsf{Enc}_{pk}(m; r).$$

- $\mathsf{Dec}(sk, c)$. With the secret key sk and a ciphertext c, the decryption algorithm retrieves the original plaintext message m:

$$m = \mathsf{Dec}_{sk}(c).$$

- $(pk, f, \{c_i\}_{i=1}^n)$. Given the public key pk, a function $f \in \mathcal{F}$, and a set of ciphertexts $\{c_i\}_{i=1}^n$, this algorithm produces a ciphertext $c' \in \mathcal{C}$ such that:

$$\mathsf{Dec}_{sk}(c') = f(\{\mathsf{Dec}_{sk}(c_i)\}_{i=1}^n).$$

  This implies the function f is executed over encrypted data, and its result is encrypted as c'.

3.2.6 System Overview

Our platform is built with modularity in mind. It includes the quintessential components of a rollup, alongside new and specific components needed to support (Threshold) FHE. In this section, we briefly describe the different components, as they are illustrated in Figure 3.3.

- **Settlement and Data Availability (DA).** These components are served by the layer-1. In our implementation both will leverage Ethereum directly, but we note that any other layer-1 would work, with the understanding that we inherit its security.

- **Sequencer.** Just like with other layer-2 architectures (e.g., [40]), transaction ordering is done by an entity known as the sequencer. In our design, the same entity is also in charge of transaction (and smart contract) execution, but we note that this two roles could be separated. The sequencer is in-charge of submitting layer-2 state updates periodically. A state update is the result of a batched execution of potentially many transactions.

- **Validators.** These are other nodes in the layer-2 network which observe state-updates submitted by the sequencer. If any validator observes an incorrect state-update (i.e., the sequencer cheated), they can trigger a dispute to the layer-1. In this case, the parties engage in an interactive protocol between the sequencer, challenging validator (the *challenger*) and the layer-1 which acts as an arbitrator (as described in [40], [63], [64]). At the end of the protocol, there is consensus on whether the sequencer or the challenger cheated. To further prevent cheating, it is common to impose financial penalties on the cheating party. The fraud-proof mechanism is further described in Section 3.2.9.

- **Threshold Services Network.** Our platform uses Threshold FHE under the hood, which implies users can encrypt their transactions using a single public FHE key. The secret key, however, is shared across a network of nodes we denote as the *threshold services network* (TSN). The TSN is is also in charge of any decryption or re-encryption operations that need to happen from time to time. We discuss this in more detail in Section 3.2.7.

3.2.7 Threshold Services Network (TSN)

A key component in our design is the Threshold Services Network (TSN). This network is separate from the layer-1 or other rollup components and plays several key roles. In particular, we assume that in a setup phase the TSN generates the network's Threshold FHE key-pair (sk, pk) (using a distributed key-generation protocol), and that pk is published onto the rollup's first block, making it available to all users. In contrast, the secret key has to remain private, and is secret-shared across the TSN participants. There are several threshold decryption protocols to choose from, and our current implementation utilizes [65], which uses Shamir secret-sharing to split sk into n shares, out of which $t+1$ shares are needed to reconstruct.

Threshold Decryption and Re-encryption

Occasionally, the network will need to decrypt certain results, or re-encrypt them to a designated user. The TSN is in-charge of any such request coming from the rollup.

To illustrate why this is needed, consider the following two examples. First, imagine a private voting contract deployed on the rollup. When users vote for a certain candidate, they encrypt their votes, and the contract tallies these votes (all encrypted). At a certain point in time, a functionality in the contract should be able to decrypt the tally and announce the winner. This request is routed to the TSN, which decrypts it and returns the result to be stored in the contract's state. This flow is illustrated in Figure 3.4.

A similar example is that of a user who owns a NFT with private metadata only they can see. Such metadata is stored in the contract's state under the network's key. When a user tries to access that information, the contract should threshold re-encrypt it so only the designated user would be able to decrypt and get access to the underlying data.

3.2.8 Security

It is important to note that the underlying confidentiality guarantees of the entire system are closely related to the trust assumptions of the TSN. Anything that relates to keeping the threshold decryption key safe, correctly decrypting/re-encrypting ciphertexts, etc., is under the responsibility of the TSN.

Currently, the state-of-the-art protocols by [65] require the TSN to have at most $t < \frac{n}{3}$ malicious corruptions for a fast and robust protocol, or $t < \frac{n}{2}$ if we are willing to settle for security with abort.

3.2.9 Fraud Proofs

The key to Optimistic Rollups lies in their fraud proof mechanism. But how do we fit that mechanism, in particular Ethereum's EVM, to work with smart contracts that execute FHE circuits over encrypted data?

First, observe that FHE, unlike other encrypted computation techniques such as MPC, natively allows anyone to verify that a computation was done correctly, without breaking the privacy guarantees. This is because an adversary holding the encrypted inputs and outputs learns nothing about the underlying encrypted data (if this were not the case, then the encryption scheme would not be semantically secure).

This makes verifying FHE computations compatible with the idea of Optimistic Rollups, at least in theory. As mentioned earlier, Optimistic Rollups are based on posting the full transaction data on the layer-1 (or some other data availability service), alongside the output. In this case, both are encrypted. Just like with plaintext data, any off-chain validator could take the encrypted transaction data, re-execute the transactions, and make sure that they receive the same encrypted output. If this is not the case, then an honest validator could submit a dispute and start the arbitration process with the layer-1.

However, the whole fraud proving mechanism is rooted on the layer-1's ability to determine unequivocally whether the layer-2 node that posted the state update or the disputing verifier (i.e., the *challenger*) is cheating. To do this, the layer-1 needs to be able to run a single computational step of the underlying computation. Since our solution relies on the security of the layer-1, we wish to use Ethereum in our implementation as it is the most secure smart-contract platform. But this introduces a new challenge: How can Ethereum, or any other layer-1, validate execution on FHE primitives without inherent support for FHE operations?

To overcome this challenge, we utilize Arbitrum's Nitro fraud prover [5], which has an Ethereum contract on-chain that can verify the correctness of a single WebAssembly (WASM)

[5]https://docs.arbitrum.io/inside-arbitrum-nitro

opcode. This is sufficient, as we can now compile the underlying FHE libraries (which are written in Rust) to WebAssembly as well, and avoid requiring changes to the layer-1 itself.

Addressing performance concerns, it is rational to assume that if FHE computations are inherently intensive, simulating them in a WASM runtime atop EVM might incur significant performance penalties. While this is a valid concern, it is important to remember that initiating these computations is only mandatory in a dispute scenario, and real-time speed is not an absolute necessity given a sufficient dispute window. Considering that the standard practice allocates approximately seven days for it, we estimate that this is more than sufficient time to settle any such disputes. However, we did not empirically validate this hypothesis, and we note that doing so in the future is important.

3.2.10 Evaluation

We implement a proof-of-concept of our layer-2 FHE rollup system [6]. Our architecture is based on the Arbitrum stack [40], and uses Ethereum as the layer-1. For FHE capabilities, we use the *tfhe-rs* [7] library, which is written in Rust. We then implement a Solidity wrapper on top, such that we can write standard Solidity smart contracts. Figure 3.6 is an example of such a contract for private voting.

As a first step, we benchmarked elementary FHE addition and multiplication gates under different VMs, and for (plaintext) integers between 8-256 bits. As can be seen in Table **??**, the performance gap between the VMs increases with the bit length. This is expected due to the underlying TFHE scheme [62]. However, on a per dollar basis, choosing a stronger VM does not justify the cost.

Table 3.1: FHE addition times using different VMs for 8-256bit plaintext integers.

System			**Addition (ms)**					
VM	**Cores**	**Price/yr**	**8**	**16**	**32**	**64**	**128**	**256**
I9-13900K	32	$1,000	47.527	71.752	122.07	202.26	440.57	975.29
R7i (x32)	32	$18,230	49	77.98	121.4	159.05	343.57	770.72
hpc7a.96xlarge	96	$62,208	63.201	83.203	102.75	120.27	145.52	192.26
m6id.metal	128	$52,790	70.5	100	132	186	249	334

Table 3.2: FHE multiplication times using different VMs for 8-256bit plaintext integers.

System			**Multiplication (ms)**					
VM	**Cores**	**Price/yr**	**8**	**16**	**32**	**64**	**128**	**256**
I9-13900K	32	$1,000	99.034	207.35	589.84	2084.2	7715.5	28946
R7i (x32)	32	$18,230	102.49	205.44	493.72	1587.2	5861.6	21777
hpc7a.96xlarge	96	$62,208	119.14	164.41	229.74	412.67	1059.4	3446.3
m6id.metal	128	$52,790	144	216	333	832	2500	8850

[6]https://github.com/FhenixProtocol/tfhewasm, https://github.com/FhenixProtocol/go-tfhe
[7]https://github.com/zama-ai/tfhe-rs

We also implemented three confidential Solidity smart contracts of real-world use cases. These include a confidential ERC-20 token contract (allowing for private transactions), a private voting contract, and a sealed-bid auction contract. For all contracts, we used 32-bit encrypted integers. The results are summarized in Table 3.3, and they illustrate that our solution is practical and concretely efficient.

Table 3.3: Benchmarking different Solidity Contracts using FHE

Contract Name	**Transaction**	**Time (ms)**
FHERC20	transfer()	832
Voting	vote()	620
Auction	bid()	3500

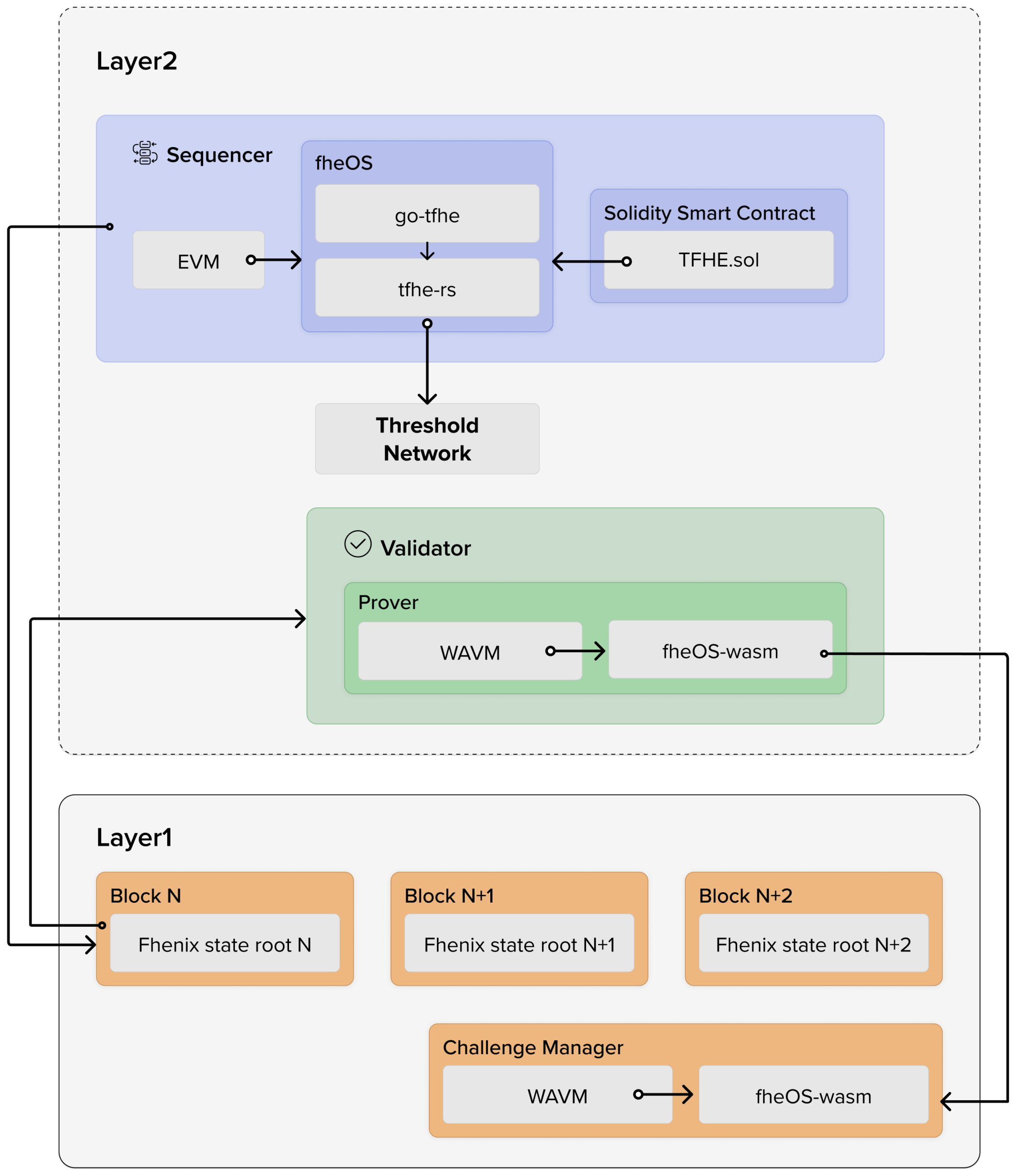


Figure 3.3: Overview of an FHE Rollup architecture

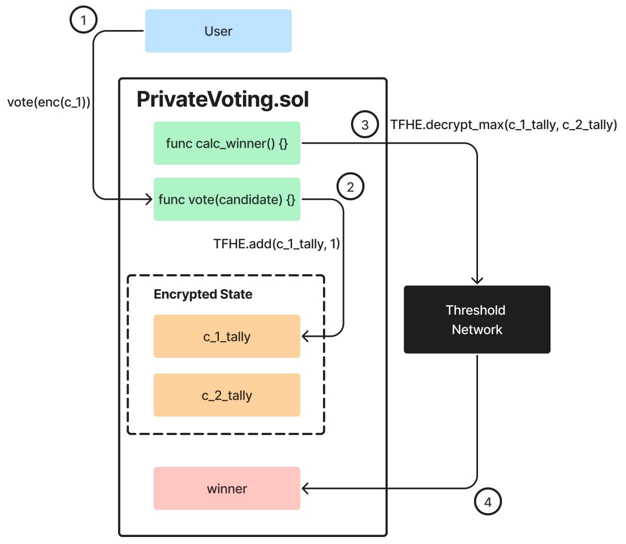
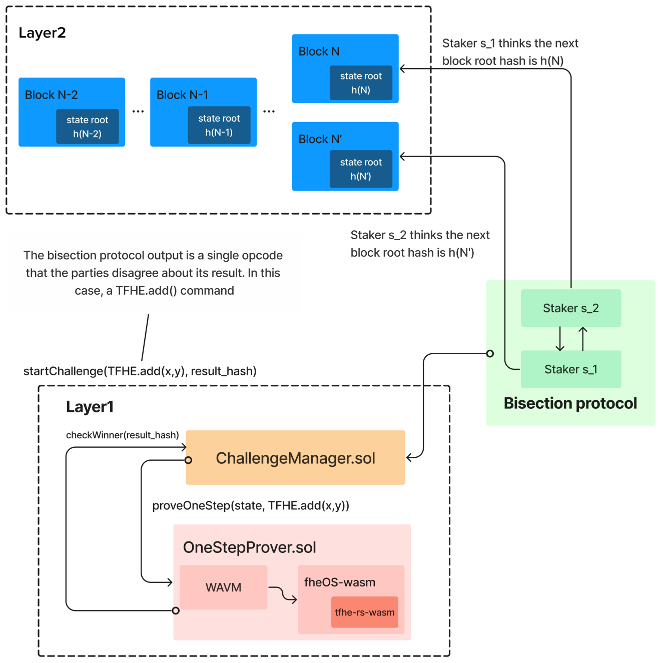


Figure 3.4: Smart contract request to decrypt FHE-encrypted data on the rollup



Layer2
Block N-2
state root h(N-2)
Block N-1
state root h(N-1)
Block N
state root h(N)
Block N'
state root h(N')
Staker s_1 thinks the next block root hash is h(N)
Staker s_2 thinks the next block root hash is h(N')
The bisection protocol output is a single opcode that the parties disagree about its result. In this case, a TFHE.add() command
startChallenge(TFHE.add(x,y), result_hash)
Staker s_2
Staker s_1
Bisection protocol
Layer1
checkWinner(result_hash)
ChallengeManager.sol
proveOneStep(state, TFHE.add(x,y))
OneStepProver.sol
WAVM
fheOS-wasm
tfhe-rs-wasm

Figure 3.5: Fraud proof engine for FHE instructions

Figure 3.6: Excerpt of a Voting Contract.

Chapter 4

Unstoppable Wallets: Chain-assisted Threshold ECDSA and its Applications

In the previous chapter, we discussed how a blockchain can implement a decentralized access-control mechanism, and have shown how that idea extends to confidential smart contract blockchains. In both of these cases, and in all blockchains more generally, there is a requirement for users to authenticate by digitally signing messages using a secret key that they own. The most common digital signature scheme used in blockchains is ECDSA (Elliptic Curve Digital Signature Algorithm). Securing a secret-key is potentially one of the biggest challenges in decentralized systems today, because if you lose your key (or if it is stolen), there is no trusted authority that can restore it for you. This chapter focuses on a novel solution to this problem, by designing new threshold ECDSA protocols that leverage the existence of a confidential smart contract blockchain, which we introduced earlier.

More specifically, threshold ECDSA is a cryptographic technique that enables multiple parties to jointly sign a message using a shared secret key. It has gained increasing importance in the custody of cryptocurrencies and Web3 due to its ability to improve security and control over private key management. By enabling multiple parties to jointly control a single private key, threshold ECDSA allows for the creation of multisignature (multisig) accounts that require multiple approvals before a transaction can be signed. This enhances security by reducing the risk of funds being stolen or lost due to a single point of failure. Additionally, threshold ECDSA enables the creation of flexible and customizable access policies.

Current deployments of threshold ECDSA (e.g., Fireblocks, Coinbase Wallet, BitGo, Zengo and others) rely on a third party service provider for availability and are often limited by closed-sourced vendors [66]. In practice, relying on a service provider for availability has proven itself inadequate, and in certain cases even disastrous, leading to significant loss of funds (e.g., FTX[1] and Prime Trust[67] incidents, to name a few). Furthermore, even beyond availability, trusting a service provider with *correctness* is difficult, as for example, a compromised service provider may ignore a client's policy and convince the client to sign an unintended transaction (e.g., by changing the user interface). Such attacks are not theoretical, see MyEtherWallet for example [68].

To overcome these challenges, we introduce *unstoppable wallets* as a novel concept that

[1]https://en.wikipedia.org/wiki/FTX

addresses the limitations of current threshold ECDSA systems. An unstoppable wallet is a threshold ECDSA wallet where the counterparty co-signing transactions with the user (or a set of users) is not a singular third-party, but rather a blockchain itself, which naturally provides strong availability and correctness. This enables the creation of programmable wallets that are controlled directly by a smart contract, such as those being explored through the concept of *account abstraction*[2]. Unstoppable wallets push this idea further, as they can operate cross-chain and are not limited to Ethereum or EVM chains only.

Since generally speaking, blockchains cannot keep a private state, we require the use of blockchains that support *confidential smart contracts*, which we introduced earlier.

Unstoppable wallets also have other implied benefits. Similar to how smart contracts eliminate the need for excessive trust in intermediaries, enabling novel applications like Decentralized Finance, unstoppable wallets further enable new use-cases that require similar levels of trust. To illustrate this, consider a decentralized lending protocol such as Compound[3]. In Compound, the smart contract acts as a trusted escrow between lenders and borrowers. In contrast, many centralized service providers offering the same functionality have recently failed, leading to billions in customer funds lost. Similarly, with unstoppable wallets, one can securely implement use-cases such as a *wallet exchange*, which allows users to atomically trade *wallets*. With an unstoppable wallet construct, an underlying smart contract atomically escrows and facilitates the process (just like lending in DeFi). Identifying a wallet as an asset on its own rather than a vehicle to store assets has many advantages. For example, one may transfer all its token portfolio in one transaction, even if part of it is staked, locked or lent; in addition, wallets can accrue reputation according to their activities, which may increase/decrease their value. In some sense, one may see a wallet as a non-fungible token (NFT).

4.1 Revisiting the Server-aided Model of MPC

Another main contribution of this work is showing how confidential smart contract blockchains are a pragmatic real-world realization of the *server-aided MPC* model. Cryptographic protocols in the server-aided model are common in the literature and real-world deployments, for both generic and application specific functionalities (see [69], [70] and references within). In this section we take this model further and argue that replacing the server with a blockchain serves as a better real-world realization of that model, in particular when availability is necessary.

Designers of cryptographic protocols have been increasingly relying on blockchains as their broadcast channel infrastructure [71]–[74], as they may assist in achieving desired properties (e.g., [73]), and work around known impossibility results (e.g., [74]). Such a transition has prompted researchers to explore other benefits that can be derived from blockchains.

At their core, blockchains provide such benefits due to the strong *availability* and *correctness* properties that they provide. One might also wish for a blockchain that entirely handles sensitive information, such as cryptographic keys, and is able to confidentially perform operations (like sign and decrypt) using the keys. Confidential smart contracts-enabled

[2]https://eips.ethereum.org/EIPS/eip-4337
[3]https://compound.finance/

blockchains aim to offer exactly that, by relying on an underlying MPC protocol or TEEs. Whichever assumption is used, it is important to note that while active faults in cryptographic multiparty protocols can be publicly detected and attributed, privacy breaches (in general) are not. Consider, for instance, a publicly auditable MPC protocol (in short, a protocol in which everyone can determine if a party faithfully follows the protocol steps or not). In case an attacker corrupts a sufficient number of parties it can "silently" break privacy; however, even such an attacker is not able to break the correctness of the protocol.

While we can rely on blockchains for correctness and availability, In our case of threshold signatures, we rely on the blockchain to store a partial secret, which is only a share of the actual underlying signing key. By doing so, breaking the blockchain security layer only reveals that share of the secret, and not the full key.

Equipped with this intuition, we present a Threshold ECDSA protocol for n parties, out of which at most $t < n$ are malicious and colluding, to generate an ECDSA key-pair and sign messages, with the aid of a blockchain as described above. We assume the blockchain supports confidential smart contracts, meaning that it uses privacy-preserving techniques to protect a contract state from outside actors. To keep the model as general as possible, we do not prescribe which technique the blockchain uses to achieve confidentiality. Instead, we capture the properties described above by modeling the blockchain as an additional semi-honest and non-colluding party, referred to as P_c. Such a party can easily play the role of a broadcast channel (by simply relaying a message to all other parties) and hold and operate on secrets.

Another benefit of this model, is that parties do not necessarily need to know each other in advance, or set up complex ad-hoc communication networks with point-to-point channels across each set of parties, or an underspecified broadcast channel, as is common with MPC protocols. Moreover, parties can come and go as they please, even mid-execution of a protocol, since all coordination is done on-chain, which is guaranteed to be robust. Finally, our protocols support detection of cheating parties, which can be immediately translated to a monetary punishment on-chain.

Lifting all communication on-chain is advantageous at a high level because it simplifies protocol implementation in practice, as each node only reads and writes to a single endpoint, regardless of the number of counterparties. Specifically, by relying on a blockchain, one does not need to take care of network synchronization, and 'proofs of silence' (i.e., a proof that a participant did not send a message) are taken for granted[4]. There are several other benefits to this, such as pseudonimity, higher degree of censorship-resistance, public accountability (e.g., in the context of DAO multi-sigs), etc. Finally, recall that in some settings, and in the dishonest majority setting that we address in particular, implementation of a broadcast channel is impossible. Thus, this blockchain assisted model implicitly outsources the broadcast channel operation to an external entity.

In this new communication model every message, either peer-to-peer or broadcast, is translated to a blockchain transaction, which is inherently a broadcast message. On one hand, broadcasting on chain may entail significantly larger latency than a plain broadcast

[4]This should not be interpreted as everything being perfect when using a blockchain; rather, we argue that using a blockchain obscures these problems away from the developer. Indeed, a block lacking a message from a user does not necessarily mean the user did not send that message; for example, the recent blockchain block's validator/miner may have censored that message.

that is implemented among the parties. On the other hand, in such model all messages are permanently public, which allows for publicly verifiable protocols that encourages honesty of the parties. In addition, On the other hand, all messages are available to the participants whenever they are ready to consume them. This enables an easy recovery and auditability by participants that experienced a temporary offline period.

This necessitates the reassessment of the concept of rounds – a crucial performance metric used to evaluate protocols in the standard MPC model (without the aid of blockchains). The number of rounds informally measures the longest sequence of interdependent messages sent between parties. In this modified model, each round consists of one or more parties writing to the blockchain, followed by all parties reading from it. Although one might assume that this would typically involve decomposing each round into two separate rounds, we must recognize that writing to a blockchain is significantly more costly than reading, as it necessitates consensus and updating a replicated state.

As a result, our primary objective in this model is to minimize the total number of messages, with a specific focus on reducing the number of sequential *writes*, simply referred to as 'writes' hereafter. While in previous works on threshold ECDSA the number of writes is equivalent to the number of rounds, this does not hold for the protocols presented in this work, highlighting the importance of identifying a common performance metric.

Figure 4.1 illustrates the model we described. All parties are connected via a slow *write* channel to the blockchain party P_c, which also acts as a public bulletin board they can read from (specifically, we assume P_c has a public state anyone can read from). Finally, being one of the computing parties, P_c also maintains its own private state.

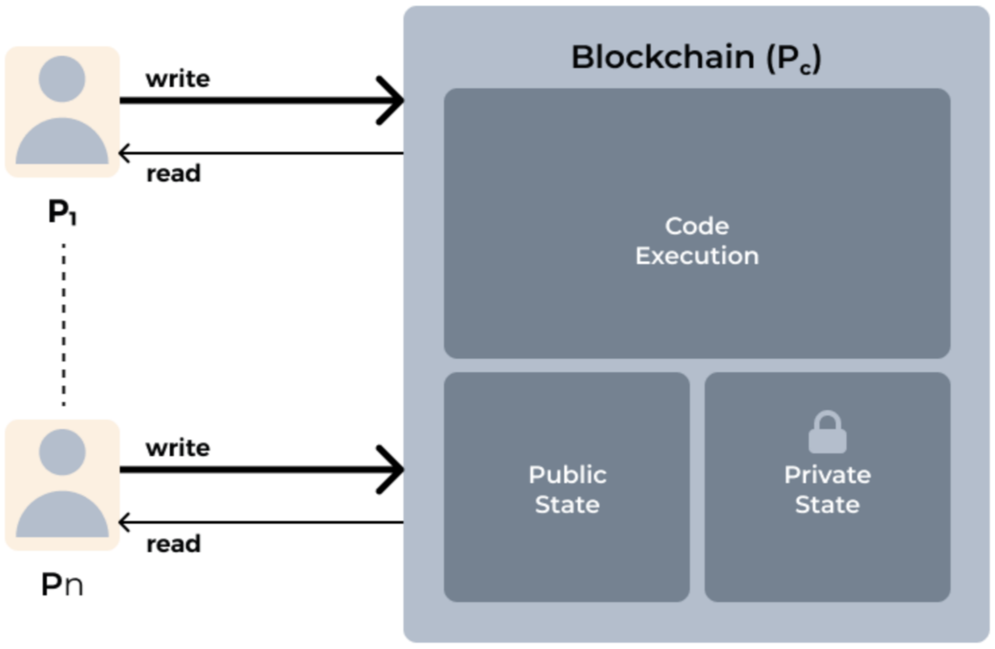


Figure 4.1: Communication Model Illustrated

4.2 Main contributions

In this chapter, we make the following main contributions:

- We revisit the problem of threshold ECDSA by considering a real-world 'server-aided' model, and construct new protocols for threshold ECDSA in that model. Defining messages *to the server* as 'write' and messages *from the server* as 'read', our protocols enjoy the minimal number of 'writes', compared to previous works (see Table 4.1). In particular, our robust threshold ECDSA incurs only a single write message from the parties, and another one from the signature initiator, compared to previous protocols that incur at least four writes. We also greatly reduce communication and computation, by avoiding the use of expensive cryptographic primitives such as Paillier encryption and costly zero knowledge proofs over Paillier ciphertexts. We provide a full proof of security of our protocols in the server-aided model, treating the server as a semi-honest non-colluding party, and show that the protocols offer both *fairness* and *robustness*. That is, we achieve *robustness* in the sense that if $t + 1$ parties agree to sign on a message (and hence participate in the protocol faithfully), then they will obtain the signed message.

- We implement these protocols as smart contracts and deploy them on a functioning blockchain with confidential smart contract capabilities. By doing this, we introduce the concept of *unstoppable wallets* - programmable threshold ECDSA wallets where the counterparty co-signing transactions with the user (or a set of users) is not a singular third-party we need to rely on, but rather a confidential smart contract.

- We ran benchmarks of our protocols, ranging from $n = 2$ to $n = 15$ signers, and prove their real world applicability by reporting on their respective gas costs and fees. To show case the importance of wallet programmability we develop two applications: a *multisignature wallet with policy checks* and a *wallet exchange*.

- We revisit the server-aided MPC model and show that a *confidential smart contract* can efficiently realize the server in this model. This should motivate more MPC protocols to build in this model, since it is both more efficient than dishonest majority protocols and has a real world implementation.

4.2.1 Related Work

Our work builds upon the existing body of research on concretely efficient threshold ECDSA protocols in the dishonest majority setting. Previous works in this setting can be grouped into several categories:

- Protocols using Paillier's Homomorphic Encryption (HE) with a small number of rounds but high computational cost [75]–[79]. These also require expensive zero-knowledge proofs over Paillier ciphertexts. Optimized variants for the two-party variants also exist (e.g., [80], [81]).

- Replacing HE with class group-based schemes as in [82]–[84], which improves the efficiency of zero-knowledge proofs but not the number of rounds, while introducing different assumptions on class groups of imaginary quadratic fields.
- Oblivious transfer (OT)-based protocols [85], [86], that reduce cryptographic assumptions and computational overhead but increases round complexity.
- Protocols that are based on generic MPC; in particular in such protocols multiplication triplets are pre-processed [87], [88]. These protocols typically increase the overall number of rounds (and hence, the number of writes) and in some cases introduce newer assumptions such as LPN [88].

In contrast to prior work, our protocols are designed to be chain-friendly, by reducing the number of writes without resorting to heavyweight cryptographic tools like HE and expensive ZKP that are likely too inefficient to run in a constraint blockchain environment.

Our protocol also achieves two often overlooked properties for threshold ECDSA: *fairness* and *robustness*. The current state-of-the-art honest majority threshold ECDSA protocol by Damgard et al., [89] achieves fairness in six writes, as opposed to 1-2 writes in our work, and by well-known impossibility results, dishonest majority protocols (without blockchain assistance) cannot hope to achieve fairness at all [74], [90].

As to robustness, since the original work of Gennaro et al., on threshold (EC)DSA for a super-honest majority ($n \geq 4t+1$) more than two decades ago [91], most known efficient protocols in the dishonest majority setting (e.g., [75], [76], [78], [79]) sacrifice robustness for additional efficiency gains. These protocols move from threshold to additive secret sharing as soon as pre-signing starts, leaving no room to handle faults mid-execution. Recently, attempts to partially address robustness have been proposed. Gagol et al. [92] suggested a robust scheme which requires all parties to participate honestly in the pre-signature phase, while others proposed schemes with identifiable aborts instead (e.g., [79] [84]). In a concurrent and independent work, Wong et al., [93] achieve a stronger notion of robustness they call 'self-healing robustness', where as long as the signers in the online-phase are a subset of the signers in the pre-signature phase, their scheme is either robust (for an honest majority) or gracefully falls back to identifiable aborts otherwise. In contrast, in this work we achieve the standard plain notion of robustness, where signers in pre-signing and signing can be disjoint.

For a comprehensive comparison of our work with the existing literature, please refer to Table 4.1. Note that we also describe a scenario unique to our work, where there is a single signing party involved (i.e., $n = 1$). This is an interesting scenario, as it allows a single user to increase their wallet security by having P_c as a co-signer. Similarly, some use cases, like wallet exchange, may make more sense under this setting. However, for this scenario, we describe a modified version of [80] and show that while it is less efficient than our main protocol, it can still run on-chain.

Other works address *generic* MPC with fairness and public verifiability via bulletin boards (that can be implemented with blockchains). Bentov et. al, Kumerasen et. al, and Baum et. al [47], [94]–[96] achieve a revised form called 'fairness with penalties' using gradual release mechanisms and deposits using a blockchain. Choudhuri et al. [74] showed how to use blockchains to achieve the standard notion of fairness (without penalties), by leveraging either witness encryption, which is too expensive in practice, or off-chain TEEs. Baum et al. and Rivinius et al. show how to achieve public verifiablity and robustness using a public bulletin board [97]–[99]. Similarly, a long line of works of MPC-as-a-service systems inspired by blockchains have emerged in recent years [3], [4], [48], [71]–[73], [100]–[104]. While they address how a blockchain can help with the general MPC problem (or how MPC can add confidentiality to blockchains), our work, as far as we know, showcases the first threshold ECDSA protocol that effectively provides both fairness and robustness, by relying on an external blockchain.

Finally, in contrast to all prior works solving threshold ECDSA, we are the first to consider a model for MPC protocols that leverage a special non-colluding semi-honest party. We show how such a model greatly increases efficiency, simplifies protocol design, and achieves properties of interest – in our scenario, fairness and robustness. We further show how this model is realized in practice through the use of a confidential smart contracts, which have garnered significant interest in recent years [4], [5], [7]–[11], [29], [30], [46], [53], [54], [105]–[111].

Protocol	Parties	Writes	Messages	Primitives	Properties
LN18 [76]	n	8	$O(n^2)$	Paillier	
CGGMP20 [79]	n	4	$O(n^2)$	Paillier	IA
DKLS19 [86]	n	log(t) + 6	$O(n^2)$	OT	
BMP22 [112]	n	4	$O(n)$	Paillier	
CCLST20 [83]	n	8	$O(n^2)$	CL-HE	
CGCL+23 [84]	n	7	$O(n^2)$	CL-HE	IA, Fairness (Honest Majority)
WMYC23 [93]	n	5	$O(n^2)$	Paillier	Self-healing
Lindell17 [80]	2	2	$O(1)$	Paillier	
XAXYC21 [81]	2	3	$O(1)$	HE/OT	
CCLST19 [82]	2	3	$O(1)$	CL-HE	
DKLS18 [85]	2	7	$O(1)$	OT	
This work	n	1-2	$O(n)$	Group	Fairness
This work	n	1-2	$O(n^2)$	Group	Robustness
This work	1	1	$O(1)$	Paillier	

Table 4.1: Comparison with related work. For protocols that support pre-signing – the number of writes consists of both pre-sign and sign phase, ignoring amortization.

4.3 Threshold ECDSA Protocol

As explained in Section 5.0.2, current threshold ECDSA protocols require the use of expensive primitives (like HE or OT) and require at the very least four rounds of interactions, which in our model, translate to four consecutive writes to the blockchain. That kind of latency, and more importantly, the implied requirement from each user to sign four transactions in a row in order to produce a signature is too burdensome in practice.

In-line with our goals, we seek to construct a protocol that would be chain-friendly, and would minimize the number of writes each party has to perform. For all parties except one, we achieve the optimal of a single write per-party, which can be done non-interactively. A designated party (the signature initiator) needs to write twice. . The blockchain is modeled as an additional semi-honest and non-colluding party, denoted P_c. This enables us to take a different approach and leverage techniques from honest-majority MPC, even though the adversary may corrupt the majority of the parties $P_1, \ldots, P_n$. We do this by assigning n shares to the parties, and t additional shares are held by P_c, for a total of $N = n + t$ shares. Our protocol ensures that as long as there are $t + 1$ honest signers they will generate a valid signature; otherwise, no information is revealed.

In that sense, our protocol resembles the one by Damgard et al. [89], which is secure in the honest majority setting; however, we make significant changes to their protocol, greatly improving the number of writes and the communication costs. In particular, their protocol requires six writes (or four writes without fairness, which we obtain anyway), which is significantly more than ours.

For readability reasons, in the protocols below we write that P_i sends P_j a message although it is understood that P_i only communicates through P_c. That is, P_i sends a ciphertext to P_c under P_j's encryption key, and then P_j decrypts that message (implicitly implying PKI).

