A Privacy-preserving Decentralized Storage with Payments Based on a Blockchain

Dissertation zur Erlangung des Doktorgrades Dr. rer. nat. der Fakultät für Ingenieurwissenschaften, Informatik und Psychologie der Universität Ulm

Henning Johannes Gustav Kopp

aus Villingen-Schwenningen

2018

Institut für Verteilte Systeme
Universität Ulm, Deutschland
Amtierender Dekan: Prof. Maurits Ortmanns
Gutachter: Prof. Frank Kargl
Gutachter: Prof. Frederik Armknecht
Tag der Promotion: 20.12.2018
Recently, the paradigm of cloud storage has seen wide acceptance in industry and for personal use. One of its core principles is to outsource storage, such that users can be billed flexibly by their actual demand.

However, outsourcing storage such as private data or business secrets leads to privacy problems, as control over the data is lost to the storage provider. This is intensified by the fact that often privacy is considered only as an afterthought in these systems and not integrated into the design from the beginning.

Privacy-preserving alternatives to these centralized cloud storage providers are peer-to-peer systems like Freenet or GNUnet. In these systems, participants can donate storage to other users of the system. Privacy plays a vital role in these systems, as, e.g., participants are unable to access data of other users if they are not authorized to do so, even if the data of the other users resides on their own hard disk. However, these decentralized systems suffer from limited contribution due to a lack of incentives to participate.

Naively enhancing these systems with the possibility of payments such that storage providers can earn money, infringes privacy, since tracing of payment flows provides links between users and their storage providers. Consequently, a form of anonymous payment needs to be considered when enhancing such storage systems with remuneration.

A very similar problem of providing incentives for protocol adoption is solved by the digital currency Bitcoin which rewards participants by uncloneable currency units which are themselves called bitcoins. Although Bitcoin is not privacy-preserving, the scenario of such blockchain architectures is very similar to a peer-to-peer storage platform and thus provides a good starting point.

We provide two designs of a decentralized storage system with incentives for the participants to contribute. Our first design exchanges the proof of work mining process of a blockchain by publicly verifiable proofs of storage. These are cryptographic proofs that allow a storage provider to convince a verifier that it has stored some data of another participant.

While this approach leads to a working system, it is not able to provide the envisioned privacy and security guarantees. Although the link between storage provider and user is severed, the link between a storage provider and its stored file is still observable. Further, the sender and receiver are revealed for transfers of digital cash which are unrelated to a file, as in Bitcoin.
Improving on our first design, we provide a second design of a privacy-preserving storage. Here, the senders and receivers of transactions are anonymous due to our use of linkable ring signatures and one-time payment addresses. The storage is managed via smart storage contracts which are special transactions that grant the storage provider a financial reward if the storage provider is able to provide proofs of storage at predetermined times. Since storage contracts constitute a special form of transactions, linkable ring signatures and one-time payment addresses are applicable to storage smart contracts as well, where they provide anonymity of the user and storage provider.

After introducing our two designs of a privacy-preserving storage system, we focus on selected building blocks of our architecture.

We provide a thorough comparison between different publicly verifiable proof of storage schemes. Further, we design a new publicly verifiable proof of storage scheme from a modification of the Guillou-Quisquater identification protocol whose security is based on the RSA-assumption. We implemented all discussed proof of storage schemes and provide benchmarks regarding their real-world performance.

Since our second storage system design relies on the wasteful proof of work mining process of Bitcoin, we propose a novel mining algorithm based on proofs of human work. These are solutions to problems that only humans are able to solve efficiently and thus prove cryptographically that an amount of human work has been performed. In contrast to a CAPTCHA, these proofs of human work are publicly verifiable. Our construction of a proof of human work relies on secure multiparty computation, which is a well-known cryptographic primitive with multiple known feasible instantiations. The only other instantiation of a proof of human-work known to date relies on indistinguishability obfuscation—a cryptographic primitive that is only speculated to exist and where no secure instantiation is currently known.

As a third building block, we turn our attention to routing in payment channels. Payment channels are widely regarded as a solution to the low transaction throughput of blockchains. As an example, Bitcoin allows for around 7 transactions per second globally. Payment channels are bilateral channels, which include a balance that can be updated. Only the opening and the closing of the channel is written to the blockchain. Inside the channel, there can be multiple updates to the payment which are not written to the blockchain. The design of payment channels guarantees that no participant can cheat by persisting older transaction states to the blockchain. A network of payment channels operates as an overlay network to the blockchain and allows for routing of transactions over multiple hops. Up to now, routing algorithms in this setting have only been discussed from a purely technological perspective without taking into account the economical impact, like capacity constraints, routing fees, cost of locked capital, and the focus on the cheapest route (as opposed to the shortest route). Our treatment provides the first economic-technical analysis of routing in payment channels. We provide measurements of the cheapest route and the number of failed transactions due to capacity constraints using our custom simulator.

In summary, our results show that it is possible to design a decentralized privacy-preserving storage system with remuneration for it participants. With the addition of anonymous payments we hope to increase participation beyond that of traditional decentralized storage systems.
Short contents

1 Introduction 1
2 Background 9
3 Basic Design 37
4 Privacy Enhanced Design 59
5 A Comparison of Publicly Verifiable Proofs of Storage 85
6 uMine: A Mining Algorithm based on Human Work 113
7 The Economical Impact on Routing in Payment Channel Networks 133
8 Conclusion & Outlook 149
Author’s Publications 157
Bibliography 159
## Contents

1 Introduction  
   1.1 Research Questions and Contribution ........................................ 2  
   1.2 Outline ......................................................................................... 6  
   1.3 Notation ......................................................................................... 8  

2 Background  
   2.1 Privacy ....................................................................................... 9  
      2.1.1 Privacy by Design ................................................................. 9  
      2.1.2 Privacy Patterns ................................................................... 11  
      2.1.3 Privacy Design Strategies ...................................................... 11  
      2.1.4 Privacy Enhancing Technologies (PETs) ................................ 13  
   2.2 Blockchain Technology ................................................................. 13  
      2.2.1 Bitcoin and the Blockchain .................................................... 13  
      2.2.2 Beyond Bitcoin ................................................................. 19  
   2.3 Proofs of Storage ......................................................................... 28  
   2.4 Related Work .............................................................................. 31  
      2.4.1 Permacoin and Retricoin ....................................................... 32  
      2.4.2 Filecoin ................................................................................. 32  
      2.4.3 Anonymous Storage Systems .............................................. 34  
      2.4.4 Summary ............................................................................. 35  

3 Basic Design  
   3.1 Our Scheme ............................................................................... 38  
      3.1.1 Overview ................................................................................ 39  
      3.1.2 The Blockchain and Mining Process ...................................... 41  
      3.1.3 The Store Transaction ........................................................... 44  
      3.1.4 Fetching Files ....................................................................... 45  
   3.2 Security and Privacy Analysis ..................................................... 48  
      3.2.1 Security ............................................................................... 48  
      3.2.2 Privacy ............................................................................... 51  
   3.3 Conclusion ................................................................................... 58  

4 Privacy Enhanced Design ............................................................... 59  
   4.1 Building Blocks .......................................................................... 61
CONTENTS

4.1.1 Linkable Ring Signatures 61
4.1.2 One-time Payment Addresses 66
4.2 Roles 68
4.3 Design of our System 68
4.3.1 Mining 69
4.3.2 Anonymous Payments 70
4.3.3 File Handling 72
4.4 Security and Privacy Analysis 76
4.4.1 Privacy Properties 76
4.4.2 Security Analysis 79
4.5 Additional Considerations 81
4.5.1 Files 81
4.5.2 Blockchain Transmissions 81
4.5.3 File Transmissions 82
4.5.4 Summary 82
4.6 Conclusion 82

5 A Comparison of Publicly Verifiable Proofs of Storage 85
5.1 Preliminaries 88
5.1.1 Identification Protocol 88
5.1.2 Proof of Storage 90
5.2 Proofs of Storage 94
5.2.1 Proof of Storage from Guillou-Quisquater 94
5.2.2 Shoup Proof of Storage 98
5.2.3 Shacham-Waters Proof of Storage 99
5.3 Evaluation 103
5.3.1 Method 103
5.3.2 Results and Discussion 104
5.3.3 Verification Preprocessing 105
5.4 Modifications for Use in our Storage System 107
5.5 Future Work 109
5.5.1 Anonymous Proofs of Storage 110
5.5.2 Post-quantum Security 110
5.6 Conclusion 112

6 uMine: A Mining Algorithm based on Human Work 113
6.1 Building Blocks 114
6.1.1 Slow Hash Functions 114
6.1.2 Captchas 115
6.1.3 Proof of Human-work Puzzles 118
6.1.4 Multiparty Computation Protocol 119
6.2 Our Construction 120
6.2.1 Overview 120
6.2.2 Our Proof of Human-Work 120
6.2.3 Block Generation 124
6.2.4 Choosing the Captcha Generators 126
6.3 Security 128
6.4 Notes on Implementing our Scheme 130
6.5 Related Work 131
6.6 Conclusion 132
# 7 The Economical Impact on Routing in Payment Channel Networks

7.1 Background .......................................................... 135  
7.2 Challenges ............................................................ 137  
7.3 Our Approach .......................................................... 138  
  7.3.1 Notation ............................................................ 138  
  7.3.2 Cheapest Path as a Linear Program ................................ 139  
  7.3.3 Practical Computation of the Cheapest Path .................. 140  
7.4 Evaluation .............................................................. 141  
  7.4.1 Method ............................................................ 141  
  7.4.2 Results and Discussion ........................................ 143  
7.5 Conclusion & Future Work ........................................... 144  

# 8 Conclusion & Outlook

8.1 Summary of Contributions ........................................... 149  
8.2 Outlook ................................................................. 150  

**Author’s Publications** ................................................. 157  

**Bibliography** ............................................................. 159
Introduction

A persistent trend in computing is the outsourcing of storage to external storage providers. These cloud providers, like Dropbox and Google Drive, satisfy the demand for cheap and flexible storage capacity, since the storage resources can be dynamically allocated to users. Thus, external storage providers can offer their resources cheaper than local storage, since users only pay by demand and do not need to host their own infrastructure.

However, a big problem among cloud service providers is that privacy impacts have not been taken into consideration at the design phase and so cloud providers suffer from a lack of privacy. In data sharing systems like Dropbox, the storage provider has access to the plain data and discourages encryption, since that impedes deduplication and thus reduces profit. Further, the Snowden revelations showed that the current system designs are the target of state-sponsored surveillance, with the goal of industrial espionage. Even when encryption is allowed, the access patterns can leak sensitive information such as search patterns. Additionally, since the user is known to the service provider it is easy for the storage provider to link the files of the user to its real-world identity.

Consequently, users are often afraid to entrust their private data to cloud storage providers.

An alternative to these centralized external storage systems are peer-to-peer–based privacy-preserving storage systems.

In these systems like Freenet, GNUnet or the Osiris serverless portal system, anyone can join the network and donate storage. Other participants can use this storage for storing their files in a distributed fashion. Most of these decentralized systems focus on privacy. Storage providers cannot recover the data of other participants if they are not allowed to do so, even if the data resides on their own disk. Since these peer-to-peer systems do not use any centralized or other trusted services, they are very robust and resilient against censorship.

One of the difficulties in these systems is the lack of contribution, since there are no incentives other than the hope that others donate storage capacity as well. In the peer-to-peer file sharing system Gnutella almost 70% of users...
do not contribute to the network, and nearly 50% of all responses are returned by the top 1% of sharing hosts \[1\]. While centralized storage providers can charge money for usage of their storage, there are no similar business models for decentralized storage solutions.

Consequently, there is a conflict between privacy-preserving solutions with few contributors on the one side and centralized storage providers with business models on the other hand.

Current academic research fails to resolve this conflict and rather focuses on isolated solutions. Among them are special kinds of encryption to facilitate searching in encrypted data \[41, 214\], techniques to hide data access patterns \[59, 96, 99, 145, 180, 186, 215, 216\], or encryption techniques which allow operations on encrypted data \[92, 197\]. However, current academic research does not offer holistic solutions.

We set out to combine these two contrasting poles of anonymous decentralized storage systems, and centralized storage systems with business models. The goal of our system is to provide a distributed storage system with financial rewards for its participants and strong privacy properties. In particular, we assume that there is no trusted central authority in our system.

Decentralized systems offer a good starting point, since often privacy has been integrated directly into their design and is not realized as an optional add-on. But these systems need to be enhanced by additional payment mechanisms, since storage providers are unlikely to give storage for free without remuneration. Naive approaches infringe privacy, since payments create a link between a user and its storage provider. Thus, storage providers are able to access names of the users, and additionally banks can infer the size of storage the users bought. Consequently, there needs to be a form of anonymous payment in our envisioned system.

Our main research question is thus:

How can we design a decentralized privacy-preserving storage system with integrated anonymous payment?

This question lead to the creation of the PriCloud project supported by the Baden-Württemberg Stiftung which funded main parts of our research. The PriCloud project consists of two parts. The first part is discussed in this thesis and concerns the interaction of the anonymous payment with the storage. The second part focusses on a privacy-preserving network layer for the decentralized storage system. The network part is envisioned to be covered in the dissertation of my colleague David Mödinger.

1.1 Research Questions and Contribution

In order to answer our main research question from the previous paragraph, we break it apart into more detailed questions. One part clearly concerns the introduction of anonymous payments. Regarding digital payment schemes, there are basically two flavors: approaches building on Chaumian E-Cash \[52\] and blockchain based approaches \[167\]. Chaumian E-Cash assumes a semi-honest centralized bank and thus is not suitable for our scenario, since we assume a peer-to-peer network and try to avoid any forms of centralization. Blockchain
1.1. RESEARCH QUESTIONS AND CONTRIBUTION

Based currencies like Bitcoin\textsuperscript{[167]} on the other hand are decentralized in the sense that their system architecture does not require any centralized special participants. Thus, blockchain based currencies seem like a suitable fit for solving our problem.

Additionally to the anonymous payment, our system requires the management of storage. Since a blockchain inherently provides incentives for participation\textsuperscript{[140]}, it may be possible to integrate the storage system directly into the reward architecture of the blockchain layer, instead of a separate abstraction layer. A natural research question thus is the following.

1. Is it possible to integrate a distributed storage system into a blockchain based digital currency to incentivize participation?

In Chapter\textsuperscript{[5]} we provide a system design corresponding to that question. There, the wasteful proof of work process which is needed to maintain the transaction ledger is substituted by proofs of storage, i.e., proofs that some file of a participant is stored. This leads to a blockchain where the miners are rewarded by storing files instead of finding partial collisions in a hash function.

Additionally to supporting file storage, there needs to be a mechanism that allows users to retrieve their data again. However, this poses an inherent problem. If the user pays first, storage providers have no incentive any more to transmit the file, since they already received their payment. If the storage provider first transmits the data, the user loses its incentive to pay. While cheating storage providers could be punished by an appropriate reputation mechanism, this is hard to do in an anonymous system, since reputation systems link a reputation score to the identity of participants.

A mechanism using a trusted third party providing escrow is trivial. Both, storage provider and user give the file and the payment to the custodian, respectively, who then proceeds to give the user its data and the storage provider the remuneration. However, in our setting there is no trusted third party, which leads to the following research question.

2. How can we design a mechanism to allow users to retrieve their files from storage providers and pay for them without depending on a trusted third party?

The class of cryptographic protocols which implement this functionality is called secret exchange or simultaneous exchange of secrets\textsuperscript{[10]}. In our application the secrets which should be exchanged are the file and a signature on a transaction.

It is hard to ensure that the exchanged secrets are exactly those the other party expects. In particular, the user should really provide the payment as input into the secret exchange mechanism, and the storage provider should provide the original file and not some arbitrary data.

In our scenario simultaneous exchange of secrets is infeasible, since it is proven that a strongly fair exchange without a trusted third party is impossible in general\textsuperscript{[181]}. In this case “strong” means that at the time of protocol termination either both participants get what they want or neither of them does.

We overcome this theoretical limitation by introducing additional assumptions on the behavior of participants. Since our system deals with remuneration
in order to incentivize a particular behavior—contributing to a storage system—it makes sense to take additional assumptions from a related field of study. In game theory, a field which studies the decision-making processes of individuals in scenarios of cooperation, the assumption of rational agents is fundamental. A rational agent is an individual in a system which seeks to maximize its profit. As this notion of rational agents captures the lack of incentives of traditional anonymous peer-to-peer file storage systems, and what we aim to improve, we decide that this is a fitting, yet weak additional assumption. Thus, we introduce a game-theoretic mechanism for secret exchange without the necessity of a trusted third party, as explained in Section 3.1.4.

Our design so far consists of a distributed storage with financial incentives for its participants. However, we need to take privacy into account to allow for anonymous file storage and anonymous payments. Thus, we conduct a privacy impact analysis for our first design which reveals issues regarding linkability and identifiability of senders and recipients of transactions. Additionally, as our first design is vulnerable to nothing at stake attacks resulting from a resource friendly mining process, we need to reintroduce mining with proofs of work to fix this issue. Thus, we separate the functionality of the blockchain and remuneration of storage providers. Incidentally, this separation of the remuneration for mining and for storing files allows for an easier integration of privacy in our system, due to a looser coupling of the components. Since payment transactions, as well as storage contracts can be seen as different variations of a smart contract, we can achieve privacy for both variations simultaneously by creating a mechanism for private smart contracts. This leads to the following research question.

3. How can smart contracts on a blockchain be handled in a privacy preserving way?

Privacy-preserving in this case means that we want to hide the senders and receivers of money transfers. We answer this research question by enhancing storage smart contracts with linkable ring signatures to hide the senders and one-time payment addresses to hide the receivers in Chapter 4. This results in an improved privacy preserving system for decentralized file storage with payments.

However, by separating the abstraction layer of the blockchain and the payments, we cannot remunerate the storage providers using the mining process of the blockchain anymore. Thus, we need to find a new way to make sure that potentially selfish storage providers are remunerated if and only if they store data of some participants in the network. Consequently, we ask the following.

4. How to make sure that the storage provider is remunerated only if it actually stores the data?

To tackle this problem we make again use of proofs of storage as a building block in storage smart contracts. Proofs of Storage enable a storage provider to cryptographically convince a verifier that it stores a file, and thus that the storage provider is eligible for payment.

Current research focusses mainly on privately verifiable proofs of storage. However, in our system we require a publicly verifiable proof of storage in order to allow public verification for its use in a blockchain. Moreover, current
research provides mostly asymptotic performance measurements of publicly verifiable proofs of storage and does not measure real world performance.

Consequently, in Chapter 5 we design a novel publicly verifiable proof of storage and compare its performance to other currently known publicly verifiable proofs of storage. Thereby we answer the following research question.

5. What is the practical performance of publicly verifiable proofs of storage?

A main drawback of blockchain based systems is their low scalability in terms of energy consumption. This issue affects our second system where we reintroduced proofs of work in order to mitigate nothing at stake attacks. Bitcoin’s power consumption is estimated to be 56.89 TWh which corresponds to 0.25% of the world’s electricity consumption. Other sources estimate Bitcoin’s power consumption to be comparable to the electricity consumption of Ireland. This is clearly a disadvantage to its widespread adoption.

Bitcoin’s huge energy consumption is a result of its mining process. Since blockchains allow for dynamic participants they need to resist Sybil attacks. A solution to this problem is to bind digital identities to a scarce physical resource—processing power in the case of Bitcoin. While our initial design in Chapter 3 makes use of proofs of storage in the mining process, our privacy enhanced design uses proofs of work and thus also suffers from this limited scalability. Naturally, this leads to the following research question.

6. How to design an alternative mining mechanism which uses less power than proof of work?

One possibility of a mining process which does not require as much power as Bitcoin is a proof of human work. The main idea of a proof of human work is to provide a solution to a problem which can be efficiently solved by a human only. In the mining process this implies having what distinguishes humans from machines as the scarce resource in the system to which the digital identities are bound.

Currently, there is only one construction of a proof of human work known. However this design relies on indistinguishability obfuscation, which is a cryptographic mechanism where no instantiation is currently known. This leads to the following research questions.

7. How can we design a realizable construction for a proof of human-work?

We answer these questions in Chapter 6, where we propose a novel instantiation of a proof of human work which relies on multiparty computation. In contrast to the construction of a proof of human work by Blocki and Zhou, our scheme requires that the participants are online. However, this is no limitation when used as a mining algorithm, since the miners need to be online anyway in order to validate transactions and blocks.

Additionally to the inefficiency of Bitcoin’s original mining algorithm, a further scalability problem of blockchains is their low transaction throughput. Bitcoin achieves a maximum throughput of approximately 7 transactions per second globally. This is certainly not enough to power a global currency system and fails to meet the demands of our storage network.
CHAPTER 1. INTRODUCTION

One envisioned solution to the problem of low transaction throughput are payment channels [156]. These are bilateral channels between participants over which incremental payments can be sent without interaction with the blockchain. Since these channels in theory allow arbitrary computable smart contrast, they can be applied to our storage smart contracts as well. Payment channels are designed in such a way that later transactions have priority over previous transactions and thus provide cheating resistance. Only the opening of the channel and its closing with the final balances are recorded on the blockchain.

Since these channels are only between two parties each, a routing mechanism is needed to provide true scalability benefits. Thus it becomes possible to forward payments via multiple payment channels and thus a payment network emerges. While there are many proposals of payment channels that support routing [64, 161, 187, 222] none of these take economic implications into account. Namely, routing hubs may demand remuneration in form of routing fees, and thus, routing should not be done via a shortest route but rather via a cheapest route. Further, depending on the volume of a transaction, a given channel may not have sufficient capacity to support that transaction.

Apart from technical privacy considerations on the level of payment channels, it is to be investigated if payment channels are desired at all, since they may provide high centralization pressure. In the worst case, economic incentives alone could lead to centralized routing hubs over which the majority of payments are routed, independently of any further technologies to enhance privacy. This provides an imbalance of power and is contrary to the idea of a decentralized peer-to-peer network. Consequently, the following research question emerges.

8. What are the economic implications of routing in payment channels?

We treat this question in Chapter 7 where we provide measurements regarding the economic impact on transaction routing in a custom simulator we have implemented. Our result is that the economic perspective should not be neglected when designing routing protocols for payment channels.

1.2 Outline

An overview of the structure of this thesis, together with the relevant publications is given in Figure 1.1. Chapter 2 gives the necessary background about privacy, blockchains, and proofs of storage which is needed to understand this thesis. Our first design of a distributed storage without incentives is given in Chapter 3. We enhance this design to provide privacy for its participants in Chapter 4. Then we go on to discuss selected building blocks of our design. These chapters can be read in any order. Chapter 5 introduces a novel proof of storage and provides performance comparisons. Chapter 6 proposes a novel approach to limit the huge energy consumption of blockchains by introducing an alternative mining process. Another scalability issue, namely, the low transaction throughput of blockchains is tackled in Chapter 7. In particular we take a look at payment channel networks, a popular proposal to increase transaction throughput, and highlight the economic impact on routing in these overlay networks. Finally, Chapter 8 concludes our work and provides directions for future work.

The basis of this thesis is formed by the following publications.
1.2. OUTLINE

- Chapter 2: Background
  - RQ 1, RQ 2
- Chapter 3: Basic Design
- Chapter 4: Privacy-Preserving Design
  - RQ 3, RQ 4
- Chapter 5: Proofs of Storage
  - RQ 5
- Chapter 6: Proofs of Human Work
  - RQ 6, RQ 7
- Chapter 7: Routing in Payment Channels
  - RQ 8
- Chapter 8: Conclusion

Figure 1.1: Roadmap of this thesis, together with the relevant publications and research questions


1.3 Notation

We write $a \leftarrow \mathcal{A}(x)$ to assign to $a$ the output of running the randomized algorithm $\mathcal{A}$ on the input $x$. We denote with $a \leftarrow \mathcal{A}(x; r)$ the deterministic result of running $\mathcal{A}$ on input $x$ with the fixed randomness $r$. We say that an algorithm $\mathcal{A}$ is ppt if it runs in probabilistic polynomial time. Although every deterministic algorithm can be seen as a non-deterministic algorithm which does not use its source of randomness, we write $a := \mathcal{A}(x)$ if we want to stress that $\mathcal{A}$ is deterministic. With $\mathbb{Z}/p\mathbb{Z}$ we denote the residue classes of the integers $\mathbb{Z}$ modulo $p \in \mathbb{N}$. We write $x \in_r S$ if we choose an element $x$ from a finite set $S$ uniformly at random. Vectors are written in bold typeface. We call a function $f$ negligible in $n$ if for all positive polynomials $p$ there is a natural number $N \in \mathbb{N}$ such that for all $n > N$ it holds that $|f(n)| < 1/p(n)$. We sometimes write $\text{negl}(n)$ to denote a negligible function. Hash functions are denoted by the symbol $\mathcal{H}$. For a function $f$ we use $\text{argmin} f$ to denote the point of the domain of $f$ where the function $f$ has its minimum if it exists, i.e., the point $x$ such that $f(x) \leq f(x')$ for all $x' \neq x$. Similarly we define $\text{argmin}\{x_1, \ldots, x_n\}$ as the index $k$ such that $x_k \leq x_\ell$ for all $\ell \in \{1, \ldots, n\}$ with $\ell \neq k$. 
In this chapter, we will explain some background information which we believe is necessary to understand later parts of this thesis. Since our work deals with the design of a privacy friendly system, in the next section we will explain the role privacy plays in the engineering process of such IT systems. We will explain Privacy by Design, and the more current approach using privacy patterns and privacy design strategies. Section 2.2 gives the necessary background for understanding Bitcoin and its blockchain. We will go on to detail selected issues and improvements of blockchain architectures. Section 2.3 gives an introduction to proofs of storage and related mechanisms which constitute relevant building blocks for our constructions. Finally, Section 2.4 gives an overview over related projects.

2.1 Privacy

In this section, we give an overview of the building blocks used to design privacy-aware systems. In particular Privacy by Design, as well as the more current pattern-based approach to privacy engineering is detailed. Parts of this section have been taken from [40].

2.1.1 Privacy by Design

Until the mid-1990s, privacy was rarely considered a relevant feature of IT systems. Even if it was, the integration of privacy-preserving mechanisms was often conducted only as an additional requirement added later to the system. The widespread notion of “privacy as an afterthought” contradicted the properties of privacy as an integral part of an IT system and often yielded extensive or insufficient changes to the system.
To overcome these deficits, a joint team of the Information and Privacy Commissioner of Ontario, Canada; the Dutch Data Protection Authority; and the Netherlands Organisation for Applied Scientific Research advocated a more integral approach that included privacy considerations into the whole development cycle \[110\]. In 1995, they introduced the so called Privacy by Design approach, which is summarized by the following seven foundational principles:

1. Proactive not reactive; Preventative not remedial: Privacy invading events should be prevented before they happen.

2. Privacy as the default setting: Privacy should be built into the system by default. There should be no action necessary on the user-side to ensure privacy.

3. Privacy embedded into design: Privacy should be a part of the core functionality of the system and not implemented as an add-on.

4. Full functionality – positive-sum, not zero-sum: Privacy by Design should prevent false dichotomies, like privacy vs. security and should incorporate both.

5. End-to-end security – full life-cycle protection: Personal data should be protected throughout its whole life cycle, beginning with the secure collection and ending with the secure deletion of the data involved.

6. Visibility and transparency – keep it open: Stakeholders should be assured that the system works according to the stated promises. This should be independently verifiable.

7. Respect for user privacy – keep it user-centric: The privacy of the user should be held uppermost by integrating strong privacy defaults and appropriate user notifications.

These principles have been a major milestone in the design of privacy-preserving systems as they provide general guidance. For that reason, these concepts have been included in several privacy legislations today, like the European General Data Protection Regulation \[194\].

A frequent criticism regarding Privacy by Design and its seven foundational principles is that they are too abstract to be applied directly in a development process. The principles neither provide concrete advice, nor do they address the varying needs of specific domains, such as the Internet of Things, Healthcare systems, or Car-2-Car communication. A system designer is still required to have a thorough understanding of privacy and the forces involved, to design a privacy friendly system. Clearly, more guidance and a more methodological approach is required to establish a privacy engineering process as, for example, worked on by the PRIPARE project \[175\] in which we participated.
2.1.2 Privacy Patterns

One major element of modern privacy engineering approaches are so called privacy patterns. The main idea of privacy patterns is to improve the drawbacks of the Privacy by Design principles, i.e., that they are not actionable \[72, 104, 196\]. Privacy patterns are reusable solutions for commonly occurring problems in the realm of privacy. Essentially, they are design patterns, as introduced by the Gang of Four \[89\], for achieving or improving privacy. Privacy patterns provide guidance for engineering and development, and target the needs of specific domains, such as back end implementations or user interface design. By providing a well-structured description of a problem and its solution using a standardized template, patterns can, in theory, easily be looked up and applied by system engineers without a deep knowledge of privacy. Since these patterns include references to specific use-cases and possibly implementations, engineers can directly find the resources needed to implement them in their own context.

One well-known example of a privacy pattern that can be implemented in multiple domains is the “stripping of metadata” that is not necessary for the functionality of the service. This procedure enhances privacy, since metadata often includes personally identifiable information. Further, this solution is reusable, since it is not bound to a specific instance of a problem. Thus, stripping of metadata constitutes a privacy pattern that can be applied, e.g., to a website managing digital photographs.

Another example of a pattern which can be implemented in multiple domains is the usage of pseudonyms instead of identities. This provides a solution to a commonly occurring problem, namely the management of linkable identities in a system. These identities are able to establish a reputation. In such a system the use of pseudonyms enhances privacy, since the connection between the identities of the participants in the system and their real-world identities are not immediately obvious. The usage of pseudonyms is a reusable pattern since it is not bound to a specific instance of a problem. Consequently, usage of pseudonyms constitutes a privacy pattern.

Similar to the well-known design pattern catalogs from software engineering, privacy patterns can be collected into a catalog describing a number of relevant problems and suitable patterns. These catalog can be applied during the design phase of privacy friendly systems to aid in creation of the system architecture. There are multiple collections of privacy patterns from academic research \[72, 105, 199, 205\] as well as online repositories \[https://privacypatterns.eu, http://privacypatterns.org/\] that are more accessible for practitioners. One of these online repositories, privacypatterns.eu, was set up by us as part of the PRIPARE project \[http://pripareproject.eu/\] which has received funding from the European Union’s Seventh Framework Programme.

2.1.3 Privacy Design Strategies

In a typical system development process, privacy patterns are applied during the design stage. However, there are more general architectural building blocks improving privacy, that can be applied at an even earlier stage, i.e., during requirement analysis and architectural design. This basically continues the
Privacy by Design philosophy—to include privacy considerations into the entire development process.

These general architectural building blocks are called *Privacy Design Strategies*. Whereas a privacy principle states the goals of a system, a privacy design strategy in contrast states how to achieve these goals. According to Hoepman [109], a privacy design strategy is on a more general level than a privacy pattern and “describes a fundamental approach to achieve a certain design goal. It has certain properties that allow it to be distinguished from other (fundamental) approaches that achieve the same goal.”

Hoepman [109] defines the following eight privacy design strategies.

- **Minimize**: Data minimization is a strategy which insists that the amount of personal information that is processed should be minimal. Data that is not needed for the original purpose should not be collected.

- **Hide**: Hide takes place after data collection. Whereas Minimize forbids the collection of needless information, Hide suggests that any personal data that is processed should be hidden from plain view.

- **Separate**: The approach of the privacy strategy Separate is to process any personal information in a distributed fashion if possible. Thus, interrelationships between personal data vanish in contrast to a centralized processing.

- **Aggregate**: When implementing Aggregate, personal information is processed at a high level of aggregation. This level should only be so high as to remain useful, however. Details that are not needed for the functionality of the service vanish. This process could include statistical aggregation such that the individual details of identities are blurred.

- **Inform**: The privacy strategy Inform states that data subjects should be adequately informed whenever their personal information is processed.

- **Control**: A common requirement of software systems is that data subjects should be in control of the processing of their personal information. Whenever this is ensured, we are dealing with the privacy strategy Control. However, Hoepman states that he is not aware of any patterns implementing this strategy.

- **Enforce**: Enforce states that a privacy policy that is compatible with legal requirements should be in place and should be enforced.