4.3.1 Key Generation

Our key generation protocol (Protocol 4) begins with a standard joint random secret sharing generation protocol having two dealers: P_1 and P_c. Given that the blockchain is semi-honest and non-colluding, we can avoid a more expensive coin-tossing protocol. This is a recurring theme we use in all of our protocols. After both P_1 and P_c deal their shares, each party computes their final share of the secret key $[x]_i$ and sends their share of the public key ($X_i := [x]_i \cdot G$) to P_c. Finally, P_c ensures that all shares of the public key are consistent by interpolating in the exponent. If any of the parties cheated, it aborts, otherwise it sends the generated public key X to all parties, which concludes the protocol successfully.

4.3.2 Signing Protocol

Similarly to key generation, the signature protocol (Protocol 5) begins with a two-dealer random secret-sharing protocol between P_1 and P_c, who jointly generate all required randomness for a single execution. These include t-sharings of fresh random values k, a, and $2t$-sharings of zero, denoted as z, z'. Intuitively, k is the usual ECDSA nonce produced for every signature, and the other values are used internally to mask $2t$-shares that are the product of two t-shares. For concrete efficiency, the protocol does not check consistency of any of these values. In fact, it may even be that the parties hold inconsistent sharings, or that $R \neq [k] \cdot G$. In the proof we show that the adversary cannot learn anything even if it cheats, and so it can only cause an abort.

After the parties obtain these sharings and $r := R.x$, they can locally compute their share of s_1, s_2, such that $[s_1]_i := [a]_i(m + r[x]_i) - [z]_i \mod q$ and $[s_2]_i := [k]_i[a]_i - [z']_i \mod q$. Notice that each s_1 and s_2 has a multiplicative depth of one, meaning that the resulting shares are lifted from a degree t polynomial to a degree $2t$ one. Furthermore, as these shares may no longer be properly random, each party also uses their share of z, z' to rerandomize their resulting shares.

PROTOCOL 4. (*Key-Generation:* KeyGen)

1. **Users' dealing**:
   (a) Party P_1 samples a random $x_u \leftarrow Z_q$.
   (b) Party P_1 computes $[x_u] \leftarrow \mathsf{SS.Share}(x_u, t, N)$.
   (c) Party P_1 sends $[x_u]_i$ to P_i for all $i \in [1, n]$ and $[x_u]_i$ to P_c for all $i \in [n+1, N]$.

2. **Center's dealing**:
   (a) Party P_c samples a random $x_c \leftarrow Z_q$.
   (b) Party P_c computes $[x_c] \leftarrow \mathsf{SS.Share}(x_c, t, N)$, and sends $[x_c]_i$ to P_i for $i \in [1, n]$.

3. **Compute key share**:
   (a) For each $j \in n+1, ..., N$, P_c computes $[x]_j = [x_u]_j + [x_c]_j \mod q$ and $X_j \leftarrow [x]_j \cdot G$.
   (b) Each party P_i ($i \in [1, n]$) computes $[x]_i = [x_u]_i + [x_c]_i \mod q$ and $X_i = [x]_i \cdot G$.
   (c) Each party P_i ($i \in [1, n]$) sends X_i to P_c.

4. **Public key**:
   (a) Let P be the polynomial defined by the $t + 1$ points $(n, [x]_n), (n + 1, [x]_{n+1}) \ldots, (N, [x]_N)$, and let $\lambda_n^j, \lambda_{n+1}^j, \ldots, \lambda_N^j$ be the Lagrange coefficients s.t. $P(j) = \sum_{k=n}^{N} \lambda_k^j \cdot [x]_k$.
   (b) Party P_c verifies that the keys are consistent: For every $j \in [1, n-1]$ compute $X'_j = P(j) \cdot G = \sum_{k=n}^{N} \lambda_k^j \cdot X_k$, then, abort if $X'_j \neq X_j$.
   (c) Othrewise (if all key shares are consistent) P_c broadcasts the public key $X = P(0) \cdot G = \sum_{k=n}^{N} \lambda_k^0 \cdot X_k$.

Finally, each party sends $([s_1]_i, [s_2]_i)$ to P_c. After receiving $t + 1$ shares, P_c can itself generate additional t shares of these values, and having $2t+1$ total shares of each, reconstruct s_1, s_2 to obtain the final $s := s_1 \cdot s_2^{-1} \mod q$. Finally, if (r, s) is a valid signature, P_c sends it to all parties.

It should be clear that the protocol takes only a single write (for producing the signature) by each party. The only exception is the dealer P_1, who needs to write twice (and can be pre-processed).

Fairness. Our protocol provides fairness, since we make sure that the first party to see a valid signature is P_c, which we know follows the protocol. Therefore, if P_c releases the signature to others, then we know it is indeed a valid signature. Another benefit of our construction is that P_c, a de-facto smart contract, can use incentives (e.g., penalties) to encourage parties to behave correctly and participate in the protocol [95], [96].

We prove the following theorem in Section 4.6.1.

Theorem 4.3.1. *Protocols 4-5 securely compute the ECDSA functionality (Functionality 1) with perfect security with abort, against a static malicious adversary who corrupts at most t parties (which are the majority) of $\{P_1, \ldots, P_n\}$ or a semi-honest adversary who corrupts P_c.*

Security follows since we can perfectly simulate the adversary's view by picking random values for its shares. One challenge is to align all parties' shares (those of the adversary as well as those of the honest parties) with the values obtained in from the ECDSA functionality (like the public key X, the random nonce R and the signature s), in which case we first make sure that the adversary's shares are consistent with those values, and then 'interpolate' the other parties' shares to reside on the same, fully determined, polynomial. Another challenge is that P_u picks a secret and shares it first (before this is done by P_c), however, when simulating P_c we need to know P_c's secret (be it x_c in the key generation protocol or k_c in the signing protocol) before simulating P_u's dealing. To this end, in the protocol we instruct P_c to derive its secret from $\mathcal{H}$, which is modeled as a random oracle that is programmable by the simulator. Interestingly, since P_c is semi-honest (and follows the protocol) we can program the random oracle apriori. That is, we can choose the secret values x_c and k_c on behalf of P_c even before it queried the random oracle for them. This was not possible if P_c is malicious, since P_c could have query the random oracle multiple times (or not at all), and the simulator could not know which one was the right one.

From ROM to the standard model. We stress that the protocol can be described in a way that is secure in the standard model, without the random oracle, by having P_c commit to a PRF key as a first step in the key generation protocol, and then this PRF can be used as a random oracle. The simulator extracts that PRF key, as it takes the role of the commitment functionality, and can reproduce any value that P_c produces during the protocol.

4.4 Robust Threshold ECDSA

Note that Protocols 4 and 5 are fair, but not robust. They are fair because either all or none of the parties $P_1, \ldots, P_n$ obtain the result verification key X and signatures. However, robustness is not guaranteed, that is, if P_1 cheats in its dealing then the protocols abort and the parties will not learn the public key or signatures. We can overcome that by using a publicly verifiable secret sharing (cf. Section 2.2.1) in two different approaches: (1) Let P_1 be the only dealer (apart from P_c) as before, and if it cheats, repeat with P_2 as the dealer, and so on. This process will end by at most $t+1$ writes, as at least one of $P_1, \ldots, P_{t+1}$ is honest; (2) Let all $P_1, \ldots, P_{t+1}$ be dealers simultaneously which ensures that by one write this dealing is complete. While optimistically the first approach entails only one party to write to the blockchain, and hence the overall protocol's message complexity is $O(n)$ (i.e., we consider P_i sending a share to P_j as one message), in the worst case there are $O(t)$ rounds and $O(n^2)$ messages. In the second approach there is still $O(n^2)$ messages, but they are all happen in parallel and so this approach is completed in one round. Protocols 6 and 7 follow the second approach.

Note that ensuring correctness of sharing is not sufficient for robustness - one has to make sure that the computation of $s_1 = a(m + rx)$ and $s_2 = ka$ of the partial signatures by each party are computed correctly. Since these values are the result of a non-linear function, they could not be verified against existing values, m, r, A, K and X, that are already public. To this end, the parties provide additional auxiliary information M_1 and M_2, such that $M_1 = \log(A) \cdot \log(X) \cdot G$ and $M_2 = \log(A) \cdot \log(K) \cdot G$, then, everyone can check that s_1 and s_2 are computed correctly by verifying the equalities $s_1 \cdot G = r \cdot M_1 + m \cdot A$ and $s_2 \cdot G = M_2$. The last piece is verifying that M_1 and M_2 are indeed computed correctly. This can be done by having the parties provide a simple zero-knowledge proof that (A, X, M_1) and (A, K, M_2) are Diffie-Helman tuples (DHT), where the DHT relation is defined by

$$R_{\mathsf{DHT}} = \{(A, B, C) \text{ s.t. } a = \log(A), b = \log(B), ab = \log(C)\}.$$

Note that we use PVSS for the computation of P_c even though it is not needed as P_c is semi-honest, we do this as the interface already gives us the public values required for the messages of parties $1, \ldots, n$ to be publicly verified.

We prove the following in Section 4.6.2.

Theorem 4.4.1. *Assuming the decisional Diffie-Helman (DDH) problem is hard relative to $(\mathbb{G}, G, q)$, Protocols 6 and 7 securely compute the ECDSA functionality (Functionality 1) with guaranteed outupt delivery, against a static malicious adversary who corrupts at most t parties (which is the majority of) of $\{P_1, \ldots, P_n\}$ or a semi-honest adversary who corrupts P_c.*

In addition to the challenges aforementioned above for the non-robust protocol, which we solve in the same way here, simulating the robust protocol introduces a new challenge because the use of Shoenmakers's PVSS scheme, which involves El-Gamal encryptions. This extra challenge is introduced only when P_c is corrupted, since when it is not (and we are in the first case in which a subset of $P_1, \ldots, P_n$ are corrupted, and so the simulator simulates message arriving from P_c) the simulator has to simulate only P_c's messages, which are not publicly verifiable, but are guaranteed to be correct due to the fact that P_c behaves honestly, thus, there is no need to simulate encryptions of unknown plaintexts. In contrast, when P_c is corrupted, we need to simulate publicly verifiable messages from parties $P_1, \ldots, P_{t+1}$, let's focus on one of them, P_u. Then, in the key generation, the simulator knows the public key X (as received from the ECDSA functionality) as well as the complementary part of the public key X_c (which is extracted by the technique described above), therefore the simulator knows $X_u = X - X_c$. However, for a perfect simulation the simulator has to share $x_u = \log(X_u)$ using the PVSS scheme. Now, in contrast to the non-publicly verifiable secret sharing in which each receiver receives its own share only, in PVSS the dealer has to broadcast the encryptions of *all* shares under their respective key, and prove that they are consistent with the commitment of the polynomial. In our case, the simulator does not know x_u and so it cannot produce a polynomial P s.t. $P(0) = x_u$. Instead of providing encryptions of the shares $P(1), \ldots, P(N)$, which are obviously unknown to the simulator, the simulator picks random shares $[x_u]_{n+1}, \ldots, [x_u]_N$ intended for P_c and encrypts those correctly. Then, the simulator produces the commitment to the polynomial $A_0, \ldots, A_t$, where $A_0 = X_u$ since the polynomial must evaluate to x_u at 0, and the values $A_1, \ldots, A_t$ are computed from the linear system with

t equations and t variables, where the j-th equation is $\sum_{j=0}^{t} i^j \cdot A_j = [x_u]_i \cdot G$. By solving that system the simulator obtains $A_1, \ldots, A_t$ and so it has all information required to make all P_c's values be consistent with X and X_c. Finally, for the encryptions of parties $P_1, \ldots, P_n$, that are also sent to P_c, the simulator simply encrypts the value $0 \in Z_q$, which is indistinguishable from an encryption of the actual value $P(i)$ that should have been encrypted, from the CPA-security of El-Gamal.

4.5 A Solution for a Single User

So far the chain-assisted protocols were designed to support a *group of signers*, but are not extended to the case in which there is *only one signer*. To see this, observe that for the smallest possible threshold $t = 1$, we need at least two parties that are not P_c. We therefore need to utilize a different protocol between the user and P_c directly. This reduces to a two-party ECDSA protocol between a user P_u and P_c. One of the current state of the art protocols for two-party ECDSA is that of Lindell's [80]. Luckily, when taking into account that our model allows for one of the parties to be semi-honest, we can gain some performance improvements for this setting as well, discussed shortly.

First note that the functionality is a bit different than a typical 2PC ECDSA: since P_c is only an assistant, party P_u is the only one who can ask for key generation or signatures. The formal description appears in Functionality 2. Second, note that we employ the same technique for extracting P_c's secret inputs x_c, k_c as done in the multiparty protocols above. As explained, however, this technique can be replaced with a standard model technique using a commitment on a PRF key. Third, since P_c is semi-honest in our model, and so it is guaranteed to choose its nonce randomly and independently of P_u's message, which is not the case in Lindell's protocol. This way, in our model the two-party protocol enjoys *non-interactive signing*, or in other words, requires only one write. As briefly discussed below, that fact also enables simulation of both parties without the additional non-standard 'Paillier-EC' assumption that is used in [80]. The reason for that is that we assign P_c the role of the party who performs the linear evaluation on the encryption of P_u's secret key share (c_{key}). Now, since P_c follows the protocol's description, it is guaranteed to not cheat and produce an encryption of $(k_c)^{-1}(m + xr)$ exactly as described. This removes the need of (1) guessing whether P_c will abort or not, (2) adding an expensive zero-knowledge proof on P_c's last message, or (3) relying on a non-standard assumption as Paillier-EC. Except of the changes mentioned above, our protocol resembles that of Lindell. See Section 2.4.2 for a formal description of the Paillier encryption scheme.

We prove the following theorem in Section 4.6.3.

Theorem 4.5.1. *Protocols 8 and 9 securely compute the ECDSA functionality (Functionality 2) against a static malicious adversary who corrupts P_u or a semi-honest adversary who corrupts P_c.*

4.6 Security Proofs

4.6.1 Proof of Theorem 4.3.1

The proof below is separated to the two cases mentioned in the Theorem, for each of which we present a perfect simulation. Note that the use of $\mathcal{H}$ in the protocol is merely to easily extract P_c's randomly chosen x_c. It is possible to remove this random oracle usage by standard commitment techniques. In both cases it is easy to see that the joint distributions of the honest parties' output and the adversary's view in the real and ideal worlds are identically distributed.

Case 1. Let $\mathcal{A}$ be a malicious real world adversary who corrupts P_1 and a subset of $\{P_2, \ldots, P_n\}$ of size $t-1$. Denote the set of corrupted parties by C and the rest of the parties by $H = \{P_1, \ldots, P_n\} - C$. We present an ideal world adversary $\mathcal{S}$ that does as follows.

- **Key Generation.**
  1. Send (keygen) to $\mathcal{F}_{\mathrm{ECDSA}}$.
  2. Run $\mathcal{A}$ internally and simulates all other parties:
     (a) Receive all shares $[x_u]_i$ for all $i \in H \cup [n+1, N]$.
     (b) Reconstruct x_u using the above $|H| + t$ shares.
     (c) If reconstruction fails then send $(\mathsf{keygen}, \mathsf{abort})$ to $\mathcal{F}_{\mathrm{ECDSA}}$ and halt. Otherwise, compute $[x_u]_i$ for all $P_i \in C$.
     (d) Compute $[x_c] \leftarrow \mathsf{SS.Share}(x_c, t, N)$ for a random $x_c \leftarrow Z_q$, and send $[x_c]_i$ to every party $P_i \in C$.
     (e) Compute secret key shares $[x]_i = [x_u]_i + [x_c]_i \mod q$ for all $i \in [1, N]$. (Note that this x is not the actual secret key $\log_G(X)$ obtained by the functionality, however, the simulator uses it in order to checks whether the adversary cheats when computing the signature.)
     (f) Receive X_i for all $i \in C$ and compute $X_i = [x]_i \cdot G = ([x_u]_i + [x_c]_i) \cdot G$ for all $i \in H \cup [n+1, N]$.
     (g) Check consistency of all X_i as done in the protocol, if the check fails then send $(\mathsf{keygen}, \mathsf{abort})$ to $\mathcal{F}_{\mathrm{ECDSA}}$ and halt.
     (h) Send $(\mathsf{keygen}, \mathsf{continue})$ to $\mathcal{F}_{\mathrm{ECDSA}}$ and obtain X and H_q.
     (i) Broadcasts X and H_q.
     (j) Output whatever $\mathcal{A}$ outputs.

- **Sign.**
  1. Send $(\mathsf{sign}, \mathsf{sid})$ to $\mathcal{F}_{\mathrm{ECDSA}}$ and obtain R.
  2. Run $\mathcal{A}$ internally and simulates all other parties:
     (a) Receive all shares $[k_u]_i$ and $[a_u]_i$ for all $i \in H \cup [n+1, N]$.

(b) Receive all shares $[z_u]_i$ and $[z'_u]_i$ for all $i \in H \cup [n+1, N]$.

(c) Receive R_u.

(d) Sample $k_c, a_c \leftarrow Z_q$ and compute $[k_c] \leftarrow \mathsf{SS.Share}(k_c, t, N)$, $[a_c] \leftarrow \mathsf{SS.Share}(a_c, t, N)$, $[z_c] \leftarrow \mathsf{SS.Share}(0, 2t, N)$ and $[z'_c] \leftarrow \mathsf{SS.Share}(0, 2t, N)$

(e) Send $[k_c]_i, [a_c]_i, [z_c]_i, [z'_c]_i$ to P_i for all $i \in C$.

(f) Compute $[k]_i = [k_u]_i + [k_c] \mod q$ and $[a]_i = [a_u]_i + [a_c] \mod q$ for all $i \in H \cup [n+1, N]$.

(g) Send R to all $P_i \in C$.

(h) Receive $[s_1]_i$ and $[s_2]_i$ from all $P_i \in C$.

(i) Compute $[s_1]_i$ and $[s_2]_i$ using values r, m and the shares $[k]_i, [a]_i, [x]_i$ for all $P_i \in H \cup [n+1, N]$.

(j) Reconstruct s_1 and s_2 using the the shares received from the adversary (for parties in C) and the shares computed above (for the parties in $H \cup [n+1, N]$). If reconstruction (of a $2t$-degree polynomial) failed then send $(\mathsf{sign}, \mathsf{sid}, \mathsf{abort})$ and halt.

(k) Check whether r and $s = s_1 \cdot s_2^{-1} \mod q$ is a valid signature on M using the secret key x that was computed in the key-generation phase (recall, this is not the actual secret key used by the functionality).

(l) If the check fails then send $(\mathsf{sign}, \mathsf{sid}, \mathsf{abort})$ and halt.

(m) Send $(\mathsf{sign}, \mathsf{sid}, \mathsf{continue})$ and obtain (r, s). Broadcast (r, s) and output whatever $\mathcal{A}$ outputs.

Case 2. Let $\mathcal{A}$ be a semi-honest real world adversary who corrupts P_c. We present an ideal world adversary $\mathcal{S}$ that does as follows:

- **Key Generation.**

  1. Send (keygen) and $(\mathsf{keygen}, \mathsf{continue})$ to $\mathcal{F}_{\text{ECDSA}}$, and obtain X.
  2. Run $\mathcal{A}$ internally and simulate parties $(P_1, \ldots, P_n)$:

     (a) Sample $x_u \leftarrow Z_q$, compute $[x_u] \leftarrow \mathsf{SS.Share}(x_u, t, N)$ and send $[x_u]_i$ to P_c, for all $i \in [n+1, N]$.

     (b) Receive $[x_c]_i$ from P_c for all $i \in [1, n]$, reconstruct x_c (always succeeds because $\mathcal{A}$ follows the protocl) and compute $[x_c]_i$ for all $i \in [n+1, N]$.

     (c) Let λ_0^j and $\{\lambda_i^j\}_{i \in [n+1,N]}$ be the Lagrange coefficients for a polynomial evaluation on j, using points at 0 and the indices in $[n+1, N]$ ($t+1$ points in total).

     (d) For every $j \in [1, n]$ compute $X_j = \lambda_0^j \cdot X + \sum_{i \in [n+1,N]} \lambda_i^j \cdot X_i$.

     (e) Send X_j to P_c for every $i \in [1, n]$. (The above computation ensures that the consistency verification goes through.)

     (f) Output whatever $\mathcal{A}$ outputs.

- **Sign.**

  1. Send $(\mathsf{sign}, \mathsf{sid})$ and $(\mathsf{sign}, \mathsf{sid}, \mathsf{continue})$ to $\mathcal{F}_{\text{ECDSA}}$ and obtain R and (r, s).

2. Run $\mathcal{A}$ internally and simulates all other parties:
   (a) Sample $k_u, a_u \leftarrow Z_q$ and compute $[k_u] \leftarrow \mathsf{SS.Share}(k_u, t, N)$, $[a_u] \leftarrow \mathsf{SS.Share}(a_u, t, N)$, $[z_u] \leftarrow \mathsf{SS.Share}(0, 2t, N)$ and $[z'_u] \leftarrow \mathsf{SS.Share}(0, 2t, N)$
   (b) Send $[k_u]_i, [a_u]_i, [z_u]_i, [z'_u]_i$ to P_c for all $i \in [n+1, N]$.
   (c) Sample $k_c \leftarrow Z_q$ and program $\mathcal{H}(x_c \| \mathsf{sid}) \leftarrow k_c$.
   (d) Compute $R_c = k_c \cdot G$ and $R_u = R - R_c$.
   (e) Send R_u to P_c.
   (f) Receive all shares $[k_c]_i$ and $[a_c]_i$ for all $i \in [1, n]$.
   (g) Receive all shares $[z_c]_i$ and $[z'_c]_i$ for all $i \in [1, n]]$.
   (h) Receive R.
   (i) Compute $[\alpha]_i = [\alpha_u]_i + [\alpha_c]_i \mod q$ for all $i \in [n+1, N]$ and for all $\alpha \in \{k, a, z, z'\}$.
   (j) Compute $[s_1]_i = [a]_i(m + r[x]_i) - [z]_i \mod q$ and $[s_2]_i = [k]_i[a]_i - [z']_i \mod q$ for all $i \in [n+1, N]$.
   (k) Sample random $2t$-degree polynomials S_1 and S_2, such that $S_b(0) = s_b$ and $S_b(i) = [s_b]_i$, for all $i \in [n+1, N]$ and $b \in \{1, 2\}$.
   (l) Send $(m, [s_1]_i, [s_2]_i)$ to P_c for all $i \in [1, n]$, where $[s_1]_i = S_1(i)$ and $[s_2]_i = S_2(i)$.

4.6.2 Proof of Theorem 4.4.1

The proof below is separated to the two cases mentioned in the Theorem, for each of which we present a perfect simulation. As mentioned above, we use $\mathcal{H}$ as a random oracle in order to easily extract P_c's randomly chosen x_c, but it is possible to replace it with standard commitment techniques.

Case 1. Let $\mathcal{A}$ be a malicious real world adversary who corrupts P_1 and a subset of $\{P_2, \ldots, P_n\}$ of size $t-1$. Without loss of generality, let that subset be $P_1, \ldots, P_t$. We present an ideal world adversary $\mathcal{S}$ that does as follows.

- **Key Generation.**

1. Send (keygen) to $\mathcal{F}_{\text{ECDSA}}$, then send $(\mathsf{keygen}, \mathsf{continue})$ to $\mathcal{F}_{\text{ECDSA}}$ and obtain X and H_q.
2. Run $\mathcal{A}$ internally and simulates all other parties (knowing their encryption key-pair, so it is possible to decrypt ciphertexts under their key):
   (a) Choose $x_{t+1} \leftarrow Z_q$, and send

$$(\{c^{\ell}_{t+1}\}_{i=1}^{N}, \{A^{t+1}_j\}_{j=0}^{t}, \pi^{t+1}) \leftarrow \mathsf{PVSS.Share}_{t,N}(x_{t+1}),$$

   to the adversary.
   (b) Receive $(\{c^{\ell}_{\ell}\}_{i=1}^{N}, \{A^{\ell}_j\}_{j=0}^{t}, \pi^{\ell})$ from the adversary for all $\ell \in [1, t]$.

(c) Let $u \in [1, t+1]$ be the first index for which

$$1 = \mathsf{PVSS.CheckDealer}(\{c_i^u\}_{i=1}^N, \{A_j^u\}_{j=0}^t, \pi^u).$$

Denote these values by $\{c_i\}_{i=1}^N, \{A_j\}_{j=0}^t$ (i.e., dropping the supertext u). Note that there must be such u, as the above certainly holds for $u = t+1$ (as this is the honest party simulated here.

(d) Extract the secret x_u by decrypting c_i for $t+1$ parties (which is possible because there are at least $t+1$ parties under the control of the simulator). Note that this also enables obtaining $\log(A_j)$ for all $j \in [0, t]$ sent by P_u.

(e) Compute $[x_c] \leftarrow \mathsf{SS.Share}(x_c, t, N)$ for a random $x_c \leftarrow Z_q$.

(f) Send $[x_c]_i$ to the adversary for every $i \in [1, t]$.

(g) Set $X_0 = X$ and compute $X_i = ([x_u]_i + [x_c]_i) \cdot G$ for every $i \in [1, t]$. Then compute $X_i = \sum_{j=0}^t i^j \cdot X_j$ for every $i \in [t+1, n]$.

(h) Broadcast X and X_i for every $i \in [1, n]$.

(i) Output whatever $\mathcal{A}$ outputs.

- **Sign.**

1. Send $(\mathsf{sign}, \mathsf{sid})$ to $\mathcal{F}_{\mathrm{ECDSA}}$ and obtain R, then send $(\mathsf{sign}, \mathsf{sid}, \mathsf{continue})$ and obtain (r, s).
2. Run $\mathcal{A}$ internally and simulates all other parties:

(a) Choose $k_{t+1}, a_{t+1} \leftarrow Z_q$, and send to the adversary

$$(\{c_{k,i}^{t+1}\}_{i=1}^N, \{K_j^{t+1}\}_{j=0}^t, \pi_k^{t+1}) \leftarrow \mathsf{PVSS.Share}_{t,N}(k_{t+1}),$$
$$(\{c_{a,i}^{t+1}\}_{i=1}^N, \{A_j^{t+1}\}_{j=0}^t, \pi_a^{t+1}) \leftarrow \mathsf{PVSS.Share}_{t,N}(a_{t+1}),$$
$$(\{c_{z,i}^{t+1}\}_{i=1}^N, \{Z_j^{t+1}\}_{j=0}^t, \pi_z^{t+1}) \leftarrow \mathsf{PVSS.Share}_{2t,N}(0),$$
$$(\{c_{z',i}^{t+1}\}_{i=1}^N, \{Z'^{t+1}_j\}_{j=0}^t, \pi_{z'}^{t+1}) \leftarrow \mathsf{PVSS.Share}_{2t,N}(0).$$

(b) For every $i \in [1, t]$, receive from the adversary

$$(\{c_{k,i}^i\}_{i=1}^N, \{K_j^i\}_{j=0}^t, \pi_k^i) \leftarrow \mathsf{PVSS.Share}_{t,N}(k_i),$$
$$(\{c_{a,i}^i\}_{i=1}^N, \{A_j^i\}_{j=0}^t, \pi_a^i) \leftarrow \mathsf{PVSS.Share}_{t,N}(a_i),$$
$$(\{c_{z,i}^i\}_{i=1}^N, \{Z_j^i\}_{j=0}^{2t}, \pi_z^i) \leftarrow \mathsf{PVSS.Share}_{2t,N}(0),$$
$$(\{c_{z',i}^i\}_{i=1}^N, \{Z'^i_j\}_{j=0}^{2t}, \pi_{z'}^i) \leftarrow \mathsf{PVSS.Share}_{2t,N}(0).$$

(c) Let $u \in [1, t+1]$ be the first index for which all sharings above are verified.

(d) Denote the public values of P_u by $\{K_j, A_j\}_{j=0}^t$ and $\{Z_j, Z'_j\}_{j=0}^{2t}$.

(e) Extract the values k_u, a_u and z_u, z'_u (the values z_u and z'_u are extractable via the zero knowledge functionality).

(f) Generate the sharings $[k_c], [a_c], [z_c]$ and $[z'_c]$ as in the protocol, and send the adversary $\{[k_c]_i, [a_c]_i, [z_c]_i, [z'_c]_i\}$ for every $i \in [1, t]$.

(g) Broadcast R (as received from the ECDSA functionality).

(h) Set $K_0 = R$ and compute $K_i = ([k_u]_i + [k_c]_i) \cdot G$ for every $i \in [1, t]$. Then compute $K_i = \sum_{j=0}^{t} i^j \cdot K_j$ for every $i \in [t+1, n]$.

(i) Compute $A_i = ([a_u]_i + [a_c]_i) \cdot G$, $Z_i = ([z_u]_i + [z_c]_i) \cdot G$ and $Z'_i = ([z'_u]_i + [z'_c]_i) \cdot G$ for every $i \in [1, n]$.

(j) Broadcast (K_i, A_i, Z_i, Z'_i) for every $i \in [1, n]$.

(k) Send $(\mathsf{proof}, \mathsf{sid}\|1, A_{t+1}, X_{t+1}, M_{t+1,1})$ and $(\mathsf{proof}, \mathsf{sid}\|2, A_{t+1}, K_{t+1}, M_{t+1,1})$ to the adversary, in addition, receive and verify the adversary's proof on its $M_{i,1}, M_{i,2}$ for every $i \in [1, t]$.

(l) When received $t + 1$ messages $([s_1]_i, [s_2]_i, M_{i,1}, M_{i,2})$ for i for which the proof is verified, broadcast the signature (r, s) as received from the ECDSA functionality.

(m) Output whatever $\mathcal{A}$ outputs.

First note that the honest parties's output are identically distributed in both real and ideal world. We now argue that the adversary's views in both world are computationally indistinguishable. The only difference between the views is that in the simulation the values X_i and K_i for $i \in [t+1, n]$ that are observed by the adversary (since P_c broadcasts them) are not computed correctly by $([x_u]_i + [x_c]_i) \cdot G$ and $([k_u]_i + [k_c]_i) \cdot G$; rather, they are computed (interpolated) directly from the values $X_0, \ldots, X_t$ and $K_0, \ldots, K_t$ (if they were not interpolated this way then it would have been easy to detect this). Now, since the adversary does not have any information about $([x_u]_i + [x_c]_i)$ or $([k_u]_i + [k_c]_i)$ it cannot tell the difference and so the views are identically distributed.

Case 2. Let $\mathcal{A}$ be a semi-honest real world adversary who corrupts P_c. We present an ideal world adversary $\mathcal{S}$ that does as follows:

- **Key Generation.**

1. Send (keygen) and $(\mathsf{keygen}, \mathsf{continue})$ to $\mathcal{F}_{\text{ECDSA}}$, and obtain X.
2. Run $\mathcal{A}$ internally and simulate parties $(P_1, \ldots, P_n)$:

(a) Choose $x_c \leftarrow Z_q$ (on behalf of P_c).

(b) Compute $X_u = X - x_c \cdot G$.

(c) Choose random values $[x_u]_i \leftarrow Z_q$ and compute $c_i \leftarrow \mathsf{EG.Enc}_{\mathsf{ek}_i}([x_u]_i)$ for $i \in [n+1, N]$; and $c_i \leftarrow\leftarrow \mathsf{EG.Enc}_{\mathsf{ek}_i}(1)$ for every other $i \in [1, n]$. Finally compute $A_1, \ldots, A_t$ such that $\sum_{j=0}^{t} i^j A_j = [x_u]_i \cdot G$ for every $i \in [n+1, N]$ (this is a linear system of t equations with t variables).

(d) Broadcast $\{c_i\}_{i=1}^{N}$, $\{A_j\}_{j=0}^{t}$, and π, where π is generated by the HVZK simulator associated with the zero-knowledge proof.

(e) Receive a call to $\mathcal{H}$ from the adversary and respond with x_c chosen above.

(f) Receive $[x_c]_i$ from the adversary for every $i \in [1, n]$.

(g) Receive X and X_i for every $i \in [1, n]$.

(h) Output whatever the adversary outputs.

- **Sign.**

1. Send $(\mathsf{sign}, \mathsf{sid})$ and $(\mathsf{sign}, \mathsf{sid}, \mathsf{continue})$ to $\mathcal{F}_{\mathrm{ECDSA}}$ and obtain R and (r, s).
2. Run $\mathcal{A}$ internally and simulates all other parties:
   (a) Choose $k_c \leftarrow Z_q$ (on behalf of P_c).
   (b) Compute $R_u = R - k_c \cdot G$.
   (c) Choose random values $[k_u]_i \leftarrow Z_q$ and compute $c_i \leftarrow \mathsf{EG.Enc}_{\mathsf{ek}_i}([k_u]_i)$ for $i \in [n+1, N]$; and $c_i \leftarrow\leftarrow \mathsf{EG.Enc}_{\mathsf{ek}_i}(1)$ for every other $i \in [1, n]$. Finally compute $K_1, \ldots, K_t$ such that $\sum_{j=0}^{t} i^j K_j = [k_u]_i \cdot G$ for every $i \in [n+1, N]$ (this is a linear system of t equations with t variables).
   (d) Broadcast $\{c_{k,i}\}_{i=1}^{N}$, $\{K_j\}_{j=0}^{t}$, and π_k, where π_k is generated by the HVZK simulator associated with the zero-knowledge proof.
   (e) Choose random $a_u \leftarrow Z_q$ and compute

$$(\{c_{a,i}\}_{i=1}^{N}, \{A_j\}_{j=0}^{t}, \pi_a) \leftarrow \mathsf{PVSS.Share}_{t,N}(a_u),$$
$$(\{c_{z,i}\}_{i=1}^{N}, \{Z_j\}_{j=0}^{t}, \pi_z) \leftarrow \mathsf{PVSS.Share}_{2t,N}(0),$$
$$(\{c_{z',i}\}_{i=1}^{N}, \{Z'u_j\}_{j=0}^{t}, \pi_{z'}) \leftarrow \mathsf{PVSS.Share}_{2t,N}(0).$$

   (f) Broadcast the PVSS results above.
   (g) Receive a call to $\mathcal{H}$ from the adversary and respond with k_c chosen above.
   (h) Receive $([k_c]_i, [a_c]_i, [z_c]_i, [z'_c]_i)$ from P_i for $i \in [1, n]$, and extract a_c (z_c and z'_c could not be extracted since they are shared using a sharing of degree $2t$).
   (i) Receive K and (K_i, A_i, Z_i, Z'_i) for all $i \in [1, n]$.
   (j) At this point the simulator knows the values $[s_1]_i, [s_2]_i$ for every $i \in [n+1, N]$ that are computed by the adversary in the local computation step.
   (k) The simulator generates random sharings of degree $2t$ for random values s_1, s_2 such that: (1) the shares at points $i \in [n+1, N]$ are those computed by the adversary; (2) it holds that $s_1 \cdot s_2^{-1} = s$ and s is the value received from the ECDSA functionality.
   (l) The simulator also compute the values $M_{i,1}, M_{i,2}$ according to the constraints implied in the protocol. Note that these values will not meet the constraints required by the zero-knowledge proof, however, the proof will be successfully verified since it is simulated using the HVZK simulator associated with it.
   (m) The simulator sends $[s_1]_i, [s_2]_i, M_{i,1}, M_{i,2}$ to the adversary for all $i \in [1, n]$.
   (n) Receive s from the adversary and output whatever it outputs.

Note that here the view of the adversary under the simulation is identical to its view in the real world, except the fact that the ciphertext that are published under the encryption keys of parties $P_1, \ldots, P_n$ are incorrect, that is, they encrypt 0 instead of the actual value. That value that should have been encrypted is unknown to the simulator and hence could not be used. This however is computationally indistinguishable by the adversary and hence it will proceed with the protocol exactly as it would have proceed if these ciphertext were encrypting the correct messages, as otherwise we could have used that adversary in order to break the CPA-security of El-Gamal (which relies on the DDH assumption).

4.6.3 Proof of Theorem 4.5.1

The two-party $\mathcal{F}_{\text{ECDSA}}$ is slightly different than the one presented in Functionality 1. For the two-party, the functionality works only with P_u, P_c and an adversary $\mathcal{S}$, *who cannot abort the execution* (but is mentioned in the functionality solely to emphasize this). This is possible because the first (and only) message sent in the protocol from P_u to P_c fully determines whether the adversary will abort or not (by verifying the zero-knowledge proofs), and if so, the honest party refuses to participate. In the ideal world, such refusal is expressed by not invoking $\mathcal{F}_{\text{ECDSA}}$ at all. Finally, since this case could not be translated to a honest majority protocol we could not achieve fairness, and only P_u obtains the result signature from the functionality. For completeness, the modified version is presented in Functionality 2.

We separately present a simulator to the case of malicious P_u and semi-honest P_c.