- **Demonstrate**: The privacy strategy Demonstrate demands that data controllers are able to demonstrate compliance with their privacy policy and any applicable legal requirements. A good example of a pattern implementing this strategy is the use of audits.

Later in the development process, a privacy design strategy can be refined with privacy patterns implementing one or more privacy design strategies. Thus, privacy design strategies can be used to classify privacy patterns. This enables system designers to find the necessary privacy pattern more efficiently in a collection, since they are only interested in the ones implementing their chosen privacy strategy.
2.1.4 Privacy Enhancing Technologies (PETs)

If privacy design strategies are the most abstract building blocks of a system that protects privacy, privacy enhancing technologies are the most concrete ones. Privacy enhancing technologies can be seen as implementations of a privacy pattern and describe the concrete technology used.

Assume that during collection of the system architecture requirements the developers decide to make use of the privacy strategy \texttt{Hide} to conceal the data of the users from untrusted parties. During the architecture design phase they may agree to use the privacy pattern “symmetric encryption” to encrypt data and hide their contents from all parties who are not in possession of the keys. When implementing, the developers have to choose a PET which implements this pattern, in this case, a concrete symmetric encryption scheme like AES or Blowfish.

Thus, privacy strategies, patterns, and PETs can be used together to address privacy requirements throughout the whole development cycle.

2.2 Blockchain Technology

In this section we give the necessary background for understanding Bitcoin and its ecosystem. We continue by highlighting some of the improvements and enhancements in this field.

For a more detailed description of Bitcoin we refer the interested reader to the original white paper \cite{nakamoto2008}. Additionally, there are several good surveys on the topic \cite{bhattacharya2016,bhattacharya2017,fan2017} as well as a book by Antonopoulos \cite{antonopoulos2018}.

2.2.1 Bitcoin and the Blockchain

In 1990, Chaum \cite{chaum1990} proposed a purely digital electronic cash scheme. One major benefit over previous schemes like credit cards or paper banknotes, where one can in theory trace the serial numbers, was the untraceability of Chaumian cash. One of the main problems of digital cash revolves around the problem of \textit{double spending} \cite{chaum1983}, i.e., making digital copies of the cash and spending them on two different transactions. As an example consider that Alice has a coin C. She makes a copy of C, which is easy, since C is just digital data. Next, Alice can try to spend the coin C and its copy with different receivers. One of the major innovations of Chaum’s scheme is the ability to detect this type of fraud. A bank needs to provide blind signatures \cite{chaum1982} on the transactions, i.e., it needs to approve that coins have not been spent before. Other than that, the bank learns nothing about the coin. Despite this achievement, the bank needs to be available and semi-honest and thus provides a single point of failure.

In 2009 Satoshi Nakamoto\footnote{It is now widely assumed that the name Nakamoto is a pseudonym. It is even speculated that there is a group behind the pseudonym Satoshi Nakamoto.} presented Bitcoin \cite{nakamoto2008}, the first truly decentralized cryptocurrency. Bitcoin tackles the problem of double spending by storing all valid transactions, i.e., changes of possession of bitcoins instead of valid coins, in a publicly verifiable ledger. All valid transactions are stored in a global public sequence of blocks in the peer-to-peer network called the \textit{blockchain} which is stored by each miner. This sequence of blocks goes back to
Transactions

If user Bob wants to send bitcoins to Carol he creates a new transaction. A transaction consists of one or more inputs and one or more outputs. In the input of this transaction Bob references a transaction output of a previous transaction which includes his public key, e.g., a transaction from Alice where she has send money to Bob. In the output of Bob’s newly created transaction he includes the public key of the recipient Carol. To prove that Bob is authorized to spend the funds of the previous transaction output he signs his transaction. The simplified process is given in Figure 2.1. Since Bob is the only one possessing the private key corresponding to the public key in the referenced transaction output this proves his intention and authorization of sending his funds to Carol. Finally Bob broadcasts his transaction into the network, where it will be included in a block.

If Bob does not want to give the whole amount denoted in the referenced transaction output to Carol, he can include an additional output in his current transaction which serves as change. This allows Bob to divide money into multiple smaller amounts. On the other hand, it is possible to merge different amounts by having multiple transaction inputs which each reference one previous transaction output and then providing signatures with the keys corresponding to the outputs in each input respectively.

More exactly, transaction outputs in Bitcoin are realized by a domain specific language which specifies the terms under which the transaction output can be spent. The transaction inputs then need to provide the data such that the conditions defined in the transaction outputs are fulfilled. This allows more complex terms of the transactions as the standard transactions described above. E.g., custodians can be set up by a transaction which can be spent only under the condition that two signatures are provided. Threshold schemes can be implemented by requiring $t$ out of $n$ signatures to spend a transaction output.
Blocks and Mining

In an abstract way, the mining process is a distributed consensus protocol where the identities of the participants are not known in advance.

The miners receive transactions which are then bundled into blocks. These blocks are challenges for proofs of work and thus the miners vote with their computational power on the validity and order of transactions. As a result the miners agree on the global state of the accounts in the system and prevent double spending. In contrast to classical consensus algorithms where the participants are known, in an open system like Bitcoin there is the additional possibility of Sybil attacks, i.e., fake identities which manipulate the voting process. However, in Bitcoin this is solved by the proof of work mining mechanism. This mechanism binds digital identities to scarce computational resources and thus restricts the creation of identities [11, 73, 74].

In order to mine a block the miner first aggregates the transactions which are not yet persisted in the blockchain into a Merkle tree [158, 159] as can be seen in Figure 2.2. The parent of each node in the tree contains the hash of their siblings. The root $H_0$ of the Merkle tree, which is comparable to a fingerprint of all the contained transactions, is included in the block headers which are necessary for verification of the blockchain. This is an important implementation detail which provides scalability. It allows the removal of transactions from the storage of the miner if they have been spent, since removing transactions does not change the root hash of the Merkle tree. Since the root hash is preserved, removing spent transactions does not change the validity of the blockchain. This process of removing spent transactions is called pruning.

A whole block consists of the block header and all the transactions which are aggregated in it. Each block header contains the following:

- The Merkle root of the transactions aggregated in the block.
- A reference to the previous block header realized as a hash value.
- Other fields like a time stamp and the version number.
- Data relevant to the consensus: a nonce and a difficulty parameter.

A simplified mining process works as follows: Let $B_1, \ldots, B_n$ denote the block headers in the blockchain. Let $H$ denote a cryptographically secure hash function and $tx_i$ are transactions.
function. First the miner includes the Merkle root of the valid transactions he received previously in his new candidate block header. To generate a new block with header $B_{n+1}$, participants try to find a nonce for $B_{n+1}$, such that $\mathcal{H}(B_{n+1})$ is below the threshold difficulty. This process is depicted in Algorithm 1. It is restarted if new transactions are received or if a new valid block is received or found by the miner itself.

Algorithm 1 Mining new Blocks

Input: newest block header $B_n$, merkle_root of unconfirmed transactions, difficulty.
Output: next block $B_{n+1}$

1: $B_{n+1} \leftarrow \text{Block(merkle_root, } \mathcal{H}(B_n) \text{)}$ $\triangleright$ Create new block
2: Choose $B_{n+1}.\text{nonce}$ uniformly at random.
3: while $\mathcal{H}(B_{n+1}) < \text{difficulty}$ do
4: $B_{n+1}.\text{nonce} \leftarrow \text{nonce} + 1$
5: end while
6: return $B_{n+1}$

The new block, i.e., the block header $B_{n+1}$ together with its transactions, is broadcast to the other miners. On receiving a block a miner checks the validity of the new block, i.e., whether

$$\mathcal{H}(B_{n+1}) < \text{difficulty}$$

and if the included transactions are correctly signed and valid. If the received block is valid the miners append it to their local copy of the blockchain and continue to mine the next block $B_{n+2}$. Otherwise, they reject the block and continue mining on top of the old block $B_n$.

The parameter difficulty is agreed upon dynamically by the miners. It is included in the respective blocks and adjusted every 2016 blocks to account for fluctuations in the overall hash rate of the network. Thereby, the expected block generation rate of the network is one block every ten minutes, assuming no changes in the total hash rate. A stable block mining rate is necessary for the functionality of the system due to the non-zero propagation delay in the network. If the difficulty is not adjusted, a rise in overall computing power could lead to the situation that block generation time is lower than the propagation delay. Then some nodes would not be able to see the current block and thus would have no chance to mine the next one.

To incentivize users to participate in the system miners are financially rewarded in form of a special transaction, called coinbase which they are allowed to include in their blocks. The coinbase transaction has no input but an output with a fixed amount of bitcoins which can be spent by the miner of that particular block. In the genesis block this amount was 50 bitcoins. Every 210,000 blocks the reward halves by consensus rules, thus resulting in an exponentially decreasing block reward. This also leads to a total of 21 million bitcoins which can be mined. Additionally to providing incentives for participation in the system by compensating for the computational effort spent, the coinbase transaction serves as an initial wealth distribution mechanism. A second type of reward for miners is in the form of transaction fees. If the sum of the outputs of a transaction is less than the sum of the inputs, then the
2.2. BLOCKCHAIN TECHNOLOGY

difference can be spent by the miners which includes that transaction into a block. This type of remuneration prevents that rational miners save the cost for validating transactions and mine blocks without including any transactions.

When multiple miners find a new block simultaneously both of them broadcast it. The other miners then have two possible valid blocks on which they can continue to mine. This situation is called a fork. The miners mine on one of the blocks until eventually one of the chains is longer. An example of a blockchain with some forks is shown in Figure 2.3. Each arrow denotes a reference to the previous block, and consequently the blockchain grows to the right.

Bitcoin assumes that the honest miners control the majority of the computational resources of the network and thus try to extend the longest chain. With this assumption the network converges to one blockchain and therefore one global state of the accounts.

If the assumption of an honest majority is not given, double spending becomes possible. An adversary with more than 50% of the computational resources of the network can buy goods with a transaction on the main chain, fork the chain at some point in the past, and extend the length of his fork beyond the main chain. When the chain not containing his transaction is the longest one he effectively reverses his transaction, since the honest miners switch to the longer chain. From their perspective it seems that the network had split and they had been in the part with a minority of computational resources. An attacker with less computational resources cannot execute this specific attack, since the main chain will in expectation grow faster than his chain.

However, it has been shown that by selectively withholding blocks only 25% of the mining power is needed to gain an unfair advantage in mining [79]. The main idea of this attack is to waste the mining power of the public network which still mines on the old block, after the attacker has found a new block. Another work assumes that the attacker can exploit the peer-to-peer network to separate miners and control their view of the network [107]. This allows an attacker to double spend. In order to successfully mount this attack, no threshold in mining power is necessary, but the attacker needs to control enough addresses in different IP ranges. Nayak et al. [169] combine these two approaches and expand the strategy space for the optimal strategy for mining and withholding blocks. Nevertheless, the misleading term 50% attacker continues to be used in literature, even though considering network effects less than 50% of the mining power are needed to gain an unfair advantage in Bitcoin.

**Mining Pools**

While the original idea of Bitcoin was that one cpu should equal one vote in the consensus protocol, this is no longer the case. Currently, Bitcoin mining is done on specialized hardware, namely application-specific integrated circuits (ASICs),
and is controlled by only few large corporations. For individuals, mining Bitcoin is not feasible anymore when done on a regular cpu, since current cpus achieve only between 2 and 60 MH/s. Instead, participants need to rent or buy ASICs which are then put into a computing center. For comparison, the current AntMiner S9 ASIC is advertised to have 14,000,000 MH/s\(^4\).

Since the verification of transactions can be done separately from the proof of work, so called mining pools emerged. In a mining pool there is a pool coordinator and a number of workers. The pool coordinator validates and assembles the transactions and prepares the block header without the nonce. The result is sent to the workers who try to find the nonce by guessing and checking that the hash of the nonce and the block header is smaller than the difficulty. Since there is no complex validation logic involved, the workers can be realized as hardware chips. If a correct nonce is found, it is sent to the pool coordinator who completes the block and broadcasts it.

Additionally to correct nonces, the workers submit “partial” nonces, which are nonces, where the hash is not smaller than the difficulty parameter required by the blockchain but instead smaller than a custom difficulty parameter set by the pool coordinator. This allows the pool coordinator to reward workers based on their contribution of mining power, even if they are unlucky and do not find a block.

Connecting to a mining pool lowers the variance of the rewards of the participants and thus is done nearly universally, in line with basic portfolio optimization theory. However, mining pools provide two main disadvantages. First, transaction validation is only done by the pool coordinator. The workers never have access to the transactions which they approve of. Thus, malicious pool coordinators can, e.g., mine another currency with the same proof of work algorithm without the workers’ knowledge \(^5\). Secondly, due to the aggregation of mining power 50% attacks become more feasible.

Mining pools provide a strong centralization pressure to Bitcoin and their current role and the emerging attacks are still subject to ongoing research \(^7\) \(^8\) \(^11\) \(^3\) \(^4\) \(^5\) \(^6\) \(^7\) \(^8\) \(^11\) \(^3\) \(^4\) \(^5\) \(^6\).

Privacy in Bitcoin
Privacy in Bitcoin is very limited, since the (hashes of the) public keys of the receivers are included in the transaction outputs and thus publicly visible. This allows anyone to track the current balance of each user by processing the transaction data stored in the blockchain.

One naively implemented countermeasure is to use multiple public keys for the same user. This does not help much, since if a transaction has multiple inputs it is very probable that these belong to the same entity. Thus, multiple addresses can be clustered as belonging to the same person. This heuristic is very well known \(^4\) \(^5\) \(^6\). However, there are additional privacy issues in Bitcoin.

Sometimes it is possible to distinguish between the real recipient and the address used for change in the system \(^4\) \(^5\) \(^6\). Take as an example a single transaction with two inputs, each spending 2 bitcoins, and two outputs, one

\(^4\) A comparison between the hashrates of different cpus and ASICs can be found at \(https://en.bitcoin.it/wiki/Mining_hardware_comparison\) and \(https://en.bitcoin.it/wiki/Non-specialized_hardware_comparison\); accessed 2018-06-18.
with 1 and the other with 3 bitcoins. Now let us assume that the output with 1 bitcoin is the real recipient and the other output with 3 bitcoin is the change address. Then it would have sufficed if there had been only one input with 2 bitcoins. This is a contradiction, since there are two inputs. Thus our assumption was wrong and the output with 1 bitcoin could not have been the real output. Consequently, the output with 3 bitcoins is the real sender and the addresses in the inputs and in the output corresponding to the payment of 1 bitcoin are all controlled by the same entity.

Additionally to the missing privacy on the blockchain layer, the network layer also lacks privacy. It is possible to link public keys of senders to IP addresses in the network since broadcast communication spreads roughly at the same speed in all directions and so its center is easy to locate [29, 133]. Further, using Bitcoin over the anonymity network Tor does not remedy the problem, since an attacker can gain full control over the information flows between the participants [30].

### 2.2.2 Beyond Bitcoin

Soon after the emergence of Bitcoin multiple enhancements or changes to its design have been proposed. These can be categorized into two main types of proposals. The first type of proposals suggests changes to the Bitcoin system by enhancing it with additional features. Often these are designed as applications on top of Bitcoin. The second type of proposals suggest the design of alternative blockchain based systems. The innovations of these alternative currencies called altcoins ranges from simple changes of the hash function used in Bitcoin, like Litecoin or Dogecoin, to deep changes in the way transactions are handled, like Ethereum [229]. These blockchain based transaction systems, together with Bitcoin are often called cryptocurrencies. A good overview of second generation blockchain technology can be found in the survey of Bonneau et al. [38].

In this section we dig deeper into some selected aspects of Bitcoin and suggested or implemented improvements, or novel applications. These have been chosen due to their synergies with the remaining thesis and provide parallel developments to our work, as well as open issues in the state of the art.

#### Multiparty Computation

In multiparty computation, multiple participants collaboratively want to evaluate a function $f(x_1, \ldots, x_n)$ on their inputs $x_1, \ldots, x_n$. Additionally, each participant does not want to reveal his input $x_i$ to the other participants. Since the seminal work of Yao [234] and Goldreich et al. [98] it is possible to achieve this for any computable function, under the assumption of honest participants. However, it is known that the properties of fairness (either all parties get the final output or none) and robustness (delivery of the output is guaranteed), are impossible to achieve unless a majority of the parties is honest.

To get around this cryptographic impossibility result, it is necessary to look at a related area of research which studies the collaborative behavior of participants with different motives: game theory. In game theory, the assumption of an adversary with a polynomial bound on his computational effort is substituted by the assumption of rational actors [9, 70, 125]. These are actors which aim to maximize their own utility. Assuming rational actors taking
part in a multiparty computation protocol, it is possible to punish malicious
participants which do not adhere to the protocol and thus it is possible to
incentivize the participants to comply.

Since Bitcoin offers a way to compute with money and thus to selectively
administer financial punishments to the participants in a more complex protocol
it is well suited for candidate constructions in this area. Andrychowicz et
al. [5] introduce a novel cryptographic primitive based on Bitcoin called a
timed commitment, where the committer has to reveal his secret within a
certain time frame, or to pay a fine. This can be used to punish aborts of the
multiparty computation protocol and thus to obtain fairness. Fairness in this
case is understood in the sense that any party that aborts after learning the
output pays a financial penalty to all parties that did not learn the output. In
particular Andrychowicz et al. construct a protocol for a distributed lottery
without trusted third party.

Subsequent work provides different formalizations, lowers the number of
messages needed [27, 119], or provides protocols with additional properties like
verifiability or restricted leakage [136]. Kumaresan et al. [137] generalize the
model to allow the payoff of the participants to depend on the computation.
This leads to a novel cryptographic primitive called secure cash distribution
with penalties which the authors use to update the well-known mental poker
game of Shamir et al. [209] to include payments. Their mental poker protocol
guarantees fairness in the sense that someone has to pay a penalty to all other
parties if he aborts the protocol.

In summary, Bitcoin allows for a novel way of financially punishing misbehav-
ing participants in a protocol and thus allows for secure multiparty computation
in the face of a malicious majority under the assumption of rational actors.

Improvements to the Mining Algorithm
The mining algorithm of Bitcoin suffers from multiple problems and there have
been many suggested changes to it. We will now highlight some of these.

Energy consumption One of the problems of proof of work is the high
energy consumption needed to power the network. In 2014 it was estimated
that the power consumption of the Bitcoin blockchain alone was comparable to
the electricity consumption of Ireland [176].

As explained above, the proof of work mechanism ties the digital identities to
a physically scarce resource: computing power. In order to substitute the proof
of work by another mining process, an alternative scarce resource is needed.

Proof of stake is an alternative mining process, where the coins in the system
itself serve as a scarce resource to which the digital identities are tied [26, 28, 118,
122]. Participants in the system are elected to mine new blocks with probability
proportional to their account balance. A working proof of stake system has
been a holy grail in the community for a long time. The first proposal of a proof
of stake altcoin was in 2014, when King and Nadal introduced Peercoin [122].
In Peercoin, blocks can be mined by destroying coin age, i.e., currency amount
times holding period. Let us say that Alice grants Bob 20 coins. Bob then holds
the coins for 10 days, resulting in Bob having 200 coin days. If the difficulty
of mining a block is below 200 coin days, Bob can destroy his coin age—the
holding time is set to zero but the currency amount is preserved—and mine a block through this destruction of coin age.

However, Peercoin was vulnerable to so called nothing at stake attacks, where in the case of a fork, miners try to extend both ends of the chain at the same time. Consequently, consensus over the transactions never emerges since the system does not converge to a single longest chain. In the case of proof of work systems, nothing at stake attacks are impossible, since the computing power per fork decreases if is divided between multiple forks. However, in the original proof of stake, there is no drawback for miners which try to extend multiple forks. There have been some proposals of introducing financial punishments if miners trying to extend multiple chains are witnessed. However, to our knowledge none of them have been used widely. Peercoin remedies the problem of nothing at stake attacks by introducing checkpoints in the code for blocks at specific heights, thus leading to more centralization in their system, since developers are able to decide on the correct chain and censor blocks.

The first provably secure proof of stake is a recent invention due to Kiayias et al. [118]. There, the randomness for the election of the leader who needs to mine the next block is generated by a multiparty coin-flipping protocol instead of by the blockchain itself.

Additionally to proof of stake there have been some other proposals for mining algorithms which do not use computational power. Spacecoin [182, 183] for example, uses proofs of space [12, 75], i.e., proofs that a significant amount of disk space has been allocated, instead of proof of work.

Useful Proof of Work Another objection against the proof of work mechanism in Bitcoin is that it is essentially a waste of energy, since it contributes nothing beyond maintaining a distributed ledger. Although economic arguments for the infeasibility of proof of work with additional utility, based on market equilibria between the revenue from mining and its cost can be made, we will not delve into those.

One of the first altcoins with the motivation to have a meaningful proof of work was Primecoin [121]. The computational problem their mining algorithm is based on is finding Cunningham chains—sequences of prime numbers with special properties—which are interesting from a research perspective [235].

Ball et al. [18] give a framework for useful proof of work, where computational tasks which reduce to Orthogonal Vectors, 3SUM or the All-Pairs Shortest Path problem can be delegated as challenges to the miners. Thus, the hardness of the proof of work scheme is immediate, in contrast to the computation of Cunningham chains used in Primecoin, whose complexity is unknown.

Improving Mining Pools

Some suggestions to improve blockchains concern the role of mining pools, as these are considered to be a risk to the blockchain system due to their centralization pressure.

One approach is to exchange the hash function used in the mining process with a nonoutsourceable proof of work [163]. This is a special proof of work mechanism such that coalitions among miners are deterred. Concretely, the workers are able to steal the block reward from the pool coordinator without revealing their identity. Since pooled mining is economically desired by the
Figure 2.4: Example of a Mixing Transaction. \( pk_{Alice} \) and \( pk'_{Alice} \) are different public keys of Alice which cannot be linked by an observer. The same holds for the addresses \( pk_{Bob} \), \( pk'_{Bob} \), and \( pk_{Carol} \), \( pk'_{Carol} \).

participants—it lowers their variance of rewards and thus their risk—it is unclear if a system where pooled mining is impossible will be accepted by its participants.

Another line of work decentralizes the pool operator and thus adds transparency and decreases the power of the operator. P2Pool\(^5\) is a mining pool where the pool operator is distributed in a peer-to-peer fashion. Smartpool\[^147]\ is a mining pool based on a separate blockchain with lower difficulty. The reward distribution is managed on the real blockchain. These proposals provide a feasible alternative to central mining pools.

**Privacy**

Since Bitcoin lacks privacy\[^4, 29, 30, 133, 157, 200]\, there have been multiple proposals to increase the privacy of users in the bitcoin system. These approaches can be divided into measures which work on top of Bitcoin and altcoins which integrate privacy by design.

**Privacy on top of Bitcoin** One of the main ways in which to increase privacy in Bitcoin transactions is so called *coin mixing*\[^39]\ or *coin shuffling*\[^201]\. These protocols allow a set of participants to collaboratively generate a transaction such that each of the participants contributes an input with the same amount. There is the same number of outputs as participants. Each output gives a participant its money back but with a different order than in the inputs and with different public keys as shown in Figure 2.4. This protocol can be compared to a bag where each participant inserts a coin. Afterwards, each participant takes out one coin. Thus the payment flows are not traceable anymore. In particular for each output address the probability of success in linking it to its input address is not better than random guessing.

In Mixcoin\[^39]\ the interaction is managed by a mixer which collects and distributes the money in such a way that he cannot cheat. The mixer receives remuneration in the form of a mixing fee. In Coinshuffle\[^201]\ the mixing protocol is completely decentralized. This saves the fee for the mixer and thus is cheaper.

\[^{5}\]http://p2pool.org/ \(\text{accessed 2018-01-29}\)
Coin mixing and shuffling only hides the senders and receivers of the transactions. To additionally hide the amounts in transactions an enhancement called confidential transactions was proposed. In confidential transactions instead of the amounts, transactions contain additively homomorphic commitments in their amounts.

A commitment scheme allows a participant Alice to commit to a value and then later reveal the commitment to Bob. Commitment schemes are binding, i.e., they do not allow the committing party Alice to change the committed value before the revealing phase. Moreover they are hiding which means that Bob cannot learn the value committed to before it is revealed.

In detail this works as follows. Alice computes a function $C(\text{blind}, \text{value})$, where $\text{value}$ is the value she wants to commit to and returns the output to Bob. The value $\text{blind}$ is called the blinding factor. Later Alice can give $\text{blind}$ to Bob and he can recover $\text{value}$.

A homomorphic commitment scheme is a commitment scheme $C$, where

$$C(\text{blind}_1, \text{value}_1) + C(\text{blind}_2, \text{value}_2) = C(\text{blind}_1 + \text{blind}_2, \text{value}_1 + \text{value}_2)$$

holds for all possible values and blinding factors. Furthermore we require $C(0, 0) = 0$. In fact, for confidential transactions it suffices if it is “easy” to detect commitments on zero. Pedersen commitments satisfy this property. In a Pedersen commitment, a cyclic group $G$, e.g., an elliptic curve, and two generators $P$ and $Q$ are used. We define a commitment on $\text{value}$ as $C(\text{blind}, \text{value}) = \text{blind} \cdot P + \text{value} \cdot Q$. Proving that this is a homomorphism is straightforward. The security of Pedersen commits rests on the discrete logarithm problem in $G$.

The idea of confidential transactions is to include a commit on the amount $C(\text{blind}_\text{in}, \text{amount}_\text{in})$ in the transaction input instead of the transaction amount $\text{amount}_\text{in}$ itself, thus, effectively hiding the amount. The output of the transaction is a committed value $C(\text{blind}_\text{out}, \text{amount}_\text{out})$. The blinding factor $\text{blind}_\text{out}$ is chosen by the sender of the transaction and needs to be revealed to the receiving party so that it can check that the correct amount was paid. Additionally, the receiver needs the blinding factor to spend the funds, since for spending he needs to choose special blinding factors for his transaction outputs such that all blinding factors cancel when added. This can be achieved in multiple ways, the easiest is encrypting the blinding factor with the public key of the recipient and appending it to the transaction.

Since there can be multiple inputs $\text{amount}_\text{in}^{(1)}, \ldots, \text{amount}_\text{in}^{(m)}$ and outputs $\text{amount}_\text{out}^{(1)}, \ldots, \text{amount}_\text{out}^{(n)}$ in a single transaction in Bitcoin one would need to check that no additional money is created, i.e., that

$$\sum_i \text{amount}_\text{in}^{(i)} = \sum_j \text{amount}_\text{out}^{(j)}$$

In confidential transactions there are no input and output amounts but instead commits $C\left(\text{blind}_\text{in}^{(1)}, \text{amount}_\text{in}^{(1)}\right), \ldots, C\left(\text{blind}_\text{in}^{(m)}, \text{amount}_\text{in}^{(m)}\right)$ and $C\left(\text{blind}_\text{out}^{(1)}, \text{amount}_\text{out}^{(1)}\right), \ldots, C\left(\text{blind}_\text{out}^{(n)}, \text{amount}_\text{out}^{(n)}\right)$ respectively. The blinding factors in the output need to be chosen in such a way that $\sum_i \text{blind}_\text{in}^{(i)}$ –
\[ \sum_j \text{blind}_\text{out}^{(j)} = 0. \] Then, due to the homomorphic property of the commitment scheme it suffices to check if
\[ \sum_i C(\text{blind}_\text{in}^{(i)}, \text{amount}_\text{in}^{(i)}) - \sum_j C(\text{blind}_\text{out}^{(j)}, \text{amount}_\text{out}^{(j)}) = C(0, 0) \]

But as required above, commitments on zero are easy to detect.

Consequently, the transaction amounts are not stored in the distributed ledger anymore. But since the transaction amounts are now hidden, we need to be certain that they are not negative. Commitment schemes usually work over the multiplicative group of finite field or over an elliptic curve, where integer overflow can happen which could lead to negative amounts. To ensure that none of the transaction amounts is negative, a cryptographic proof by the sender is needed. In fact, it suffices to ensure non-negativity of transaction outputs. Bulletproof [44] provides a more efficient scheme to compress these range proofs than the original article about confidential transactions [154].

Confidential transactions are currently not widely used, because the additional proofs increase the size of the transaction and thus their needed fee. Used together with coin mixing, confidential transactions can hide the participants in a transaction, as well as the transferred amounts, and thus achieve high levels of privacy for Bitcoin.

**Privacy-focussed Altcoins** Among the altcoins focussing on privacy, there are two main systems, Zerocash and Monero, which we will explain in the following.

Zerocash is an enhancement on top of Zerocoin. In Zerocoin [160] the blockchain is augmented to allow for additional private transactions. In addition to non-private transactions as in Bitcoin, there is a separate minting transaction in the blockchain which converts a zerocoin to an anonymous zerocoin. These anonymous zerocoins can then be transferred or redeemed into non-private zerocoins. The redemption of anonymous zerocoins is basically a non-interactive zero-knowledge proof [84, 204] that it has been minted before in some transaction, without revealing which one, together with a proof that it has not yet been redeemed. Additionally, when transferring anonymous zerocoins, these two proofs are attached. Since there is no link between minting and redemption the coins are untraceable. However, in Zerocoin only one denomination is possible and the recipients are revealed.

These two drawbacks are solved by Zerocash [23] which allows for different denominations by using commitments on transaction amounts. Zerocash uses zero-knowledge succinct non-interactive arguments of knowledge [31] instead of non-interactive zero-knowledge proofs due to efficiency reasons to prohibit double spending of a minted coin without detection. However succinct non-interactive arguments of knowledge require a trusted third party to generate the initial parameters. In the implementation of Zerocash, this is solved by generating the parameters using multiparty computation. I.e., multiple parties collaboratively compute the parameters and can recover the trapdoor only when all of them collaborate [22]. Thus, if at least one of the participants is honest, the parameters have been generated in a correct way, such that the trapdoor which would allow double spending cannot be recovered.

Danezis et al. [63] proposed an alternative construction to Zerocash called Pinocchio which bases its security on bilinear pairings of elliptic curves instead
of the discrete logarithm problem and the strong RSA assumption used in Zerocash.

A completely different approach to privacy in cryptocurrencies is given by CryptoNote and Monero. In 2013 Nicolas van Saberhagen\footnote{As Satoshi Nakamoto, Nicolas van Saberhagen is widely believed to be a pseudonym.} published the cryptocurrency construction kit CryptoNote\cite{202}. After providing it with some variables like the total money supply, the emission curve, and the seed nodes, it compiles a version of an anonymous cryptocurrency. Monero is probably the CryptoNote clone with the most active community and their developers have implemented many improvements over the original CryptoNote code base.

CryptoNote and thus Monero use two main technologies to achieve anonymity: \textit{linkable ring signatures} provide sender anonymity, and \textit{one-time keys} provide recipient anonymity.

A ring signature proves that a document is signed by a member of a group without revealing the identity of that member. In contrast to group signatures, ring signatures do not require a setup procedure and there is no group manager which can revoke the anonymity of participants.

For usage of ring signatures in a cryptocurrency, multiple fake inputs are included in the transactions additionally to the real input. Then, instead of a signature which proves that the sender is authorized to spend the real input, a ring signature proving that the sender is authorized to spend one of the inputs without disclosing which one, is included. For an observer it is thus impossible to distinguish the real input from the other fake inputs. To prevent double spending, a \textit{linkable} ring signature is used, such that signatures created with the same private key can be linked, again without revealing the identity. Note that the additional fake inputs need to have the same amount as the real input, since otherwise it is impossible to check the amounts in the transaction. CryptoNote uses a variation of the FS linkable ring signatures\cite{88}. Monero originally used the same signatures as CryptoNote, then switched to a variation of LWW signatures\cite{144} and currently uses multilayered linkable spontaneous anonymous group signatures to enable \textit{ring confidential transactions}\cite{171,172}. These ring confidential transactions combine the idea of ring signatures with confidential transactions as explained above and thus additionally hide the amounts contained in the transactions.

The second main privacy enhancing technology used in CryptoNote hides the recipient of the transactions. In Bitcoin, if transactions are sent to the same receiver they can be linked since they have the same public key in the output field. In CryptoNote, each participant has a long-time key pair. When sending a transaction, the sender derives a one-time public key from the long-term public key and sends the money to the one-time public key. The receiver can use his private long-time key to recover the private one-time key and hence spend the money. An observer cannot distinguish if different one-time keys have been derived from the same long-term key or not. Thus, the anonymity of the recipient is ensured, since two transactions cannot be linked as belonging to the same recipient.
Transactions

As shortly mentioned above, transaction outputs in Bitcoin contain code in a domain specific language which specifies the conditions under which the output can be spent. The original scripting language used in Bitcoin is a stack based language comparable to Forth and not Turing-complete, since it does not allow for loops or recursion. It is speculated that this is done on purpose, since otherwise there could be transactions whose validation does not terminate.