Case 1. Let $\mathcal{A}$ be a malicious real world adversary who corrupts P_u, consider an ideal world adversary $\mathcal{S}$ that does as follows:

- **Key Generation.**
  1. Run $\mathcal{A}$ internally and simulate the honest party P_c:
     (a) Receive (X_u, pk, c_{key}) and $(\mathsf{prove}, c_{key}, pk, X_u, x_u, P, Q)$ from P_u, set $sk = (P-1)(Q-1)$ and verify that (1) $X_u = x_u \cdot G$, (2) P, Q are primes of length κ', (3) $N = PQ$, (4) $x_u = \mathsf{Dec}(sk, c_{key})$. If verification fails then halt, otherwise continue.
     (b) Send (keygen) to $\mathcal{F}_{\text{ECDSA}}$ and receive X.
     (c) Compute $X_c = (x_u)^{-1} \cdot X_u$ and send X to $\mathcal{A}$.
     (d) Output whatever $\mathcal{A}$ outputs.

- **Sign.**
  1. Run $\mathcal{A}$ internally and simulate the honest party P_c:
     (a) Receive R_u and $(\mathsf{prove}, \mathsf{sid}, R_u, k_u)$ from P_u, verify that $R_u = k_u \cdot G$. If verification fails then halt, otherwise continue.
     (b) Send $(\mathsf{sign}, \mathsf{sid}, M)$ to $\mathcal{F}_{\text{ECDSA}}$ and receive R and (r, s).
     (c) Choose $\rho \leftarrow Z_{q^2}$ and $\tilde{r} \leftarrow Z_N^*$, and compute $c_2 = \mathsf{Enc}(pk, \rho q + [k_u \cdot s \mod q])$, where s is the signature received from $\mathcal{F}_{\text{ECDSA}}$.
     (d) Send c_2 to $\mathcal{A}$ and output whatever $\mathcal{A}$ outputs.

Observe that the view of P_u under simulation and in the real execution are identically distributed, except of the value c_2: in the simulation it is an encryption of $z'_1 = \rho q + [k_u \cdot s \mod q]$ whereas in the real execution it is an encryption of $z'_2 = \rho q + [(k_c)^{-1} m \mod q] + [(k_c)^{-1} r x_c \mod q] \cdot x_u$, where ρ is a random value from $\{0, \ldots, q^2 - 1\}$. Denote by z_1, z_2 the values wihtout the addition of a random multiple of q, that is, $z_1 = k_u \cdot s \mod q$ and $z_2 = [(k_c)^{-1} m \mod q] + [(k_c)^{-1} r x_c \mod q] \cdot x_u$. Note that we consider z_1 and z_2 over the integers, rather than over Z_q. In [80] the values z'_1 and z'_2 are shown to be statistically close (as long as all conditions on X_u, pk and c_{key} are met, which is guaranteed by using an ideal functionality for zero-knowledge). We present this analysis here for completeness.

Consider the real world value z_2, it is an integer result of the addition of an element from Z_q (namely $(k_c)^{-1}m \mod q$) with the product of of two elements from Z_q (namely $[(k_c)^{-1}rx_c \mod q] \cdot x_u$), and we know that by reducing that integer modulo q we get $k_u \cdot s \mod q$ (where (r, s) the ECDSA signature on M obtained by the functionality), thus there exists some $\ell \in N$ such that $[k_u \cdot s \mod q] + \ell \cdot q = z_2$. Also, note that $0 \leq \ell < q$ since $z_2 < q(q-1)$, so the difference between the simulation and the real world is:

- Real: ciphertext c_2 encrypts $z_2' = [k_u \cdot s \mod q] + \ell \cdot q + \rho \cdot q$, and
- Simulation: ciphertext c_2 encrypts $z_1' = [k_u \cdot s \mod q] + \rho \cdot q$.

We show that with a random choice of $\rho \in Z_{q^2}$ the values z_1' and z_2' are statistically close. Fix k_u and s, then for every $0 \leq \zeta < q$ define $v = [k_u \cdot s \mod q] + \zeta \cdot q$, we have:

- If $0 \leq \zeta < \ell$ then $\Pr[z_1' = v] = 1/q^2$ but $\Pr[z_2' = v] = 0$ (because $z_2' > [k_u \cdot s \mod q] + \ell \cdot q$).
- If $q^2 - 1 < \zeta < \ell + q^2$ then $\Pr[z_2' = v] = \Pr[\rho = q^2 - 1 - \ell] = 1/q^2$ but $\Pr[z_1' = v] = 0$ (because $z_1' \leq [k_u \cdot s \mod q] + (q^2-1)q$).
- If $\ell \leq \zeta \leq q^2 - 1$ then $\Pr[z_1' = v] = \Pr[\rho = \zeta] = 1/q^2$ and $\Pr[z_2' = v] = \Pr[\rho = \zeta - \ell] = 1/q^2$.

We get that $\Delta(z_1', z_2') = \sum_{\zeta=0}^{\ell+q^2-1} |\Pr[z_1' = v] - \Pr[z_2' = v]| = \frac{2\ell}{q^2}$, which is negligible.

Case 2. Let $\mathcal{A}$ be a semi-honest real world adversary who corrupts P_c, consider an ideal world adversary $\mathcal{S}$ that does as follows:

- **Key Generation.**
  1. Run $\mathcal{A}$ internally and simulate the honest party P_u:
     (a) Receive the oracle call and obtain v, forward v to the RO and obtain v_x, forward v_x back to $\mathcal{A}$.
     (b) Receive v_x from $\mathcal{A}$.
     (c) Compute $x_c = \mathcal{H}(v \| \mathsf{keygen})$, $X_c = x_c \cdot G$ and $X_u = (x_c)^{-1} \cdot X$.
     (d) Generate a Paillier key-pair (pk, sk) where $pk = N = P \cdot Q$, with κ'-bit primes P, Q, and compute $c_{key} = \mathsf{Enc}(pk, 0)$.
     (e) Send (X_u, pk, c_{key}) and $(\mathsf{proof}, c_{key}, N, X_u)$ to P_c.
     (f) Send $(\mathsf{proof},$
     (g) Receive X from $\mathcal{A}$ and output whatever $\mathcal{A}$ outputs.
- **Sign.**
  1. Run $\mathcal{A}$ internally and simulate the honest party P_c:
     (a) Receive R from $\mathcal{F}_{\text{ECDSA}}$.

(b) Compute $k_c = \mathcal{H}(v \| \mathsf{sid})$, and computes $R_u = (k_c)^{-1} \cdot R$.

(c) Send R_u and $(\mathsf{proof}, \mathsf{sid}, R_u)$ to $\mathcal{A}$.

(d) Receive c_2 from $\mathcal{A}$ and output whatever $\mathcal{A}$ outputs.

The views of $\mathcal{A}$ in the real execution and under the simulation of the key generation protocol are computationally indistinguishable: the value X_u (and therefore X) are identically distributed in G and the key-pairs generated in both worlds are identically distributed. The only difference is in the generation of ciphertext c_{key}: in the real execution this is an encryption of x_u and in the simulation this is an encryption of zero, and since Paillier encryption scheme is CPA-secure it follows that that the two views are computationally indistinguishable.

In addition the views of $\mathcal{A}$ in the real execution and under the simulation of the signing protocol are identically distributed, in both cases it only receives R_u and $(\mathsf{proof}, \mathsf{sid}, R_u)$, such that $k_c \cdot R_u = R$, with R chosen by the functionality. Note that unlike in [80], since we assume $\mathcal{A}$ is semi-honest it always reply with a ciphertext that holds a correct evaluation on c_{key} and so we do not need to guess whether to abort or not, neither to rely on the 'Paillier-EC' assumption [80, Def. 5.2].

4.7 Applications

Unstoppable wallets serve as a foundational component for a diverse array of applications. To demonstrate their applicability, we developed and implemented two examples of applications that possess real-world value. These applications were deployed to Secret Network's mainnet under contract addresses: (1) *secret1lge6kdh078u7yc778whz8wjdc39ce78knqjfjh*; (2) *secret1lkvhyg4723fxreeyrm0mk7pkzgd4qaztmx4ztw*. At their core, these wallets are governed by a smart contract, meaning that they may have all kinds of other use-cases as well.

4.7.1 Multisignature Wallet with Policy Checks

In the traditional banking system, accounts often have various checks and limits on spending to enhance security and control. One can imagine a similar use case for cryptocurrency transactions, integrating such checks and constraints within a multisignature wallet.

Threshold ECDSA inherently supports a multisignature transaction approval structure already, necessitating $(t + 1)$-out-of-n parties to endorse signing a transaction. On top of this, with unstoppable wallets, we can introduce further layers of spending policies into the smart-contract component of the protocol, such as per-transaction spending limits, daily spending limits, or a combination of both. These policies offer increased control and security over transactions involving cryptocurrency.

One can think of more elaborate schemes and use-cases as well, that clearly benefit from the blockchain's role as a public bulletin board. For example, decentralized autonomous organizations (DAOs) are often assumed to be governed by all token holders, but their treasuries are in practice controlled by a small committee of signers[5]. By leveraging unstoppable

[5] As a concrete example, as of Sep, 2022, Frax treasury of 1.2B USD was unilaterally controlled by the team's multisig (https://www.blockworksresearch.com/research/risk-assessment-frax-governance).

wallets, the community could define clear spending limits in a smart contract to prevent a DAO committee from abusing their mandate.

To demonstrate the concept of a multisig wallet with policy checks, we developed a contract that not only requires a quorum of at least $t + 1$ approvals, but also verifies the transaction as a valid Ethereum transaction with a spending limit of 1 ETH. We detail the contract flow in Diagram 4.2.

4.7.2 Wallet Exchange

Typically, users exchange cryptocurrencies, such as swapping BTC for ETH between two parties. However, here we propose an alternative model: instead of exchanging assets, what if we could exchange the wallet itself directly? This concept, a wallet exchange, is not merely theoretical. For instance, venture capital funds often enter illiquid deals for tokens that do not yet exist or have a certain lockup, making selling the asset itself infeasible.

One could envision a wallet exchange platform that allows sellers to list their wallets instead of their assets, and sell these to buyers, who can be reassured that the seller provably loses access after the transaction concludes. In light of the recent collapse of large exchanges and centralized lenders like FTX and Celsius, an exchange that allows creditors to sell their claims (likely at a discount) becomes more appealing. Such exchanges have already started to emerge[6], and a wallet exchange mechanism could provide a more secure way to facilitate this process.

Equipped with this motivation, we present an implementation of a contract that enables selling a wallet from the current owner (the seller) to an interested buyer. Initially, the wallet is jointly held by the seller and the chain. A prospective buyer can send a bid to the contract governing the wallet, which the seller can either accept or ignore. The buyer can set a timeout to release their deposited bid if they have not received a response from the seller after some time.

If the seller accepts the bid, they must re-encrypt their share of the key with the buyer's key and send it to the contract in a separate transaction that concludes the sale. The chain, after verifying that neither party has cheated, assists in refreshing the shares and revoking the seller's share. The contract also atomically finalizes the payment, completing the wallet exchange process securely. We detail the contract flow in Diagram 4.3.

4.8 Implementation and Evaluation

In this section, we provide an overview of the implementation and evaluation of our proposed underlying threshold ECDSA protocols. We implement the main threshold ECDSA protocol in 4, 5, and the protocol for a single user. Using these as building blocks, we implement the applications discussed in Section 4.7. We also discuss the practical aspects of implementing cryptographic primitives on a confidential smart-contract enabled blockchain and delve into the performance analysis of our approach in terms of gas costs associated with on-chain transactions, which is the main performance bottleneck in addition to the number of consecutive writes (i.e., transactions) each user has to perform.

[6]https://opnx.com/

4.8.1 Implementation Details

Our implementation is tied and optimized for the *secp256k1* curve, as that is the most commonly used curve related to cryptocurrencies. However, our protocols are generic and our implementation can be extended to support other curves as well. The implementation is divided into two main parts: the local execution by users, and the on-chain execution on the blockchain. Our code is written in Rust, but it is important to note that any language could be used for the client.

For the on-chain part of our proposed protocols, the spectrum of options is more constrained, as we needed a blockchain that supports confidential smart contracts. We chose the Secret Network [28], a blockchain platform that relies on TEEs for confidentiality and has been running in production for several years. Another benefit of choosing Secret Network is that it exemplifies a blockchain that guarantees correctness, availability and privacy with *different levels of confidence*. Namely, while the system's correctness and availability guarantees have never been broken, its privacy guarantees were broken multiple times due to attacks on the underlying hardware (e.g., [113], [114]). Attacks on privacy (but not on the availability nor the correctness) may happen even when confidential smart contracts are implemented using (publicly auditable) MPC protocols. In such cases corrupted parties may 'silently' break privacy, but cannot break correctness or availability. This is the source for our motivation to not store the entire signing key within the smart contract.

Secret Network is built on top of Cosmos SDK and Tendermint consensus algorithm [115], and it features a smart contract framework based on CosmWasm, which enables developers to write and deploy smart contracts using Rust, ensuring compatibility with the local execution part of our protocols. Communication between users and the blockchain is established directly through transactions, which are used for broadcasting data and writing it into the chain's state, and queries, which facilitate data retrieval from the chain's current state. Compared to our formal terminology, transactions are writes (and are therefore slow), and queries are reads.

Our entire implementation is open-source [7], fostering transparency and allowing for peer review. In total and including our modifications below to existing repositories, our implementation comprises roughly 6,500 lines of code.

4.8.2 Cryptographic Primitives on Chain

In order to allow our protocols to run inside of a smart contract, we needed to implement several cryptographic building blocks in a way that allows them to run on-chain. In particular, we needed libraries that support secret sharing (over secp256k1's specified field), elliptic curve operations (over the same curve), and Paillier encryption.

This turned out to be especially challenging, since we had to make sure these building blocks are efficient, do not use randomness generated by the operating system, and do not use floating-point types. The last two are practical constraints present in any blockchain environment, which needs to be deterministic due to consensus. Porting existing cryptographic libraries was especially challenging, since practically all libraries need to generate randomness at one point, and this issue propagates up the dependency tree. We modified all

[7]https://github.com/anonauthors01001/unstoppable-wallets

relevant libraries to take in a custom PRG instead, and we used that as a hook to plug in a deterministic PRG that is purpose-built for Secret Network contracts. Overall, we modified approximately 1,350 lines of code across five open-source repositories.

4.8.3 Performance Evaluation

In this subsection, we assess the performance of our proposed threshold ECDSA protocols by focusing on the gas costs associated with on-chain transactions. Gas costs represent the computational resources necessary to execute a transaction on a blockchain, and are a popular cost metric on all smart-contracts chains, starting with Ethereum [2]. These costs not only impact users monetarily but also impose limitations on the number of gas-intensive transactions a blockchain can process in a single block, as blockchains have inherent constraints in terms of computational resources.

Multiparty Protocol Evaluation

In Table 4.2a we show an evaluation for $n = 5$, $t = 4$. *init* marks the contract's initialization (for each wallet we deploy a different contract), *keygen* is the dealing portion of the key generation protocol, *presig* marks the dealing part of the signing protocol where shared randomness and the nonce are produced, and *sign_i* marks the cost for each signing party. On a per user basis, the costs are negligible at the time of writing, and amount to roughly one-tenth of a cent per user (with the exception of the dealer who pays roughly three-tenths of a cent). Since the actual cost was calculated based on the price of SCRT, a volatile asset used to pay fees in Secret Network, it is also useful to compare the unitless gas used metric between threshold wallets and other common types of smart contract executions. We reference these in Table 4.3 and note that surprisingly our results are very appealing given that we have essentially implemented an MPC protocol on-chain.

Table 4.2: Benchmarks for Multiparty and Two-party ECDSA

(a) Table (a)

Tx Type	Time (ms)	Tx size (bytes)	Gas Used	Tx Cost (¢)
init	0.07	43	45,227	0.04¢
Keygen	7.93	1,206	132,792	0.11¢
Presig	11.65	4,335	237,195	0.19¢
Sign_1	1.62	295	138,865	0.11¢
Sign_2	1.55	295	140,599	0.11¢
Sign_3	1.51	295	142,328	0.11¢
Sign_4	1.85	295	144,046	0.12¢
Sign_5	12.95	295	187,238	0.15¢

(b) Table (b)

Tx Type	Time (ms)	Tx size (bytes)	Gas Used	Tx Cost (¢)
Keygen	175.35	2,707	856,051	0.68¢
Sign	313.75	287	1,882,619	1.51¢

We also found that costs scale very well (practically linearly, as expected) with the number of parties, making this scheme highly efficient in terms of scalability. We capture this close-

Table 4.3: Gas cost baselines

Tx Type	Gas Used
Token transfer	55,877
NFT Mint/Transfer	150,833
Token Swap (direct)	595,916
Token Swap (2-hops)	1,553,937

to-linear relation in Figure 4.4, which examines how the average gas expenditure changes (on average) per party, as we increase the number of parties (and assume the maximum corruption threshold of $n = t - 1$). We make the same comparison for a fixed $n = 15$ and a dynamic threshold in Figure 4.5, and reach a similar result.

Two-Party Protocol Evaluation

Interestingly, as shown in 4.2b, our performance evaluation reveals that the multiparty protocol, even when accommodating numerous parties, incurs significantly lower costs per party compared to the two-party protocol. This finding can be attributed to the relatively resource-intensive Paillier Encryption used in the two-party protocol, which is used for a single user. It is also worth mentioning that we have not implemented the expensive zero-knowledge proofs necessary for this protocol on-chain, which would undoubtedly widen the gap even more. Based on our results, and assuming the maximum amount of corruptions, we extrapolate that it would take around $n = 82$ users for the gas costs of the multiparty protocol to match the two party one.

Also, given current gas limits in Secret Network, and given that state-of-the-art multiparty threshold ECDSA protocols (e.g., [79]) requires even more homomorphic operations and many more zero-knowledge proofs, it is fair to assume any existing multiparty variant would not even run on-chain. These results support the need of devising chain-friendly threshold ECDSA protocols, as demonstrated in this work.

PROTOCOL 5. (*Signing:* $\mathsf{Sign}\,(M,(G,G,q),\mathsf{sid})$)

Inputs.

1. Each party P_i, $i \in [1,n]$, holds $([x]_i, X)$.
2. Party P_c holds X and $[x]_i$ for all $i \in [n+1,N]$.
3. The parties Compute $m = H_q(M)$ and verify that sid has not been used before (otherwise the protocol is not executed).

The protocol.

1. **Users' dealing**:
   (a) Party P_1 samples a random $k_u, a_u \leftarrow Z_q$.
   (b) Party P_1 computes $[k_u] \leftarrow \mathsf{SS.Share}(k_u,t,N)$ and $[a_u] \leftarrow \mathsf{SS.Share}(a_u,t,N)$.
   (c) Party P_1 computes $[z_u] \leftarrow \mathsf{SS.Share}(0,2t,N)$ and $[z'_u] \leftarrow \mathsf{SS.Share}(0,2t,N)$.
   (d) Party P_1 sends $([k_u]_i,[a_u]_i,[z_u]_i,[z'_u]_i)$ to party P_i where $i \in [1,n]$ and to P_c where $i \in [n+1,N]$.
   (e) Party P_1 sends $R_u = k_u \cdot G$ to P_c.
2. **Center's dealing**:
   (a) Party P_c computes $k_c = \mathcal{H}(x_c \| \mathsf{sid})$.
   (b) Party P_c samples a random $a_c \leftarrow Z_q$.
   (c) Party P_c computes $[k_c] \leftarrow \mathsf{SS.Share}(k_c,t,N)$ and $[a_c] \leftarrow \mathsf{SS.Share}(a_c,t,N)$.
   (d) Party P_c computes $[z_c] \leftarrow \mathsf{SS.Share}(0,2t,N)$ and $[z'_c] \leftarrow \mathsf{SS.Share}(0,2t,N)$.
   (e) P_c sends $([k_c]_i,[a_c]_i,[z_c]_i,[z'_c]_i)$ to party P_i for $i \in [1,n]$.
   (f) P_c sends $R = k_c \cdot G + R_u$ to everyone.
3. **Partial signature.**
   (a) Every party P_i for $i \in [1,n]$, and P_c for $i \in [n+1,N]$:
      i. Computes $[\alpha]_i = [\alpha_u]_i + [\alpha_c]_i \mod q$, for $\alpha \in \{k,a,z,z'\}$.
      ii. Computes $[s_1]_i = [a]_i(m + r[x]_i) - [z]_i \mod q$ and $[s_2]_i = [k]_i[a]_i - [z']_i \mod q$.
   (b) P_i for $i \in [1,n]$ sends $(m,[s_1]_i,[s_2]_i)$ to P_c.
4. **Finalization.** Upon receiving $t+1$ messages, $\{(m,[s_1]_{i_j},[s_2]_{i_j})\}_{j=1}^{t+1}$, party P_c:
   (a) Computes $s_1 = \mathsf{SS.Reconstruct}(\{[s_1]_{i_j}\}_{j=1}^{t+1}, \{[s_1]_j\}_{j=n+1}^{N})$ and $s_2 = \mathsf{SS.Reconstruct}(\{[s_2]_{i_j}\}_{j=1}^{t+1}, \{[s_2]_j\}_{j=n+1}^{N})$.
   (b) Computes $s = s_1 \cdot s_2^{-1} \mod q$.
   (c) Broadcasts (r,s) if it is a valid signature on `MSG`, otherwise it broadcasts $\perp$.

PROTOCOL 6. (*Robust Key-Generation:* KeyGen)

1. **User's dealing:** Every P_ℓ, ($\ell \in \{1, \ldots, t+1\}$):
   (a) Samples $x_\ell \leftarrow Z_q$ and computes and broadcasts
   $$(\{c_i^\ell\}_{i=1}^N, \{A_j^\ell\}_{j=0}^t, \pi^\ell) \leftarrow \mathsf{PVSS.Share}_{t,N}(x_\ell).$$
   (b) Let $u \in [1, t+1]$ be the first index for which
   $$1 = \mathsf{PVSS.CheckDealer}(\{c_i^u\}_{i=1}^N, \{A_j^u\}_{j=0}^t, \pi^u).$$
   Denote these values by $\{c_i\}_{i=1}^N, \{A_j\}_{j=0}^t$ (i.e., dropping the supertext u)
2. **Center's dealing**:
   (a) P_c computes $[x_c] \leftarrow \mathsf{SS.Share}_{t,N}(x_c)$ for $x_c \leftarrow \mathcal{H}(\tilde{x})$ where $\tilde{x} \leftarrow \{0,1\}^\kappa$.
   (b) P_c sends $[x_c]_i$ to P_i for $i \in [1, n]$.
   (c) P_c broadcasts $X = x_c \cdot G + A_0$ and $X_i = [x_c]_i \cdot G + \sum_{j=0}^t i^j \cdot A_j$ for $i \in [1, n]$.
3. **Compute secret key shares**: Each party P_i computes $[x_u]_i = \mathsf{EG.Dec}_{\mathsf{dk}_i}(c_i)$ and $[x]_i = [x_u]_i + [x_c]_i \mod q$.

PROTOCOL 7. (*Robust Signing:* $\mathsf{Sign}(M,(G,G,q),\mathit{sid})$)

Inputs.

1. Each party P_i, $i \in [1,n]$, holds $([x]_i, X)$.
2. Party P_c holds X and $[x]_i$ for all $i \in [n+1,N]$.
3. The parties Compute $m = H_q(M)$ and verify that sid has not been used before (otherwise the protocol is not executed).

The protocol.

1. **User's dealing:** Every P_ℓ, $(\ell \in \{1,\dots,t+1\})$:
   (a) Samples $k_\ell, a_\ell \leftarrow Z_q$ and computes and broadcasts
   $$(\{c^\ell_{k,i}\}_{i=1}^N, \{K^\ell_j\}_{j=0}^t, \pi^\ell_k) \leftarrow \mathsf{PVSS.Share}_{t,N}(k_\ell),$$
   $$(\{c^\ell_{a,i}\}_{i=1}^N, \{A^\ell_j\}_{j=0}^t, \pi^\ell_a) \leftarrow \mathsf{PVSS.Share}_{t,N}(a_\ell),$$
   $$(\{c^\ell_{z,i}\}_{i=1}^N, \{Z^\ell_j\}_{j=0}^t, \pi^\ell_z) \leftarrow \mathsf{PVSS.Share}_{2t,N}(0),$$
   $$(\{c^\ell_{z',i}\}_{i=1}^N, \{Z'^\ell_j\}_{j=0}^t, \pi^\ell_{z'}) \leftarrow \mathsf{PVSS.Share}_{2t,N}(0).$$
   (b) Let $u \in [1,t+1]$ be the first index for which
   $$1 = \mathsf{PVSS.CheckDealer}(\{c^u_{\alpha,i}\}_{i=1}^N, \{\alpha^u_j\}_{j=0}^t, \pi^u_\alpha)$$
   for all $\alpha \in \{k,a,z,z'\}$.
2. **Center's dealing**:
   (a) P_c computes $k_c = \mathcal{H}(x_c \| \mathsf{sid})$, samples $a_c \leftarrow Z_q$ and computes $[k_c] \leftarrow \mathsf{SS.Share}_{N,t}(k_c)$, $[a_c] \leftarrow \mathsf{SS.Share}_{N,t}(a_c)$, $[z_c] \leftarrow \mathsf{SS.Share}_{N,2t}(0)$, and $[z'_c] \leftarrow \mathsf{SS.Share}_{N,2t}(0)$
   (b) P_c sends $([k_c]_i, [a_c]_i, [z_c]_i, [z'_c]_i)$ to P_i for $i \in [1,n]$
   (c) P_c broadcasts $K = k_c \cdot G + K^u_0$ and (K_i, A_i, Z_i, Z'_i) for all $i \in [1,n]$, where $E_i = [e_c]_i \cdot G + \sum_{j=0}^t i^j \cdot E_j$ for every $(E,e) \in \{(K,k),(A,a),(Z,z),(Z',z')\}$.
3. **Local computation.**
   (a) P_i $(i \in [1,N])$ computes $[\alpha]_i = [\alpha_u]_i + [\alpha_c]_i \mod q$ for $\alpha \in \{k,a,z,z'\}$, where $[\alpha_u]_i = \mathsf{EG.Dec}_{\mathsf{dk}_i}(c^u_{\alpha,i})$.
   (b) P_i $(i \in [1,N])$ computes $[s_1]_i = [a]_i(m + r[x]_i) - [z]_i \mod q$ and $[s_2]_i = [k]_i[a]_i - [z']_i \mod q$.
   (c) P_i $(i \in [1,n])$ computes $M_{i,1} = ([a]_i \cdot [x]_i) \cdot G$ and $M_{i,2} = ([a]_i \cdot [k]_i) \cdot G$.
   (d) Everyone computes $r = K.x \mod q$.
4. **Partial signature.**
   (a) P_i $(i \in [1,n])$ sends $(\mathsf{prove}, \mathsf{sid}\|1, A_i, X_i, M_{i,1}, a_i, x_i)$ and $(\mathsf{prove}, \mathsf{sid}\|2, A_i, K_i, M_{i,2}, a_i, k_i)$ to $\mathcal{F}_{\mathsf{zk}}^{R_{DHT}}$.
   (b) P_i $(i \in [1,n])$ sends $(m, [s_1]_i, [s_2]_i, M_1, M_2)$ to P_c.
5. **Finalization.** Upon receiving at least $t+1$ messages $(m, [s_1]_i, [s_2]_i, M_{i,1}, M_{i,2})$ for which $[s_1]_i \cdot G = r \cdot M_{i,1} + m \cdot A_i - Z_i$, $[s_2]_i \cdot G = M_{i,2} - Z'_i$, and proofs $(\mathsf{proof}, \mathsf{sid}\|1, A_i, X_i, M_{i,1})$ and $(\mathsf{proof}, \mathsf{sid}\|2, A_i, K_i, M_{i,2})$ were received from $\mathcal{F}_{\mathsf{zk}}^{R_{DHT}}$, denote these indices by I. Then party P_c:
   (a) Computes $s_1 = \mathsf{SS.Reconstruct}(\{[s_1]_i\}_{i \in I}, \{[s_1]_j\}_{j=n+1}^N)$ and $s_2 = \mathsf{SS.Reconstruct}(\{[s_2]_i\}_{i \in I}, \{[s_2]_j\}_{j=n+1}^N)$.
   (b) Broadcasts $s = s_1 \cdot s_2^{-1} \mod q$.

PROTOCOL 8. (*Two-Party Key-Generation:* KeyGen)

1. **P_c's randomness setup.**
   (a) P_c picks a random value $v \leftarrow \{0,1\}^{\kappa}$ and computes $v_x = \mathcal{H}(v)$.
   (b) P_c sends v_x to P_u.
2. **Party P_u's message**:
   (a) P_u samples a random $x_u \leftarrow Z_q^*$ and computes $X_u = x_u \cdot G$.
   (b) P_u generates a Paillier key-pair (pk, sk) where $pk = N = P \cdot Q$ with κ'-bit primes P, Q, and computes $c_{key} = \mathsf{Enc}_{pk}(x_u)$. ($\kappa'$ is the bit-length of the factors of N for the Paillier encryption scheme to be secure).
   (c) P_u sends X_u, $pk = N$ and c_{key} to P_c.
   (d) P_u proves in zero-knowledge that $N \in L_P$ and that it knows a witness (x_u, P, Q) such that $(c_{key}, N, X_u) \in L_{PDL}$, by sending $(\mathsf{prove}, c_{key}, N, X_u, x_u, P, Q)$ to $\mathcal{F}_{zk}^{\mathsf{keygen}}$.
3. **Party P_c's message**: Upon receiving $(\mathsf{proof}, c_{key}, N, X_u)$ from $\mathcal{F}_{zk}^{\mathsf{keygen}}$:
   (a) Verify that $c_{key} \in Z_{N^2}^*$ and that N is of length at least $2\kappa'$.
   (b) P_c computes $x_c = \mathcal{H}(v \| \mathsf{keygen})$, and $X_c = x_c \cdot G$ and $X = x_c \cdot X_u$.
   (c) Send X to P_u.
4. **Output:**
   (a) P_u outputs (pk, sk, x_u, X).
   (b) P_c outputs (pk, x_c, X, c_{key}).

PROTOCOL 9. (*2P Signing:* $\mathsf{Sign}\,(M, (G, G, q), \mathit{sid})$)

Inputs.

1. Party P_u holds (pk, sk, x_u, X).
2. Party P_c holds (pk, x_c, X, c_{key}).
3. The parties Compute $m = H_q(M)$ and verify that sid has not been used before (otherwise the protocol is not executed).

The protocol.

1. **Party P_u's message**:
   (a) P_u chooses $k_u \leftarrow Z_q$ and computes $R_u = k_u \cdot G$.
   (b) P_u sends R_u to P_c.
   (c) P_u sends $(\mathsf{prove}, \mathsf{sid}, R_u, k_u)$ to $\mathcal{F}_{zk}^{\mathsf{DL}}$ to proves knowledge of k_u.
2. **P_c's message:** Upon receiving $(\mathsf{proof}, \mathsf{sid}, R_u)$ from $\mathcal{F}_{zk}^{\mathsf{DL}}$:
   (a) P_c computes $k_c = \mathcal{H}(v \| \mathsf{sid})$ and computes $R = k_c \cdot R_u$ and $r = R.x \mod q$.
   (b) P_c chooses $\rho \leftarrow Z_{q^2}$ and $\tilde{r} \leftarrow Z_N^*$.
   (c) P_c computes:
      i. $c_1 = \mathsf{Enc}_{pk}(\rho q + [(k_c)^{-1} m \mod q], \tilde{r})$,
      ii. $v = (k_c)^{-1} \cdot r \cdot x_c \mod q$,
      iii. $c_2 = c_1 \oplus (v \odot c_{key})$
   (d) P_c sends R and c_2 to P_u.
3. **Output:**
   (a) P_u computes $s' = (k_u)^{-1} \cdot \mathsf{Dec}(sk, c_2) \mod q$ and $r = R.x \mod q$.
   (b) P_u outputs (r, s) where $s = \min(s', q - s')$.