Nevertheless, one obvious suggestion to enhance blockchain systems is to substitute that limited scripting language by a Turing-complete one. This has been done by the project Ethereum. In Ethereum, additionally to transactions there are so called smart contracts, which go back to an idea of Szabo. These smart contracts are automated Ethereum wallets on the blockchain which can hold ether, the currency in the Ethereum network, and create transactions according to conditions which are specified in the smart contract. Ethereum contracts are programmed in a high level language like Solidity or Serpent and then compiled to code for the Ethereum virtual machine (EVM) and published in the blockchain. The EVM program code are the op-codes included in the smart contracts. Each contract provides a list of functions which can be invoked by sending money to them. Additionally, smart contracts are able to store internal state via variables. Computations in the EVM are executed by the miners when validating the transactions.

The problem of infinite loops is solved by equipping each contract invocation with a fee based on the op-codes that need to be executed. If a computation runs out of money, it is stopped and the fee is collected by the miner.

Abstractly speaking, each transaction in a blockchain is a state transition from some global state to another one. In Bitcoin these state transitions change the balance of the users. In Ethereum the state transitions correspond to computations in the EVM.

Additionally to Ethereum, a second proposal to realize smart contracts in a blockchain setting is called Hawk. In Hawk, financial transactions are not stored in clear on the blockchain but instead in an “encrypted” form, informally speaking. To achieve this, the compiler generates smart contracts which require cryptographic primitives such as zero knowledge proofs to interact with them.

Scalability

Another drawback of blockchain systems is their lack of scalability. In Bitcoin, there is a maximum blocksize to thwart DoS attacks which lead to the problem that there is a backlog of transactions. Hence transactions take 10 minutes or longer to confirm. The maximum throughput of Bitcoin is 7 transactions per second. As a comparison, the mainstream payment processor Visa processes 2000 transactions per second on average, with a peak capacity of 56,000 transactions per second.

Increasing the maximum block size or the block generation frequency does not completely solve this problem, since there is a natural trade off between...
transaction processing speed and security due to latency issues. Further, larger blocks increase the transaction throughput only linearly.

Sompolinsky and Zohar suggested to use a tree of blocks instead of a blockchain to lower the impact of latency issues. At each fork the heaviest subtree rooted at the fork is chosen for conflict resolution instead of the chain with the most proof of work backing it. Their protocol is called *Ghost* which stands for greedy heaviest observed subtree and is currently used in Ethereum to achieve a block generation time of 12 seconds.

Another approach to increase scalability is the development of an upper layer functionality over which multiple transactions can be sent without persisting all of them in the blockchain. The blockchain is only used for settlement of the aggregated transactions. This approach is known under the term payment channel. Users open a bilateral channel by locking money with a special channel opening transaction in the blockchain. Then a series of transactions can be sent off-chain without persistence in the blockchain. This is designed in such a way that cheating is impossible, i.e., that the transactions can theoretically be persisted into the blockchain but newer transactions have priority over older ones. Afterwards the channel can be closed and the final balances are written to the blockchain. It is possible to route transactions over multiple connected payment channels, thus effectively serving as an overlay payment network.

For Bitcoin there are two proposed implementations for payment channels which are based on different mechanisms. Duplex Micropayment Channels use multi signature transactions and time locks to achieve the priority of newer transactions. The lightning network uses so called Poon-Dryja channels named after their developers, where persistence of older intermediate transactions reveal a secret such that the honest user receives all the money which was initially locked in the channel.

For Ethereum, the envisioned payment channel network is called Raiden and allows for additional functionality like delegation of channels or refilling of already opened channels with more money. Raiden is realized as a number of smart contracts in the Ethereum virtual machine.

Malavolta et al. improve payment channels in Bitcoin in that they introduce privacy such that when routing a channel along multiple paths the sender and receiver cannot be linked.

For payment channels in Zerocash there is the additional requirement that the use of a payment channel should not violate the privacy of the user, since Zerocash is an altcoin focussed on privacy. *Bolt* which stands for blind off-chain lightweight transactions is a payment channel proposal which hides transaction data in payment channels. More exactly, upon receiving a payment the receiver learns nothing beyond the fact that a valid payment has occurred on an open channel. The network learns only that a channel of some balance has been opened or closed. While their authors write that Bolt can be used on Zerocash, their paper is written using any distributed ledger with sufficient functionality. The concrete construction of using Bolt in Zerocash is not specified in their work. Z-Channels by Zhang et al. are another approach for payment channels over Zerocash with comparable functionality. In contrast to Bolt, Zhang et al. have implemented their protocol and provide benchmarks.

It remains to be seen if payment channel networks really provide the boost to scalability in blockchain systems that is claimed, since the routing nodes can demand remuneration in form of a routing fee and the cost of locked capital.
in the channels is not free either. These two factors may suggest that the widespread usage of payment channels may lead to a more centralized and thus less privacy preserving network.

2.3 Proofs of Storage

A proof of storage or proof of retrievability is a protocol between a storage provider who acts as a prover, and a verifier, that allows the storage provider to cryptographically prove to the verifier that he has stored a special file without tampering or deleting parts of it. Approaches which are used in local settings like the use of checksums fail when applied to a remote setting due to a potentially malicious storage provider. On the one hand, if the storage provider stores both the file and the checksum it can delete parts of the file and modify the checksum such that the deletion is not noticed. On the other hand, if the checksum is stored at the client and the file is stored at the storage provider, each proof involves transmitting the whole file, which would convince the verifier, but is not very efficient. Thus, proofs of storage are typically required to be smaller than the file itself. In contrast to more general proofs of knowledge the verifier in a proof of storage has possibly learned the knowledge, i.e., the file already. Thus, the main challenge in proofs of storage is to encode the proof with length smaller than the file itself.

Types that exist are privately and publicly verifiable proofs of storage. In a privately verifiable proof of storage, only the uploading party can verify the proof, whereas a publicly verifiable proof allows verification by any third party, e.g., an external auditor. In particular, in publicly verifiable proofs of storage no secret knowledge is necessary for the verification.

The systems discussed later all use publicly verifiable proofs, preferably of constant length. However, for the sake of completeness and clarification we discuss some privately verifiable schemes in this section, since these are easier to understand.

Merkle Proofs of Storage The first notion of proofs of storage is possibly due to Merkle [158, 159]. Although Merkle trees have originally been invented to efficiently handle many Lamport one-way signatures [138], they are currently used in peer-to-peer systems to check that a partial file has been downloaded without modification. Assume that a client wants to download a file consisting of the chunks $m_1, \ldots, m_n$ which are aggregated in a Merkle tree as shown in Figure 2.3. The client is in possession of the root $H_0$ which serves as file descriptor. Now assume that the data chunk $m'_1$ has been downloaded and the client wants to check if it is the correct data chunk, i.e., if $m_1 = m'_1$ without first downloading the whole file. In this case, the client can ask its peers for the hashes $H_8, H_4$, and $H_2$ and reconstruct a root $H'_0$ by recomputing hash values $H'_7$, $H'_3$, and $H'_1$ along the path from $m_1$ to the root. If the recomputed root $H'_0$ is different from the real root $H_0$, at least one of the transmitted hashes or $m'_1$ is wrong. The ingenuity in this scheme lies in the fact that due to the hash $H$ being a one way function, the client cannot be convinced to accept a wrong $m'_0$, i.e., it is computationally infeasible to find $H'_6$, $H'_4$, and $H'_2$, such that the root $H'_0$ which was recomputed using $m'_1 \neq m_1$ is the real root $H_0$. 
2.3. PROOFS OF STORAGE

This can be used to construct a proof of storage as follows. The user takes his file $f = (m_1, \ldots, m_n)$ and aggregates the data in a Merkle tree. The file, together with the tree is given to the storage provider and the user keeps only the root of the Merkle tree. A challenge for a proof of storage consists of an index to a chunk $c \in \{1, \ldots, n\}$ chosen uniformly at random. The storage provider, if he stores the file honestly, is then able to provide $m_c$, along with the hash values needed to reconstruct the hash values on the path up to the root. This constitutes a proof of storage. For verification, the user recomputes the root hash and checks if it is the same as his local copy. If this is the case, the proof of storage is valid. Otherwise, the proof is considered invalid. Note that due to the one way property of the hash function it is infeasible for the storage provider to cheat, i.e., to produce a valid proof without storing the file. Further, the size of the proofs is only logarithmic in the size of the file and thus asymptotically smaller than the file. This scheme is publicly verifiable in the sense that only the Merkle root which is public knowledge is needed to validate a proof.

Juels Kaliski Proofs of Retrievability

A first formal treatment of proofs of retrievability was given by Juels and Kaliski [113] in 2007. In their scheme the client encodes the file by including so called sentinel-blocks into a file at locations unknown to the prover. The verifier stores the sentinels, as well as their positions. Creating a proof of retrievability requires the verifier to request the data at the position of a sentinel-block. The verifier then compares the response with the sentinel he expects at this position. Since the storage provider does not know where the sentinels are, he needs to store the whole file. Otherwise the storage provider could delete the file and only store the sentinels to produce valid proofs.

This scheme supports only a finite number of proofs, since there are only finitely many sentinels embedded into the file which can be queried. After all of the sentinels have been queried, the prover knows them and can delete the file, keeping only the sentinels to produce valid proofs. Further, the proofs are not publicly verifiable, since some secret knowledge, namely the sentinel-blocks and their positions are needed to verify the proof. A counterintuitive property of this scheme is that the knowledge used to compute the proofs, i.e., the sentinels, is different from the proven knowledge, i.e., the file. Thus, neither the prover...
nor the verifier need to actually have knowledge of the file to create a valid proof or validate one. This makes proofs of retrievability difficult to formalize.

**Provable Data Possession** A similar formalism using knowledge extractors (see [21, 83]) to formalize the knowledge of the file has been independently developed by Ateniese et al. [14] under the term provable data possession. There, the file is split into chunks which are signed by the user using a homomorphic signature scheme. These chunks together with the signatures are uploaded to the verifier. The challenge consists of a set of references to the chunks. Generating a proof involves combining the referenced chunks and signatures homomorphically, thereby creating a valid signature for the combined chunks. This signature together with the combined chunks then serves as a proof of data possession. The scheme of Ateniese et al. allows an unlimited number of verifications in contrast to the scheme of Juels and Kaliski [113] described above, but neither admit public verifiability. The formalization of provable data possession is easier, since the chunks themselves are used in the proof. Thus, soundness can be proven by constructing an extractor algorithm which is able to extract the original file from proofs.

Remark 2.3.1 (Proofs of Storage, Proofs of Retrievability, and Provable Data Possession). The terms proof of storage, proof of retrievability, and proof of data possession became increasingly blurred due to inconsistent usage.

A proof of retrievability [8, 42, 47, 71, 113, 210, 227, 231, 236] is a challenge-response protocol that enables a cloud provider to demonstrate to a client that a file is retrievable, i.e., recoverable without any loss or corruption. Proofs of data possession [14, 15, 58, 237, 239] are related proofs that still validate successfully if there is only a small amount of data corruption. Put differently, they are unable to detect small amounts of data corruption. In a proof of data possession only the existence of a knowledge extractor is required which can extract knowledge of the file, whereas a proof of storage [16, 208, 232] additionally requires the knowledge extractor to be efficiently computable. In particular it needs to have expected polynomial runtime.

Throughout this thesis, it makes almost no difference which of these concepts we use, since our constructions work with all of them and thus we will use the term proof of storage as an umbrella term for all of these concepts. A notable exception is Chapter 5, where we describe a novel proof of storage and benchmark it together with other proof of storage schemes. There, we mean proofs of storage with an efficient knowledge extractor.

There are many extensions and variants of these types of proofs. Dynamic proofs of storage allow for updating (insertion, modification, deletion) of chunks [47, 210] or even support revision control [237]. There is research covering distributed proofs of storage, where storage of multiple replicas of a file can be proven [58], or storage providers can prove that they cooperatively store a file, and each storage provider needs to store only parts of the file [239]. We will not delve into these special cases, and rather provide an informal but workable definition, which may not cover all special cases of proofs of storage. Even though this definition should be sufficient to understand most of our thesis in Chapter 5, we need to refine the definition. The formal definition of a publicly verifiable proof of storage is given in Definition 5.1.3.
Definition 2.3.1 (Proof of Storage, informal). A proof of storage scheme consists of four algorithms (Gen, Encode, Prove, Verify). The algorithm Gen is executed by the user in order to set up the scheme. Next, the user executes Encode to encode his file and add additional information. In the case of the Merkle proof of storage, this adds the tree structure, in the case of the Juels Kaliski proof of retrievability, Encode adds the sentinels to the file. In the case of provable data possession, the added information are the homomorphic signatures of the chunks. We will call the additional data authenticators. The encoded file, i.e., the file together with its authenticators is sent to the storage provider. A challenger, which in some schemes may be different from the user, sends a challenge value $c$ to the storage provider who then executes Prove to generate a proof of storage. The proof of storage can be verified using the algorithm Verify. Although in some schemes, only the user can verify the proof, since secret knowledge is needed for the verification, for our case only publicly verifiable proofs of storage, i.e., proofs of storage where everyone is able to verify the proof are important.

Remark 2.3.2 (Limitations of Proofs of Storage). A proof of storage is just a cryptographic proof of possession of a file. In particular, it does not guarantee confidentiality of the file. The data over which the proof of storage is created is known to the prover. However, confidentiality can be achieved on top of the proof of storage by standard cryptographic mechanisms.

Further, a proof of storage does not prevent modifications or loss of the file. A valid proof of storage however guarantees that the file is stored in an unmodified fashion. If the file is modified or parts are lost, the original file cannot be recovered by a proof of storage. However, the prover is then unable to generate further valid proofs of storage without possessing the whole original file. To impede modification or loss of the data, an erasure code can be applied on top of the proof of storage.

A proof of storage does not guarantee file retrievability. The storage provider can generate valid proofs without allowing other participants to retrieve the file.

2.4 Related Work

In this section we take a look at some related systems for anonymous distributed storage. In particular, we will discuss Permacoin, Retricoin, and Filecoin which provide incentives for their participants, and briefly sketch some anonymous storage systems without incentives. During the blockchain hype, a number of other altcoins which include decentralized storage emerged, like MaidSafe\(^{10}\), Storj\(^{11}\), and Sia\(^{12}\). However, their whitepapers often do not provide a full description of the system as important details are missing. Mostly these papers are not scientific and targeted more towards investors rather than researchers. Consequently, we cannot discuss all of these systems and focus on the ones mentioned above.

\(^{10}\)https://maidsafe.net/; accessed 2018-06-26
\(^{11}\)https://storj.io/; accessed 2018-06-26
\(^{12}\)https://sia.tech/; accessed 2018-06-26
2.4.1 Permacoin and Retricoin

According to our literature search, Permacoin \cite{162} is the first approach to combine the ideas of decentralized file storage and a blockchain. In Permacoin miners prove retrievability of a large publicly valuable digital archive where single miners are unlikely to have the resources to store all the data. Miller et al. \cite{162} mention as motivation the ancient library of Alexandria which was destroyed by fire as an example for such a publicly valuable archive.

In Permacoin the challenges for the proofs of retrievability are generated pseudorandomly using the previous block, as well as the public key of the miner, and a seed chosen by the miner. If the miner is unable to find a new block since it cannot provide the necessary proof of retrievability, i.e., it does not store the requested part of the file, a new seed is chosen. Thus, the scheme still contains some elements of proof of work, due to the choice of seeds.

The motivation of Permacoin is very different from ours and consequently the design decisions are very different. In Permacoin, the digital archive is fixed at the time of setup of the blockchain and no changes to it are possible. For the initial distribution of the file a trusted dealer is needed. In particular, users cannot upload or download data in contrast to our designs.

Retricoin \cite{206} offers some efficiency improvements over Permacoin but inherits their main design decisions. In particular, Retricoin uses the proof of retrievability of Shacham and Waters \cite{207, 208} and thus has less storage overhead. Further, Retricoin introduces a protocol by which a group of miners can form a pool with this new mining algorithm.