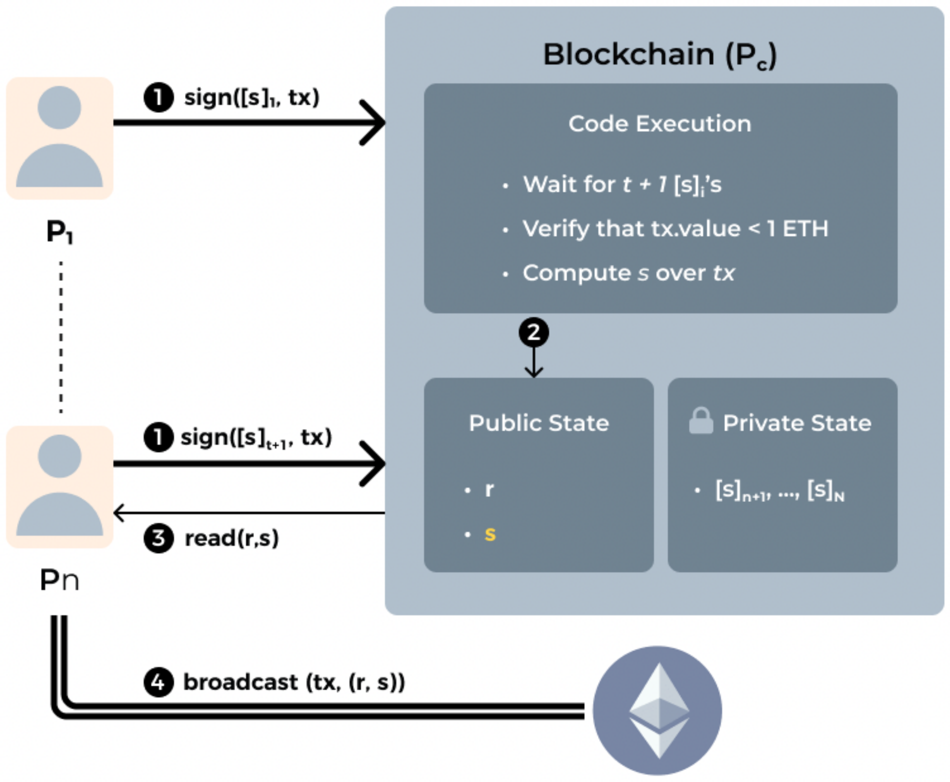
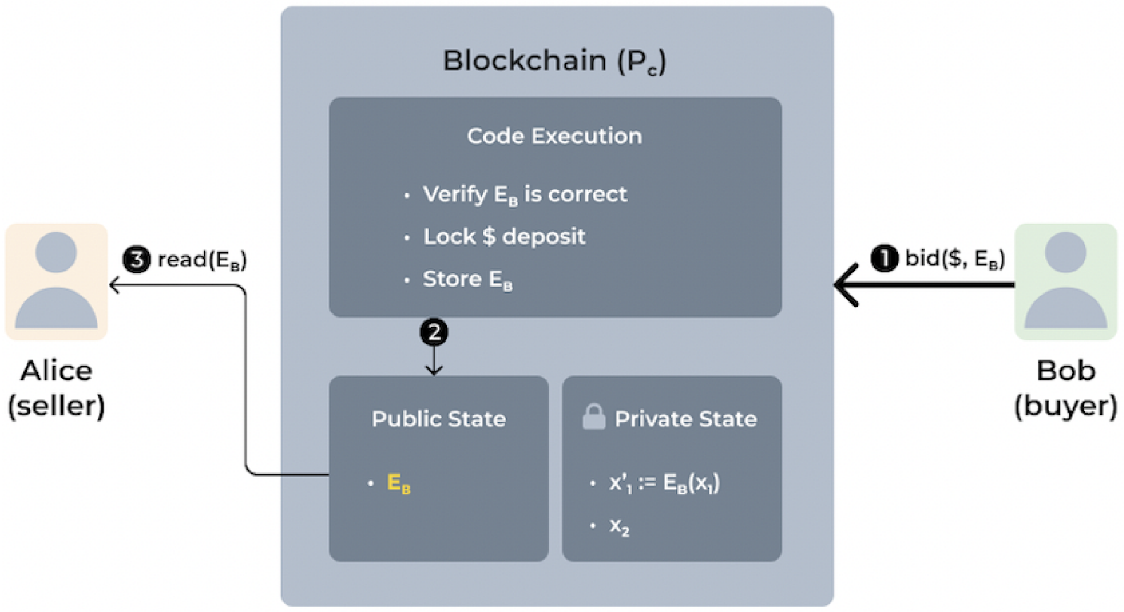
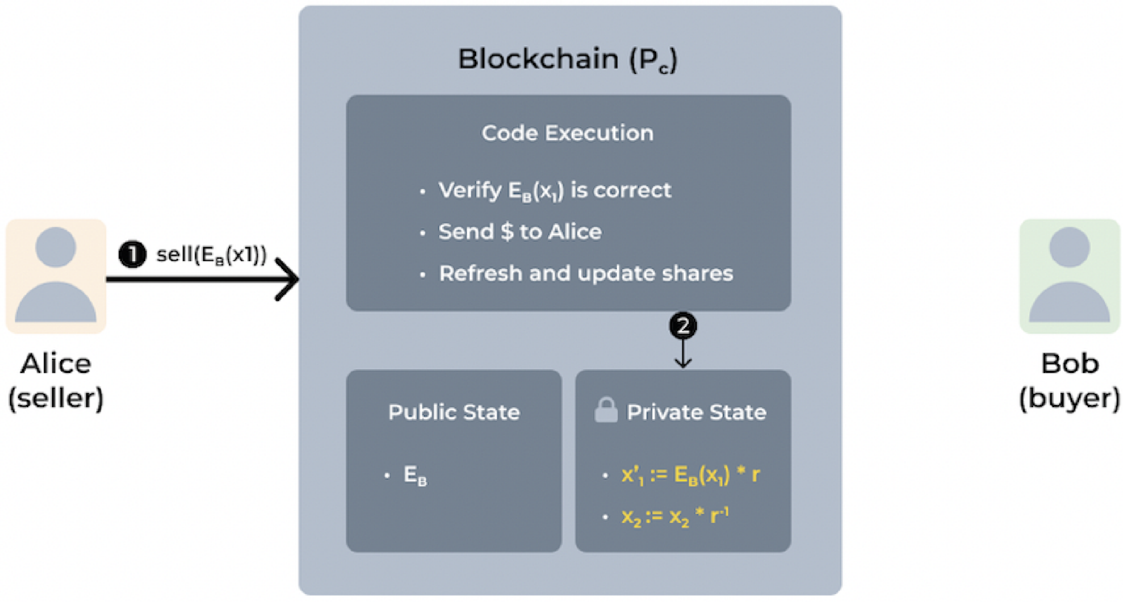


Figure 4.2: Multisignature Wallet with Policy Checks



(a) Step 1: Buyer initiates a bid for the wallet



(b) Step 2: Seller approves the sale

Figure 4.3: Wallet Exchange Application Flow

4.8.4 Scalability across n and t

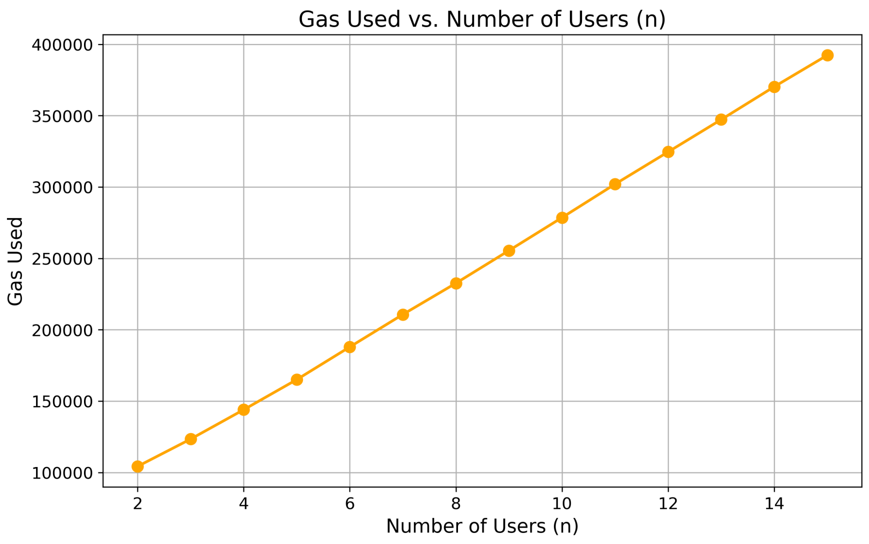


Figure 4.4: Gas Used vs. Number of Users (n)

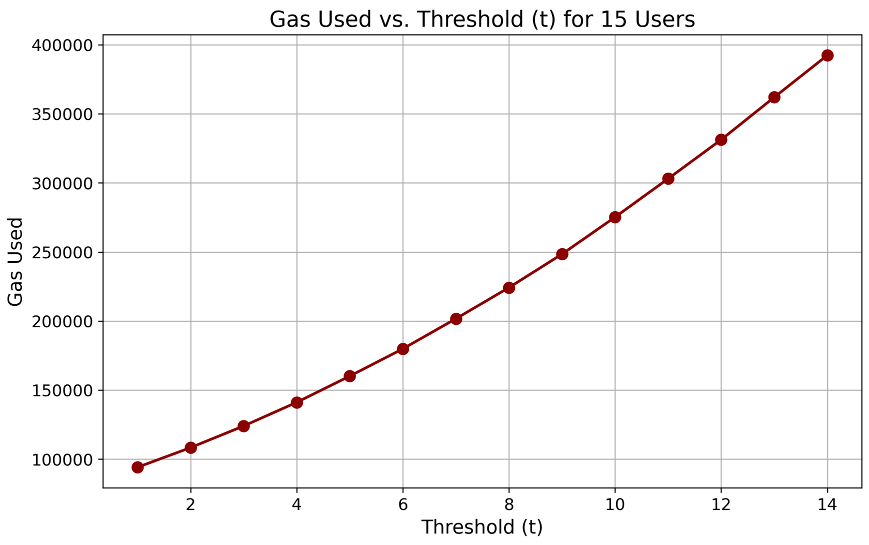


Figure 4.5: Gas Used vs. Threshold (t) for 15 users

Chapter 5

High-throughput Three-Party DPF and its application to ORAM and Digital Currencies

As described in Section 2.6, a *point function* $f_{\alpha,\beta}$ is a function that evaluates to zero everywhere, except for one point, α, at which it evaluates to a (possibly) non-zero value β. A DPF, introduced by [116], [117], is a cryptographic technique to share a point function to multiple receivers, such that each receiver can locally obtain a share of the evaluation of the function at the point of interest. In the two receiver setting, a DPF scheme allows one to generate two additive shares of $f_{\alpha,\beta}$, called f_0 and f_1, such that for every x in the domain it holds that $f_0(x) + f_1(x) = f(x)$. The performance of a DPF scheme is measured by the size of the shares (also known as 'keys') f_i given to the receivers and the incurred computational complexity to evaluate $f(x)$ for some x in the domain. In the two-party case (i.e., when the point function is shared toward two receivers) the shares size as well as the time to evaluate are logarithmic in the domain size.

The DPF primitive is increasingly recognized as a fundamental tool in various applications, including private information retrieval (PIR), anonymous communication, privacy-preserving machine learning (PPML) and even as a building block for general secure computation for RAM programs [118]–[126]. Having said that, the DPF constructions today are practical in the setting of two receivers, therefore, most of the applications mentioned above were demonstrated efficient when employing a two-party construction of a DPF; leaving the challenge of extending them to a larger number of parties unlocked.

A natural and important milestone is to efficiently extend DPFs to three receivers with honest majority. While a few works have addressed this setting, none of them reached a performance that matches that of the two receivers.[1] In this work we present the first DPF construction with three-receivers that accomplishes that goal, and as shortly explained, *even surpasses* the performance of the two receivers setting.

[1] The works in [127], [128] achieve a three-party protocol with a matching performance, however, we still consider it in the two receivers domain as one of the parties plays as a non-colluding server.

Database operations via DPFs We quickly revisit the importance of DPFs for PIR, PIW and ORAM. Most of the applications that use DPFs as building blocks focus on either reading from or writing to a database. Specifically, consider the 'replicated database' private information retrieval (PIR) setting with a (public) database D. In that setting, a client may privately read the α-th database entry by sharing a point function $f_{\alpha,1}$, and sending the shares f_0 and f_1 to the PIR servers. Then, the i-th server computes $d_i = \sum_x f_i(x) \cdot D[x]$ and hands d_i back to the client. Finally, the client can reconstruct the result by computing $d = d_0 + d_1$. Note that evaluating $f(x)$ over the entire domain $|D|$ creates a shared one-hot-vector, so d correctly encodes $D[\alpha]$. Obviously, this procedure does not work when the database itself is secret shared.

On the other hand, when the database is secret shared among the two servers, i.e., there are two databases D_0 and D_1 s.t. $D[x] = D_0[x] + D_1[1]$, a DPF can be used in order to privately *update* an entry in that database (also known as private information writing, or PIW). Specifically, a client sends the shares f_0 and f_1 for some point function $f_{\alpha,\beta}$, and then the i-th server adds $f_i(x)$ to the value at $D[x]$, for all x in the domain. This way, the client can privately add β to the α-th entry of the shared database.

Supporting both private information retrieval and writing operations, however, is more challenging, as privately reading an entry requires multiplication between the shared entries of the one-hot vector to those of the database, which is achieved via communication in the information theoretic setting, or via expensive public-key primitives; both approaches limit the size of the databases that a system may support. For example, the FLORAM system by Doerner and Shelat [129] requires $O(\sqrt{|D|})$ communication per private operation (read or write).

In contrast, given a DPF construction for three receivers, where sharing $f_{\alpha,\beta}$ results in f_1, f_2, f_3 s.t. $f_1(x)$, $f_2(x)$ and $f_3(x)$ form a 2-out-of-3 sharing of $f(x)$, we can efficiently achieve both private retrieval and update operations. This is due to the fact that the shares now have redundancy and have one degree of multiplicative homomorphism. The only work that proposed such a construction is [130], where the resulting shares form a replicated sharing. However, their construction achieves sub-optimal performance, both concretely and asymptotically. That is, their shares size as well as the computational complexity per evaluation is $O(\log^2 |D|)$. Furthermore, it does not protect against a malicious adversary. In this work, we propose the first three party DPF construction that matches (and even concretely improves) the two party one; additionally, it offers security against a malicious adversary. On its own, our three party DPF lends itself as the main tool for our efficient three-server ORAM, which has state-of-the-art performance compared to previous DPF-based ORAM constructions. Also, to the best of our knowledge, ours is the only maliciously secure construction.

Revisiting private and account-based digital currencies The UTXO (Unspent Transaction Output) model (as used by Bitcoin) tracks individual unspent coins from previous transactions. In contrast, the account model (as used by Ethereum) resembles traditional banking, with each user having an account and a clear balance. Transactions adjust these balances directly, simplifying operations for complex programs like smart contracts.

Existing works on account-based privacy-preserving digital currencies have typically been constrained by a limited anonymity set size [54], [131]–[135]. Consequently, the focus has predominantly been on the UTXO model, as evidenced by the plethora of works ([12], [53], [136]–[139] to name a few). However, the UTXO model has several significant shortcomings. For example, it limits programmability and auditability [140], [141], which are crucial for building financial applications and fraud-prevention. Additionally, in the privacy-preserving setting, the UTXO set infinitely grows, which poses scalability issues.

In light of this, using our DPF-based ORAM construction, we aim to rekindle interest in account-based privacy-preserving digital currencies, addressing its inherent limitations and proposing a viable alternative to private UTXO-based systems. We place a particular emphasis on the application of our techniques to Central Bank Digital Currencies (CBDCs). Given their rising prominence [137], [141]–[144] and the ongoing debate around their privacy [138], [139], [145], [146], our goal is to demonstrate the potential of our methods in developing a privacy-centric CBDC. Such a system would not only safeguard individual privacy but also remain conducive to necessary auditability.

5.0.1 Main Contributions

We summarize our main contributions below.

- We present a novel three-party verifiable DPF (VDPF) construction in the honest majority setting that is secure against a malicious server who may collude with the client. Our construction maintains the same asymptotic and concrete overheads (share size and evaluation time) as the state-of-the-art two-party DPF of Boyle et al. [20]. Compared to the state-of-the-art three-party DPF, our construction improves the DPF's shares size and evaluation by a factor of $O(\log n)$ asymptotically, translating to $48 - 120\times$ improvement for domains between $2^{16} - 2^{40}$.

- As the lion share of the overhead in a three-server ORAM is incurred by the DPF, we get the most efficient DPF-based three-server ORAM construction, in both communication and computation. We analytically compare ourselves against state of the art three-party protocols using DPF and ORAM [128], [130], [147]–[149]. We analyze all with respect to a semi-honest ORAM application and observe that we have better client-server communication, overall computation and storage costs, and our model does not have the limitations of [127], [128] (server-aided), and [149] (writes are not supported; only appends). Our comparison results are summarized in Table 5.1 [2,3].

- Expanding on our main construction, we introduce a three-party VDPF with semi-honest security, that uses a sublinear amount of PRF calls in full-domain evaluation. This construction is $\sim 2.2\times$ faster than the leading two-party DPF during full-domain evaluation, with potential for further improvement using GPU acceleration due to CPU-optimized AES instructions. However, similar to two-party DPFs, this construction is confined to separate read or update operations and is not suitable as a complete ORAM construction.

[2] For readability, we ignore constants related to the security parameter or the output group size.

[3] We estimate our results generalize to DORAM as well, but we leave exploring that to future work.

- Motivated by the need for privacy-preserving account-based digital currencies [138], [139], [145], [146], we apply our DPF-based ORAM to a three-party private CBDC protocol. Our implementation demonstrates substantial improvements in throughput over prior works in the account model. Specifically, our implementation supports $500, 200$ and 58 (resp. $825, 300$ and 105) transactions per second (tps) for anonymity sets of $2^{16}, 2^{18}$, and 2^{20} accounts, with protection against a malicious adversary (resp. a semi-honest adversary). We compare to the previous state-of-the-art, Solidus [131], and find that for these settings our protocol's throughput is $11 - 141\times$ higher.

- One building block in our maliciously secure DPF construction is a newly introduced primitive we call *updatable VDPF* (or UVDPF). An efficient UVDPF construction is used in order to directly compute a dot-product between (the compressed) full-domain evaluation of a point function and a vector (or of two point functions). Our dot-product protocol is constant-round and incurs communication overhead that is sub-linear in the functions' domain (compared to linear in a naive implementation). We believe that the UVDPF primitive and the DPFs dot-product protocol will find interest in other applications as well.

Protocol	**Key Size**	**Read**	**Update**	**DB Copies**	**Malicious**	**Model**
[130]	$3\sqrt{n}$	$4n$	$4n$	2	No	3-party
[148]	$12\log^2 n$	$O(n\log n)$	$O(n\log n)$	2	No	3-party
[149]	$3\log n$	$2n$	Appends only	2	Yes	3-party
[127], [128]	$3\log n$	$2n +$ Interaction	$3n$	3	No	Server-aided
Ours	$2\log n$	$2n$	$2n$	1	Yes	3-party

Best	Neutral	Poor	Worst

Table 5.1: Comparison of three-party ORAM protocols using DPFs. Key Size captures client-server communication, Reads and Updates are non-interactive (except for [127], [128]) and focus on overall computation costs. Results are for semi-honest, but we remark that we and [149] provide malicious implementations.

5.0.2 Related Work

DPF Constructions and three-party ORAM As observed by previous works [130], [148], [149], $(2, 3)$-DPFs are better suited for applications that combine PIR with PIW (ORAM), as they offer one degree of multiplicative homomorphism. Also, all $(3, 3)$-DPFs (withstanding two corruptions) are significantly less efficient, as they require $O(\sqrt{n})$-sized keys, and either have much higher constants or require public-key (or lattice-based) operations [20], [120], [121], [130], [150].

The state-of-the-art for $(2, 3)$-DPF [127], [128], [148], [149] combines replicated secret sharing (RPSS) with $(2, 2)$-DPFs, and then uses these to build different ORAM-related applications. Bunn et al., [148] proposes a $(2, 3)$-DPF similar to us, but their construction has asymptotically larger keys and evaluation time, which makes it $48 - 120\times$ less efficient, in both computation and communication, for domains between $2^{16} - 2^{40}$ (and this gap widens further for larger domains). Alternatively, Waldo, Duoram and PRAC [127], [128], [149] take a different approach and use (2,2)-DPFs to construct their ORAM application (a private time-series database and DORAM respectively), but their solutions are more limited as they do not develop new dedicated three-party DPFs. Waldo cannot do writes or updates (only appends), while Duoram and PRAC operates in the server-aided model. Only Waldo offers malicious security as we do, but they rely on public access control, rather than the privacy preserving authentication that our protocol offers.

A different line of work focused on *verifiable DPFs*, essentially achieving security for DPFs against potentially malicious clients (dealers). [20], [119] suggested to use sketching techniques, and more recently [151] presented a lightweight solution based on hashing. Similarly, [120], [152] show how to do private access-control for DPFs. We similarly build verifiability and access-control capabilities into our (2,3)-DPF construct. Our construction protects against both malicious clients and servers, whereas prior work only protected against semi-honest servers.

Privacy-preserving digital currency A large body of work has been dedicated to ensuring transaction-privacy in blockchains. The majority of works [12], [53], [136], [137], [153]–[156] focused on the UTXO-model, which offers a limited programmability and auditability [140], add end-user and developer complexity [12], [141], and cause an infinite growth of the permanent database (the nullifiers). This led to the transition into the account-based model (e.g., Ethereum), which we follow in this work. Solutions in this model mostly build upon general MPC techniques that enable the addition of additional layers on top of it (like smart contracts, auditability, etc.) [3], [4], [6], [157].

The challenge for account-based ledgers with private transactions Solving privacy for account-based cryptocurrencies remains a challenge, and prior work has significant limitations compared to UTXO-based solutions. QuisQuis [134] and Zether [54], [135] operate in the account-model but are limited to extremely low k-anonymity sets (64-256) [4]. Very recently, [158] presented a theoretical solution using Fully Homomorphic Encryption (FHE), but they neither provide an implementation or an evaluation. The heavyweight cryptography used indicates it might not be concretely efficient.

An increasing body of research is focused on constructing CBDCs [137]–[139], [141]–[144]. Yet, to the best of our knowledge, we are the first to provide a private account-based solution. Most similar to our research, several works have proposed building private bank-to-bank cryptocurrencies (e.g., [131]–[133]). In these works, banks transact with each other on behalf of users, but the number of banks in all of these models is very small (10-100). While Solidus is more scalable, zkLedger and MiniLedger are geared more towards auditability. We compare against Solidus which is the state-of-the-art and show that our work has $11 - 141\times$ higher throughput up to an anonymity set of 2^{20}. Additionally, while our protocol provides full anonymity, the rest of these works only provides bank-level anonymity, which is orders of magnitude smaller (equal to the total number of banks). Conversely, compared to these other systems, our system is not publicly verifiable and relies on MPC assumptions.

[4]Estimates from [158]

5.1 MPC Functionalities

5.1.1 Overview

We use of the following well-established ideal functionalities in this work.

- $\mathcal{F}.\mathsf{Rand}()$, $\mathcal{F}.\mathsf{DRand}()$, and $\mathcal{F}.\mathsf{Zero}()$, which return a shared random value, shared random with degree t and $2t$, and a sharing of zero, respectively.
- $\mathcal{F}.\mathsf{A2B}([x])$. Converts an arithmetic shamir sharing to a binary Shamir sharing.
- $\mathcal{F}.\mathsf{Mult}([x], [y])$. Returns a (t, n)-Shamir sharing of the product $x \cdot y$.
- $\mathcal{F}.\mathsf{CheckZero}([x])$. Returns 1 if $[x] = 0$, or 0 otherwise.
- $\mathcal{F}.\mathsf{LTE}([x], [y])$. Returns 1 if $[x] \leq [y]$ and 0 otherwise.

In addition, we define a procedure, $Open([x])$ as a one-round (assuming broadcast) protocol where all parties send their shares to each other and reconstruct a secret.

5.1.2 Access Policy ORAM Functionality

We define an application called an *ORAM with Access Policy*, and define the corresponding Functionality $\mathcal{F}.\mathsf{APORAM}$, in Figure 5.7

5.1.3 Sum-of-Products Functionalities

We use two Sum-of-Products functionalities to compute the dot product. $\mathcal{F}.\mathsf{Product}$ refers to the maliciously secure functionality, whereas $\mathcal{F}.\mathsf{SoP}$ refers to a dot-product functionality in which the adversary can add an additive error. Both are defined in Figures 5.8 and 5.9, respectively.

5.1.4 Implementation of Known Functionalities

Here we provide implementation details for the (well established) MPC functionalities we use in this work. We focus on semi-honest implementations, since they can be transformed to malicious security using MACs [159].

Protocols for $\mathcal{F}$.Rand, $\mathcal{F}$.DRand, and $\mathcal{F}$.Zero

In the 3-party setting, these functionalities are implemented using $PRSS$ and $PRZS$ respectively as shown in [160]. The functionalities can be implemented with any PRF (e.g., AES) and require no communication and $O(1)$ computation.

Setting: The functionality interacts with servers S_1, S_2, S_3, clients $c_1, \dots, c_n$, and an adversary $\mathcal{S}$.

Parameters: The functionality is initialized with a zero-initialized array $D = (D_1, \dots, D_N) \in F^N$, and is parameterized with a policy verification function: $\mathsf{PVerify}$ that is given the request's arguments and returns `accept` or `reject`. Parameter m refers to the number of entries read-/update commands support.

- On input $(c_i, \mathsf{register})$ from client c_i, mark c_i as 'registered'.
- On input $(c_i, \mathsf{read}, (\ell_1 \dots, \ell_m))$ from a registered client c_i:
  - Send (read, m) to $\mathcal{S}$ and wait to its response; if $\mathcal{S}$ returns abort then send $\perp$ to $\mathcal{S}$ and all servers, otherwise (if $\mathcal{S}$ returns $\mathsf{continue}$) continue.
  - If $\mathsf{accept} = \mathsf{PVerify}(c_i, \mathsf{read}, (\ell_1 \dots, \ell_m))$ then output $(D_{\ell_1}, \dots, D_{\ell_m})$ to c_i.
- On input $(c_i, \mathsf{update}, (\ell_1, v_1), \dots, (\ell_m, v_m))$ from a registered c_i:
  - Send (update, m) to $\mathcal{S}$ and wait to its response; if $\mathcal{S}$ returns abort then send $\perp$ to $\mathcal{S}$ and all servers, otherwise (if $\mathcal{S}$ returns $\mathsf{continue}$) continue.
  - If $\mathsf{accept} = \mathsf{PVerify}(c_i, \mathsf{update}, ((\ell_1, v_1), \dots, (\ell_m, v_m)))$ then, update $D_{\ell_j} = D_{\ell_j} + v_j$ for every $j \in \{1, \dots, m\}$.

Figure 5.1: Functionality $\mathcal{F}_{\mathsf{APORAM}}$

Setting: The functionality interacts with parties P_1, P_2, P_3 and an adversary $\mathcal{S}$.

Inputs: P_i inputs $[V]_i, f_i$, for $i \in \{1, 2, 3\}$.

- For $j \in \{1, 2, 3\}$, Expand $[F]_j \leftarrow \mathsf{VDPF.Eval}(j, f_j)$.
- For all $i \in \{1, 2, ..., N\}$:
  - $F_i \leftarrow \mathsf{Reconstruct}_{1,3}([F_i])$
  - $V_i \leftarrow \mathsf{Reconstruct}_{1,3}([V_i])$
  - $Z_i \leftarrow F_i \cdot V_i$.
  - Store in $[Z]$ the i-th sharing: $[Z_i] \leftarrow \mathsf{Share}1, 3(Z_i)$.
- Wait for an input from $\mathcal{S}$, if it is $\perp$ then output $\perp$ to all parties, otherwise continue.
- Output $[Z]_j$ to P_j, for $j \in \{1, 2, 3\}$.

Figure 5.2: Functionality $\mathcal{F}.\mathsf{Product}$

Upon receiving $[X_i]_j, [Y_i]_j$ from the honest parties P_j, as their shares of X_i, Y_i, for every $i \in \{1, \ldots, N\}$ and an additive error ε from the adversary: Compute $z = \varepsilon + \sum_{k=1}^{N} X_i \cdot Y_i$, then, output $[z] \leftarrow \mathsf{Share}_{2,3}(z)$ to the parties.

Figure 5.3: Functionality $\mathcal{F}_{\mathsf{SoP}}$

Protocols for $\mathcal{F}$.Mult and $\mathcal{F}$.SoP

We use the DN multiplication scheme of [161] and its sum-of-products variant. Both of these schemes allow to efficiently realize $\mathcal{F}.\mathsf{Mult}$ and $\mathcal{F}.\mathsf{Product}$. We present the sum-of-products protocol below. Multiplication is the same, but involves scalar values instead of vectors.

Protocol Π_{SoP}:

- **Public parameters:** Prime field F or a binary field ; n parties; t the corruption threshold.
- **Inputs:** $[\boldsymbol{x}], [\boldsymbol{y}]$.
- Each party computes:
  - $[r_t], [r_{2t}] \leftarrow \mathcal{F}.\mathsf{DRand}()$
  - $[c] \leftarrow [\boldsymbol{x}] \cdot [\boldsymbol{y}] - [r_{2t}]$
  - $c \leftarrow Open([c])$
  - Output $[z] = c + [r_t]$

Protocol for $\mathcal{F}$.CheckZero

We use a known protocol for realizing the functionality $\mathcal{F}.\mathsf{CheckZero}$ (e.g., [159]):

Protocol $\Pi_{CheckZero}$:

- **Public parameters:** Prime field F or a binary field ; n parties; t the corruption threshold.
- **Inputs:** $[x]$.
- Each party computes:
  - $[r] \leftarrow \mathcal{F}.\mathsf{Rand}()$
  - $[c] \leftarrow \mathcal{F}.\mathsf{Mult}([x], [r])$
  - $c \leftarrow Open([c])$
  - Output $c \overset{?}{=} 0$

Protocols for $\mathcal{F}$.LTE and $\mathcal{F}$.A2B

There are many protocols in the literature that realize $\mathcal{F}$.LTE and $\mathcal{F}$.A2B [162]–[166]. As we stated in Section 5.6, bandwidth is not a major concern in our scheme, so we prioritize minimizing rounds over communication. Luckily, many of these protocols (e.g., [162]–[164]) offer constant-round protocols. For example, in [163], a 3 (online) rounds and $(k+1)q$-bits of communication protocol is provided. To illustrate, for 32-bit representations, $k = 32$ and $\lceil log_2(q) \rceil = 64$, we would only need 264B of communication per party per comparison gate, which is marginal. $\mathcal{F}$.A2B is somewhat more expensive but follows a similar analysis.

5.2 $(2, 3)$-Verifiable DPF

In this section, we formally define the notion of a $(2, 3)$-VDPF, and provide our construction as well as a security proof. To this end, we move away from additive shares and require that the individual evaluations by keys f_1, f_2, f_3 on input α form a valid Shamir sharing of the target value β. This is in contrast to other works [148], [149] where those values form a replicated secret sharing. We start by defining a (2,3)-VDPF:

Definition 5.2.1. *A* $(2, 3)$*-VDPF, denoted* $F^{(2,3)}_{\alpha,\beta}$*, is defined by algorithms:*

- $(f_1, f_2, f_3) \leftarrow \mathsf{VDPF.Gen}(1^\kappa, \alpha, \beta)$ *and*
- $y_b \leftarrow \mathsf{VDPF.Eval}(b, f_b, x)$ *(*$b \in \{1, 2, 3\}$*),*
- $\pi_b = \mathsf{VDPF.Prove}(b, f_b, r)$*, (*$b \in \{1, 2, 3\}$*),*
- $\{\mathtt{accept}, \mathtt{reject}\} \leftarrow \mathsf{VDPF.Verify}(\pi_1, \pi_2, \pi_3)$

such that:

Correctness. Similar to the $(2, 2)$ *case, except that we use Shamir's reconstruction rather than mere group addition. Let* $y_b(x) = \mathsf{Eval}(b, f_b, x)$*, then:*

- $\mathsf{SS.Reconstruct}(y_1(x), y_2(x), y_3(x)) = \beta$ *for* $x = \alpha$*, and*
- $\mathsf{SS.Reconstruct}(y_1(x), y_2(x), y_3(x)) = 0$ *for all* $x \neq \alpha$*.*

Privacy. For every $b \in \{1, 2, 3\}$ *there exists a simulator* $\mathcal{S}$ *such that*

$$(f_b, \pi_{b'}, \pi_{b''}) \overset{c}{\equiv} \mathcal{S}(1^\kappa, b, n)$$

where $\{b, b', b''\} = \{1, 2, 3\}$*,* $(f_1, f_2, f_3) \leftarrow \mathsf{VDPF.Gen}(1^\kappa, \alpha, \beta)$ *and the distribution is over the coin tosses of algorithms* $\mathsf{VDPF.Gen}$ *and* $\mathcal{S}$*.*

Verifiability. Let $\boldsymbol{y}_x = \mathsf{SS.Reconstruct}(y_1(x), y_2(x), y_3(x))$*, then* $\mathtt{accept} = \mathsf{VDPF.Verify}(\pi_1, \pi_2, \pi_3)$ *iff* $\boldsymbol{y} = \boldsymbol{e}_{\alpha'} \cdot \beta'$ *for some* α', β'*.*

We note that in the above definition β as well as the outputs of algorithm Eval are drawn from a prime field F.

In Section 5.2.2 we present our construction for $(2, 3)$-VDPF, which uses $(2, 2)$-VDPF$^+$ from Section 5.2.1 as a building block. In Section 5.2.3 we provide a security proof for our VDPF construction.

5.2.1 Building Block: $(2,2)$-VDPF$^+$

One tool we use in order to construct our $(2,3)$-VDPF is a $(2,2)$-VDPF$^+$ (or an *enhanced* VDPF); its non-verifiable version was introduced in [148] and our addition of verifiability is straightforward, assuming we use (2,2)-VDPFs of [151] internally.

Definition 5.2.2. *A $(2,2)$-VDPF$^+$, denoted $F^{(2,2)}_{\alpha,\beta_0,\beta_1}$, is a $(2,2)$-VDPF, as defined in Section 2.6, with the following additional constraint: It must hold that $\mathsf{VDPF.Eval}(b, f_b, \alpha) = \beta_b$ for $b \in \{0,1\}$.*

The additional constraint ensures that at the special point α, party $b = 0$ receives β_0 and party $b = 1$ receives β_1. In Figure 5.4 we provide the construction for $F^{(2,2)}_{\alpha,\beta_0,\beta_1}$ from [148], given a 'normal' DPF construction. Note that the resulting VDPF is a normal VDPF for parameters α, β, where all individual evaluations are shifted by z. Then, obviously since both parties XOR their evaluation with z, this does not change the combined evaluation on any point (as it simply adds $z \oplus z = 0^l$; while on point $x = \alpha$, it causes f_0's (resp. f_1's) evaluation be β_0 (resp. β_1), which is easy to verify.

```
VDPF⁺.Gen($1^\kappa, \alpha, \beta_0, \beta_1$)
    1. $\beta \leftarrow \beta_0 \oplus \beta_1$
    2. $(f_0, f_1) \leftarrow \mathsf{VDPF.Gen}(1^\kappa, \alpha, \beta)$
    3. $z \leftarrow \beta_0 \oplus \mathsf{VDPF.Eval}(0, f_0, \alpha)$
    4. $f_b \leftarrow (z, f_b)$ for $b \in \{0,1\}$
    5. Output: $(f_0, f_1)$.
VDPF⁺.Eval($b, f_b, x$)
    1. $(z, f'_b) \leftarrow f_b$.
    2. $y \leftarrow \mathsf{VDPF.Eval}(b, f'_b, x) \oplus z$.
    3. Output: $y$
VDPF⁺.Prove($b, f_b$)
    1. $(z, f'_b) \leftarrow f_b$.
    2. $\pi'_b \leftarrow \mathsf{VDPF.Prove}(b, f'_b)$.
    3. Output $\pi_b = (\pi'_b, z)$
VDPF⁺.Verify($\pi_0, \pi_1$)
    1. Output accept iff $\pi_0 = \pi_1$.
```

Figure 5.4: Protocol for $(2,2)$-VDPF$^+$

5.2.2 Our $(2,3)$-VDPF Construction

Our $(2,3)$-VDPF construction is formally given in Figure 5.5.

We use two independent instantiations of a $(2,2)$-VDPF$^+$ (of Section 5.2.1) and output three keys f_1, f_2, f_3 where f_i is a pair of $(2,2)$-VDPF$^+$ keys, one from each instance. Specifically, denote the first and second $(2,2)$-VDPF$^+$ keys by (g_0, g_1) and (k_0, k_1), respectively, then our three $(2,3)$-VDPF keys are $f_1 = (g_0, k_0)$, $f_2 = (g_1, k_0)$ and $f_3 = (g_1, k_1)$. Evaluating f_i on input x is done by first interpreting it as the keys g and k of a $(2,2)$-VDPF$^+$; then, evaluating g and k independently on x and adding the results (over F_{2^l}).

The $(2,2)$-VDPF$^+$ instances above are adjusted so that evaluating g_0 and g_1 on α results with v_0 and v_1, respectively; similarly, evaluating k_0 and k_1 on α results with v_2 and v_3, respectively. It is required that the three values (1) $\beta_1 = v_0 \oplus v_2$, (2) $\beta_2 = v_1 \oplus v_2$ and (3) $\beta_3 = v_1 \oplus v_3$ be valid Shamir sharing of the target value $\beta_0 = \beta$; namely, that there is a degree-1 polynomial P s.t. $P(i) = \beta_i$ for $i \in \{0,1,2,3\}$. Fixing a random Shamir sharing $(\beta_1, \beta_2, \beta_3)$ of β, equations (1)-(3) above always have a solution (assignments to v_0, v_1, v_2, v_3); moreover, since the sharing is random, by drawing a random v_0 we get that the two values obtained by evaluating f_i (i.e., $(v0, v_2)$, (v_1, v_2) and (v_1, v_3)) are distributed uniformly in $F_{2^l} \times F_{2^l}$.

A subtle issue in the construction is that the shares $\beta_1, \beta_2, \beta_3$ are from a prime field F whereas v_0, v_1, v_2, v_3 are from F_{2^l}, thus, equations (1)-(3) above do not 'compile'. To reconcile that, we define the operation that takes an element x from a prime field F and an element y from a binary field F_{2^l}, 'embeds' x into F_{2^l} by simply using its binary representation to form x', and outputs $x' \oplus y$. In addition, for x and y as above, we define the operation $\odot$ that embeds y into F to form y' and outputs $x \cdot y'$ over F.

For these operations to work as expected, and to not raise a security concern, we must have that the binary representation of an element in F be well defined over F_{2^l} and that the arithmetic representation of an element in F_{2^l} be well defined over F. This is not true in general, however, we can pick a prime and binary fields for which the above almost always holds. Concretely, using $l = \kappa$ and F with a prime very close to 2^κ will achieve the desired result. In this manner, we get $\frac{2^\kappa - |F|}{2^\kappa} \approx 2^{-\kappa}$ and so values are drawn (either at random or as a result of a computation) from the gap between the fields only with negligible probability.

The above establishes that evaluation of keys f_1, f_2, f_3 on input $x = \alpha$ results with a Shamir sharing of the target value β. For completeness, we now show that evaluation on every other input (i.e., $x \neq \alpha$) results with a Shamir sharing of zero. Since the $(2,2)$-VDPF$^+$ is defined over a binary field F_{2^l}, evaluation of g_0 and g_1 (respectively of k_0 and k_1) on $x \neq \alpha$ results with the same value (so adding them results with 0^l). Thus, evaluation of $f_i = (g_{i_g}, k_{i_k})$ ($i \in \{1,2,3\}, i_g, i_k \in \{0,1\}$) results with the same value for all i's; let that value be y. Then, an interpolation using the points $(1, y), (2, y), (3, y)$ results with an horizontal line at height y, which leads to the secret y. To get a secret 0 instead, we multiply the result, y, by the index of the key, which now results with the points $(1, y), (2, 2y), (3, 3y)$, resulting with a line that crosses through $(0,0)$ and so hides the secret 0.

Finally, that multiplication by i requires fixing the target values we give to the VDPF$^+$ instances: instead of working with the shares $\beta_1, \beta_2, \beta_3$, we work with the values $\beta'_1 = \beta_1 \cdot 1^{-1} = \beta_1, \beta'_2 = \beta_2 \cdot 2^{-1}$ and $\beta'_3 = \beta_3 \cdot 3^{-1}$.

5.2.3 Security analysis

Correctness can be verified from the protocol, in the following we prove verifiability and privacy.