2.4.2 Filecoin

Filecoin is an approach to incorporate a file system into a cryptocurrency. The design of Filecoin was significantly altered between its first white paper \cite{85} from 2014 and its second white paper \cite{190} from 2017. With the second design, its developers have been able to collect 257 million USD through an ICO\footnote{https://www.coindesk.com/257-million-filecoin-breaks-time-record-ico-funding/}, and thus Filecoin may be the most promising project of its kind. In both designs, it is possible to store and fetch files chosen by the users. As a drawback, none of the designs takes the topic of privacy into account and neither the first nor the second white paper is currently implemented.

First Filecoin Design  The design described in the first Filecoin white paper from 2014 \cite{85} allows the upload of data selected by users by a special transaction called *put*. Another transaction called *get* allows the download of data. However, the get transaction is not enforced by cryptographic mechanisms and is solely dependent on the philanthropy of the miners which additionally serve as storage providers.

Storage of files is guaranteed by demanding miners to create a proof of retrievability in addition to a proof of work in the mining process.

In a put transaction, additionally to uploading a file, the reward for a miner proving possession of this file is specified. The authors of Filecoin suggest that for each proof of storage the reward decreases linearly over time. Thus the user can decide how important his files are, by setting the rewards in the
2.4. RELATED WORK

put transaction appropriately. Furthermore, the user can balance between a rapid spreading of his chunks and the lifetime of the data by increasing or decreasing the linear factor of the decrease of the storage reward. Since the reward decreases, after a specified time, miners have no more incentive to store old files and thus delete them from their storage.

Regarding the mining algorithm, Filecoin extends the classic hash-based proof of work mining of Bitcoin with an additional proof of retrievability. Thus valid blocks contain two proofs with two separate difficulty parameters to regulate the growth speed of the blockchain. One difficulty parameter controls the hash-based proof of work while the second determines the difficulty of the proof of retrievability. In the Filecoin white paper it is not explained how these two difficulty parameters are designed to interact or how they are adjusted.

The difficulty parameter for the proof of retrievability is realized by the number of files of which miners need to prove retrievability. Beyond a certain difficulty parameter, small miners are never able to mine new blocks because they do not have the necessary storage. This leads to centralization pressure, since these small miners are unable to mine blocks beyond a certain difficulty and consequently will not use the system. Note that in Bitcoin’s design this is theoretically not the case, since there, all miners can mine new blocks with a probability proportional to their share of mining power with respect to the total mining power of all participants. Thus, in Bitcoin even miners with little mining power are able to mine new blocks.

Another problem of the first Filecoin design is that this scheme incentivizes file hoarding. If a node stores a file that no one else has, only this node can create correct proofs of retrievability for this particular file and thus has no incentive to distribute the file further in the network. An ineffective guard proposed in the white paper is to demand that each file referenced in the currently mined block has to be broadcast into the network. So, theoretically, for each proof of storage the corresponding files get distributed. However, since the broadcast of files is not cryptographically verified, this countermeasure does not work. Thus, a hoarding miner can simply omit sending the files without any repercussions.

Regarding privacy, the users and their files are linked because they are contained in the same put transaction. Additionally, the miners and their stored files are linked through the information contained in the blocks. Filecoin does not offer any transactional privacy improvements beyond Bitcoin. Admittedly, privacy has never been stated as a design goal in the Filecoin system.

Nevertheless, despite suffering from some technical problems, the first Filecoin white paper has provided the community with a leap towards a distributed storage service with financial payments.

Second Filecoin Design  The second Filecoin white paper from 2017 [190] offered drastic changes to their system architecture. Instead of proofs of retrievability, the second Filecoin system uses a proposed proof of replication which is a proof that any replica of data is stored in physically independent storage. Additionally, the second Filecoin design includes proofs of spacetime which guarantees that data is stored throughout a range of time instead of at a single point in time like in proofs of retrievability. However, only the definitions are given, without any proposed constructions. Even their dedicated technical
report about proofs of replication \cite{192} provides no construction. Chapter four in this report is called “Realizable Constructions (TODO)” as of January 2018. Consequently, it is unknown if a feasible proof of replication or a proof of spacetime can be achieved.

Furthermore the second Filecoin white paper suggests to introduce a novel consensus protocol and references some technical reports discussing these in depth. In their work about power fault tolerance \cite{191} byzantine fault tolerance is generalized, such that some participants can have more influence over the consensus process. For their novel consensus protocol called expected consensus an unpublished technical report \cite{189} is given. According to the white paper of Filecoin \cite{190} the basic idea of their consensus protocol is to deterministically, unpredictably, and secretly elect a small set of leaders at each epoch which are allowed to mine blocks, comparable to proof of stake.

Finally, for storing and retrieving files Filecoin introduces two markets to allow for price discovery. The respective order books are published on the blockchain. This allows the settlement process to be executed by the miners. However, some problems such as frontrunning by miners, i.e., reordering the transactions before persisting them in the blockchain to gain a financial advantage, are not addressed. Nicola Greco, a member of the Filecoin team claims in private communication \cite{100} that there are ways to solve the frontrunning problem and cites the work by Clark et al. \cite{53}. However, that article states that it is difficult to mitigate such manipulations and instead makes a claim for embracing front running \cite[Section 6.1]{53} in contrast to common economic knowledge.

Building a fully decentralized exchange where frontrunning is infeasible is an ambitious aim in itself and current systems either rely on trusted arbiters \cite[223]{67} who publish proofs that they match orders correctly, or on multiparty computation \cite{35} where $t$ out of all $n$ participants are assumed to be honest.

In summary, the second publication of the Filecoin team clearly does not constitute a working solution for the problem of decentralized storage and leave many open questions.

### 2.4.3 Anonymous Storage Systems

There are a number of anonymous storage systems like GNUnet \cite[102]{25} or Freenet \cite{54} which do not rely on payment to attract storage providers but instead on voluntary contribution of resources of its participants. These architectures are very focussed on privacy, in contrast to the systems mentioned previously. While GNUnet is geared towards anonymous and censorship-resistant file sharing, Freenet is envisioned as an anonymous replacement for the Internet. Hence, in Freenet it is possible to publish anonymous web sites, though only static sites are possible. Routing in GNUnet is based on the addresses of files and uses a system based on a distributed hash table similar to Kademlia \cite{155}. In Freenet, files with similar addresses cluster by design in similar regions of the network, since files are replicated at each hop when queried. It is considered a feature, that data which is only rarely queried slowly disappears over time from the Freenet system due to a lack of replication. In particular, in both GNUnet and Freenet there is no single storage provider responsible for a specific file who could be remunerated. Hence, there are no storage guarantees in contrast to
systems with payment which can provide storage guarantees assuming rational actors.

2.4.4 Summary

In summary, it can be said that none of the discussed systems solves the problem of a distributed file system with remuneration. Although, Permacoin, Retricoin, and Filecoin combine the idea of payments based on a blockchain, each of these systems has their drawbacks.

Permacoin and Retricoin do not allow the upload and download of data. However, in our systems we require that participants should be able to upload and retrieve data. Filecoin’s first design allows for uploading data, but suffers from technical problems as explained above. Filecoin’s second design clearly does not provide a working system. Further, all of the three systems Permacoin, Retricoin, and Filecoin do not take the topic of privacy into account. This is due to the fact that participants are remunerated only for storing files. Thus, the remuneration links the participant to its stored file.

Anonymous storage systems like GNUnet and Freenet do not provide rewards for its participants and thus cannot reveal links between rewards for participants and files of the rewarded participant. Further, no single storage provider is responsible for a specific file which makes it hard to build remuneration on top of these systems, since it is unclear who should be remunerated. GNUnet and Freenet have a different motivation from our system. GNUnet’s focus is on providing file sharing. Freenet’s focus is on anonymous web sites. Thus, both of these systems are geared toward content distribution instead of simply providing storage.

Our system in the next chapter aims to be a working alternative to Filecoin. It can also be thought of as Permacoin/Retricoin with files defined by the participants. As discussed, it is hard to augment an anonymous storage system with payments since it is unclear who should be remunerated. Consequently, our second privacy-preserving storage system augments a privacy aware blockchain based payment system with storage rather than augmenting an anonymous storage system with remuneration.
As we have seen, one of the current problems of peer-to-peer-based file storage systems like Freenet is missing participation, especially of storage providers. Users are expected to contribute storage resources but may have little incentive to do so.

In this chapter, we propose a distributed storage system inspired by Bitcoin’s blockchain, where peers can store and retrieve files. This system includes a blockchain based token system to reward those contributing storage resources. While Bitcoin’s blockchain uses computational power as a physical scarce resource to guard against Sybil attacks, our proposal uses the amount of stored chunks which have been uploaded previously. Thus, peers who store files of others are able to mine on these files and consequently have the chance to generate tokens, called koppercoins. The reward for storage is by design and users cannot withhold payments from storage providers. This is a consequence of the storage providers serving as miners of the blockchain. In addition, participants can gain koppercoins by allowing other peers to retrieve files. These tokens in turn can be sold or spent to store more files. Storing and fetching is realized by special transactions on the blockchain.

A proof of storage is used by miners to prove the contribution of storage resources. This allows miners to prove that they have stored data of other participants. However, in order to be useful in a mining algorithm, the proof of storage needs to be modified to include an adjustable difficulty parameter to adjust the block generation frequency. In our system this is achieved by requiring an upper bound on the distance of the address of a chunk whose storage is proven to a challenge value determined from the previous block.

Our mechanism creates a big advantage over traditional distributed file storage systems since in our system, users have valid incentives to contribute storage to other users. Even commercial entities can base their business model on mining, as in Bitcoin mining enterprises, but with the added benefit of

I’d like to buy some Koppercoins. When is this going to be possible? Thanks!

One of my students
contributing to the decentralized file storage.

Parts of this chapter have been previously published at ISPEC [126].

**Contribution** We claim the following contributions:

- We provide a novel design for a distributed storage system with financial incentives for its participants to contribute.

- Our design is based on a blockchain which does not rely on proof of work for its mining process. Instead, we make use of proofs of storage over data defined by the participants in order to generate new blocks.

- In order to use proofs of storage as a mining algorithm, we extend them by a difficulty parameter.

**Roadmap** In the next section we explain the design of our scheme. We discuss its properties and drawbacks in Section 3.2. Section 3.3 concludes this chapter.

### 3.1 Our Scheme

In this section we explain the proposed construction of our basic scheme. We will first provide an overview and then explain the mining process in Section 3.1.2. Section 3.1.3 explains how to store files, and Section 3.1.4 shows how to fetch them again from the network.

**Remark 3.1.1 (Possible Extensions).** Our system only provides the abstraction of storing and retrieving chunks of data. More advanced functionalities are achieved via mechanisms on higher layers.

To achieve confidentiality of the stored data, users can choose to encrypt their files before uploading it into our system using standard cryptographic mechanisms.

Data sharing between different users $U_1, \ldots, U_n$ is achieved by encrypting the file with a symmetric key which is then encrypted using the public keys of each of the users. The encrypted symmetric keys need to be prepended to the file. In order to access a file, a user decrypts the symmetric key with his private key. The symmetric key is then used to decrypt the file. A system providing this functionality that could be used in combination with our system is miniLock\(^1\).

To allow users to increase their retrieval guarantees, an application interacting with our system uses an erasure encoding. This increases redundancy, but allows for recovering the whole file even if some parts are missing. Thus, the user can decide to pay more money—since the redundancy increases the size of the data—for a higher retrieval guarantee.

\(^1\)http://minilock.io/
3.1. OUR SCHEME

3.1.1 Overview

Our scheme identifies each entity by its public key as in Bitcoin. We will occasionally use \( M \) to denote a miner and \( U \) to denote a user of the system for purposes of clarification, though each participant in the system can take on any of these roles, even at the same time. We use our own blockchain, different from the one in Bitcoin, as a global public transaction log. In contrast to Bitcoin, our scheme does not reward the miners proportionally to their computational resources, but instead proportionally to how much data of other participants in the network they store. Mining a block requires a proof of storage over some data chunk which has been uploaded previously by a user.

A simplified overview of our construction is shown in Figure 3.1. A file \( f \) is represented as a vector of chunks \( f = (f_1, \ldots, f_\ell) \) of same length, possibly padded. We always denote the pieces of a file by the term “chunk”, whereas “block” always refers to blocks in the blockchain, to prevent ambiguity. The chunks, when uploaded, cannot be linked to files, since they have identical length. A client application is needed for the splitting into chunks, tracking the identifiers of the chunks and reassembly on retrieval, together with optional encryption to achieve confidentiality. Additionally, the client application performs erasure encoding. An erasure encoding accomplishes that the file is still recoverable even if some chunks are missing. Thus, an erasure encoding can be used to achieve higher retrieval guarantees. However, the cost for the user is higher, since an erasure encoding increases the size of the file.

Our design supports all transaction types that are supported by Bitcoin, which makes it possible to transfer koppercoins to other parties in the network. Furthermore, our system introduces a new transaction store which is used to upload chunks of a user \( U \) into the network (Step 1a in Figure 3.1). It contains the public key \( pk_U \) of the user which is needed for verification of the proofs of storage, a chunk \( f_i = (f_{i1}, \ldots, f_{in}) \) consisting of \( n \) subchunks, and its authenticators \( \sigma \) which are needed to create proofs of storage. Further, it
includes a payment which determines how long the chunk should be stored. The koppercoins used in the store-transaction are removed from the network and become unspendable as will be explained in Section 3.1.3. Rewards for storing are gained only by mining or providing files to others. In parallel the user broadcasts the chunks in the underlying network (Step 1b) and miners can decide to store the chunks (Step 1c).

Storing a chunk allows miners to append blocks to the blockchain. As in Bitcoin, mining a new block grants the miner a mining reward (see Step 2a and 2b in Figure 3.1).

Mining a new block requires a publicly verifiable proof of storage over a data chunk which is “close” to a challenge value determined by the previous block header in the blockchain. The distance is computed in the address space of the chunks as will be explained later and acts as a quality parameter of the block. In particular, blocks are considered valid if this distance is smaller than a difficulty parameter. Invalid blocks are rejected by the miners on arrival as in Bitcoin. The proof of storage is computed over chunks and not over files, i.e., each chunk \( f_i \) is split into \( n \) subchunks \( (f_{i1}, \ldots, f_{in}) \) to create the proof of storage. Since all chunks \( f_i \) have the same size, the number of subchunks \( n \) is independent of the chunk.

Proof of storage schemes for our purpose are, e.g., the scheme of Shacham and Waters [207, 208] or Ateniese et al. [16], since they are publicly verifiable and of constant size. Further treatment of proofs of storage for our scenario is given in Chapter 5.

Since the challenge for the proof of storage is not known in advance, a miner who stores more chunks has a higher probability of possessing a chunk close enough to the challenge to mine a new block. To encourage users to participate in the system, a mining reward in form of koppercoins is given to the creator of a new valid block as in Bitcoin. Thus, the more chunks a miner stores the higher the probability of earning koppercoins.

The proof of storage ensures integrity of the blockchain by making it prohibitively expensive to change previous blocks, since this requires redoing many proofs of storage over arbitrary chunks. In contrast to Bitcoin, the block headers alone do not suffice to check integrity of the blocks since the public key of the user, which is included in the store-transactions, is required to check the proofs of storage. Nevertheless, not every miner has to store each block, but only the data necessary to verify integrity of the blockchain and the transactions. In particular, not every miner needs to store all chunks which are included in the store-transactions. Using Merkle trees, it is possible to remove the stored chunk without changing the hash of the aggregated transactions which is included in the blockchain headers.

The exact time of expiration of a chunk depends on the amount of koppercoins used in the initial store-transaction. In case a miner includes a proof of storage of an expired chunk into a new block, this block is considered invalid by the other miners and consequently discarded. Thus, miners have no incentive to store expired chunks and rational miners will delete them from their local storage. Thus, the expiration mechanism allows the network to regain storage space.

Lastly, users can retrieve their files by paying for them (cf. Step 3a and 3b in Figure 3.1). This consists basically of a payment on the level of the blockchain.
and the file transfer itself. File retrieval and its remuneration is designed in such a way that no party can cheat, as will be explained in Section 3.1.4.

3.1.2 The Blockchain and Mining Process

This section explains how the mining process in our system works.

The file storage in our network is designed as a key-value storage. There is a global set of keys \( K \) and a corresponding set of chunks, \( f_i, i \in I \subseteq K \). Only a subset of the keys reference chunks, such that for many keys there exists no according chunk. In a real implementation the key \( i \) is likely the hash of the chunk \( f_i \). The client application keeps track of the keys of the user’s chunks.

A valid block header in our system includes the following fields:

- The Merkle root of the transactions aggregated into the block, which we denote by \text{merkle\_root}.
- A reference to the previous block header, realized as a hash value.
- The difficulty of the mining algorithm.
- A time stamp \text{tmstmp} in an appropriate granularity, e.g., five seconds, which is used in the mining process and to readjust the difficulty parameter.
- A proof of storage \( \pi \) over a chunk \( f_i \), as well as a reference to the \text{store\_transaction} where \( f_i \) has been uploaded.

Algorithm 2 describes the mining process. This algorithm is executed by each miner every time the local time stamp at the miner advances or a new block is received. Newly computed blocks are broadcast into the network. If a new block is received it is checked for validity of the included transactions and correctness of the proof of storage. Let \( pk_M \) be the public key of the miner \( M \), which can be retrieved from the coinbase transaction contained in the block. Then, analogous to Bitcoin’s requirement of small hashes of block headers, valid blocks in our system need to fulfill the following difficulty property:

\[
\mathcal{H}(pk_M \| \text{tmstmp} \| \text{merkle\_root}) \cdot 2^{i \oplus H} \leq \text{difficulty},
\]

where \text{tmstmp} is the time stamp when the block was mined, \( H \) is the hash of the previous block, \text{merkle\_root} is the root of the Merkle tree containing the transactions, and \( i \in I \) is the index of the chunk whose storage has been proven. The symbol \( \oplus \) denotes bitwise XOR-operation and the result is interpreted as an integer [155]. As in Bitcoin, the difficulty parameter is periodically adjusted to account for changes in the underlying hash rate. The block is then accepted or rejected accordingly.

We will now explain the mining process shown in Algorithm 2 in detail. Let \( \mathcal{H}_{\text{mine}} \) be a cryptographically secure hash function which assumes values in the set of keys \( K \). The miner \( M \) computes the challenge \( H \) as a hash of the block header \( B_n \) in Line 1. In Line 2, \( M \) computes the index \( i \) of the chunk over which the proof of storage will be created. This is chosen as the index of the locally stored chunk which is closest to the challenge \( H \) in the XOR-distance. With this choice, the term \( 2^{i \oplus H} \) takes on its minimum. Since \( 2^{i \oplus H} \) needs to be smaller than some difficulty parameter, the choice of the chunk closest to the
Algorithm 2 The mining algorithm for computing new blocks in our system

**Input:** Time stamp $t_{\text{mstamp}}$, newest block header $B_n$, difficulty, root $\text{merkle\_root}$ of the Merkle tree containing the transactions.

**Output:** next block header $B_{n+1}$ if possible to compute

1. $H = H_{\text{mine}}(B_n)$ $\triangleright$ challenge for the mining process
2. $i = \arg\min_k \{ H \oplus k \mid f_k$ and its authenticators are stored locally $\}$ $\triangleright$ index closest to $H$ where the corresponding chunk is stored locally
3. if $H(pk_M||t_{\text{mstamp}}||\text{merkle\_root}) \cdot 2^{i \oplus H} \leq \text{difficulty}$ then
4. $c = H_{\text{por}}(B_n)$ $\triangleright$ challenge for the proof of storage
5. $\pi =$ proof of storage of the chunk $f_i$ with challenge $c$
6. return new block with aggregated transactions and proof of storage $\pi$.
7. end if
8. return next block is not mineable.

challenge $H$ is optimal to mine a block. In Line 3 the miner tests if the index of his chunk is near enough to the challenge $H$ and thus if it is possible for him to mine the next block. If this is the case the proof of storage is created in Line 4 and 5 and the new block is broadcast into the network. Otherwise, the next block is currently not mineable for $M$ and he has to wait until the timestamp advances, a valid block of another participant is received, or new transactions are received.

While the mining process needs a challenge derived from the previous block header $B_n$ to guarantee that the proofs of storage become invalid, if the old blocks are modified, an additional challenge $c$ is needed for the proof of storage itself as explained in Definition 2.3.1. Using the Fiat-Shamir transformation \[84\] to obtain non-interactivity for our proof of storage we can generate $c$ by a pseudorandom number generator. In particular, in our system the challenge $c$ is generated by applying a pseudorandom function $H_{\text{por}}$—mapping block headers to challenges—to the current block header $B_n$ in Line 4. The resulting proof of storage $\pi$ is published in the header of the newly mined block $B_{n+1}$.

Note that if the same chunk $f_i$ is challenged in two different blocks $B_s$ and $B_t$ with $s \neq t$ the two challenges $c = H_{\text{por}}(B_s)$ and $c' = H_{\text{por}}(B_t)$ used in the proof of storage should be different. If this is not the case, replay attacks are possible by reusing the proof of storage of the old block to mine the new one. But the same chunk $f_i$ is challenged in different blocks only with negligible probability, since the generation of the same challenge implies a collision of a cryptographically secure hash function $H_{\text{por}}(B_s) = H_{\text{por}}(B_t)$.

When a miner receives a block it first checks if the condition on the difficulty is satisfied, i.e., if $H(pk_M||t_{\text{mstamp}}||\text{merkle\_root}) \cdot 2^{i \oplus H_{\text{mine}}(B_n)} \leq \text{difficulty}$ holds. Next, the miner checks if the time stamp lies in an appropriate window, as in Bitcoin. Next, the miner verfies the proof of storage contained in the block. Finally, the transactions are validated. In particular it needs to be checked if the transactions contain valid signatures, the amounts are correct, and the referenced transaction outputs have not been spent before. If any of these checks fail, the block is considered invalid and discarded. Otherwise, the block is valid and mining continues on top of it.

Our choice of the “quality” of the block and therefore of the difficulty of
mining a block as
\[
Q(B_n, pk_M) = H(pk_M || \text{tmstmp} || \text{merkle\_root}) \cdot 2^{i \oplus H_{\text{mine}}(B_n)}
\]

has several desired properties:

(i) \( Q \) intuitively behaves like a difficulty parameter, since the smaller the distance between the key of a chunk whose storage is proven and \( H_{\text{mine}}(B_n) \), the smaller the result of our mapping \( Q \) is. Further, the more chunks a miner stores, the higher is its probability of storing a chunk whose address is close to \( H_{\text{mine}}(B_n) \). Consequently, the probability of mining a block is proportional to how many files the miner stores:

\[
P \left[ \frac{\text{pk}_M' \neq \text{pk}_M : Q(B_n, \text{pk}_M) < Q(B_n, \text{pk}_M')} {\# \text{ files in the system}} \right] = \frac{\# \text{ files stored by pk}_M}{\# \text{ files in the system}}
\]

(ii) The mapping \( Q \) is dependent on the miner. If two or more miners prove storage of a chunk with the same distance to the challenged \( H_{\text{mine}}(B_n) \), they get different values for \( Q \) since \( \text{pk}_M \) is included in the hash function. This property is important, since if the mapping does not depend on the miner such a situation allows both miners to create a new block and thus lead to a fork of the blockchain.

(iii) It is impossible to end up with a block \( B_n \) where no-one can successfully append a next block \( B_{n+1} \), since \( Q \) changes if the slotted time stamp changes. This provides liveness of the blockchain, i.e., it is always possible to find a subsequent block after sufficient time has passed.

We have chosen the XOR-distance \( d(x, y) = x \oplus y \) as a distance function, since it is unidirectional \cite{155}. Thus, for each distance \( \delta \) and each fixed bit sequence \( x \) there is exactly one \( y \) satisfying \( d(x, y) = \delta \). In our system this implies having a different distance to each chunk and therefore a clearly defined priority over which chunk storage needs to be proven.

Regarding integrity of transactions, if an already persisted transaction is changed, the root \text{merkle\_root} of the Merkle tree changes unpredictably. Since this is included in a cryptographically secure hash function to compute the “quality” of the block as \( H(pk_M || \text{tmstmp} || \text{merkle\_root}) \cdot 2^{i \oplus H} \), each bit changes with probability 1/2. Consequently, it is infeasible to modify transactions which are already included in the blockchain, if there are enough files in the system and thus integrity of the transactions is guaranteed.

As in Bitcoin, occasionally it can happen that two blocks are mined simultaneously by different miners \( M \) and \( M' \), thus creating a fork in the chain. In this case the miners try to extend the chain at the block where the value which is compared against the difficulty parameter, i.e., \( H(pk_M || \text{tmstmp} || \text{merkle\_root}) \cdot 2^{i \oplus H} \) or \( H(pk_M' || \text{tmstmp} || \text{merkle\_root'}) \cdot 2^{i' \oplus H} \), is smaller. When the two chains differ in length they are mining on the longest chain by the protocol rules. Thus, this chain grows faster, since it is backed by more resources and eventually the miners abandon the shorter chain.

In Bitcoin, if some malicious miner controls the majority of computational resources, he can extend both chains at the same speed, thereby preventing consensus. In our system this situation can also happen, but the attacker needs
more than half of the storage resources of the network, instead of computational resources. We assume that this is infeasible if our network is big enough. Additionally, an entity controlling a majority of storage resources will perhaps prefer to comply with protocol rules, since otherwise trust in the system will disappear and therefore the koppercoins, which the adversary would be able to mine, become worthless.

3.1.3 The Store Transaction

The store transaction allows participants to store chunks. store takes as input a chunk $f_i$, its authenticators $\sigma$ computed by the user, the public key $pk_U$ of the user $U$, as well as an amount of koppercoins. These koppercoins vanish from the network and cannot be spent anymore. This payment is necessary to avoid denial-of-service attacks, since without the payment an attacker can upload an arbitrary number of chunks for free and thereby exceed the total available storage in the network.

When receiving a store-transaction, miners can choose to store the chunk together with its authenticators to be able to create a proof of storage over this chunk in the future. If the authenticators are invalid, the store-transaction is rejected to guard against cheating users. After the store-transaction is persisted in the blockchain, miners need to store the public key $pk_U$ of the user for verification purposes. The miners do not need to store all files and are possibly not even able to do so. The incentive to store chunks is of economical nature, since by storing chunks, miners can increase their chance of mining new valid blocks and thus collect mining rewards.

Beyond these financial incentives there are no further mechanisms for achieving storage guarantees. The retrieval guarantees can be increased arbitrarily by the user by applying an appropriate erasure code on the file to be uploaded. Assume that a file $f$ consists of $\ell$ chunks and each chunk is stored with a probability of $p$. Then the probability that the whole file is stored is $p^\ell$. An erasure code increases the overhead such that in total $(1 + \delta) \cdot \ell$ chunks need to be uploaded to the network. But only $\ell$ out of those $(1 + \delta) \cdot \ell$ chunks need to be stored to recover the full original file. Thus, the probability that the file is recoverable is $1 - P(\delta \cdot \ell + 1$ chunks are lost) $= 1 - (1 - p)^{\delta \ell + 1}$. Since this converges to one for large $\delta$ the retrieval guarantees can be increased arbitrarily close to one by choosing an appropriate $\delta$. Note however, that the cost of storing an erasure encoded file increases linearly in $\delta$.

The storage period of the chunk is linearly dependent on the amount of koppercoins spent when issuing the store-transaction. The reason for this is, that the resources used from the network are proportional only to the storage duration. Note especially that the payment is independent from the size of the chunks since the chunks are of constant size.

After the storage period has passed, new blocks which include a proof of storage over that particular chunk are not considered valid any more. Assuming the majority of miners does not accept such blocks, there is no incentive to store these expired chunks any longer and miners can delete them from their storage. Since the blockchain already provides a lose synchronization of time all miners can agree on when the requested storage period has passed.
3.1. OUR SCHEME

3.1.4 Fetching Files

Miners may not be willing to give users their files back, since this consumes bandwidth and the miners do not gain an advantage from it. Thus, there needs to be a mechanism in place to allow users to remunerate miners for their expenses for each file retrieval.

Before we go on to describe our solution, let us note that a straightforward solution allows different forms of cheating depending on the order of payment and file transfer during retrieval. If the user pays first, the miner has no incentive anymore to transfer the file. However, if the storage provider transfers the file first, the problem is not solved either since in this case the user loses its incentive to pay the storage provider.

One cryptographic primitive which is often used in these situations is called fair exchange or simultaneous exchange of secrets \[34, 62, 146, 178\]. This mechanism allows two parties to exchange two digital items simultaneously, although neither trusts the other. However, fair exchange needs to address the expectation of the participants of the respectively other’s item, i.e., the inputs need to be verified \[43\]. In our case, the user expects to receive its file and the miner expects to receive a valid transaction. If this is not taken into account the cheating participant can exchange garbage against the payment or the file. Another problem is that there is no prevention against participants aborting the protocol \[219\]. Current protocols distinguish between strong and weak fairness \[10\]. In strong fairness, at the time of protocol termination either both participants get what they want or neither of them does. In weak fairness, an honest participant can prove to an external arbiter that the other participant has received the item it expected. Our case requires strong fairness, since there are no fixed identities in our system and thus a reputation system is infeasible. However, it has been proven that a strongly fair exchange without a trusted third party is impossible \[181\] by reducing it to the famous impossibility result of Fischer, Lynch, and Paterson \[86\]. Since we do not have a trusted third party in our architecture, we need to work around this cryptographic impossibility result by loosening the requirements.

For example, in Bitcoin there are strong fair exchange protocols without a trusted third party, due to the fact that in Bitcoin, redeeming a transaction leaks a preimage of a hash \[45, 66, 143\]. Thus, the requirement of non-invasiveness, i.e., that the fair exchange protocol does not impose any requirements on the form of the items being exchanged, is violated.

We propose a simpler approach for exchanging payments against files, using the assumption of rational actors, which is nonstandard in the cryptographic literature. However, this assumption is commonly used in game theory. A rational actor seeks to maximize its utility function. Thus, the assumption of rational actors formalizes that users in our system are profit-seeking. Game theory as well as the field of cryptographic protocols examine the collaborative interactions between mutually distrusting parties. Consequently, there is some overlap between these different areas of research, despite their different goals and mechanisms \[9, 70, 125\]. Since our use case deals with mutually distrusting parties as well, we think that the assumption of rational actors is justified.

To deter both parties from cheating we make use of security deposits. On a high level both parties are brought into a situation, where they have a game-theoretic disadvantage if they do not behave honestly, i.e., they lose their
security deposits. This idea is heavily inspired by the cold war doctrine of mutually assured destruction, where participants are deterred from using their nuclear arsenal, since this leads to complete destruction of the attacker and the defender. To our knowledge the idea of using such a mechanism to exchange goods was first introduced by the decentralized trading platform NashX\textsuperscript{2}.

In particular, our system utilizes a special payment transaction for each file retrieval, called \textit{mutual assured destruction (MAD) transaction}. A MAD-transaction consists of a \textit{multi signature transaction}, i.e., a transaction that needs to be signed by multiple parties in order to be spent. In particular we use a 2-2 multi signature transactions which can be spent if and only if two out of two parties (the user and the miner) agree to spend them.

In order to fetch a file, the client application needs to know the identifiers of the corresponding chunks. The file is restored by retrieving sufficiently many chunks. For successful retrieval not all chunks have to be fetched, depending on the erasure code that has been applied before storing the file in the network. The erasure code solves the problem of missing chunks and storage providers taking files hostage by demanding unrealistically high prices for chunk retrieval.

Let $U$ be the user who wants to retrieve a chunk. Note that there are probably different miners which store the chunk, so $U$ can choose among them. In particular, $U$ broadcasts a request for the chunk into the network containing the key of the chunk. Each miner storing the chunk answers the user $U$ with an offer. Next, $U$ chooses one of the miners, possibly the cheapest one, to retrieve its chunk. This penalizes miners which quote high prices, since it leads to them not earning anything through the file retrieval process. Thus, miners cannot hold chunks hostage. Suppose that $M$ is the cheapest miner offering retrieval of the chunk at the price $p$. Suppose further that $U$ wants to pay the amount $p$ for retrieving its chunk. Then $U$ and $M$ create a 2-2 multi signature transaction where the user has an input of $D_U + p$ koppercoins and $M$ has an input of $D_M$. The amounts $D_M$ and $D_U$ are security deposits. In a next step $M$ sends the chunk to $U$. The user $U$ checks if he has received the correct chunk. If the miner has not sent the correct chunk to $U$, the user $U$ does not free the funds locked previously in the multi signature transaction. Thus, the user effectively punishes the miner $M$ for its misbehavior by destroying the security deposit.