Theorem 5.2.3. *The construction in Figure 5.5 is a VDPF (Def. 5.2.1), secure against a malicious client and a single malicious server.*

Proof. Verifiability. We show that if the verification function outputs `accept` then the functions f_1, f_2, f_3 are well formed (e.g., they evaluate to a Shamir sharing of zero at all points except of at most one, where they evaluate to a non-zero value). Since the VDPF$^+$ outputs `accept` in both verifications, this means that the functions (g_0, g_1) and (k_0, k_1) are well formed (g_0, g_1, k_0 and k_1 have only one non-zero value).

Denote by $y_{g_b}(x)$ (resp. $y_{k_b}(x)$) the result of evaluation of g_b (resp. k_b) at point x. The above guarantees that y_{g_0} and y_{g_1} differ on at most one point. Let that point be α_g and denote the evaluation by $y_{g_0} = \beta_{g,0}$ and $y_{g_1} = \beta_{g,1}$, respectively. Similarly, let α_k be the point at which y_{k_0} and y_{k_1} differ, and denote the evaluation by $y_{k_0} = \beta_{k,0}$ and $y_{k_1} = \beta_{k,1}$, respectively.

We show that if $\alpha_g \neq \alpha_k$ then our verification rejects. Assume $\alpha_g \neq \alpha_k$, that the parties partially exchanged $H(t_b)$, where H is a collision-resistant hash function, before exchanging the rest of their proof (this serves as a commitment and is needed for malicious servers), and that the verification procedure accepted. This implies that the vector $\boldsymbol{y}$ has two non-zero positions, as it the result of adding (over) two vectors with a single non-zero value. Assume these values are β_{α_g} and β_{α_k}. From this, we get that the calculation of t in line 7 of Prove yields: $t = (\beta_{\alpha_g} u_{\alpha_g} + \beta_{\alpha_k} u_{\alpha_k})^2 \quad - (\beta_{\alpha_g} + \beta_{\alpha_k}) \left(\beta_{\alpha_g} u^2_{\alpha_g} + \beta_{\alpha_k} u^2_{\alpha_k} \right) = 0$. This is since we assumed the verification accepts. After re-arranging, this equation is simplified to $(u_{\alpha_g} - u_{\alpha_k})^2 = 0$, which implies $u_{\alpha_g} = u_{\alpha_k}$, in contradiction.

Privacy Per Definition 5.2.1, the view of each server b is: $(f_b, \pi_{b'}, \pi_{b''})$, where f_b consists of two different VDPF+ keys, each of which is further composed of a (2,2) VDPF key and a value z. The underlying (2,2) VDPF keys are all indistinguishable from random by the privacy of those VDPF keys.

It is also easy to see that for each VDPF+ key, z is pseudorandom, given that our selection of $v_0, v_1, v_2, v3$ is pseudorandom. However, recall that our construction evaluates two (2,2)-VDPF+ keys, and then adds (over) their result together. At every point $x \neq \alpha$, we are adding two pseudorandom shares together, but at the point $x = \alpha$, by construction, we obtain $\beta'_b \in F$. We therefore need to ensure that β'_b is indistinguishable from a random in . Otherwise, an adversary controlling b could run a full domain evaluation of its key, and look for a value that is distinguishable from all others (which are pseudorandom in). This potentially leaks both β'_b and α, so the simulation would fail.

Thus, to conclude the proof, we focus on showing that β'_b is indistinguishable from a random in . W.l.o.g., we will assume we are looking at the view of party $b = 0$, which holds β'_0. We note that the analysis for party $b = 1$ and $b = 2$ is similar. First, observe that β'_0 itself is random in F, from the security of Shamir sharing. Then, by our assumption, the probability that a randomly generated value falls in the gap between the two fields is: $\frac{||FF|-2^\kappa|}{2^\kappa} \approx 2^{-\kappa}$, which implies that β'_0 also appears random in , since $l = \kappa$ as well.

□

5.2.4 (2,3)-VDPF with sublinear PRF calls

The first application of our (2,3)-VDPF construction is to build, in a black-box manner, a (2,3)-VDPF with sublinear PRF calls (the main computational bottleneck in evaluating DPFs). Note that in this construction, calling Eval on x produces a degree-2 sharing of $y = F(x)$. This has two implications: first, is that this construction can still withstand malicious clients (due to verifiability), but only semi-honest servers; second, we lose our one *free* multiplication, so we can no longer combine PIR with PIW in a simple way, similar to (2,2) VDPFs. In other words, this DPF is a potential (faster) drop-in replacement for applications using (2,2) VDPFs (e.g., [118], [120], [121], [124]), designed for the three-party model.

Our construction is described in Figure 5.6. The main idea is to re-interpret each DPF as a square matrix (instead of a vector), where α is defined by $\alpha_{row}, \alpha_{col}$. Then, for a DPF with domain of size $n = |D|$, we can share two DPFs that have a domain of size $\sqrt{n}$ instead. One for the row and one for the column. A similar idea was used in prior works (e.g. [148], [150]). To perform a full-domain evaluation efficiently, we can first run a full-domain evaluation on the row and column DPFs. We then obtain two one-hot-vectors, which we use to expand the full DPF. This is done by taking each value in the row vector, and multiplying it with each value in the column vector. It is easy to see that the result is a sharing of a vector $\boldsymbol{y}^n$, with a sharing of β in index $\alpha_{row} \cdot \sqrt{n} + \alpha_{col} = \alpha$, and 0 everywhere else. The full construction is described in Figure 5.6. We only show Eval in its full-domain version, to illustrate that we only need $O(\sqrt{n})$ PRF calls[5].

[5]We remark that for security, each party should additionally re-randomize its final vector share, but in practice this can be deferred until there is a communication round for greater efficiency.

```
VDPF.Gen$(1^\kappa, \alpha, \beta)$
    1. Parse $\alpha$ as $(\alpha_i, \alpha_j)$, which represents the (row, column) index of $\alpha$ in a $(\sqrt{n}, \sqrt{n})$-square matrix.
    2. $(g_1, g_2, g_3) \leftarrow$ VDPF.Gen$(1^\kappa, \alpha_i, 1)$
    3. $(k_1, k_2, k_3) \leftarrow$ VDPF.Gen$(1^\kappa, \alpha_j, \beta)$
    4. Output: $((g_1, k_1), (g_2, k_2), (g_3, k_3))$.
VDPF.Eval$(b, f_b)$
    1. $(g_b, k_b) \leftarrow f_b$.
    2. $\boldsymbol{y_g} \leftarrow$ VDPF.Eval$(b, g_b)$.
    3. $\boldsymbol{y_k} \leftarrow$ VDPF.Eval$(b, k_b)$.
    4. $\boldsymbol{y} \leftarrow \{\}$.
    5. For every $y_i \in \boldsymbol{y_g}$:
        • $\forall y_j \in \boldsymbol{y_k}$: $\boldsymbol{y} \leftarrow \boldsymbol{y} \cup \{y_i \cdot y_j\}$.
    6. Output: $\boldsymbol{y}$
VDPF.Prove$(b, f_b)$
    1. $(g_b, k_b) \leftarrow f_b$.
    2. Output: $\pi_b = (VDPF.Prove(b, g_b), VDPF.Prove(b, k_b))$
VDPF.Verify$(\pi_1, \pi_2, \pi_3)$
    1. Parse each proof as the two underlying proofs and verify them both. Output accept if both accept.
```

Figure 5.6: Protocol for $(2, 3)$-VDPF

5.3 ORAM with Policy

We define an *ORAM with Policy* functionality, which our CBDC protocol below instantiates. We note that this functionality may have other applications as well, but we do not explore them in this work. The main functionality is described in Figure 5.7.

5.3.1 Additional Functionalities

We define the remaining functionalities we use in this chapter: $\mathcal{F}$.Product and $\mathcal{F}$.SoP in Figures 5.8 and 5.9, respectively.

5.4 Account-based Digital Currency

A specific application of the functionality F_{APORAM} is a privacy-preserving account-based digital currency system. We maintain that our construction is specifically relevant to CBDCs, which are growing in popularity [137], [141]–[144]. CBDCs have the potential to greatly improve the efficiency of our financial system, as well as assist with preventing fraud and money-laundering. On the other hand, this raises concerns around individual privacy [138], [139], [145], [146]. Thus, a system like ours that can achieve both is highly desirable.

We note that systems that distribute trust to two (or three) parties, such as Prio [118], have seen real-world deployment at scale [6]. Similarly, a Central Bank can use our solution to build a private CBDC by distributing its management to three different trust zones (e.g., by having the parties be the Central Bank itself, the Department of Treasury, and a nonprofit focused on privacy rights[7]). Other deployment models are possible, as we explore in Appendix **??**.

We formalize the CBDC application using the AP-ORAM functionality (See Functionality 5.7), with the following parameters: The number of locations to read is $m_{\mathsf{read}} = 1$ and the number of locations to update is $m_{\mathsf{update}} = 2$; these values replace the single parameter m used in the functionality. In addition, the PVerify parameter function is defined by:

$$\mathsf{PVerify}(c_i, \mathsf{read}, \ell_1) = \begin{cases} \texttt{accept} & \text{if } i = \ell_1 \\ \texttt{reject} & \text{otherwise} \end{cases} \tag{5.1}$$

$$\mathsf{PVerify}(c_i, \mathsf{update}, \ell_1, v_1, \ell_2, v_2) = \begin{cases} \texttt{accept} & \text{if } i = \ell_1 \text{ and } v_1 = v_2 \text{ and } D_i \geq v_1 \\ \texttt{reject} & \text{otherwise} \end{cases} \tag{5.2}$$

where D_i is part of the state held by the functionality, see Figure 5.7. That is, the procedures of interest in a CBDC application are "moving" funds from one account to the other (a *transfer*), or reading a balance. Reading a balance is allowed to the account owner and so the function verifies that the client ID (index) matches the row to be read. Moving funds is translated into two memory updates, the first subtract the value in one entry and the second increase the value in another (possibly same, if one is paying itself) entry. It is required that one can pay only from its own account, and so the same matching verification is done as in the balance check procedure; additionally, it is required that the subtracted value at the payer's account and the added value at the payee account are equal. In the rest of this work, we denote $\mathcal{F}_{\mathsf{AP-ORAM}}$ with the above parameters by $\mathcal{F}_{\mathsf{CBDC}}$.

5.4.1 Registration & Access Control for DPFs

Recently, Servan-Schreiber et al., [120], [152] presented a mechanism called *private access control lists* (PACL) to verify that a private access to a database (or a vector) D succeeds

[6] https://blog.mozilla.org/en/products/firefox/partnership-ohttp-prio

[7] For example, EFF: https://www.eff.org

only if the client requesting that access is permissioned. The privacy of such access, as in our case, is provided via a DPF which enables hiding the location and the value to be written (in case of an update). One drawback of their scheme is that it requires the servers to perform $O(N)$ ($N = |D|$) public-key operations per check, as each server requires to compute an inner-product in-the-exponent. Although their scheme incurs only $O(1)$ communication, performing so many public-key operations per check (in case N is large) is prohibitive.

Instead, we make the observation that since we are in an honest-majority setting, we can modify their protocol to use secret-shared values (instead of values in the exponent) and still compute inner-products with $O(1)$ complexity, and with cheap information-theoretic operations, using the sum-of-products trick (e.g., [150], [167]). We describe the protocol and notation below. Security follows from the same arguments as in [152] and the security of sum-of-products.

The registration procedure, as described in Figure 5.10, is very simple: each new client receives a random value, λ, from the servers; this random value is known only to the client (and is secret shared at the servers). This value is used by the client each time it wishes to access the servers. The servers maintain a database, denoted Λ, for those secret shared values $[\lambda_1], [\lambda_2], \ldots$, where λ_i is known to client c_i.

Access control for CBDC. In the CBDC application, the i-th entry of the database D 'belongs' to the i-th client, c_i, which means that c_i is the only one who can read it, and decrease the value stored there (up to zero). Suppose that client c_i wants to read D_i (recall that all entries in D are secret shared by the servers). Client c_i now sends a new secret sharing of λ_i, namely $([\lambda_i]_1, [\lambda_i]_2, [\lambda_i]_3)$, to the servers, as well as a VDPF (f_1, f_2, f_3) that encodes the value 1 at location i. This way, the servers can evaluate the DPF to obtain a new shared database, denoted T, that hides 1 at location i and 0 elsewhere. The servers can compute the dot-product between Λ and T, which results with the secret authentication value stored for the client at location i, call that value $\tilde{\lambda}_i$. If the client indeed has access to location i then it must hold that λ_i that is shared by the client at the time of the protocol is equal to $\tilde{\lambda}_i$ that is already stored at location i in Λ. This is a proof that the request sender knows the required authentication secret for some location i that is 'one-hot' encoded in the database T. Then, computing a dot-product between T and D results with the balance b of that exact location that the client proved it knows the authentication secret for.

5.4.2 The CBDC Protocol

The protocol is described in Figures 5.10-5.12. If the underlying MPC protocols realizing the well-known functionalities we use (e.g., $\mathcal{F}.\mathsf{Mult}, \mathcal{F}.\mathsf{CheckZero}, \mathcal{F}.\mathsf{Product}$) are maliciously secure (resp. semi-honest), then our entire protocol protects against malicious servers (resp. semi-honest). Except for the protocol $\mathcal{F}.\mathsf{Product}$, we can use semi-honest protocols and turn them maliciously-secure by running them over authenticated inputs (i.e., using MACs) [159], [168], which is exactly what we do in practice. However, doing so for $\mathcal{F}.\mathsf{Product}$ would require $O(N)$ communication per requrest, which is prohibitively expensive. We solve that problem via a novel maliciously-secure protocol for a dot-product between DPFs (see Section 5.5).

The protocol is described in the ORAM language, with Read and Update requests. In the CBDC context, the Read request is actually only used by a client c_i to get its balance.

Since the client does not reveal which database entry it reads, yet it proves (using the authentication secret described above) that it is allowed to read that entry, this action is anonymous. This is important, as it reduces the attack surface of network correlation attacks that may exist if requesting the balance would not be anonymous. Then, the Update request is actually only used by a client in order to pay another client. For a client c_i to pay amount v to another client c_j, the client runs the ORAM protocol with inputs (ℓ_1, v_1) and (ℓ_2, v_2) with $\ell_1 = i, \ell_2 = j$ and $v_1 = v_2 = v$. Security against a corrupted server follows the fact that the underlying building blocks and functionality are secure; a formal simulation based proof is given below. Security against a corrupted client follows the authentication technique described above, which we briefly expound on now. The client sends two VDPFs, each is verifiable on its own and therefore it is guaranteed that each represents a vector with at most one entry that is non-zero. To verify that the two non-zero values are the same, the hidden values are 'extracted' (these are values a and b) and their difference is later handed to $\mathcal{F}.\mathsf{CheckZero}$. Note that in contrast to the Read operation, where the client provided a sharing of its authentication secret λ, in Update the client provides a sharing of $\lambda \cdot v_1$ (i.e., the authentication secret times the amount to transfer). Then, the protocol obtains another instance of that value by computing a dot-product between Λ and f (the vector shared by (f_1, f_2, f_3)), and again, the difference between these results is checked using $\mathcal{F}.\mathsf{CheckZero}$. Finally, to check that the amount to transfer is no greater than the balance of the user, the protocol computes v' – the user's balance times the amount to transfer, and a_2 the amount to transfer squared. Obviously, it must hold that the latter is no greater than the former, which is checked via a call to $\mathcal{F}_{\mathsf{LTE}}$. We note that because of that check, it is must be ensured that the balances in the system do not exceed the square root of the underlying field size (i.e. $\sqrt{|F|}$).

Security An intuition to the security of the protocol was given above; in this section we give a formal simulation-based proof of security. We prove the following:

Theorem 5.4.1. *Given a verifiable enhanced distributed point function scheme* VDPF^+*, protocol* $\Pi = (\Pi_{\mathsf{CBDC}}.\mathsf{register}, \Pi_{\mathsf{CBDC}}.\mathsf{read}, \Pi_{\mathsf{CBDC}}.\mathsf{update})$ *(from Figures 5.10-5.12) securely computes* $\mathcal{F}.\mathsf{CBDC}$ *(Figure 5.7 with* $\mathsf{PVerify}$ *from Equations 5.1-5.2) in the* $\mathcal{F}_{\mathsf{X}}$*-hybrid model, for all* $\mathsf{X} \in \{\mathsf{Rand}, \mathsf{Zero}, \mathsf{Mult}, \mathsf{Product}, \mathsf{CheckZero}, \mathsf{LTE}\}$.

Proof. We show a proof against a malicious adversary. Let be an adversary who corrupts server S_c ($c \in \{1, 2, 3\}$) and any subset of the clients. We present a simulator $\mathcal{S}$ that runs internally and produce's a simulated adversarial view and output set associated with the honest parties (the servers S_{h_1}, S_{h_2} s.t. $\{h_1, h_2, c\} = \{1, 2, 3\}$). The resulting adversarial view and the honest parties' outputs are computationally indistinguishable from those in the real execution of the protocol.

In the following we simulate the commands as if the adversary controls the client as well, and later we discuss what is changed in the simulation in case the client is honest. Whenever the simulator halts in the below description, it sends abort to the $\mathcal{F}_{\mathsf{CBDC}}$ functionality, with which it interacts.

Register. On input $(c, \mathsf{register})$, if $\mathsf{ctr} = N$ send `full` to the client and halt, otherwise increment ctr and continue. Simulate $\mathcal{F}.\mathsf{Rand}$ by computing $[\lambda] \leftarrow \mathsf{Share}_{2,3}(\lambda)$ for a uniform

$\lambda \in F$, hand $[\lambda]_c$ to S_c and $[\lambda]_{h_1}, [\lambda]_{h_2}$ to the client.

Read. Given, $f_{h_1}, f_{h_2}, [\lambda]_{h_1}, [\lambda]_{h_2}$, simulate $\mathcal{F}.\mathsf{Zero}$ by computing $[z] \leftarrow \mathsf{Share}_{2,3}(0)$ and hand $[z]_c$ to S_c; then, do exactly as in the protocol, for $b \in \{1, 2\}$:

- Compute $T^{h_b} \leftarrow \mathsf{VDPF}^+.\mathsf{Eval}(h_b, f_{h_b})$.
- Compute $[t]_{h_b} = \sum k = 1^N T_k^{h_b}$.
- Compute $\pi_{h_b} = \mathsf{VDPF}^+.\mathsf{Prove}(h_b, f_{h_b}, [z]_{h_b})$, send π_{h_b} to S_c and receive π_c from S_c.

Then, simulate $\mathcal{F}.\mathsf{Product}$ by receiving $[\Lambda]_c$ and f_c from S_c, then, halt if $[\Lambda]_c$ together with $[\Lambda]_{h_1}$ and $[\Lambda]_{h_2}$ do not form a valid Shamir sharing for a vector (note that the secrets in Λ are completely determined by $[\Lambda]_{h_1}$ and $[\Lambda]_{h_2}$ and so the secrets could not be adversarially changed), or f_c together with f_{h_1} and f_{h_2} do not form a valid point function (there exists at most one entry α at which the shared value is non-zero, and all entries form valid Shamir sharings), or $\mathsf{reject} = \mathsf{VDPF}^+.\mathsf{Verify}(\pi_1, \pi_2, \pi_3)$. Otherwise (the above checks pass), compute $\tilde{\lambda} = \Lambda \cdot T$ and hand $[\tilde{\lambda}]_c$ to S_c where $[\tilde{\lambda}] \leftarrow \mathsf{Share}_{2,3}(\tilde{\lambda})$. Simulate the first instance of $\mathcal{F}.\mathsf{CheckZero}$ by receiving $[t]_c$ from S_c, checking that it is consistent with $[t]_{h_1}, [t]_{h_2}$ computed above and verifying that $t = 1$. Similarly, simulate the second instance of $\mathcal{F}.\mathsf{CheckZero}$ by receiving $[\lambda - \tilde{\lambda}]_c$ from S_c, checking that it is consistent with $[\lambda - \tilde{\lambda}]_{h_1}, [\lambda - \tilde{\lambda}]_{h_2}$ and verifying that $\lambda - \tilde{\lambda} = 0$. Finally, simulate $\mathcal{F}.\mathsf{Product}$ by computing $b = D \cdot T$ and $[b] \leftarrow \mathsf{Share}_{2,3}(b)$, then sending $[b]_{h_1}, [b]_{h_2}$ to the client and $[b]_c$ to S_c.

Update Given, $f_{h_b}, g_{h_b}, [\lambda]_{h_b}$ for $b \in \{1, 2\}$, simulate $\mathcal{F}.\mathsf{Zero}$ by computing $[z] \leftarrow \mathsf{Share}_{2,3}(0)$, $[z'] \leftarrow \mathsf{Share}_{2,3}(0)$, and hand $[z]_c, [z']_c$ to S_c; then, do exactly as in the protocol, for $b \in \{1, 2\}$:

- Compute $F^{h_b} \leftarrow \mathsf{VDPF}^+.\mathsf{Eval}(h_b, f_{h_b})$
  and $G^{h_b} \leftarrow \mathsf{VDPF}^+.\mathsf{Eval}(h_b, g_{h_b})$.
- Compute $[a]_{h_b} = \sum k = 1^N F_k^{h_b}$ and $[b]_{h_b} = \sum k = 1^N G_k^{h_b}$.
- Simulate $\mathcal{F}.\mathsf{Mult}$ by receiving $[a]_c$ from S_c; if it is consistent with the $[a]_{h_b}$'s then compute $[a_2] \leftarrow \mathsf{Share}_{2,3}(a^2)$ and hand $[a_2]_c$ to S_c.
- Compute $\pi_{h_b} = \mathsf{VDPF}^+.\mathsf{Prove}(h_b, f_{h_b}, [z]_{h_b})$ and $\pi'_{h_b} = \mathsf{VDPF}^+.\mathsf{Prove}(h_b, g_{h_b}, [z']_{h_b})$, send (π_{h_b}, π'_{h_b}) to S_c and receive π_c, π'_c from S_c.
- Simulate $\mathcal{F}.\mathsf{Product}$ twice by receiving $[\Lambda]_c$ and f_c from S_c, then, halt if $[\Lambda]_c$, when combined with the $[\Lambda]_{h_b}$'s, does not form a valid Shamir sharing for a vector, or f_c together with f_{h_1} and f_{h_2} do not form a valid point function, or reject is returned when computing $\mathsf{VDPF}^+.\mathsf{Verify}(\pi_1, \pi_2, \pi_3)$ or $\mathsf{VDPF}^+.\mathsf{Verify}(\pi'_1, \pi'_2, \pi'_3)$. If not halted, compute $[\lambda'] \leftarrow \mathsf{Share}_{2,3}(\Lambda \cdot F)$ and $[v'] \leftarrow \mathsf{Share}_{2,3}(D \cdot F)$, and hand $[\lambda']_c$ and $[v']_c$ to S_c.

Simulate the first instance of $\mathcal{F}.\mathsf{CheckZero}$ by receiving $[\lambda - \lambda']_c$ from S_c, checking that it is consistent with the $[\lambda - \lambda']_{h_b}$'s and verifying that $\lambda - \lambda' = 0$. Then, simulate the second instance of $\mathcal{F}.\mathsf{CheckZero}$ by receiving $[a - b]_c$ from S_c, checking that it is consistent with the $[a-b]_{h_b}$'s computed above and verifying that $a-b=0$. Finally, simulate $\mathcal{F}.\mathsf{LTE}$ by receiving $[a_2]_c, [v']_c$ from S_c, checking that they are consistent with the $[a_2]_{h_b}$'s and $[v']_{h_b}$'s, and verifying that $a_2 \leq v'$ (which is equivalent to verifying that $a \leq D_i$). If any of the above verifications fail then halt, otherwise, update $D^{h_b} = D^{h_b} - F^{h_b} + G^{h_b}$ send Ok to the client.

The resulting view of the adversary and the output for the honest servers are perfectly simulated, that is, these views under the simulation and in the real execution of the protocol are identically distributed. We note that this is a perfect simulation even though the DPF construction is only computationally secure; this is due to the fact that when the client is corrupted then the adversary itself produces it, and so the VDPF's simulator does not come into play.

The probability that the adversary's client succeeds in submitting a malformed VDPF and still pass the verification, or pass the authentication verification without submitting a correct $\lambda = \Lambda_i$ (for some $i \in \{1, \ldots, N\}$ equals in the simulation and real execution, and are both negligible in κ (the former is computationally negligible and the latter is statistically negligible).

Simulating an honest client. Here we use the VDPF's simulator (see Definition 5.2.1). The only difference between this case and the above (when the client is under the control of) is that now $\mathcal{S}$ has to produce f_c in the simulation of read (or f_c, g_c in the simulation of update). This is done by invoking the VDPF's simulator, and then sending S_c the simulated point function's share $f_c \leftarrow \mathsf{VDPF}^{+}.\mathcal{S}(1^\kappa, c, N)$. Combining with the rest of the simulation, the views under real execution and simulation become computationally indistinguishable (rather than identical as they were in the case the client was corrupted). It is easy to see that we can reduce the security of protocol Π to that of the VDPF^{+} construction. □

5.5 Efficient, Malicious Dot-Product

The functionality $\mathcal{F}.\mathsf{Product}$ (Figure 5.8) is an important one in the above CBDC protocol; it receives shares of a vector $[V]$ and a point functions $f = (f_1, f_2, f_3)$ (alternatively, it can receive two point functions f, g and expand g into $[V]$) from the parties, and returns the dot product of V and F, where F is the result of a full-domain evaluation of f, that is the sharings $[F] = [F_1], \ldots, [F_N]$.

A naive implementation of $\mathcal{F}.\mathsf{Product}$ would call $\mathcal{F}.\mathsf{Mult}$ on the pairs $([V_i], [F_i])$ for every $i \in \{1, \ldots, N\}$. This, however, incurs $O(N)$ communication between the parties, a cost we highly wish to avoid (otherwise the protocol could not scale well with the number of clients). Our goal is to achieve a secure implementation of $\mathcal{F}.\mathsf{Product}$ with communication sub-linear in N, namely, with $O(\log N)$ or even constant communication. In this section we show how to do that using a new primitive we call *updatable VDPF* (or UVDPF). An updatable VDPF allows the parties who already hold some VDPF $f = (f_1, f_2, f_3)$ for some point function $F_{\alpha,\beta}$, to update the target value at entry α; that is, to produce an updated VDPF $f' = (f'_1, f'_2, f'_3)$ for the point function $F_{\alpha,\beta'}$ for some β' that is also secret shared by them. If an implementation of that primitive can be done in sub-linear communication in N, then so can the dot product. We remark that for simplicity and better generalization, we present a protocol that only has black-box access to (2,2)-VDPFs. This protocol has

$O(\log N)$ communication. For our implementation, we make an optimization that achieves $O(1)$ communication but without black-box access.

In Section 5.5.1 we formalize the functionality for a UVDPF, for both the $(2, 2)$ and the $(2, 3)$ cases. Then, in Section 5.5.2 we show how to use a UVDPF to implement the dot-product functionality.

5.5.1 Updatable $(2, 3)$-VDPF

In Figure 5.13 we present the updatable $(2, 2)$-VDPF$^+$ functionality, denoted $\mathcal{F}^+_{(2,2)-\mathsf{UVDPF}}$. A secure implementation of (a slightly different version) of that functionality was proposed in [147] (we note that in that paper this primitive is called 'deferred DPF').

Then, in Figure 5.14 we present the $(2, 3)$-threshold variant of that functionality, denoted, $\mathcal{F}^+_{(2,3)-\mathsf{UVDPF}}$. Finally, using $\mathcal{F}^+_{(2,2)-\mathsf{UVDPF}}$, we construct, in Figure 5.15, a protocol to securely compute $\mathcal{F}^+_{(2,3)-\mathsf{UVDPF}}$.

The $(2, 2)$-updatable VDPF$^+$ functionality (Figure 5.13), is a two-party functionality that, given VDPF$^+$ shares f_0 from the first party and f_1 from the second party, for point function $F_{\alpha,\beta_0,\beta_1}$, and the sharings of new target values $[\beta'_0], [\beta'_1]$ (these are $(2, 2)$ sharings), outputs updated shares f'_0 to the first party and f'_1 to the second party for a new point function $F_{\alpha,\beta'_0,\beta'_1}$.

Similarly, the $(2, 3)$-updatable VDPF$^+$ functionality (Figure 5.14), is a *three-party* functionality that, given VDPF$^+$ shares f_1, f_2, f_3 from P_1, P_2, P_3, respectively, for a point function $F_{\alpha,\beta}$, and the sharings of a new target value $[\beta']$ (a Shamir $(2, 3)$ sharing), outputs updated shares f'_1, f'_2, f'_3 to point function $F_{\alpha,\beta'}$.

We note that the three-party functionality gives the adversary the opportunity to abort whereas the two-party one does not. This is due to the fact that our realization of the three-party functionality can work with a protocol for the two-party functionality that is only secure against semi-honest adversaries, as a verification is performed in the three-party protocol.

The construction begins when the servers hold shares for a $(2, 3)$-VDPF, which, in our construction each share is essentially composed of two $(2, 2)$-UVDPF$^+$ shares (note that our construction in Figure 5.5 uses VDPF, however, the same construction could work with an updatable VDPF). Specifically, there are two UVDPF$^+$s that are already shared, namely, (f_0, f_1) encode $(\alpha, \beta_0, \beta_1)$ and (g_0, g_1) encode $(\alpha, \gamma_0, \gamma_1)$. Recall that each party P_i $(i \in \{1, 2, 3\})$ has shares of two $(2, 2)$-UVDPF$^+$s, that is, P_1 has (f_0, g_0), P_2 has (f_1, g_0) and P_3 has (f_1, g_1). It holds that $(\beta_0 \oplus \gamma_0), (\beta_1 \oplus \gamma_0) \odot 2, (\beta_1 \oplus \gamma_1) \odot 3$ form a valid Shamir sharing of β.

Let δ be the value that the parties wish to plug at location α instead of β, we assume that the parties hold the $(2, 3)$-sharing $[\delta]$. The parties' goal is to obtain the sharings $[\beta'_0], [\beta'_1], [\gamma'_0], [\gamma'_1]$ s.t.

$$[\delta]_1 = (\beta'_0 \oplus \gamma'_0),\ [\delta]_2 = (\beta'_1 \oplus \gamma'_0) \odot 2, [\delta]_3 = (\beta'_1 \oplus \gamma'_1) \odot 3 \qquad (5.3)$$

form a valid Shamir sharing of δ, and hand those sharings to the two instances of $\mathcal{F}_{(2,2)-\mathsf{UVDPF}^+}$. That is, the first (f_0, f_1) should be updated with $[\beta'_0], [\beta'_1]$ and (g_0, g_1) should be updated with $[\gamma'_0], [\gamma'_1]$. In protocol $\Pi_{(2,3)-\mathsf{UVDPF}^+}$ (Figure 5.15) the parties obtain the appropriate

$(2, 2)$-sharings of $[\beta'_0], [\beta'_1], [\gamma'_0], [\gamma'_1]$ that are later fed to the two instances of $\mathcal{F}_{(2,2)-\mathsf{UVDPF}^+}$ and obtain the new shares of the $(2, 3)$-VDPF$^+$.

5.5.2 The Dot-Product Protocol

Given a $(2, 3)$-UVDPF, the construction of our maliciously secure dot-product is presented in Figure 5.16. It accepts a vector and a DPF, expands the DPF, and computes a sharing of their dot product. We note that the protocol relies on an additional functionality, $\mathcal{F}.\mathsf{SoP}$ (Figure 5.9) for computation of sum-of-products. We note that $\mathcal{F}.\mathsf{SoP}$ only needs to be a semi-honest functionality that allows the adversary to inject an additive error to the result, but this is not of concern in our larger protocol as we achieve the MAC'ed result using another invocation and compare the results. Since the MAC value is random and independent, the adversary cannot inject additive errors to both results such that they match (i.e., one is a MAC'ed version of the other). This technique was used in previous works for maliciously secure MPC (e.g., [159]).

5.6 Implementation, Evaluation and Applications

In this section, we implement prototypes of our VDPF constructions and a basic three-server ORAM scheme that builds on top of it. We then use these tools to construct our private Π_{CBDC} protocol presented in Figures 5.10, 5.11, 5.12. Our code is written in C++ and consists of approximately 7,000 lines of new code[8]. Our DPF implementation leverages the highly efficient (2,2)-DPF implementation from [169], but we extend their code[9] to allow for larger than 1-bit DPFs.

We ran all benchmarks on a single Azure Standard E32s (v5) VM server, which has 16 physical cores (32 vCPUs) and 256GB of RAM. We simulated network latency and bandwidth using the *tc* command.

In addition, to further illustrate the applicability of our VDPF construction, we sketch in Section 5.6.3 how it can be extended to the application of building a Distributed ORAM (DORAM), and qualitatively compare it with the state-of-the-art.

5.6.1 DPFs

We implement both verifiable and non-verifiable versions of the state-of-the-art (2,2)-DPF from [20], and our two (2,3) DPF constructions (Sections 5.2.2, 5.2.4). We also compare analytically to the (2,3)-DPF from [148].

For all of these constructions, we measure the time it takes to evaluate the entire domain for different domain sizes (shown in Figure 5.17), as well as key sizes (in Table 5.2). As expected, our constructions have key sizes that are 2x bigger than the baseline[10], but this is marginal even for very large domains ($\sim 2KB$ keys for $N = 2^{50}$). Similarly, for evaluation, our (2,3)-DPF construction is only 2x slower than the baseline, and similarly, the VDPF

[8]An anonymized version of the code is attached to this submission.

[9]https://github.com/dkales/dpf-cpp

[10]A (2,3) DPF has two (2,2) keys, and a (3,3) DPF has 4 half-sized such keys.

construction has an additional estimated 2x overhead, since we are effectively evaluating another level.

Our (2,3)-DPF from Section 5.2.4 is $\sim 2.2\times$ faster for $n > 2^{20}$ compared to a (2, 2)-DPF. We estimate that this gap will increase (in our favor) using GPUs, for two reasons: (i) AES is hardware-accelerated in CPU, but not in GPU; (ii) In contrast, GPUs are massively parallel and can perform vector operations well.

	$\mathbf{2^{20}}$	$\mathbf{2^{30}}$	$\mathbf{2^{40}}$	$\mathbf{2^{50}}$
(2,2)-VDPF [20]	357	537	717	897
(2,3)-VDPF [148] (Analytical)	51024	120804	188664	289824
(2,3)-VDPF (Section 5.2.2)	850	1210	1570	1930
(2,3)-VDPF (Section 5.2.4)	980	1340	1700	2060

Table 5.2: Comparing key sizes (in bytes) of our constructions compared to the baseline (2,2) DPF of [117] and (2,3) DPF of [148]. Since [148] does not provide an implementation, we provide an analytical estimate.

5.6.2 Account-based Privacy-preserving Cryptocurrency and CBDCs

We now benchmark our main CBDC protocol. Our results support our hypothesis that we can scale to large anonymity sets, including those exceeding a million accounts. This vastly surpasses previous systems, which only provided a k-anonymity set between 10-256. We benchmark our system against Solidus [131], the closest existing model, which also uses an ORAM for privacy in an account-based ledger and marks the current state-of-the-art.

We conducted end-to-end transaction tests comparing Solidus and our protocols (semi-honest and malicious $\Pi_{CBDC}.Update$ protocols). To create a fair comparison, we optimized Solidus parameters to leverage all available cores and system memory. Our findings, detailed in Figure 5.18, reveal that our protocols significantly outperform Solidus in transaction throughput and memory efficiency for up to $N = 2^{18}$ accounts, and continue to outperform it up to $N = 2^{22}$ accounts. While Solidus manages to close the gap at $N = 2^{24}$ accounts, it fails to offer complete anonymity like our system and it is less efficient in memory usage. Beyond $N = 2^{24}$, Solidus exhausts memory, whereas our protocols remain scalable.

Additionally, we believe our system has the potential for horizontal scaling across multiple servers, as it exhibits minimal bandwidth requirements and DPF evaluations and inner product computations (which are the bottlenecks) could be run in parallel. Furthermore, end-to-end latency, as shown in Figure 5.19, remains low across various network conditions, with acceptable delays even on slower networks.

5.6.3 Three-party Distributed ORAM (DORAM)

A closely related problem to an ORAM scheme, which our CBDC protocol above implements, is that of a Distributed ORAM (DORAM). DORAM is a fundamental building block in

MPC, as it allows securely running RAM programs directly, as opposed to converting them first to circuits, which can yield meaningful performance gains. Thus, a significant body of research was dedicated to optimizing DORAM for the two-party and three-party cases **braun2023ramen**, **falk2023gigadoram**, **noble2024metadoram**, [127]–[130], [148], with three-party DORAMs being the most efficient. There are two major branches of DORAMs in the literature: (i) those based on classical sublinear-computation ORAM constructions, with logarithmic overhead (e.g., **falk2023gigadoram**, **noble2024metadoram**), or with square-root overhead (e.g., **braun2023ramen**); (ii) and those based on DPFs [127]–[130], [148]. While the latter require linear computation, they are extremely lightweight, having sublinear communication and a very low number of rounds in comparison. For that reason, they tend to scale better for mid-to-large memory sizes (e.g., up to 2^{26} records [127], [129]), or are suited for higher-latency environments, for example when the parties communicate over the internet and are not co-located in the same data center.

Given our (2,3)-VDPF construction, our work fits in the second bucket of research targeting DPF-based DORAMs. We have already shown how our construction enables building an efficient three-server ORAM. To turn it into a DORAM, we also need the servers to generate the VDPF in MPC, since there is no client. We illustrate this process in the following protocol sketch which builds a DORAM from our DPF construction.

(2,3)-VDPF DORAM protocol:

- **Input.** The three parties start with a shamir-sharing over of the secret DPF parameters $[\alpha], [\beta]$. As shown in Figure 5.5, Line 1 of VDPF.Gen, the parties need to obtain shares (of shares) of β which they can achieve through a constant-round re-sharing protocol.

- Recall that in Figure 5.5, VDPF.Gen constructs two underlying VDPF+ instances, each of which is composed of a single two-party VDPF with auxiliary data z. Generating a VDPF in a distributed manner can be done with the well-known protocol of [129], and it takes $O(logN)$ rounds and server communication and $O(N)$ computation. After this, the parties have generated two VDPFs and have evaluated them at the same time over the entire domain.

- However, in order to turn these into VDPFs+, we also need the parties to obtain the auxiliary data $z \leftarrow \beta_0 \oplus \mathsf{VDPF.Eval}(0, f_0, \alpha)$. The parties already have a sharing of β_0, but to obtain $\mathsf{VDPF.Eval}(0, f_0, \alpha)$, they need to somehow privately evaluate each VDPF at the point α. recall that P1, P2 both have the first key of the second DPF, and similarly P2, P3 have the second key of the first DPF. This redundancy allows the third party to perform a PIR query to privately obtain the necessary value. For example, for the second VDPF, P3 can read $\mathsf{VDPF.Eval}(0, f_0, \alpha)$ by generating a DPF (locally) and performing a 2-server PIR to read the value at α from P1 and P2. Because P3 does not actually know α, P1 and P2 first shift α by some randomness r and open the result towards P3. They also shift their own vector by r positions locally, ensuring that P3 reads the correct value. Treatment of the other DPF is symmetrical. Note that this takes only a (small) constant number of rounds, $O(logN)$ communication and $O(N)$ computation, so it does not significantly add to the protocolâs overhead.

- To complete the evaluation of the (2,3)-VDPF, the parties perform local operations and obtain their final share. With this, they can perform either a private read or write over the secret-shared memory, in the same way we described for our ORAM scheme.

Overall, this protocol has $O(logN)$ communication and round-complexity, and most of the overhead is in the distributed generation of [129], which is already quite efficient and has small constants.

Comparing to the state-of-the-art. Several recent works tried to construct an efficient three-party DORAM using DPFs. In [130], [148], the authors construct three-party DPFs that are significantly less efficient than ours. Therefore, their overall communication overhead between the servers is worse. In the first, communication is $O(\sqrt{N})$ per-query (but with a constant number of rounds), while in the latter, it is $O(log^2N)$ for both communication and rounds, with high constants. More similar to our work, [127], [128] presented a $O(logN)$ communication/rounds protocol for a three-party DORAM, but their protocol is in the server-aided model and therefore not a true three-party protocol. We thus estimate that a DORAM protocol based on our VDPF would yield the most efficient three-party DPF-based DORAM protocol to date, and that compared to other DORAMs, ours would be the most efficient for mid-sized memories or in high-latency environments. We leave implementing and benchmarking such a system for future work.

```
Parameters: A prime field $F$ and a hash function $H$. Inverses are computed over the prime field $F$.
$\mathsf{VDPF.Gen}(1^\kappa, \alpha, \beta)$
    1. $(\beta_1, \beta_2, \beta_3) \leftarrow \mathsf{SS}_{2,3}.\mathsf{Share}(\beta)$
    2. $\beta'_i \leftarrow \beta_i \cdot (i)^{-1}$, for $i \in \{1, 2, 3\}$.
    3. $v_0 \overset{\$}{\leftarrow} F_{2^l}$
    4. $v_2 \leftarrow \beta'_1 v_0$
    5. $v_1 \leftarrow \beta'_2 v_2$
    6. $v_3 \leftarrow \beta'_3 v_1$
    7. $(g_0, g_1) \leftarrow \mathsf{VDPF}^+.\mathsf{Gen}(1^\kappa, \alpha, v_0, v_1)$
    8. $(k_0, k_1) \leftarrow \mathsf{VDPF}^+.\mathsf{Gen}(1^\kappa, \alpha, v_2, v_3)$
    9. Set $f_1 = (g_0, k_0), f_2 = (g_1, k_0)$ and $f_3 = (g_1, k_1)$
    10. Output: $(f_1, f_2, f_3)$.
$\mathsf{VDPF.Eval}(b, f_b, x)$
    1. Parse $(g, k) \leftarrow f_b$.
    2. Set $(b_g, b_k)$ to $(0, 0), (1, 0)$ or $(1, 1)$ if $b$ equals $1, 2$ or $3$, respectively.
    3. Compute $y_g = \mathsf{VDPF}^+.\mathsf{Eval}(b_g, g, x)$
    4. Compute $y_k = \mathsf{VDPF}^+.\mathsf{Eval}(b_k, k, x)$
    5. Compute $y = (y_g \oplus y_k) \odot b$.
    6. Output $y$.
$\mathsf{VDPF.Prove}(b, f_b, r_b)$
    1. Parse $(g, k) \leftarrow f_b$.
    2. Set $(b_g, b_k)$ to $(0, 0), (1, 0)$ or $(1, 1)$ if $b$ equals $1, 2$ or $3$, respectively.
    3. $\pi_g = \mathsf{VDPF}^+.\mathsf{Prove}(b_g, g)$ and $\pi_k = \mathsf{VDPF}^+.\mathsf{Prove}(b_k, k)$
    4. Initialize $\boldsymbol{y_b} = \{\}$, $\boldsymbol{u} = \{\}$.
    5. For $x_i \in \{x_1, \ldots, x_n\}$:
        (a) Compute $y_g(x_i) = \mathsf{VDPF}^+.\mathsf{Eval}(b_g, g, x_i)$
        (b) Compute $y_k(x_i) = \mathsf{VDPF}^+.\mathsf{Eval}(b_k, k, x_i)$
        (c) $y_i \leftarrow (y_g(x_i) \oplus y_k(x_i)) \odot b$
        (d) $\boldsymbol{y_b} \leftarrow \boldsymbol{y_b} \cup \{y_i\}$
        (e) Generate (the same) $u_i \overset{\$}{\leftarrow} F$ and let $\boldsymbol{u} \leftarrow \boldsymbol{u} \cup \{u_i\}$
    6. $\beta_b \leftarrow \sum_{i=1}^{n} \boldsymbol{y_b}[i]$
    7. $t_b \leftarrow (\boldsymbol{y_b} \cdot \boldsymbol{u})^2 - \beta_b(\boldsymbol{y_b} \cdot \boldsymbol{u}^2) - r_b$ (over $F$)
    8. Output $\pi_b = (\pi_g, \pi_k, t_b, H(t_b))$
$\mathsf{VDPF.Verify}(\pi_1, \pi_2, \pi_3)$
    1. Parse $\pi_b = (\pi_{g,b}, \pi_{k,b}, t_b, h_b)$
    2. Compute $t = \mathsf{Reconstruct}(t_1, t_2, t_3)$
    3. Output accept iff $\pi_{g,2} = \pi_{g,3}$ and $\pi_{k,1} = \pi_{k,2}$ and accept $= \mathsf{VDPF}^+.\mathsf{Verify}(\pi_{g,1}, \pi_{g,2})$ and
       accept $= \mathsf{VDPF}^+.\mathsf{Verify}(\pi_{k,1}, \pi_{k,3})$, and $t = 0$ and $\forall i \in \{0, 1, 2\}$ $H(t_b) = h_b$.
```

Figure 5.5: Our $(2, 3)$-VDPF construction

Setting: The functionality interacts with servers S_1, S_2, S_3, clients $c_1, \ldots, c_n$, and an adversary $\mathcal{S}$.

Parameters: The functionality is initialized with a zero-initialized array $D = (D_1, \ldots, D_N) \in F^N$, and is parameterized with a policy verification function: $\mathsf{PVerify}$ that is given the request's arguments and returns `accept` or `reject`. Parameter m refers to the number of entries read-/update commands support.

- On input $(c_i, \mathsf{register})$ from client c_i, mark c_i as 'registered'.
- On input $(c_i, \mathsf{read}, (\ell_1 \ldots, \ell_m))$ from a registered client c_i:
  - Send (read, m) to $\mathcal{S}$ and wait to its response; if $\mathcal{S}$ returns abort then send $\perp$ to $\mathcal{S}$ and all servers, otherwise (if $\mathcal{S}$ returns $\mathsf{continue}$) continue.
  - If $\mathsf{accept} = \mathsf{PVerify}(c_i, \mathsf{read}, (\ell_1 \ldots, \ell_m))$ then output $(D_{\ell_1}, \ldots, D_{\ell_m})$ to c_i.
- On input $(c_i, \mathsf{update}, (\ell_1, v_1), \ldots, (\ell_m, v_m))$ from a registered c_i:
  - Send (update, m) to $\mathcal{S}$ and wait to its response; if $\mathcal{S}$ returns abort then send $\perp$ to $\mathcal{S}$ and all servers, otherwise (if $\mathcal{S}$ returns $\mathsf{continue}$) continue.
  - If $\mathsf{accept} = \mathsf{PVerify}(c_i, \mathsf{update}, ((\ell_1, v_1), \ldots, (\ell_m, v_m)))$ then, update $D_{\ell_j} = D_{\ell_j} + v_j$ for every $j \in \{1, \ldots, m\}$.

Figure 5.7: Functionality $\mathcal{F}_{\mathsf{AP-ORAM}}$

Setting: The functionality interacts with parties P_1, P_2, P_3 and an adversary $\mathcal{S}$.

Inputs: P_i inputs $[V]_i, f_i$, for $i \in \{1, 2, 3\}$.

- For $j \in \{1, 2, 3\}$, Expand $[F]_j \leftarrow \mathsf{VDPF.Eval}(j, f_j)$.
- For all $i \in \{1, 2, ..., N\}$:
  - $F_i \leftarrow \mathsf{Reconstruct}_{1,3}([F_i])$
  - $V_i \leftarrow \mathsf{Reconstruct}_{1,3}([V_i])$
  - $Z_i \leftarrow F_i \cdot V_i$.
  - Store in $[Z]$ the i-th sharing: $[Z_i] \leftarrow \mathsf{Share}1, 3(Z_i)$.
- Wait for an input from $\mathcal{S}$, if it is $\perp$ then output $\perp$ to all parties, otherwise continue.
- Output $[Z]_j$ to P_j, for $j \in \{1, 2, 3\}$.

Figure 5.8: Functionality $\mathcal{F}.\mathsf{Product}$

Upon receiving $[X_i]_j, [Y_i]_j$ from the honest parties P_j, as their shares of X_i, Y_i, for every $i \in \{1, \ldots, N\}$ and an additive error ε from the adversary: Compute $z = \varepsilon + \sum_{k=1}^{N} X_i \cdot Y_i$, then, output $[z] \leftarrow \mathsf{Share}_{2,3}(z)$ to the parties.

Figure 5.9: Functionality $\mathcal{F}_{\mathsf{SoP}}$

Initialization. The servers initialize a zero-shared database $D = (D_1, \ldots, D_N)$. Denote the i-th share of the database by $D^i = (D^i_1, \ldots, D^i_N)$. Similarly, the servers maintain a sharing of $\Lambda = (\Lambda_1, \ldots, \Lambda_N)$, which is used for access control. In addition, the servers maintain a zero-initialized counter ctr, which keeps track on the number of registered clients.

1. **Register.** On input $(c, \mathsf{register})$ to the servers: if $\mathsf{ctr} = N$ then send `full` to the client and halt, otherwise:
   (a) The servers increment ctr and invoke $[\lambda] \leftarrow \mathcal{F}_{\mathsf{Rand}}$; then, each server S_j stores $\Lambda^j_{\mathsf{ctr}} \leftarrow [\lambda]_j$ and sends ctr and $[\lambda]_j$ to the client c. From this point and on, c is indexed c_{ctr}.
   (b) Client computes $\lambda = \mathsf{SS.Reconstruct}([\lambda]_1, [\lambda]_2, [\lambda]_3)$ and stores (ctr, λ).

Figure 5.10: Protocol $\Pi_{\mathsf{CBDC}}.\mathsf{register}$.

Read. On input (read, ℓ) to client c_i:

1. *Client:* The client c_i sends $(f_i, [\lambda]_i)$ to server S_i, where:
    (a) $(f_1, f_2, f_3) \leftarrow \mathsf{VDPF}^{+}.\mathsf{Gen}(1^{\kappa}, \ell, 1)$.
    (b) $([\lambda]_1, [\lambda]_2, [\lambda]_3) \leftarrow \mathsf{SS.Share}_{2,3}(\lambda_i)$.
2. *Servers:* Given $(f_i, [\lambda]_i)$, the servers invoke $[z] \leftarrow \mathcal{F}.\mathsf{Zero}$; and then, server S_j computes:
    (a) $T^j \leftarrow \mathsf{VDPF}^{+}.\mathsf{Eval}(j, f_j)$ (this is a full-domain evaluation).
    (b) $[t]_j = \sum_{k=1}^{N} T^j_k$.
    (c) $\pi_j \leftarrow \mathsf{VDPF}^{+}.\mathsf{Prove}(j, f_j, [z]_j)$.
    (d) $[\tilde{\lambda}] \leftarrow \mathcal{F}.\mathsf{Product}([\Lambda], f)$.
3. Servers verification:
    (a) Server S_j exchanges h_j from π_j, then exchanges the rest of the proof.
    (b) The servers halt if:
        i. The DPF is not valid, i.e., $\mathsf{reject} = \mathsf{VDPF}^{+}.\mathsf{Verify}(\pi_1, \pi_2, \pi_3)$,
        ii. The DPF hides a value different than 1, i.e., $\mathcal{F}.\mathsf{CheckZero}([t] - 1)$ returns false.
        iii. Client authentication fails, i.e., $\mathcal{F}.\mathsf{CheckZero}([\lambda] - [\tilde{\lambda}])$ returns false.
4. Balance retrieval:
    (a) The servers compute $[b] \leftarrow \mathcal{F}.\mathsf{Product}([D], f)$ and then server S_j sends $[b]_j$ to the client.
5. *Client:* The client outputs $b = \mathsf{SS.Reconstruct}([b]_1, [b]_2, [b]_3)$.

Figure 5.11: Protocol $\Pi_{\mathsf{CBDC}}.\mathsf{read}$.

Update. On input $(\mathsf{update}, (\ell_1, v_1), (\ell_2, v_2)$ to client c_i:

1. *Client:* The client c_i sends $(f_i, g_i, [\lambda]_i)$ to server S_i, where:
   (a) $(f_1, f_2, f_3) \leftarrow \mathsf{VDPF}^+.\mathsf{Gen}(1^\kappa, \ell_1, v_1)$.
   (b) $(g_1, g_2, g_3) \leftarrow \mathsf{VDPF}^+.\mathsf{Gen}(1^\kappa, \ell_2, v_2)$.
   (c) $([\lambda]_1, [\lambda]_2, [\lambda]_3) \leftarrow \mathsf{SS.Share}_{2,3}(\lambda_i \cdot v_1)$.
2. *Servers:* Given $(f_i, g_i, [\lambda]_i)$, the servers invoke $\mathcal{F}_{\mathsf{Zero}}$ twice to get $[z]$ and $[z']$; then, server S_j computes:
   (a) $F^j = \mathsf{VDPF}^+.\mathsf{Eval}(j, f_j)$ and $G^j = \mathsf{VDPF}^+.\mathsf{Eval}(j, g_j)$;
   (b) $[a]_j = \sum_{k=1}^{N} F_k^j$ and $[b]_j = \sum_{k=1}^{N} G_k^j$.
   (c) $\pi_j = \mathsf{VDPF}^+.\mathsf{Prove}(j, f_j, [z]_j)$, $\quad \pi'_j = \mathsf{VDPF}^+.\mathsf{Prove}(j, g_j, [z']_j)$.
   (d) $[a_2] \leftarrow \mathcal{F}.\mathsf{Mult}([a], [a])$.
   (e) $[\lambda'] \leftarrow \mathcal{F}.\mathsf{Product}([\Lambda], f)$.
   (f) $[v'] \leftarrow \mathcal{F}.\mathsf{Product}([D], f)$.
3. S_j exchanges (h'_j, h''_j) from π'_j, π''_j, then exchanges the rest of the proofs.
4. *Servers* halt if:
   (a) $\mathsf{VDPF}^+.\mathsf{Verify}(\pi_1, \pi_2, \pi_3)$ or $\mathsf{VDPF}^+.\mathsf{Verify}(\pi'_1, \pi'_2, \pi'_3)$ return reject.
   (b) $\mathcal{F}.\mathsf{CheckZero}([\lambda] - [\lambda'])$ or $\mathcal{F}.\mathsf{CheckZero}([a] - [b])$ return false.
   (c) $\mathcal{F}.\mathsf{LTE}([a_2], [v'])$ and $\mathcal{F}.\mathsf{LTE}([a], \sqrt{|F|})$ returns false.
5. Each server S_j updates $D^j = D^j - F^j + G^j$ and sends Ok to the client.

Figure 5.12: Protocol $\Pi_{\mathsf{CBDC}}.\mathsf{update}$.

Setting: The functionality interacts with parties P_0 and P_1 and an adversary $\mathcal{S}$. The functionality is initialized with a VDPF$^+$ scheme.

Inputs: P_i inputs f_i, for $i \in \{0, 1\}$. The parties input the $(2, 2)$-sharings of $\beta'_0, \beta'_1 \in F_{2^\kappa}$.

- 'Reconstruct' the function hidden by (f_0, f_1) and obtain α, β_0, β_1.
- Reconstruct the secrets β'_0 and β'_1 from their shares.
- Compute $(f'_0, f'_1) \leftarrow \mathsf{VDPF}^+(\alpha, \beta'_0, \beta'_1)$.
- Output f'_i to P_i, for $i \in \{0, 1\}$.

Figure 5.13: Functionality $\mathcal{F}^+_{(2,2)-\mathsf{UVDPF}}$

Setting: The functionality interacts with parties P_1, P_2, P_3 and an adversary $\mathcal{S}$. The functionality is initialized with a VDPF$^+$ scheme.

Inputs: P_i inputs f_i, for $i \in \{1, 2, 3\}$. The parties input a $(2, 3)$-Shamir sharing of $\beta' \in F$.

- 'Reconstruct' $F_{\alpha,\beta}$ using (f_1, f_2, f_3) and obtain α and β.
- Reconstruct the secret β'.
- Compute $(f'_1, f'_2, f'_3) \leftarrow \mathsf{VDPF}^+(\alpha, \beta')$.
- Wait for an input from $\mathcal{S}$, if it is $\perp$ then output $\perp$ to all parties, otherwise continue.
- Output f'_i to P_i, for $i \in \{1, 2, 3\}$.

Figure 5.14: Functionality $\mathcal{F}^+_{\mathsf{(2,3)-UVDPF}}$

Inputs: P_i inputs f_i, for $i \in \{1,2,3\}$. The parties input a $(2,3)$-Shamir sharing of $\beta' \in F$.

Protocol: If, at any of the following steps, any party receives $\perp$ from a functionality invocation, then it aborts.

1. Pick random bit sharings:
   - $([\beta'_{0,\kappa-1}], \dots, [\beta'_{0,0}])$.
   - $([\gamma'_{0,\kappa-1}], \dots, [\gamma'_{0,0}])$.
2. For every $i \in [0, \kappa-1]$, compute $[\delta_{1,i}] = [\beta'_{0,i}] \oplus [\gamma'_{0,i}]$.
3. Compute: $[\delta_1] = \sum_{i=0}^{\kappa-1} 2^i \cdot [\delta_{1,i}]$.

   Note that $\delta_1 = (\beta'_0 \oplus \gamma'_0)$ is S_1's share; together with δ, they completely define the degree-1 polynomial $P(\mathtt{x}) = \delta + a\mathtt{x}$ s.t. $P(1) = \delta_1 = \delta + a$, meaning that $a = \delta - \delta_1$. Given δ and a, it follows that $\delta_2 = P(2) = \delta + 2a$ and $\delta_3 = P(3) = \delta + 3a$. Thus:
4. Compute $([\delta_{2,\kappa-1}], \dots, [\delta_{2,0}]) \leftarrow \mathcal{F}.\mathsf{A2B}([\delta_2])$ and $([\delta_{3,\kappa-1}], \dots, [\delta_{3,0}]) \leftarrow \mathcal{F}.\mathsf{A2B}([\delta_3])$, where $[\delta_2] = [\delta] + 2[a]$ and $[\delta_3] = [\delta] + 3[a]$.
5. For every $i \in [0, \kappa-1]$, compute $[\beta'_{1,i}] = [\delta_{2,i}] \oplus [\gamma'_{0,i}]$ and $[\gamma'_{1,i}] = [\beta'_{1,i}] \oplus [\delta_{3,i}]$.
6. Generate random sharings of bits, denoted $[x_i], [y_i], [z_i], [w_i]$ for every $i \in [0, \kappa-1]$.
7. Open to S_1 the values x_i, y_i, z_i, w_i, for every $i \in [0, \kappa-1]$. Denote $x = (x_{\kappa-1}, \dots, x_0)$ and similarly for y, z, w.
8. Open to S_3 the value $x_i \oplus \beta'_{0,i}$, $y_i \oplus \beta'_{1,i}$, $z_i \oplus \gamma'_{0,i}$, $w_i \oplus \gamma'_{1,i}$. Denote $x' = (x_{\kappa-1} \oplus \beta'_{0,\kappa-1}, \dots, x_0 \oplus \beta'_{0,0})$ and similarly y', z', w'.
9. S_1 and S_3 call the updatable VDPF$^+$ functionality twice:
   (a) S_1 inputs f_0, x, y and S_3 inputs f_1, x', y'. S_1 receives f'_0 and S_3 receives f'_1.
   (b) S_1 inputs g_0, z, w and S_3 inputs g_1, z', w'. S_1 receives g'_0 and S_3 receives g'_1.
10. S_1 sends g'_0 to S_2 and S_3 sends f'_1 to S_2.
11. The three servers obtain $[r] \leftarrow \mathcal{F}.\mathsf{Rand}$; then they run Prove and Verify on the new shares $(f'_0, g'_0), (f'_1, g'_0)$ and (f'_1, g'_1) using their shares of $[r]$.

Figure 5.15: Protocol $\Pi_{(2,3)-\mathsf{UVDPF}^+}$

Inputs: P_i has V^i and f_i as inputs, for $i \in \{1,2,3\}$, where V^i are shares of a vector and f_i are shares for a VDPF. We assume that the VDPF was already proven valid.

Protocol:

1. The parties invoke $[m] \leftarrow \mathcal{F}.\mathsf{Rand}()$.
2. Party P_i computes $A^i = \mathsf{VDPF.Eval}(i, f_i)$ and then $[a]_i = \sum_{k=1}^{N} A_k^i$.
3. The parties compute $[ma] \leftarrow \mathcal{F}.\mathsf{Mult}([m],[a])$.
4. The parties compute $g' \leftarrow \mathcal{F}_{(2,3)-\mathsf{UVDPF}}(f, [ma])$, which results with shares f'_1, f'_2, f'_3 for a point function $F_{\alpha,ma}$.
5. Party P_i computes $B^i = \mathsf{VDPF.Eval}(i, f'_i)$ and then $[b]_i = \sum_{k=1}^{N} B_k^i$.
6. The parties compute $[va] \leftarrow \mathcal{F}_{\mathsf{SoP}}([V],[A])$ and $[vb] \leftarrow \mathcal{F}_{\mathsf{SoP}}([V],[B])$.
7. Compute $[vam] \leftarrow \mathcal{F}.\mathsf{Mult}([va],[m])$.
8. Call $\mathcal{F}.\mathsf{CheckZero}([vam]-[vb])$, if the result is false then halt, otherwise output $[va]$.

Figure 5.16: Protocol Π_{Product}

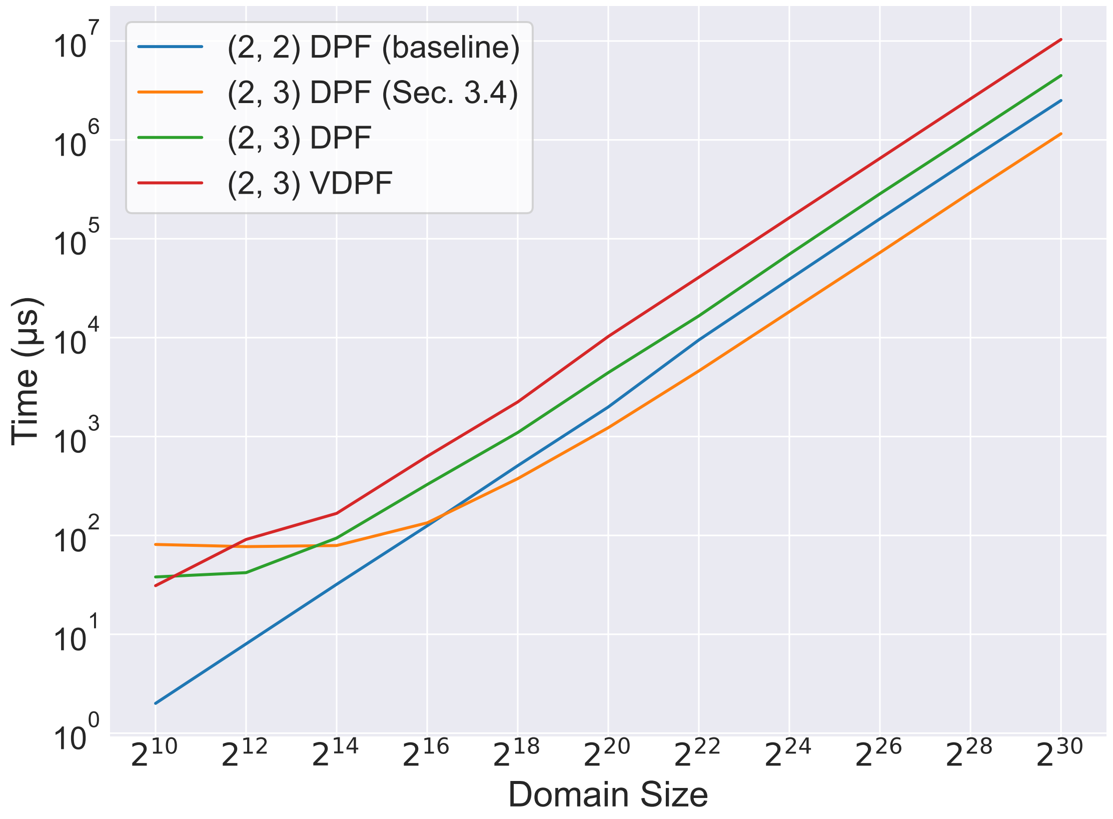


Figure 5.17: Full domain evaluation of the various DPF constructions, compared to the baseline (2,2) DPF.

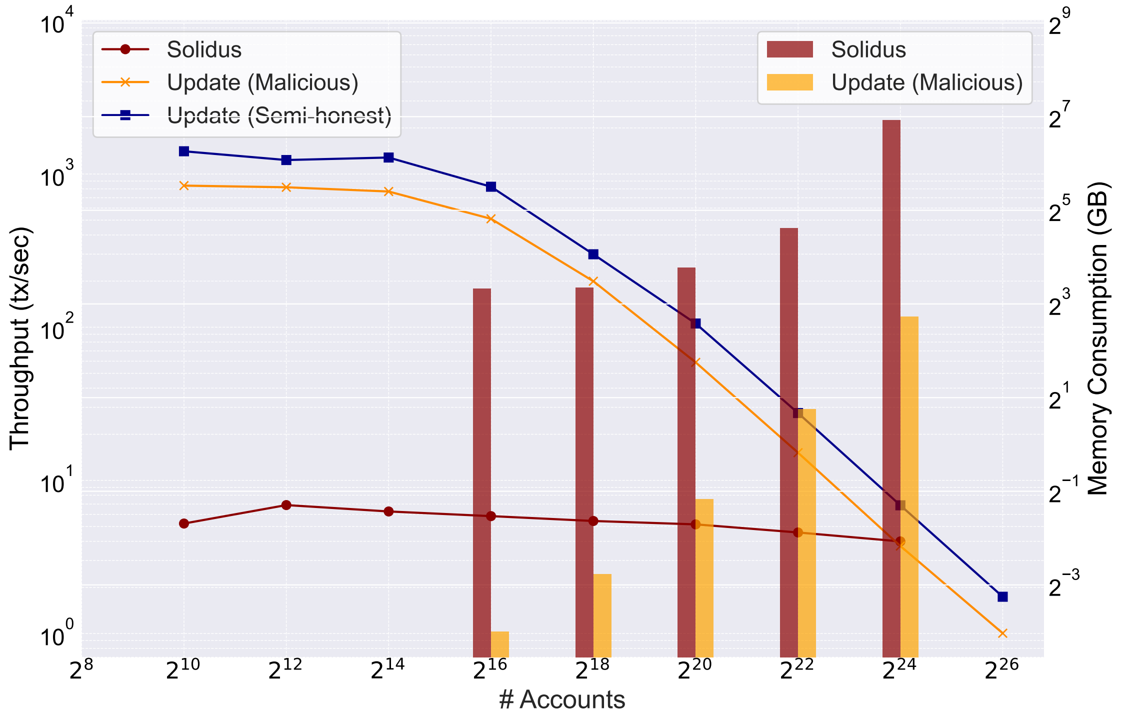
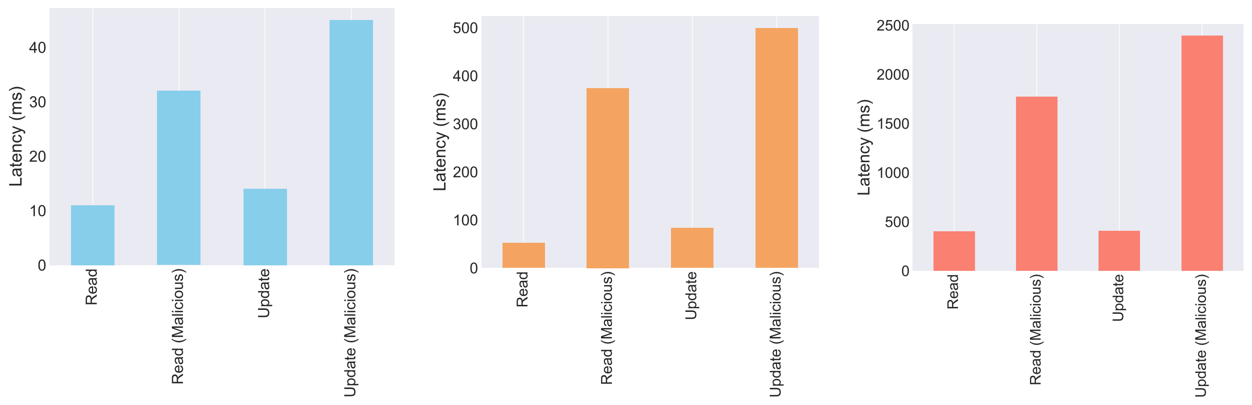


Figure 5.18: Transaction throughput and memory usage. Side-by-side comparison with Solidus.



(a) RTT = 1ms (b) RTT = 10ms (c) RTT = 100ms

Figure 5.19: Latency in processing a request under different channel latency for $N = 2^{20}$. Requests can be parallelized so this is indicative, and bandwidth changes had no effect thanks to the constant communication complexity.

Chapter 6

Private Retrieval Augmented Generation (PRAG)

Heavily pre-trained and fine-tuned Large Language Models (LLMs) have demonstrated exceptional performance on zero-shot [170] and few-shot tasks [171]. The ability of these models to generalize, combined with their costly pretraining, has shifted the focus from training ad-hoc models to perform specific tasks to utilizing these general-purpose foundational models for a wide variety of use-cases [172], [173]. These pre-trained models lack knowledge of private contexts or recent events.

To provide these LLMs with up-to-date or relevant information, methods such as Retrieval Augmented Generation (RAG) [174]–[176] are used to include external information into a generation process without needing fine-tuning on new data. This process allows LLMs to first query an external data source, retrieve relevant information (with respect to a given prompt), and then use both the prompt and the retrieved data as input to the inference phase of the LLM.

Similar to the problem of federated learning [177], it is valuable to aggregate sensitive data from multiple (perhaps many) data owners. To do that, each party should be able to guarantee that their own private data remains private even when it is utilized. On the other hand, model users should be able to query these data from many data owners without needing to share what questions they are asking.

In this work we argue that LLMs require a new model for sharing data for AI tasks. Compared to federated learning, which focuses on the training phase, LLMs should focus on the (i) retrieval phase; (ii) inference phase. Guaranteeing privacy of *both* the query and any private documents residing in the retrieval database require that both phases utilize privacy-preserving techniques and are chained together.

Alas, to the best of our knowledge all existing works only tackle the LLM inference problem [178]–[181], but provide no secure solution when retrieval is involved. In this work, we close this gap by introducing Private Retrieval Augmented Generation (PRAG). PRAG allows users to privately search a database, which in itself is private, then send the augmented query privately to any secure (or otherwise trusted) LLM, creating an end-to-end secure solution.

Our approach and contributions. We propose Private Retrieval Augmented Generation (PRAG), a secure approach to augment neural information retrieval that hides both query vectors and the retrieval database. We use a retrieval database split across a set of servers, and we ensure data remains private by using secure multi-party computation (MPC) techniques. To the best of our knowledge, we are the first to consider the problem of secure distributed retrieval in the context of LLMs, and more broadly, are the first to propose a solution for private similarity search that can protect both the query and a secret-shared (or encrypted) database. This approach can be deployed with any standard neural information retrieval (IR) embedding model to augment distance calculations (e.g., cosine, dot, euclidean) and top-k retrieval over federated vector stores, scaling to medium-size databases with very little accuracy loss (99% accuracy on real data).

We further scale the approach to much larger databases using an approximate k-nearest-neighbors approach inside MPC, replicating the accuracy of the state of the art in approximate retrieval using a first-of-its kind inverted files index inside MPC, providing significant speed improvements for retrieval. Our approach provides both theoretical and empirical improvements of value. We achieve constant communication on the clientâs side and *sublinear* communication on the serversâ side ââ the bottleneck in MPC approaches. This work is the first IR approach to work across more than two servers with minimal additional costs. We further present a ‘leakyâ version of the protocol that allows for partial privacy of queries under a privacy budget with significant improvements to speed.

We evaluate PRAG across a range of data distributions, both real and synthetic, to show it broadly maintains the performance characteristics of non-secure IR approaches. We provide a pytorch-native implementation of our system using the Crypten MPC engine and retrieval hooks for langchain and BEIR.

6.1 Overview

In this section, we present the Private Retrieval Augment Generation (PRAG) framework. The method builds from secret sharing and MPC friendly exact top-k calculations to a new MPC design of an inverted file index for efficient approximate top-k calculation.

Although a wide array of approaches exist for training document embedding models and augmenting generation with retrieved models, most neural information retrieval methods are underpinned by a step where a querier sends a query embedding to a server to calculate the distance / similarity between the query vector and the database, in order to return a document either as an embedding vector for concatenation or with the document tokens for use in LLM inference. This setup offloads the storage of large databases and their associated calculations to a more powerful server.

Recently, a significant body of research has been focusing on the problem of secure inference, which ensures that a query remains private at all times. Whether secure inference is achieved through cryptographic techniques (e.g., [178], [179], [182]–[184]), or by running the model locally [185], if the inference pipeline includes an external retrieval phase (as is often the case), then security does not hold as the query itself is leaked to the database operator.

Similarly, the database may itself hold private information, collected by many different data owners. The only way to protect their data is by making sure both the client and the vector database server(s) remain oblivious to its content.

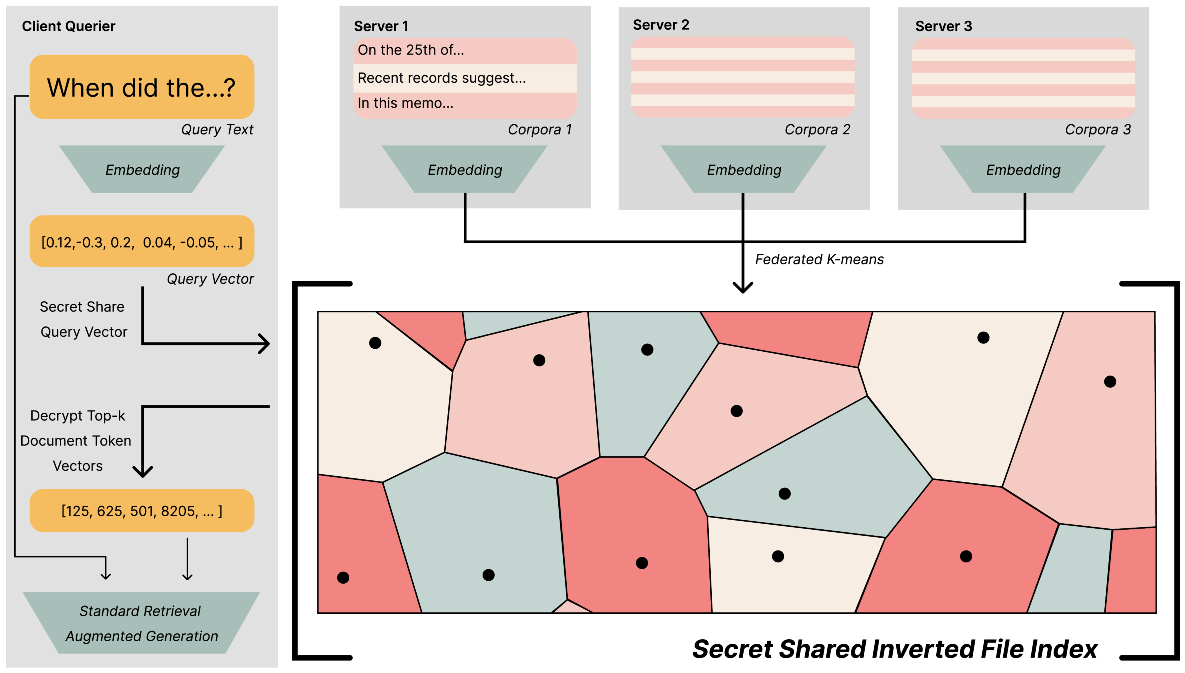


Figure 6.1: Overview of PRAG architecture using a distributed, secret-shared inverted file index (IVF), for retrieving document token vectors closely matching a privately-generated query vector in LLM-based question answering.

To formalize this, we assume our system has $n_{clients}$ clients sending queries and n_{owners} data owners. Both clients and data owners interact with a set of $n_{servers}$ vector database operators. We assume that all parties in the system are semi-honest (i.e., they follow the protocol) and that at most $t < \frac{n_{servers}}{2}$ of the servers are corrupt (the honest majority setting). In this work, we do not focus on the n_{owners} data owners privately building the server, and we assume that at some point in the past these data owners have secret-shared their data to the servers. Instead, we are focused on the inference stage, a much more frequent and real-time operation.

6.2 Exact Private Nearest Neighbors

We assume all values are shared using Shamir secret sharing [35] over a prime field F_p where $p \simeq 32$ or 64 bits. We note that our protocols could work using other secret sharing schemes suitable for the honest-majority setting (e.g., replicated secret sharing [186] over the ring $Z_{2^{32}}$ or $Z_{2^{64}}$), but Shamir is the ideal choice in our setting, as it requires the least amount of space and scales well to a large number of servers.

We further assume, as is common in secure machine learning literature [187], [188], that there is a trusted dealer that generates shared random values. However, other techniques could distribute this [166], [168], [189]. As in other works, since these protocols happen offline in a preprocessing phase and do not impact the online performance of serving a query, we do not benchmark their performance.

We denote arithmetic secret-shared values by $[x]$. A share for a specific server i is denoted as $[x]_i$. When sharings may appear once as a t-degree sharing and another as a $2t$-degree sharing, we occasionally distinguish these sharings with a superscript (e.g., $[x]^{(2t)}$). We use $[x] := \text{SS.Share}(x)$ and $x := \text{SS.Reveal}([x])$ for sharing and revealing secret shared items.

As is well known, all linear operations over secret-shared values require no interaction between the servers. For multiplication, a single round of interaction is required. Given our setting, we find the multiplication protocol by Damgård and Nielsen [161] to be the most suitable.

Since in this work we operate in the semi-honest, honest-majority setting, we encode real numbers into a field, we use the common technique of representing all underlying values as fixed-point integers [190]. In practice, this means that for any real value $\tilde{x} \in R$, we encode it as a fixed-point integer $\lfloor \tilde{x} 2^f \rfloor \in Z$ with precision f. Note that multiplying two encoded values results in a value with $2f$-precision. Therefore, truncation is needed after every multiplication to avoid causing an overflow inside the field, which would distort results.

6.2.1 Distance calculations

While there is some heterogeneity in distance measures used in neural information retrieval, the majority use dot products, cosine similarity, or L2 norms (euclidean distance). We provide MPC friendly implementations of all three.

A naive implementation of a dot product between a vector and a matrix can be provided by running the secure multiplication protocol in parallel. Both the communication and the computation complexity scale linearly with the size of the database N and embedding dimension size d_e, the latter of which is fixed in almost all cases. Round complexity remains the same (constant) regardless.

Extending the dot product gives us cosine similarity, the predominant distance measure in sentence transformer style models [191]. To save on expensive MPC computations, we pre-normalize the input vectors and matrices prior to secret sharing into MPC, allowing for cosine similarity to reduce to a simple dot product. Computing Euclidean distance can also be achieved directly through MPC, but we observe that this is a much more expensive operation, as it requires computing square roots inside the MPC circuit. For example, Crypten [188], which we use in our implementation, uses a slow Newton-Raphson approach for computing square roots, requiring multiple rounds of communication.

However, we make the observation that given that top-k calculations are the end goal of distance calculations, the monotonic square root step in L2 can be ignored completely before looking for the top-k elements in the distance vector, removing the need to compute the square root securely.

6.2.2 Fast secure dot product

We re-use the same trick as in Chapter 5, where in the honest-majority setting we can efficiently multiply two secret-shared vectors together in $O(1)$ communication cost.

6.2.3 Relation to private information retrieval

As before, we can use PIR to extract any top-k elements from a database matrix that has been secret shared, by getting a one-hot encoded secret-shared vector of the top-k indices. This allows us to extract either database embedding vectors or token arrays from inside the distributed database for return. In essence, rather than securely returning top-k indices and asking the user to separately extract them, we can return the original tokens from a secret shared database directly in MPC. This oblivious retrieval is used extensively throughout our protocols below, such as in extracting candidate vectors from clusters.

6.2.4 Exact top-k for retrieval

Retrieving the most similar documents to a query requires first ranking all documents by some similarity metric (as above) and then picking the top k documents that are closest to the query.

Our solution is conceptually similar to secure top-k circuits designed in other works [192], where $O(kN)$ comparisons are needed. These circuits operate by successively keeping an ordered list of k items, and then computing each value in the array with the minimum value in the (much smaller) sorted list. Unfortunately, this solution also requires $O(N)$ rounds for MPC based on secret-sharing.

Instead, our protocol iterates k times over a secret-shared vector $[x]$. In each iteration, we run argmax($[x]$) to extract the current minimum's index in the vector. We then obliviously scale down the selected value enough so it will be ignored in future iterations.

There are many ways to implement an MPC protocol for argmax($[x]$). Our description above assumes a recursive tree-reduction based protocol as in Crypten [188], having $O(\log_2(N))$ rounds and $O(N\log_2(N))$ total communication. This leads to an exact top-k round complexity of $O(k\log_2(N))$ and $O(kN\log_2(N))$ overall communication.

By combining this with distance calculations and oblivious private retrieval from a database we can provide an end-to-end exhaustive exact algorithm to return the top-k nearest documents to a query from a database of embeddings (and a database of tokens for exact document return).

6.3 Approximate Nearest Neighbors and Inverted Files (IVF)

At its core, the information retrieval task of top-k closest points is exactly the task of solving the *k-nearest-neighbors* (kNN) problem, which requires finding the k points in a database that are nearest to the given data point (the query). While the above exact approach achieves this, it does so at a significant speed cost (both with or without MPC), motivating the creation of approximate nearest neighbors algorithms, which only require a sublinear amount of work.

These algorithms operate by first computing a compact representation of the dataset called the *index*, and then executing queries on the index. Many approximate nearest neighbors techniques exist, and one that is particularly amenable to MPC is the *inverted files index* (IVF) [193], [194]. This technique works by first using a clustering algorithm (e.g., k-means) over the data set to find its n_c *centroids*. Then, each centroid represents a cluster holding all points associated with that cluster. In other words, this process splits the database into n_c buckets.

After this one-time step, querying the data starts by computing the nearest neighbors of the query with respect to all centroids. Then, the n_{probe} nearest inverted files are searched, looking for the k nearest neighbors among them.

During IVF generation, parameter choices in how the index is built affect the downstream performance of the queries. We choose the number of clusters to be $n_c = \alpha\sqrt{N}$ to get sublinear complexity, where α is a free parameter that can be tuned. During query time, we find the distance to all n_c centroids, and select the top n_{probe} clusters to inspect further. As we will see during experiments, this choice of n_{probe} increases the recall performance of the model, and indeed at $n_{probe} = n_c$, all clusters are inspected and the search becomes exact. However, the nature of the IVF clustering allows a smaller n_{probe} to be chosen while still achieving high accuracy.

6.3.1 Efficient approximate vector nearest neighbor search in MPC

Bringing this into MPC, the protocol Π_{IVFQuery} securely computes the approximate nearest neighbors using an inverted file index. The protocol assumes the servers pre-computed the secret-shared inverted index [IVF], which consists of n_c lists of size m, both of which are of size $O(\sqrt{N})$, ensuring the overall communication complexity is sublinear. We use the MPC distance measures established above to calculate the distance between the query vector and each of the n_c cluster means.

The parties then run a secure protocol of exact top k as described earlier to identify the n_{probe} most similar clusters. Unlike non-MPC protocols, it is critical that the servers remain oblivious as to which are the top clusters for this query. Otherwise, information about both the query and database would leak. For this reason, we require the top-k protocol to return each index as a one-hot-vector of size n_c which are collectively stored in [closest buckets].

Then, the parties perform an exact-match private information retrieval to get all the vectors in the closest buckets. These [candidates] can be obliviously found through a product of [closest buckets], a mapping of centroids indices to cluster indices in the database, [IVF indices], and the entire [IVF] vector database. By obliviously reducing the entire vector database into a much smaller search space that only includes vectors from the n_{probe} nearest clusters, we are able to achieve sublinear overall communication.

At this stage, [candidates] holds a reduced $(n_{probe} \times m) \times d$ vector matrix (where d is the embedding dimension). [candidates indices] will similarly store the mapping from each candidate to the original database index. We proceed by running an exact nearest neighbor search again, which computes the distances between the query and all candidates and then securely gets the top-k entries. Using [candidates indices], these top-k entries are mapped back to the original database records, where documents can be obviously retrieved.

Protocol 5 Π_{IVFQuery}

Public Parameters: n, k, n_c, n_{probe}, m, d
Client: query $x \in R^d$
Server: Secret-shared inverted file clusters [IVF clusters]$\in R^{n_c \times d}$, Inverted file index values [IVF] $\in R^{n_c \times m \times d}$, Inverted file index indices [IVF indices] $\in R^{n_c \times m}$ k-nearest-neighbors (approximate)
Client computation:

- $[x] := SS.\text{Share}(x)$
- Send each server i its share $[x]_i$

Servers computation:

- **In parallel** Iterate over [cluster] $\in$ [IVF clusters]
- $\quad$ $[\text{centroid distance}_i] := \text{SumProd}([x], [cluster])$
- $\quad$ $[\text{centroid distances}] := \{[\text{centroid distance}_1]^{(t)}, \ldots, [\text{centroid distance}_{n_c}]^{(t)}\}$
- Compute $[\text{closest buckets}] := \text{ExactTopk}([\text{centroid distances}], n_{\text{probe}})$
- Compute $[\text{candidates}] := \text{MatMult}([\text{closest buckets}], [\text{IVF}])$ and $[\text{candidates indices}] := \text{MatMult}([\text{closest buckets}], [\text{IVF indices}])$
- **in parallel** Iterate over [candidate] $\in$ [candidates]
- $\quad$ Compute distance using SumProd and store as $[\text{candidate distances}]$ Compute $[\text{candidate top-k indices}] := \text{ExactTopk}([\text{candidate distances}], k)$
- Compute $[\text{database top-k indices}]$ via private exact-match retrieval of $[\text{candidate top-k indices}]$ from $[\text{candidates indices}]$
- Return [database top-k indices] documents via private retrieval.

6.3.2 Sublinear Communication Complexity

The client maintains an optimal communication complexity of $O(1)$, as it only needs to communicate a share of the query vector to each server.

As to the servers, in lines 5-7 a total of $n_c := O(\sqrt{N})$ elements are communicated. Computing the exact top-k over these n_c distances requires $O(k \cdot \log_2(n_c))$ communication. Reducing the dataset obliviously costs $O(n_{probe}\frac{N}{m}d)$. With our choice of parameters, n_{probe} and d are constant, and $m = \sqrt{N}$, yielding $O(\sqrt{N})$ communication. This gives a candidate dataset that is approximately of size $n_{probe}\sqrt{N}$. Finally, we can compute the distances and exact top-k on this reduced dataset, but as it now only contains $O(\sqrt{N})$, the overall communication of that step is $O(k \cdot \log_2(\sqrt{N}))$.

Overall, we see that end-to-end the servers communicate $O(\sqrt{N} + \log_2(\sqrt{N}))$ field elements while the client communicates $O(1)$ elements (in fact, she communicates exactly d elements, as is the size of the input vector). This holds true so long as n_{probe} remains small enough to be considered a constant. As the number of candidate clusters to be probed becomes n_c, the overall complexity of the approach becomes $O(\sqrt{N} \cdot \sqrt{N}) = O(N)$, which is no better than exact search but with additional overhead operations. Hence, n_{probe} should be kept low as we will see in the experimental settings.

6.3.3 Sacrificing Privacy for Speed in MPC IVF

The fast secure dot product trick above helps significantly improve the speed of the step wherein we reduce the full database to only the n_{probe} clusters vectors relevant to the query. However, this step is still extremely costly, requiring the manipulation of a large database of vectors for lookup when the clusters are stored in a large matrix.

Instead, we can take an alternate approach, where each cluster is stored in its own secret shared database, with an exposed lookup table. The centroids of the database still remain secret shared and private, but during query time, the n_{probe} closest clusters (shuffled to avoid exposing order) are decrypted by each server to retrieve the relevant secret shared cluster matrices, which can then be concatenated before passing into the second distance-top-k calculation. This has large speed implications, dramatically decreasing the data access time and allowing for speed more competitive with non-MPC IVF.

However, this does come at the cost of privacy. Each server will now know the n_{probe} closest clusters to the query, which leaks the area in the embedding space where the query is coming from. Indeed, while the centroids are secret shared, knowing the lookup table and what a user accesses would allow an actor to determine an average point across those centroids with more queries.

To mitigate this, a query could be noised according to a privacy budget similar to differential privacy, as for sufficiently large n_{probe}, even a high noised query would likely contain the relevant closest clusters nearby. One slight advantage here is that larger choices of n_{probe} provide more privacy (and more capacity for noising), while also increasing the overall accuracy of the search (as we see in Figure 6.3).

In general, this final methodological change differs from above by no longer being fully private, but is presented as part of the spectrum from slow but exact private search to fast approximate search, and finally to fastest but leaky approximate search.

6.4 Implementation and Evaluation

To demonstrate the performance of these models we run a series of experiments on both synthetic and real data to determine performance properties of the implementations of these methods above.

We benchmark the retrieval accuracy and speed across a range of embedding sizes (256 to 8192), synthetic embedding distributions ($N(0, 0.05)$, $N(0, 1)$, $U(-1, 1)$, Binary), distance functions (cosine, dot product, euclidean), top-k values, IVF parameters, and database sizes. We perform MPC experiments on a single 2.2GHz Intel Xeon Silver CPU using Crypten's built-in communication code to spawn processes for each server.

Further to this, we test the approaches on retrieval of real neural embedding datasets from BEIR [195] using the same environment, this collection of datasets uses a range of textual document types and sizes, all of which we use a standard off-the-shelf embedding on. While there are several parallelization improvements that can be made locally within each server for MPC, our implementations of each algorithm above remain unoptimized.

6.4.1 Exact Search

Each step of the exact search approach is extremely accurate, with small numerical errors introduced during MPC. For distance measures, MPC vectors have a mean squared error difference from pytorch calculated distances of less than 10^{-5} for euclidean and 10^{-8} for cosine, going as low as 10^{-11} for euclidean distance on $N(0, 0.05)$. These errors do not change with database size, and are introduced at the numerical level of the elements.

The exact top-k approach using tree reduction applied interactive k times suffers from similar small numerical errors. For distance vectors drawn $N(0, 0.05)$, where outliers are often standalone, top-k elements are picked out with 0.99 or above recall and precision. For uniform distributions (unrealistic for embedding distance vectors) the f1 accuracy is lower for top-1 (0.842) and top-k (0.96) with recall and precision climbing for higher k. This is explained by the small distances present between the max and its nearest value when drawn from a uniform distribution, leading numerical errors to induce a loss of accuracy. Fortunately, the nature of real distance distributions means performance is high in real contexts. For small values of k, this approach can be relatively fast but increasing the choice of k dramatically increases the time cost due to communication complexity in the interactive argmax looping.

Putting distance calculations, top-k, and oblivious retrieval together, the exact search approach in MPC can identify the top-1 (argmax) most similar vector to a query with 97.5% accuracy and top-50 with 98.6% F1 score, with accuracy independent of database sizes tested up to 5×10^5. The constraint on the use of this MPC exact approach is the speed, taking up to 10 seconds for top-1 and top-5 for a 10^5 size database, and increasing dramatically for larger k as in Figure 6.2.

6.4.2 Approximate Search

Our MPC IVF implementation, using both fully secure and partially leaky clustering, returns the elements as the standard IVF implementation with an average of over 99% recall on both

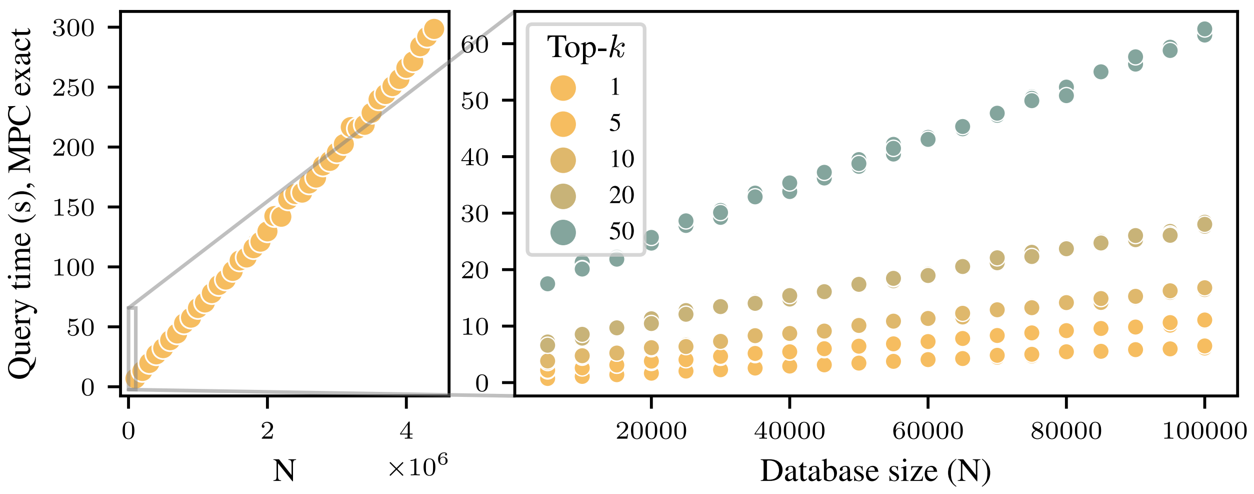


Figure 6.2: Time taken to retrieve top-k closest vectors in the database for end-to-end MPC exact search across increasing synthetic database sizes.

synthetic and real embedding data, with errors explained by numerical errors at runtime. For real data, we use embeddings from msmarco-distilbert-base-v3 from SBERT [191]. These numerical errors partly flow through from the exact search above, which is used at various points in the IVF MPC algorithm. This accuracy of the MPC IVF to non-IVF is stable across choices of n_{probe} and n_c.

While the MPC IVF matches the recall performance of the standard IVF, the underlying approximate nature of the IVF provides tradeoffs between accuracy and speed. As shown in Figure 6.2, increasing the value of n_{probe} increases the proportion of the full database that is inspected at query time, in turn increasing the overall runtime. The benefit of IVF is that we can achieve high accuracy for even a low value of n_{probe}, dramatically reducing query time at the cost of accuracy.

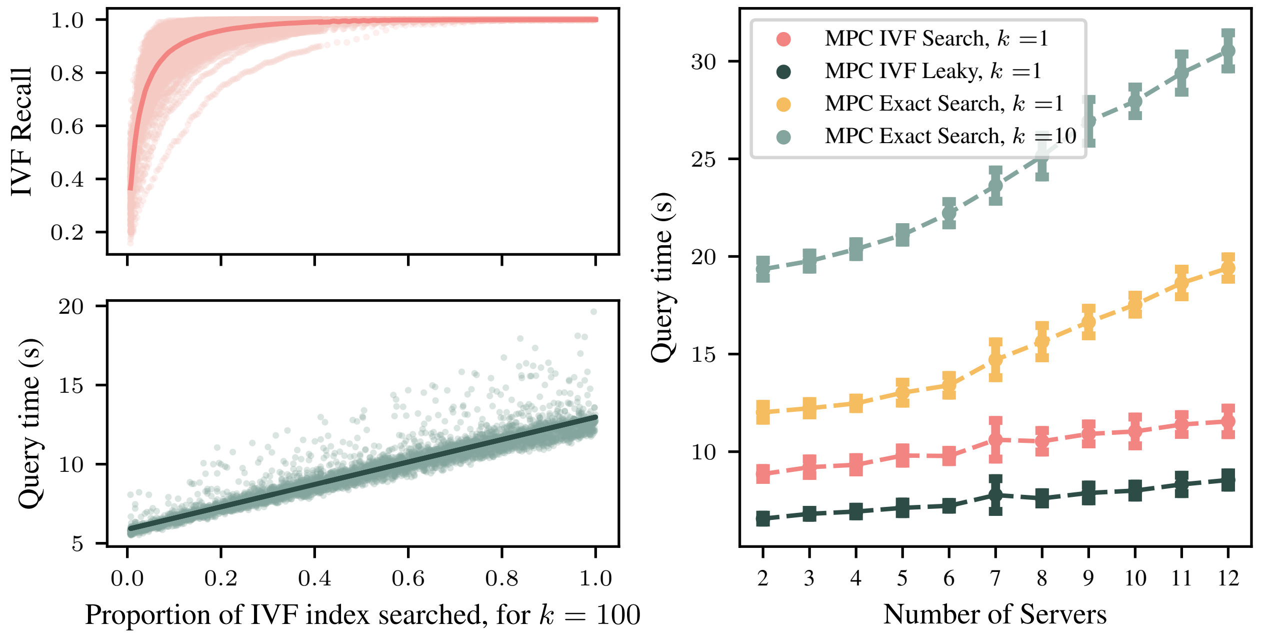


Figure 6.3: Information retrieval using IVF improves accuracy with increased n_{probe} (top left) but increases query time as a larger proportion of the index ($\frac{n_{probe}}{n_c}$) must be searched (bottom left). These retrieval approaches (both IVF and exact) scale favorably across multiple servers (right).

Chapter 7

Discussion, Conclusions and Future work

While research into distributed systems has been ongoing for several decades, the last two decades have seen the emergence of massively open decentralized systems in the wild, primarily through the use of blockchains. These systems are not operated by a single entity, but require the coordination of many parties. These parties can be known and well-established (in the permissioned setting), or pseudononymous (in the permissionless setting).

Blockchains and decentralized systems in general, have shown us that these systems have the potential to minimize trust and provide us with unparalleled integrity and availability, even in a highly adversarial environment with minimal trust assumptions. As a motivating example, the Bitcoin blockchain has been running (almost) non-stop for the last fifteen years, while maintaining an alternative financial system valued at more than a trillion dollars. This is despite the fact that there are no trust assumptions in regards to node operators, and in fact, it is assumed that if it is in the operators' best interest to cheat – they will.

A second emerging trend is that of Large Language Models (LLMs) that are trained over the entire internet, and which users can interact with using natural language. These systems are quickly changing the way we use technology, but also pose grave questions regarding our own data privacy and safety. To counter this, Decentralized Physical Infrastructure Networks (DePIN) promise to crowd-source the development and access to LLMs.

All of these decentralized systems suffer from the same problem - they lack data confidentiality. In this thesis, I attempted to tackle this problem head on from several different angles. First, in Chapter 3 I defined an access-control method for decentralized systems, using the same PKI mechanism that is already baked into all blockchains. Evolving from purely read-or-write access control over blockchains, I moved to introduce the concept of *confidential smart contracts*, which allows running arbitrary programs on a blockchain in a way that provides both integrity and confidentiality. These two concepts work well together, since they allow us to write programs that selectively reveal digested data to specific parties, without unnecessarily giving full access to the underlying data. An example for this is private voting, where specific parties should be allowed to vote, but all parties should not be able to discern individual votes, and should only be able to view the final result.

Second, in Chapter 4, I introduced how confidential smart contracts can be viewed as a real-world manifestation of the *server-aided MPC* model. In this model, we assume there is a semi-honest and non-colluding server, which the (confidential) blockchain effectively realizes. We then used this mechanism to build fair and robust server-aided Threshold

ECDSA protocols. Given the interesting results we obtained for Threshold ECDSA, it is worthwhile asking whether confidential smart contracts can be beneficial in many other server-aided MPC protocols. I leave exploring this avenue further for future work.

Third, in Chapter 5, I addressed the problem of building a scalable and private ledger that allows users to transact privately. Our approach is specifically targeted to the use case of a CBDC, which is gaining popularity, but raises significant privacy concerns. Perhaps more importantly, we were able to show that building a private ledger system in the account-model is feasible, at least with a small number of servers. Because the system uses MPC throughout, it allows us to add arbitrary functionality such as auditing against illicit activity. This is in contrast to systems based on zero-knowledge and inclusion proofs (e.g., [12]). While this work scales very well to many users in the three-server setting, extending it to more servers is an important future endeavor. Additionally, extending our work to the publicly-verifiable model, which is common in blockchains, would also be significant. This can likely be done with publicly-auditable MPC techniques such as [97].

Through the exploration of Chapter 5, we have also developed a new three-party (verifiable) DPF secure against a single corruption. We have shown for the first time that these DPFs are asymptotically (and concretely) as efficient as their two-party counterpart of [20]. While this has been hypothesized, previous works such as [196], were significantly less efficient both asymptotically and in practice. We also showed that honest-majority DPFs are interesting for efficient ORAM and DORAM applications. Solving a similar problem for honest-majority DPFs with more than three parties remains an open question, and the best techniques still require keys of size $O(\sqrt{n})$, or at best $O(n^{\frac{1}{4}})$ [196], but the latter are not applicable to our efficient (D)ORAM schemes.

Finally, in Chapter 6, I have shown how LLMs in the RAG setting can protect the privacy of a user's query, as well as a private retrieval database. Prior to our work, all others have focused on protecting user privacy in models without RAG.

7.1 Putting it All Together

To summarize, this thesis explored many different themes and components critical to adding privacy to decentralized systems. While there are many ways to use each part of this work individually or in some combination, it is useful to conclude by showing a hypothetical example where all of these pieces come together. Ideally, this hypothetical system can be viewed as a concrete way to build a privacy-first decentralized system, or at the very least, inspire future work.

In order to illustrate, we will focus on a hypothetical decentralized healthcare system. Healthcare systems and databases today are fragmented across many local and siloed systems, so combining all of these into a single (decentralized) system accessible by both patients and providers is meaningful. However, this naturally raises a lot of privacy, security and regulatory concerns. Our goal here is not to solve all of these one-by-one, but rather to show the potential of the tools developed throughout this work in addressing some of these challenges.

Our imagined healthcare system consists of several components, all of which are run by a set of operators. The specific components of the system are:

- A *confidential smart contracts* blockchain. This blockchain is used to authenticate against the healthcare system, verify access control, and pay operators (by the users) for services.

- An open-source LLM[1] deployed on the operators. Note that the operators can perform secure inference using a technique like in [179]. This is out of scope and we treat it as a black box.

- A secret-shared vector DB containing medical information, coming from a multitude of sources (hospitals, healthcare providers, etc.). For our use-case, you can imagine this vector DB includes all medical records, personnel reports, patient surveys, etc., relevant to the patients and providers who have been on-boarded into the platform.

To further illustrate, Figure 7.1 shows a patient querying the system. In this example, the patient asks: *How do my { blood test results } compare with other diabetic patients?*. Before the system proceeds to serve the patient's query, it needs to verify she is indeed a registered patient. This is where the work in Chapters 3 and 4 come into play. The patient co-signs her query with the blockchain itself (which holds the second share of the key), and the blockchain then proceeds to ensure that the combined $sig := sig_1 + sig_2$ correspond to an authorized patient.

In this case, the system proceeds to respond to the query. We assume that the query is secret-shared between the operators, and retrieval is done privately as in Chapter 6. The most relevant documents are retrieved, also in secret-shared form, from the secret-shared medical vector database. They are then appended to the original query, and an *augmented query* is fed into the LLM, which runs a secure inference protocol (not developed as part of this thesis, but solutions exist in the literature [179]). Finally, the fee provided by the patient is paid to the operators for all their work, and the answer is sent to the patient.

We note that the hypothesized system still leaks some metadata. For example, a patient asking multiple questions will be linked to the same identity. To an extent, the work in Chapter 5 addresses that, as it tackles the problem of hiding meta-data. It would therefore allow a patient to privately pay a fee to the operators, as well as to privately authenticate as a real patient, without linking themselves to any specific patient.

But, as that scheme operates differently than your run-of-the-mill blockchain, it does not compose with the proposed systems in Chapters 3 and 4. If hiding meta-data is a requirement, some more work is needed in order to combine these ideas together.

While the decentralized medical system discussed is theoretical, it effectively highlights the critical importance of privacy in decentralized systems and demonstrates how the various components developed throughout this thesis interconnect. By employing advanced techniques such as confidential smart contracts, server-aided multi-party computation, and privacy-preserving data retrieval, we can create systems that not only maintain the integrity and availability of user data but also protect the fundamental right to privacy in our digital era. As decentralized technologies progress, they possess the potential to reshape our interaction with digital services, fostering a more secure and privacy-focused future. The advancements presented in this thesis provide a solid foundation for ongoing research and innovation, encouraging the community to continue pushing boundaries. Ultimately, this work aspires to contribute to a digital ecosystem where privacy is integral, not an afterthought.

[1]Many open-source LLM models, such as LLama 2 and 3 are available through https://huggingface.co.

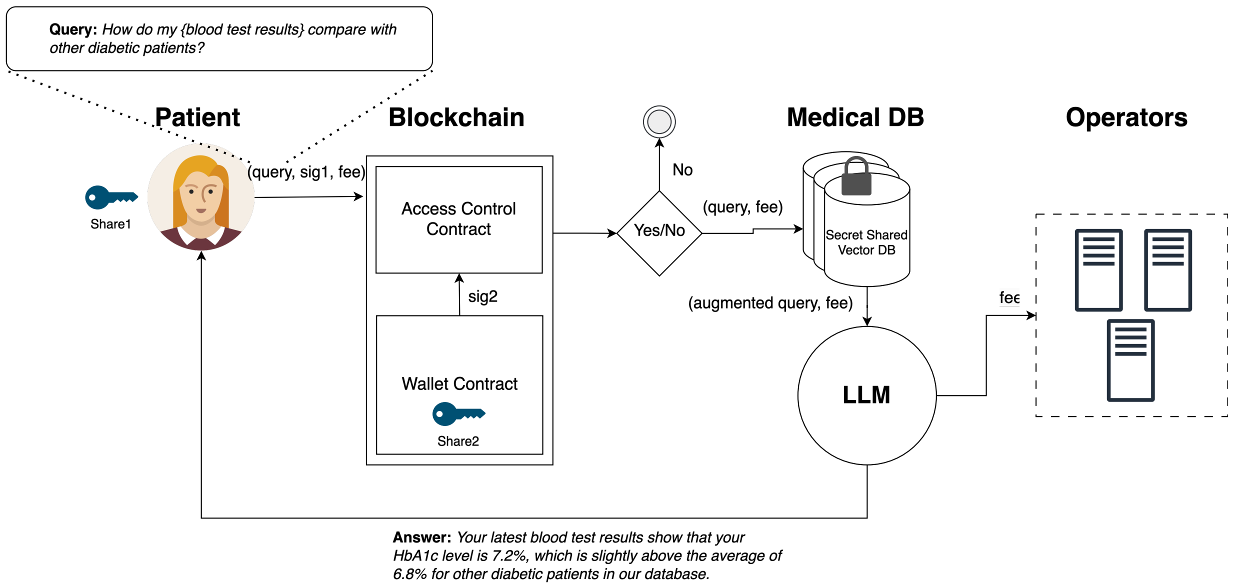

Query: How do my {blood test results} compare with other diabetic patients?
Patient
Blockchain
Medical DB
Operators
Share1
(query, sig1, fee)
Access Control Contract
sig2
Wallet Contract
Share2
No
Yes/No
(query, fee)
Secret Shared Vector DB
(augmented query, fee)
LLM
fee
Answer: Your latest blood test results show that your HbA1c level is 7.2%, which is slightly above the average of 6.8% for other diabetic patients in our database.

Figure 7.1: An illustrative decentralized medical system combining elements from this thesis to solve privacy

References

[1] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” *Consulted*, vol. 1, no. 2012, p. 28, 2008.

[2] V. Buterin, *Ethereum: A next-generation smart contract and decentralized application platform*, https://ethereum.org/en/whitepaper/, 2014.

[3] G. Zyskind, O. Nathan, *et al.*, “Decentralizing privacy: Using blockchain to protect personal data,” in *2015 IEEE security and privacy workshops*, IEEE, 2015, pp. 180–184.

[4] G. Zyskind, O. Nathan, and A. Pentland, “Enigma: Decentralized computation platform with guaranteed privacy,” *arXiv preprint arXiv:1506.03471*, 2015.

[5] R. Li, Q. Wang, Q. Wang, D. Galindo, and M. Ryan, “Sok: Tee-assisted confidential smart contract,” *arXiv preprint arXiv:2203.08548*, 2022.

[6] S. Dolev and Z. Wang, “Sodsmpc: Fsm based anonymous and private quantum-safe smart contracts,” in *2020 IEEE 19th International Symposium on Network Computing and Applications (NCA)*, IEEE, 2020, pp. 1–10.

[7] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, “Hawk: The blockchain model of cryptography and privacy-preserving smart contracts,” in *2016 IEEE symposium on security and privacy (SP)*, IEEE, 2016, pp. 839–858.

[8] A. Banerjee, M. Clear, and H. Tewari, “Zkhawk: Practical private smart contracts from mpc-based hawk,” in *2021 3rd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*, IEEE, 2021, pp. 245–248.

[9] R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. Johnson, A. Juels, A. Miller, and D. Song, “Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts,” in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, IEEE, 2019, pp. 185–200.

[10] S. Steffen, B. Bichsel, R. Baumgartner, and M. Vechev, “Zeestar: Private smart contracts by homomorphic encryption and zero-knowledge proofs,” in *2022 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2022, pp. 179–197.

[11] R. Solomon and G. Almashaqbeh, “Smartfhe: Privacy-preserving smart contracts from fully homomorphic encryption,” *Cryptology ePrint Archive*, 2021.

[12] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, “Zerocash: Decentralized anonymous payments from bitcoin,” in *2014 IEEE symposium on security and privacy*, IEEE, 2014, pp. 459–474.

[13] T. P. Pedersen, "A threshold cryptosystem without a trusted party," in *Advances in CryptologyâEUROCRYPTâ91*, Springer, 1991, pp. 522–526.

[14] B. Schoenmakers, "A simple publicly verifiable secret sharing scheme and its application to electronic," in *CRYPTO*, vol. 1666, Springer, 1999, pp. 148–164.

[15] M. Bellare, V. T. Hoang, and P. Rogaway, "Foundations of garbled circuits," in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012, pp. 784–796.

[16] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game, or a completeness theorem for protocols with honest majority," in *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, 2019, pp. 307–328.

[17] Y. Lindell, "How to simulate it–a tutorial on the simulation proof technique," *Tutorials on the Foundations of Cryptography: Dedicated to Oded Goldreich*, pp. 277–346, 2017.

[18] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," in *Advances in CryptologyâCRYPTOâ84*, Springer, 1985, pp. 10–18.

[19] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," *EUROCRYPT*, pp. 223–238, 1999.

[20] E. Boyle, N. Gilboa, and Y. Ishai, "Function secret sharing: Improvements and extensions," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 1292–1303.

[21] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan, "Private information retrieval," *Journal of the ACM (JACM)*, vol. 45, no. 6, pp. 965–981, 1998.

[22] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious rams," *Journal of the ACM (JACM)*, vol. 43, no. 3, pp. 431–473, 1996.

[23] S. Lu and R. Ostrovsky, "Distributed oblivious ram for secure two-party computation," in *Theory of Cryptography Conference*, Springer, 2013, pp. 377–396.

[24] X. Wang, H. Chan, and E. Shi, "Circuit oram: On tightness of the goldreich-ostrovsky lower bound," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 850–861.

[25] S. Zahur, X. Wang, M. Raykova, A. Gascón, J. Doerner, D. Evans, and J. Katz, "Revisiting square-root oram: Efficient random access in multi-party computation," in *2016 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2016, pp. 218–234.

[26] E. Stefanov, M. v. Dijk, E. Shi, T.-H. H. Chan, C. Fletcher, L. Ren, X. Yu, and S. Devadas, "Path oram: An extremely simple oblivious ram protocol," *Journal of the ACM (JACM)*, vol. 65, no. 4, pp. 1–26, 2018.

[27] N. Jean-Louis, Y. Li, Y. Ji, H. Malvai, T. Yurek, S. Bellemare, and A. Miller, "Sgxonerated: Finding (and partially fixing) privacy flaws in tee-based smart contract platforms without breaking the tee," *Cryptology ePrint Archive*, 2023.

[28] SCRT, *The secret network graypaper*, https://scrt.network/graypaper, 2021.

[29] M. Brandenburger, C. Cachin, R. Kapitza, and A. Sorniotti, "Blockchain and trusted computing: Problems, pitfalls, and a solution for hyperledger fabric," *arXiv preprint arXiv:1805.08541*, 2018.

[30] H. Yin, S. Zhou, and J. Jiang, *Phala network: A confidential smart contract network based on polkadot*, 2019.

[31] G. Zyskind, "Efficient secure computation enabled by blockchain technology," Ph.D. dissertation, Massachusetts Institute of Technology, 2016.

[32] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric," in *Peer-to-Peer Systems*, Springer, 2002, pp. 53–65.

[33] D. Johnson, A. Menezes, and S. Vanstone, "The elliptic curve digital signature algorithm (ecdsa)," *International Journal of Information Security*, vol. 1, no. 1, pp. 36–63, 2001.

[34] F. Information and P. Standards, "FIPS PUB 180-4 Secure Hash Standard (SHS)," no. March, 2012.

[35] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[36] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation," in *Proceedings of the twentieth annual ACM symposium on Theory of computing*, ACM, 1988, pp. 1–10.

[37] C. Gentry, "Fully homomorphic encryption using ideal lattices.," in *STOC*, vol. 9, 2009, pp. 169–178.

[38] Z. AI, *Fhevm whitepaper*, https://github.com/zama-ai/fhevm/blob/main/fhevm-whitepaper.pdf, Accessed: 27-10-2023, 2023.

[39] J. Poon and V. Buterin, *Plasma: Scalable autonomous smart contracts*, https://plasma.io/plasma.pdf, Accessed: dd-mm-yyyy, 2017.

[40] H. Kalodner, S. Goldfeder, X. Chen, S. M. Weinberg, and E. W. Felten, "Arbitrum: Scalable, private smart contracts," in *27th USENIX Security Symposium (USENIX Security 18)*, Accessed: dd-mm-yyyy, USENIX Association, 2018, pp. 1353–1370. URL: https://www.usenix.org/conference/usenixsecurity18/presentation/kalodner.

[41] O. PBC, *Optimism: Optimistic ethereum*, https://optimism.io/, Accessed: 27-10-2023, 2022.

[42] A. Viand, C. Knabenhans, and A. Hithnawi, "Verifiable fully homomorphic encryption," *arXiv preprint arXiv:2301.07041*, 2023.

[43] G. Chen, S. Chen, Y. Xiao, Y. Zhang, Z. Lin, and T. H. Lai, "Sgxpectre: Stealing intel secrets from sgx enclaves via speculative execution," in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, IEEE, 2019, pp. 142–157.

[44] S. Van Schaik, A. Kwong, D. Genkin, and Y. Yarom, *Sgaxe: How sgx fails in practice*, 2020. URL: https://sgaxe.com/files/SGAxe.pdf.

[45] P. Borrello, A. Kogler, M. Schwarzl, M. Lipp, D. Gruss, and M. Schwarz, "{Æpic} leak: Architecturally leaking uninitialized data from the microarchitecture," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 3917–3934.

[46] C. Baum, J. H.-y. Chiang, B. David, and T. K. Frederiksen, "Eagle: Efficient privacy preserving smart contracts," *Cryptology ePrint Archive*, 2022.

[47] C. Baum, B. David, and R. Dowsley, "Insured mpc: Efficient secure computation with financial penalties," in *Financial Cryptography and Data Security: 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10–14, 2020 Revised Selected Papers 24*, Springer, 2020, pp. 404–420.

[48] C. Gentry, S. Halevi, H. Krawczyk, B. Magri, J. B. Nielsen, T. Rabin, and S. Yakoubov, "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*, Springer, 2021, pp. 64–93.

[49] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game," *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pp. 218–229, 1987.

[50] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation," in *Proceedings of the twentieth annual ACM symposium on Theory of computing*, ACM, 1988, pp. 1–10.

[51] A. C. Yao, "How to generate and exchange secrets," in *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, IEEE, 1986, pp. 162–167.

[52] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: Nearly practical verifiable computation," in *2013 IEEE Symposium on Security and Privacy*, IEEE, 2013, pp. 238–252.

[53] S. Bowe, A. Chiesa, M. Green, I. Miers, P. Mishra, and H. Wu, "Zexe: Enabling decentralized private computation," in *2020 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2020, pp. 947–964.

[54] B. Bünz, S. Agrawal, M. Zamani, and D. Boneh, "Zether: Towards privacy in a smart contract world," in *International Conference on Financial Cryptography and Data Security*, Springer, 2020, pp. 423–443.

[55] A. S. Inc., *Aleo: A zero-knowledge operating system*, https://aleo.org/, Accessed: dd-mm-yyyy, 2022.

[56] Z. J. Williamson, "The aztec protocol," *URL: https://github. com/AztecProtocol/AZTEC*, 2018.

[57] W. Dai, "Pesca: A privacy-enhancing smart-contract architecture," *Cryptology ePrint Archive*, 2022.

[58] S. Steffen, B. Bichsel, M. Gersbach, N. Melchior, P. Tsankov, and M. Vechev, "Zkay: Specifying and enforcing data privacy in smart contracts," in *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, 2019, pp. 1759–1776.

[59] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Scalable zero knowledge with no trusted setup," in *Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part III 39*, Springer, 2019, pp. 701–732.

[60] V. Buterin, *Zkevm and zkrollup*, https://vitalik.ca/general/2022/08/04/zkevm.html, Accessed: dd-mm-yyyy, 2022.

[61] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," *Journal of the ACM (JACM)*, vol. 56, no. 6, pp. 1–40, 2009.

[62] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "Tfhe: Fast fully homomorphic encryption over the torus," *Journal of Cryptology*, vol. 33, no. 1, pp. 34–91, 2020.

[63] R. Canetti, B. Riva, and G. N. Rothblum, "Practical delegation of computation using multiple servers," in *Proceedings of the 18th ACM conference on Computer and communications security*, 2011, pp. 445–454.

[64] R. Canetti, B. Riva, and G. N. Rothblum, "Refereed delegation of computation," *Information and Computation*, vol. 226, pp. 16–36, 2013.

[65] M. Dahl, D. Demmler, S. El Kazdadi, A. Meyre, J.-B. Orfila, D. Rotaru, N. P. Smart, S. Tap, and M. Walter, "Noah's ark: Efficient threshold-fhe using noise flooding," *Cryptology ePrint Archive*, 2023.

[66] U. Kirstein, S. Grossman, M. Mirkin, J. Wilcox, I. Eyal, and M. Sagiv, "Phoenix: A formally verified regenerating vault," *arXiv preprint arXiv:2106.01240*, 2021.

[67] S. Sharma, *Crypto custodian prime trust frozen amid massive fund deficits and missing keys.* URL: https://beincrypto.com/crypto-custodian-prime-trust-missing-keys.

[68] R. Brandom, *Impersonation attack on myetherwallet.* URL: https://www.theverge.com/2018/4/24/17275982/myetherwallet-hack-bgp-dns-hijacking-stolen-ethereum.

[69] B. Karmakar, N. Koti, A. Patra, S. Patranabis, P. Paul, and D. Ravi, "Sfasterisk: Super-fast MPC with a friend," *IACR Cryptol. ePrint Arch.*, p. 1098, 2023.

[70] O. Nevo, N. Trieu, and A. Yanai, "Simple, fast malicious multiparty private set intersection," in *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, Y. Kim, J. Kim, G. Vigna, and E. Shi, Eds., ACM, 2021, pp. 1151–1165.

[71] S. K. D. Maram, F. Zhang, L. Wang, A. Low, Y. Zhang, A. Juels, and D. Song, "Churp: Dynamic-committee proactive secret sharing," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 2369–2386.

[72] V. Goyal, A. Kothapalli, E. Masserova, B. Parno, and Y. Song, "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*, Springer, 2022, pp. 252–282.

[73] V. Goyal, E. Masserova, B. Parno, and Y. Song, “Blockchains enable non-interactive mpc,” in *Theory of Cryptography: 19th International Conference, TCC 2021, Raleigh, NC, USA, November 8–11, 2021, Proceedings, Part II 19*, Springer, 2021, pp. 162–193.

[74] A. R. Choudhuri, M. Green, A. Jain, G. Kaptchuk, and I. Miers, “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, pp. 719–728.

[75] R. Gennaro 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, pp. 1179–1194.

[76] Y. Lindell 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, pp. 1837–1854.

[77] D. Boneh, R. Gennaro, and S. Goldfeder, “Using level-1 homomorphic encryption to improve threshold dsa signatures for bitcoin wallet security,” in *Progress in Cryptology–LATINCRYPT 2017: 5th International Conference on Cryptology and Information Security in Latin America, Havana, Cuba, September 20–22, 2017, Revised Selected Papers*, Springer, 2019, pp. 352–377.

[78] R. Gennaro and S. Goldfeder, “One round threshold ecdsa with identifiable abort,” *Cryptology ePrint Archive*, 2020.

[79] R. Canetti, R. Gennaro, S. Goldfeder, N. Makriyannis, and U. Peled, “Uc non-interactive, proactive, threshold ecdsa with identifiable aborts,” in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 1769–1787.

[80] Y. Lindell, “Fast secure two-party ECDSA signing,” in *CRYPTO, 2017*, ser. Lecture Notes in Computer Science, vol. 10402, pp. 613–644.

[81] H. Xue, M. H. Au, X. Xie, T. H. Yuen, and H. Cui, “Efficient online-friendly two-party ecdsa signature,” in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 558–573.

[82] G. Castagnos, D. Catalano, F. Laguillaumie, F. Savasta, and I. Tucker, “Two-party ecdsa from hash proof systems and efficient instantiations,” in *Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part III 39*, Springer, 2019, pp. 191–221.

[83] G. Castagnos, D. Catalano, F. Laguillaumie, F. Savasta, and I. Tucker, “Bandwidth-efficient threshold ec-dsa,” in *Public-Key Cryptography–PKC 2020: 23rd IACR International Conference on Practice and Theory of Public-Key Cryptography, Edinburgh, UK, May 4–7, 2020, Proceedings, Part II*, Springer, 2020, pp. 266–296.

[84] G. Castagnos, D. Catalano, F. Laguillaumie, F. Savasta, and I. Tucker, "Bandwidth-efficient threshold ec-dsa revisited: Online/offline extensions, identifiable aborts proactive and adaptive security," *Theoretical Computer Science*, vol. 939, pp. 78–104, 2023.

[85] J. Doerner, Y. Kondi, E. Lee, and A. Shelat, "Secure two-party threshold ecdsa from ecdsa assumptions," in *2018 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2018, pp. 980–997.

[86] J. Doerner, Y. Kondi, E. Lee, and A. Shelat, "Threshold ecdsa from ecdsa assumptions: The multiparty case," in *2019 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2019, pp. 1051–1066.

[87] A. Dalskov, C. Orlandi, M. Keller, K. Shrishak, and H. Shulman, "Securing dnssec keys via threshold ecdsa from generic mpc," in *Computer Security–ESORICS 2020: 25th European Symposium on Research in Computer Security, ESORICS 2020, Guildford, UK, September 14–18, 2020, Proceedings, Part II 25*, Springer, 2020, pp. 654–673.

[88] D. Abram, A. Nof, C. Orlandi, P. Scholl, and O. Shlomovits, "Low-bandwidth threshold ecdsa via pseudorandom correlation generators," in *2022 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2022, pp. 2554–2572.

[89] I. Damgård, T. P. Jakobsen, J. B. Nielsen, J. I. Pagter, and M. B. Østergaard, "Fast threshold ecdsa with honest majority," *Journal of Computer Security*, vol. 30, no. 1, pp. 167–196, 2022.

[90] R. Cleve, "Limits on the security of coin flips when half the processors are faulty (extended abstract)," in *STOC*, J. Hartmanis, Ed., 1986.

[91] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Robust threshold dss signatures," in *Advances in CryptologyâEUROCRYPTâ96: International Conference on the Theory and Application of Cryptographic Techniques Saragossa, Spain, May 12–16, 1996 Proceedings 15*, Springer, 1996, pp. 354–371.

[92] A. Gągol, J. Kula, D. Straszak, and M. Świętek, "Threshold ecdsa for decentralized asset custody," *Cryptology ePrint Archive*, 2020.

[93] H. W. Wong, J. P. Ma, H. H. Yin, and S. S. Chow, "Real threshold ecdsa,"

[94] I. Bentov and R. Kumaresan, "How to use bitcoin to design fair protocols," in *Advances in Cryptology–CRYPTO 2014: 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II 34*, Springer, 2014, pp. 421–439.

[95] R. Kumaresan and I. Bentov, "Amortizing secure computation with penalties," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 418–429.

[96] R. Kumaresan, V. Vaikuntanathan, and P. N. Vasudevan, "Improvements to secure computation with penalties," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 406–417.

[97] C. Baum, I. Damgård, and C. Orlandi, "Publicly auditable secure multi-party computation," in *Security and Cryptography for Networks: 9th International Conference, SCN 2014, Amalfi, Italy, September 3-5, 2014. Proceedings 9*, Springer, 2014, pp. 175–196.

[98] C. Baum, E. Orsini, P. Scholl, and E. Soria-Vazquez, "Efficient constant-round mpc with identifiable abort and public verifiability," in *Advances in Cryptology–CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part II*, Springer, 2020, pp. 562–592.

[99] M. Rivinius, P. Reisert, D. Rausch, and R. Küsters, "Publicly accountable robust multi-party computation," in *2022 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2022, pp. 2430–2449.

[100] A. R. Choudhuri, A. Goel, M. Green, A. Jain, and G. Kaptchuk, "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*, Springer, 2021, pp. 94–123.

[101] D. Lu, T. Yurek, S. Kulshreshtha, R. Govind, A. Kate, and A. Miller, "Honeybadgermpc and asynchromix: Practical asynchronous mpc and its application to anonymous communication," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 887–903.

[102] F. Benhamouda, C. Gentry, S. Gorbunov, S. Halevi, H. Krawczyk, C. Lin, T. Rabin, and L. Reyzin, "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*, Springer, 2020, pp. 260–290.

[103] R. Vassantlal, E. Alchieri, B. Ferreira, and A. Bessani, "Cobra: Dynamic proactive secret sharing for confidential bft services," in *2022 IEEE symposium on security and privacy (SP)*, IEEE, 2022, pp. 1335–1353.

[104] S. Das, T. Yurek, Z. Xiang, A. Miller, L. Kokoris-Kogias, and L. Ren, "Practical asynchronous distributed key generation," in *2022 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2022, pp. 2518–2534.

[105] T. Frassetto, P. Jauernig, D. Koisser, D. Kretzler, B. Schlosser, S. Faust, and A.-R. Sadeghi, "Pose: Practical off-chain smart contract execution," *arXiv preprint arXiv:2210.07110*, 2022.

[106] D. Demirag and J. Clark, "Absentia: Secure multiparty computation on ethereum," in *Financial Cryptography and Data Security. FC 2021 International Workshops: CoDecFin, DeFi, VOTING, and WTSC, Virtual Event, March 5, 2021, Revised Selected Papers 25*, Springer, 2021, pp. 381–396.

[107] A. L. Xiong, B. Chen, Z. Zhang, B. Bünz, B. Fisch, F. Krell, and P. Camacho, "Verizexe: Decentralized private computation with universal setup," *Cryptology ePrint Archive*, 2022.

[108] J. Lind, O. Naor, I. Eyal, F. Kelbert, E. G. Sirer, and P. Pietzuch, "Teechain: A secure payment network with asynchronous blockchain access," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, 2019, pp. 63–79.

[109] I. Bentov, Y. Ji, F. Zhang, L. Breidenbach, P. Daian, and A. Juels, “Tesseract: Real-time cryptocurrency exchange using trusted hardware,” in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 1521–1538.

[110] F. Zhang, E. Cecchetti, K. Croman, A. Juels, and E. Shi, “Town crier: An authenticated data feed for smart contracts,” in *Proceedings of the 2016 aCM sIGSAC conference on computer and communications security*, 2016, pp. 270–282.

[111] S. Steffen, B. Bichsel, and M. Vechev, “Zapper: Smart contracts with data and identity privacy,” in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 2735–2749.

[112] C. Blokh, N. Makriyannis, and U. Peled, “Efficient asymmetric threshold ECDSA for mpc-based cold storage,” *IACR Cryptol. ePrint Arch.*, p. 1296, 2022.

[113] J. V. Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, “Foreshadow: Extracting the keys to the intel SGX kingdom with transient out-of-order execution,” in *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, W. Enck and A. P. Felt, Eds., USENIX Association, 2018, pp. 991–1008. URL: https://www.usenix.org/conference/usenixsecurity18/presentation/bulck.

[114] S. van Schaik, A. Seto, T. Yurek, A. Batori, B. AlBassam, C. Garman, D. Genkin, A. Miller, E. Ronen, and Y. Yarom, *Sok: Sgx. fail: How stuff get exposed*, 2022.

[115] E. Buchman, “Tendermint: Byzantine fault tolerance in the age of blockchains,” in *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, ACM, 2019, pp. 49–61.

[116] N. Gilboa and Y. Ishai, “Distributed point functions and their applications,” 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*, Springer, 2014, pp. 640–658.

[117] E. Boyle, N. Gilboa, and Y. Ishai, “Function secret sharing,” in *Annual international conference on the theory and applications of cryptographic techniques*, Springer, 2015, pp. 337–367.

[118] H. Corrigan-Gibbs and D. Boneh, “Prio: Private, robust, and scalable computation of aggregate statistics,” in *14th USENIX symposium on networked systems design and implementation (NSDI 17)*, 2017, pp. 259–282.

[119] D. Boneh, E. Boyle, H. Corrigan-Gibbs, N. Gilboa, and Y. Ishai, “Lightweight techniques for private heavy hitters,” in *2021 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2021, pp. 762–776.

[120] Z. Newman, S. Servan-Schreiber, and S. Devadas, “Spectrum: High-bandwidth anonymous broadcast with malicious security.,” *IACR Cryptol. ePrint Arch.*, vol. 2021, p. 325, 2021.

[121] H. Corrigan-Gibbs, D. Boneh, and D. Mazières, "Riposte: An anonymous messaging system handling millions of users," in *2015 IEEE Symposium on Security and Privacy*, IEEE, 2015, pp. 321–338.

[122] T. Ryffel, P. Tholoniat, D. Pointcheval, and F. Bach, "Ariann: Low-interaction privacy-preserving deep learning via function secret sharing," *arXiv preprint arXiv:2006.04593*, 2020.

[123] G. Kaissis, A. Ziller, J. Passerat-Palmbach, T. Ryffel, D. Usynin, A. Trask, I. Lima Jr, J. Mancuso, F. Jungmann, M.-M. Steinborn, *et al.*, "End-to-end privacy preserving deep learning on multi-institutional medical imaging," *Nature Machine Intelligence*, vol. 3, no. 6, pp. 473–484, 2021.

[124] S. Servan-Schreiber, S. Langowski, and S. Devadas, "Private approximate nearest neighbor search with sublinear communication," in *2022 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2022, pp. 911–929.

[125] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, "Delphi: A cryptographic inference system for neural networks," in *Proceedings of the 2020 Workshop on Privacy-Preserving Machine Learning in Practice*, 2020, pp. 27–30.

[126] N. Trieu, K. Shehata, P. Saxena, R. Shokri, and D. Song, "Epione: Lightweight contact tracing with strong privacy," *arXiv preprint arXiv:2004.13293*, 2020.

[127] A. Vadapalli, R. Henry, and I. Goldberg, "Duoram: A bandwidth-efficient distributed oram for 2-and 3-party computation," in *32nd USENIX Security Symposium*, 2023.

[128] S. Sasy, A. Vadapalli, and I. Goldberg, "Prac: Round-efficient 3-party mpc for dynamic data structures," *Cryptology ePrint Archive*, 2023.

[129] J. Doerner and A. Shelat, "Scaling oram for secure computation," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 523–535.

[130] P. Bunn, J. Katz, E. Kushilevitz, and R. Ostrovsky, "Efficient 3-party distributed oram," in *Security and Cryptography for Networks: 12th International Conference, SCN 2020, Amalfi, Italy, September 14–16, 2020, Proceedings 12*, Springer, 2020, pp. 215–232.

[131] E. Cecchetti, F. Zhang, Y. Ji, A. Kosba, A. Juels, and E. Shi, "Solidus: Confidential distributed ledger transactions via pvorm," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 701–717.

[132] N. Narula, W. Vasquez, and M. Virza, "{Zkledger}:{privacy-preserving} auditing for distributed ledgers," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018, pp. 65–80.

[133] P. Chatzigiannis and F. Baldimtsi, "Miniledger: Compact-sized anonymous and auditable distributed payments," in *European Symposium on Research in Computer Security*, Springer, 2021, pp. 407–429.

[134] P. Fauzi, S. Meiklejohn, R. Mercer, and C. Orlandi, "Quisquis: A new design for anonymous cryptocurrencies," in *Advances in Cryptology–ASIACRYPT 2019: 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8–12, 2019, Proceedings, Part I 25*, Springer, 2019, pp. 649–678.

[135] B. E. Diamond, "Many-out-of-many proofs and applications to anonymous zether," in *2021 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2021, pp. 1800–1817.

[136] I. Miers, C. Garman, M. Green, and A. D. Rubin, "Zerocoin: Anonymous distributed e-cash from bitcoin," in *2013 IEEE Symposium on Security and Privacy*, IEEE, 2013, pp. 397–411.

[137] A. Tomescu, A. Bhat, B. Applebaum, I. Abraham, G. Gueta, B. Pinkas, and A. Yanai, "Utt: Decentralized ecash with accountable privacy," *Cryptology ePrint Archive*, 2022.

[138] E. Androulaki, M. Brandenburger, A. De Caro, K. Elkhiyaoui, A. Filios, L. Funaro, Y. Manevich, S. Natarajan, and M. Sethi, "A framework for resilient, transparent, high-throughput, privacy-enabled central bank digital currencies," *Cryptology ePrint Archive*, 2023.

[139] C. W. M. Tikvah, "A privacy-preserving central bank ledger for central bank digital currency," *Cryptology ePrint Archive*, 2023.

[140] J. Lovejoy, A. Brownworth, M. Virza, and N. Narula, "Parsec: Executing smart contracts in parallel,"

[141] J. Lovejoy, M. Virza, C. Fields, K. Karwaski, A. Brownworth, and N. Narula, "Hamilton: A {high-performance} transaction processor for central bank digital currencies," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, 2023, pp. 901–915.

[142] M. K. BRUNNERMEIER and J.-P. Landau, "The digital euro: Policy implications and perspectives," 2022.

[143] J. Xu, "Developments and implications of central bank digital currency: The case of china e-cny," *Asian Economic Policy Review*, vol. 17, no. 2, pp. 235–250, 2022.

[144] P. K. Ozili, "Central bank digital currency in nigeria: Opportunities and risks," in *The new digital era: digitalisation, emerging risks and opportunities*, Emerald Publishing Limited, 2022, pp. 125–133.

[145] K. Wenker, "Retail central bank digital currencies (cbdc), disintermediation and financial privacy: The case of the bahamian sand dollar," *FinTech*, vol. 1, no. 4, pp. 345–361, 2022.

[146] T. Ahnert, P. Hoffmann, and C. Monnet, "The digital economy, privacy, and cbdc," 2022.

[147] A. Vadapalli, F. Bayatbabolghani, and R. Henry, "You may also like... privacy: Recommendation systems meet pir.," *Proc. Priv. Enhancing Technol.*, vol. 2021, no. 4, pp. 30–53, 2021.

[148] P. Bunn, E. Kushilevitz, and R. Ostrovsky, “Cnf-fss and its applications,” in *IACR International Conference on Public-Key Cryptography*, Springer, 2022, pp. 283–314.

[149] E. Dauterman, M. Rathee, R. A. Popa, and I. Stoica, “Waldo: A private time-series database from function secret sharing,” in *2022 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2022, pp. 2450–2468.

[150] I. Abraham, B. Pinkas, and A. Yanai, “Blinder–scalable, robust anonymous committed broadcast,” in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 1233–1252.

[151] L. de Castro and A. Polychroniadou, “Lightweight, maliciously secure verifiable function secret sharing,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 2022, pp. 150–179.

[152] S. Servan-Schreiber, S. Beyzerov, E. Yablon, and H. Park, “Private access control for function secret sharing,” in *2023 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2023, pp. 809–828.

[153] M. Möser, K. Soska, E. Heilman, K. Lee, H. Heffan, S. Srivastava, K. Hogan, J. Hennessey, A. Miller, A. Narayanan, *et al.*, “An empirical analysis of traceability in the monero blockchain,” *arXiv preprint arXiv:1704.04299*, 2017.

[154] G. Fuchsbauer, M. Orrù, and Y. Seurin, “Aggregate cash systems: A cryptographic investigation of mimblewimble,” in *Advances in Cryptology–EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19–23, 2019, Proceedings, Part I 38*, Springer, 2019, pp. 657–689.

[155] M. Campanelli and M. Hall-Andersen, “Veksel: Simple, efficient, anonymous payments with large anonymity sets from well-studied assumptions,” in *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, 2022, pp. 652–666.

[156] A. Ranchal-Pedrosa and V. Gramoli, “Platypus: A partially synchronous offchain protocol for blockchains,” *arXiv preprint arXiv:1907.03730*, 2019.

[157] Y. Li, K. Soska, Z. Huang, S. Bellemare, M. Quintyne-Collins, L. Wang, X. Liu, D. Song, and A. Miller, “Ratel: Mpc-extensions for smart contracts,” *Cryptology ePrint Archive*, 2023.

[158] V. Madathil and A. Scafuro, “Prifhete: Achieving full-privacy in account-based cryptocurrencies is possible,” *Cryptology ePrint Archive*, 2023.

[159] K. Chida, D. Genkin, K. Hamada, D. Ikarashi, R. Kikuchi, Y. Lindell, and A. Nof, “Fast large-scale honest-majority MPC for malicious adversaries,” in *CRYPTO*, H. Shacham and A. Boldyreva, Eds., 2018.

[160] R. Cramer, I. Damgård, and Y. Ishai, “Share conversion, pseudorandom secret-sharing and applications to secure computation,” in *Theory of Cryptography: Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005. Proceedings 2*, Springer, 2005, pp. 342–362.

[161] I. Damgård and J. B. Nielsen, "Scalable and unconditionally secure multiparty computation," in *Annual International Cryptology Conference*, Springer, 2007, pp. 572–590.

[162] T. Nishide and K. Ohta, "Multiparty computation for interval, equality, and comparison without bit-decomposition protocol," in *Public Key Cryptography–PKC 2007: 10th International Conference on Practice and Theory in Public-Key Cryptography Beijing, China, April 16-20, 2007. Proceedings 10*, Springer, 2007, pp. 343–360.

[163] O. Catrina and S. De Hoogh, "Improved primitives for secure multiparty integer computation," in *Security and Cryptography for Networks: 7th International Conference, SCN 2010, Amalfi, Italy, September 13-15, 2010. Proceedings 7*, Springer, 2010, pp. 182–199.

[164] I. Damgård, M. Fitzi, E. Kiltz, J. B. Nielsen, and T. Toft, "Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation," in *Theory of Cryptography Conference*, Springer, 2006, pp. 285–304.

[165] E. Makri, D. Rotaru, F. Vercauteren, and S. Wagh, "Rabbit: Efficient comparison for secure multi-party computation," in *International Conference on Financial Cryptography and Data Security*, Springer, 2021, pp. 249–270.

[166] D. Escudero, S. Ghosh, M. Keller, R. Rachuri, and P. Scholl, "Improved primitives for mpc over mixed arithmetic-binary circuits," in *Advances in Cryptology–CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part II 40*, Springer, 2020, pp. 823–852.

[167] G. Zyskind, T. South, and A. Pentland, "Don't forget private retrieval: Distributed private similarity search for large language models," *arXiv preprint arXiv:2311.12955*, 2023.

[168] I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Scholl, and N. P. Smart, "Practical covertly secure mpc for dishonest majority–or: Breaking the spdz limits," in *Computer Security–ESORICS 2013: 18th European Symposium on Research in Computer Security, Egham, UK, September 9-13, 2013. Proceedings 18*, Springer, 2013, pp. 1–18.

[169] D. Kales, O. Omolola, and S. Ramacher, "Revisiting user privacy for certificate transparency," in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, IEEE, 2019, pp. 432–447.

[170] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large language models are zero-shot reasoners," *ArXiv*, 2022.

[171] T. Brown, B. Mann, N. Ryder, *et al.*, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., Curran Associates, Inc., 2020, pp. 1877–1901.

[172] T. Eloundou, S. Manning, P. Mishkin, and D. Rock, "Gpts are gpts: An early look at the labor market impact potential of large language models," *ArXiv*, 2023.

[173] OpenAI, "Gpt-4 technical report," *ArXiv*, 2023.

[174] P. Lewis, E. Perez, A. Piktus, *et al.*, "Retrieval-augmented generation for knowledge-intensive nlp tasks," *ArXiv*, 2020.

[175] V. Karpukhin, B. OÄuz, S. Min, P. Lewis, L. Y. Wu, S. Edunov, D. Chen, and W.-t. Yih, "Dense passage retrieval for open-domain question answering," in *Conference on Empirical Methods in Natural Language Processing*, 2020.

[176] Y. Mao, P. He, X. Liu, Y. Shen, J. Gao, J. Han, and W. Chen, "Generation-augmented retrieval for open-domain question answering," in *Annual Meeting of the Association for Computational Linguistics*, 2020.

[177] P. Kairouz, H. B. McMahan, B. Avent, *et al.*, "Advances and open problems in federated learning," *Found. Trends Mach. Learn.*, vol. 14, pp. 1–210, 2019.

[178] D. Li, R. Shao, H. Wang, H. Guo, E. P. Xing, and H. Zhang, "Mpcformer: Fast, performant and private transformer inference with mpc," *ArXiv*, 2022.

[179] Y. Dong, W.-j. Lu, Y. Zheng, H. Wu, D. Zhao, J. Tan, Z. Huang, C. Hong, T. Wei, and W.-C. Cheng, "Puma: Secure inference of llama-7b in five minutes," *ArXiv*, 2023.

[180] T. South, G. Zuskind, R. Mahari, and T. Hardjono, "Secure community transformers: Private pooled data for llms," 2023.

[181] F. Mo, A. S. Shamsabadi, K. Katevas, S. Demetriou, I. Leontiadis, A. Cavallaro, and H. Haddadi, "Darknetz: Towards model privacy at the edge using trusted execution environments," *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, 2020.

[182] Y. Akimoto, K. Fukuchi, Y. Akimoto, and J. Sakuma, "Privformer: Privacy-preserving transformer with mpc," in *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*, IEEE, 2023, pp. 392–410.

[183] T. Chen, H. Bao, S. Huang, L. Dong, B. Jiao, D. Jiang, H. Zhou, J. Li, and F. Wei, "The-x: Privacy-preserving transformer inference with homomorphic encryption," *arXiv preprint arXiv:2206.00216*, 2022.

[184] K. Gupta, N. Jawalkar, A. Mukherjee, N. Chandran, D. Gupta, A. Panwar, and R. Sharma, "Sigma: Secure gpt inference with function secret sharing," *Cryptology ePrint Archive*, 2023.

[185] S. Arora and C. RÃ©, *Can foundation models help us achieve perfect secrecy?* 2022.

[186] M. Ito, A. Saito, and T. Nishizeki, "Secret sharing scheme realizing general access structure," *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, vol. 72, no. 9, pp. 56–64, 1989.

[187] M. S. Riazi, C. Weinert, O. Tkachenko, E. M. Songhori, T. Schneider, and F. Koushanfar, "Chameleon: A hybrid secure computation framework for machine learning applications," in *Proceedings of the 2018 on Asia conference on computer and communications security*, 2018, pp. 707–721.

[188] B. Knott, S. Venkataraman, A. Hannun, S. Sengupta, M. Ibrahim, and L. van der Maaten, "Crypten: Secure multi-party computation meets machine learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 4961–4973, 2021.

[189] E. Orsini, N. P. Smart, and F. Vercauteren, “Overdrive2k: Efficient secure mpc over from somewhat homomorphic encryption,” in *Cryptographersâ Track at the RSA Conference*, Springer, 2020, pp. 254–283.

[190] O. Catrina and A. Saxena, “Secure computation with fixed-point numbers,” in *Financial Cryptography and Data Security: 14th International Conference, FC 2010, Tenerife, Canary Islands, January 25-28, 2010, Revised Selected Papers 14*, Springer, 2010, pp. 35–50.

[191] N. Reimers and I. Gurevych, “Sentence-BERT: Sentence embeddings using siamese BERT-networks,” in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, Nov. 2019.

[192] H. Chen, I. Chillotti, Y. Dong, O. Poburinnaya, I. Razenshteyn, and M. S. Riazi, “{Sanns}: Scaling up secure approximate {k-nearest} neighbors search,” in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 2111–2128.

[193] J. Johnson, M. Douze, and H. Jégou, “Billion-scale similarity search with gpus,” *IEEE Transactions on Big Data*, vol. 7, pp. 535–547, 2017.

[194] H. Jégou, M. Douze, and C. Schmid, “Product quantization for nearest neighbor search,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 117–128, 2011.

[195] N. Thakur, N. Reimers, A. Rücklé, A. Srivastava, and I. Gurevych, “BEIR: A heterogeneous benchmark for zero-shot evaluation of information retrieval models,” in *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

[196] P. Bunn, E. Kushilevitz, and R. Ostrovsky, “Cnf-fss and its applications,” in *IACR International Conference on Public-Key Cryptography*, Springer, 2022, pp. 283–314.