However, if the user $U$ received the correct chunk, $U$ signs a payout transaction which spends the multi signature transaction. The payout transaction has two outputs: The miner $M$ gets back its security deposit $D_M$, together with the price $p$ for the chunk. In the other output the user $U$ gets back its security deposit $D_U$. The process is illustrated in Figure 3.2. Above the arrows are the amounts and below the arrows are the owners of the respective amounts.

\textsuperscript{2}http://nashx.com/HowItWorks; accessed 2018-02-08
3.1. **OUR SCHEME**

<table>
<thead>
<tr>
<th>User $U$</th>
<th>Storage Provider $M$</th>
</tr>
</thead>
<tbody>
<tr>
<td>comply</td>
<td>$f - p, p$</td>
</tr>
<tr>
<td>cheat</td>
<td>$-D_U - p, -D_M$</td>
</tr>
<tr>
<td>comply</td>
<td>$f - D_U - p, -D_M$</td>
</tr>
<tr>
<td>cheat</td>
<td>$-D_U - p, -D_M$</td>
</tr>
</tbody>
</table>

Table 3.1: The payoff matrix for MAD-transactions

If the user $U$ wants to cheat he cannot set its security deposit $D_U$ to zero or otherwise change the first transaction which locks the deposits since this will be detected by the miner $M$ who then refuses to sign. Nevertheless $U$ can refuse to sign the 2-2 multi signature transaction after retrieving the chunk. But thereby the user loses its security deposit $D_U$.

If the miner $M$ decides to cheat it can either refuse to send the chunk or refuse to sign the release of the 2-2 multi signature transaction. In both cases $M$ will suffer a financial damage of its security deposit $D_M$ and additionally miss out on its reward $p$ for serving the chunk.

This leads to the payoff matrix show in Table 3.1. There, $f$ denotes the value of the file. Let us take a look at the strategy (comply, comply). In this case, if the user changes its strategy and cheats instead, its utility is $f - D_U - p$ which is clearly smaller than $f - p$. Thus, the user has no incentive to change its strategy. If the miner $M$ cheats instead its utility is $-D_M$ instead of $p$. Consequently, the miner has no incentive to change strategies from (comply, comply). The strategy (comply, comply) is a nash equilibrium since no party can gain an advantage when changing its strategy. An analogous line of reasoning shows that (cheat, cheat) is a nash equilibrium. These two strategy pairs—(comply, comply) and (cheat, cheat)—are the only nash equilibria in this game.

However, (comply, comply) weakly dominates the other nash equilibrium of (cheat, cheat). This means that the payoffs of each player with the strategy pair (comply, comply) are greater or equal than the payoffs of the strategy pair (cheat, cheat). Thus, rational actors play the strategy pair (comply, comply).

One idea for a user $U$ to cheat is by blackmailing the storage provider in the MAD transaction. When requesting a file $U$ can refuse to sign the second multi signature transaction freeing the funds. Instead $U$ offers to sign a different multi signature transaction granting only a small non-negative amount $\varepsilon$ to the storage provider $M$ and the remaining amount $D_M + D_U + p - \varepsilon$ to $U$. If the storage provider does not sign this transaction, its deposit $D_M$ is lost. On the other hand, if $M$ signs it, $M$ only loses $D_M - \varepsilon$, which is less. Thus, a rational storage provider will agree to the blackmailing transaction. The risk of $U$ when blackmailing the storage provider in such a way is losing its deposit $D_U$ if the miner does not agree to the blackmailing transaction.

In our system this problem of blackmailing cannot occur since in our system the storage provider $M$, and not the user, sends the second multi signature transaction. Thus, the user $U$ can only accept or refuse to sign, in which case $U$’s deposit is lost. $U$ does not have the opportunity to send a blackmailing transaction to the storage provider.
3.2 Security and Privacy Analysis

Compared to other classical proof of work blockchains, our design uses less computational resources, since the mining process additionally to providing security of the blockchain is used to power the storage system. While we have described only the two most basic file operations, it is possible to extend the design with more transactions which allow for deletion of chunks or for extending the storage period of a file.

Our architecture provides a form of censorship resistance, since it is not known in advance which peers store which chunks. The decision if a miner stores a chunk is made locally at each miner. Therefore it is impossible for an attacker to provably delete all instances of a special file by threatening the peers storing it.

Compared to classical anonymous storage systems, our system rewards early adopters with the decreasing block reward from mining. Rewarding early adopter is done since it proved to be a major factor in the widespread adoption of protocols in the past [140].

One of the disadvantages of our system is the lack of deterministic retrieval guarantees, since there is no possibility to know if a chunk has been stored by any peer at all. After a chunk has been broadcast into the network, each miner decides if it stores the chunk, based on economical considerations. In practice, each miner allocates a fixed amount of storage to the network and receives chunks, until that storage is depleted. Then the miner uses the stored chunks to generate new blocks. If the storage period of a chunks has expired the miner removes that chunk and refills its storage with new chunks.

It is possible to guard against losing a fraction of the chunks by erasure coding of the files. Nevertheless, this does not solve the problem completely and it is unclear if storage guarantees in a system without fixed identities can be achieved at all. Even classic peer-to-peer systems like BitTorrent or Freenet do not provide any storage guarantees. We may assume that participants in the system are aware of this issue and factor in the risk of losing their files in the market price of koppercoins.

3.2.1 Security

It is hard to give a formal proof of correctness for our mining rule since there is no established model for the attacker in blockchain systems, due to the mix of cryptographic and game-theoretic assumptions. Even Bitcoin does not provide formal proofs of correctness. Garay et al. [90] attempted to provide these, abstracting a blockchain protocol by the functions $V$, $R$, and $I$ for verifying, contributing to the chain, and reading the chain, respectively. However, their analysis is only valid in an idealized setting called the $q$-bounded synchronous setting.

Consequently, in the following we only discuss selected attacks.

Brute-forcing in the Mining Process Miners can cheat in the mining process by using brute-force to bias the randomness in favor of the miner.

Recall that for mining a valid block, the “quality” of a block needs to be smaller than some difficulty parameter. Thus, a lower quality parameter implies
3.2. SECURITY AND PRIVACY ANALYSIS

a higher change to mine a block.

\[ \mathcal{H}(pk_M \| tmstmp \| merkle\_root) \cdot 2^{i \oplus H} \leq \text{difficulty}, \]

Again, \( i \) is the index of a chunk close to \( H = \mathcal{H}_{\text{mine}}(B) \), because otherwise the left hand side grows larger. Since the public keys of the miners \( pk_M \) can be generated by the miners themselves, a miner can gain an unfair advantage by generating multiple public keys and trying to generate blocks with these. This is possible even if the miner \( M \) is not in possession of a chunk whose index is close to \( H \). The same is true for the root of the Merkle tree of transactions \( merkle\_root \). A miner can rearrange or omit transactions in order to yield a different \( merkle\_root \) which results in a different “quality” of the block and thus mine the block.

We will now show that although brute-forcing is possible, it only yields a limited advantage. Especially, we will show that the logarithm of the quality parameter goes to zero exponentially in the number of stored chunks, but only polynomially in the number of brute force guesses.

First, let us inspect the advantage miners have when storing more chunks. Suppose the images of \( \mathcal{H}_{\text{mine}} \) and \( \mathcal{H} \) are uniformly distributed in \( \{0, 1\}^n \). Then the expected value for the “quality” of a block with only one try is

\[
E(\mathcal{H}(pk_M \| tmstmp \| merkle\_root) \cdot 2^{i \oplus H}) = E(\mathcal{H}(pk_M \| tmstmp \| merkle\_root)) \cdot 2^{E(i \oplus H)}
\]

due to independence of the involved random variables.

Now, suppose that a miner \( M \) stores \( \ell \) chunks in total. Since the value of the “quality” of the block is smallest (and thus its chance of mining the block is highest), when the miner chooses to prove storage of a chunk near \( H \), the miner will do so. \( H \) is uniformly distributed in \( \{0, 1\}^n \) due to assumption and thus \( i \oplus H \) is uniformly random as well. Consequently, the value in the exponent is \( E(\min\{u_1, \ldots, u_\ell\}) \), where \( u_1, \ldots, u_\ell \) are uniformly distributed. We will now compute this value. As \( F(x) = x^{2^n} \) is the cumulative distribution function for all \( u_j \), we get \( 1 - (1 - F(x))^\ell \) for the cumulative distribution function of the minimum. The probability density function \( f \) of the minimum can be computed as the derivative

\[
f(x) = \frac{\partial}{\partial x} 1 - (1 - F(x))^\ell
\]
\[
= \frac{\partial}{\partial x} \left[ 1 - \left(1 - \frac{x}{2^n}\right)^\ell \right]
\]
\[
= 2^n \ell \cdot \left(1 - \frac{x}{2^n}\right)^{\ell-1}
\]

With this, the value in the exponent of the expectation of the “quality” \( E(\min\{u_1, \ldots, u_\ell\}) \) can be computed.
\( E(\min\{u_1, \ldots, u_\ell\}) = \sum_{x=1}^{2^n} x \cdot f(x) \)
\[= \sum_{x=1}^{2^n} 2^n \ell x \cdot \left(1 - \frac{x}{2^n}\right)^{\ell-1} \]

Since the above term is a sum over a constant amount of terms it is in \( O(c^\ell) \) for some constant \( c < 1 \). Thus, the term is negligible in the number of stored chunks \( \ell \). Intuitively, this means that the logarithm of the “quality” of a block is negligible in the number of stored chunks. Since the “quality” of a valid block is smaller than the difficulty parameter, this is what we wanted to show.

Next, we show that the logarithm of the “quality” of a block is only polynomial in the number of brute-force attempts, and thus brute-forcing only yields a negligible advantage compared to storing the chunks. Again, we compute the expected quality, but this time we fix the chunks.

\[ E(H(pk_M || tmstmp || merkle\_root) \cdot 2^i \oplus H) = E(H(pk_M || tmstmp || merkle\_root)) \cdot 2^{\log(2^i \oplus H)} \]

It follows that the mining attempts for brute-forcing are uniformly distributed random variables \( u_1, \ldots, u_\ell \in \{0, \ldots, 2^{2^n}\} \). Analogous to above

\[ E(\min\{u_1, \ldots, u_\ell\} \cdot \text{const.}) = \text{const.} \cdot \sum_{x=1}^{2^n} 2^n \ell x \cdot \left(1 - \frac{x}{2^n}\right)^{\ell-1} \]

This time, however, we are not computing in the exponent, as above with the chunks. Thus, the “quality” of the block is negligible in the number of tries \( \ell \), compared to above, where the same statement concerned the logarithm of the “quality”.

Thus, brute-forcing only yields a negligible advantage over honestly storing the blocks. Nevertheless, since brute-forcing yields an advantage at all, the miners may resort back to it. In practice, our computations only show a trade-off between storage and memory resources.

**Nothing at Stake** Our system is vulnerable to nothing at stake attacks. This means, that in the case of a fork malicious miners can extend both forks simultaneously instead of only one. This is possible, since the mining process uses only few resources, thus miners do not need to decide on a single chain to extend. When the time stamp advances, the miner can check both chains and mine a block if possible instead of only one chain.

The first resulting problem is that malicious miners gain a mining advantage, since the miners can mine on both chains and thus especially on the chain which will become the longer one. The second effect of the nothing at stake attack is that forks are more unlikely to be resolved. If all miners mine on all forks, then the ends of the blockchain will grow approximately with the same speed. Consequently, the consensus protocol fails since no single longest chain emerges.

As the attack stems from the fact that the mining process uses only few resources, a countermeasure which we implement in the next chapter is to switch back to the resource intensive proof of work done by Bitcoin.
3.2. SECURITY AND PRIVACY ANALYSIS

Exploiting the Proof of Storage  Another flaw was brought to our attention by Armknecht [7]. It includes various forms of a participant gaining an unfair advantage in mining by being a user and a miner simultaneously. If the private key used to encode the file in a proof of storage is known, it is possible to generate a proof of storage without storing the file. Thus, an attacker $A$ can upload multiple files and keep only the private keys used in encoding the chunks and the addresses of the chunks. This allows $A$ to mine blocks by generating valid proofs of storage without storing the chunks, whereas other participants need to store the chunks to generate blocks.

Another attack [7] involves an attacker generating bogus data and keeping the generation algorithm, e.g., by using a pseudorandom bit generator and keeping its seed. Whenever a chunk by the attacker $A$ is challenged, it can regenerate the chunk, whereas the other miners not in charge of the generation algorithm need to store the chunks.

These two attacks are possible if participants are allowed to be storage providers and users at the same time. There is no trivial solution to this problem, since it is hard to guarantee that storage providers and users are separate non-colluding entities. While an identity manager provides a solution to the problem of separating users and storage providers, this is contrary to our design goal of an open system with dynamic participants. In the next chapter we solve this problem by large changes to our architecture. Namely, storage providers are not paid by mining but instead by the user whose data they store. Thus, we need to revert back to mining by proof of work, as in Bitcoin.

3.2.2 Privacy

In order to analyze the privacy properties of our design, we employ the LIND-DUN methodology [230]. LINDDUN is an established approach to privacy threat modelling and can be used to investigate and improve the privacy of software systems. The LINDDUN approach consists of the following six steps.

Step 1: Define dataflow diagram (DFD).
Step 2: Map privacy threats to DFD elements.
Step 3: Identify threat scenarios.
Step 4: Prioritize threats.
Step 5: Elicit mitigation strategies.
Step 6: Select corresponding PETs.

Out of these we skip the fourth step of prioritizing threats, since we consider all privacy threats to be equally important in our design.

We make the following assumptions during the analysis.

1. All internal processes are only vulnerable to insider threats. This implies, that we do not consider the decision of the miner which data to store.

2. Data flows between entities, internal processes, and internal data stores are only susceptible to insider threats. Thus, we can disregard them for our analysis.
3. Data stores which are not internal are not considered confidential. This allows us to model the blockchain as a public data store.

4. We consider linkability to be an important threat in our system. In particular we do not want to be able to link users to their files or their transactions.

5. Identifiability of entities, i.e., miners and users, is considered a threat, as the system should provide anonymity of its participants.

6. Non-repudiation threats are not considered in the system, since the processes and data flows do not require plausible deniability.

7. We do not consider detectability a threat for our system as its privacy concerns are focussed on the content of the data.

8. Disclosure of information is not considered a threat, since it is a peer-to-peer system without a central service provider. Thus, most of the data is public and not treated confidentially.

9. We do not consider unawareness of the user as a threat, as this depends heavily on the graphical interface of the system and not on its technical properties.

10. We do not consider non-compliance as a threat, since this depends heavily on the specific regulation.

11. We assume that processes are not vulnerable to corruption or side channel attacks, as they are correctly implemented.

**Define Dataflow Diagram (DFD)**

In the first step of LINDDUN, a DFD needs to be created. This is shown in Figure 3.3. The users send their transactions and chunks as a broadcast to the miners, leading to the first data flow DF1. The miners send the transactions as input to the consensus protocol, i.e., the mining process. This is data flow DF2. If the consensus protocol succeeds, a new block is found and appended to the blockchain (DS1), denoted by the data flow DF3. When a miner receives a chunk of a user to store the miners needs to decide if it stored the chunks, depending on its storage space and potential reward. Thus, the miner send the chunk to the process Store (DF4). The process decides if that special chunk is stored in the local data store of the miner (DS2) or not. The corresponding data flow is called DF5. The dashed rectangle denotes a trust boundary, since everything inside happens locally at each miner. Thus it is only susceptible to insider threats from the miner, due to our assumptions.

**Map privacy threats to DFD elements**

As a next step in the LINDDUN process privacy threats need to be mapped to elements of the data flow diagram. This is done in Table 3.2. LINDDUN considers the privacy threats linkability, identifiability, non-repudiation, detectability, information disclosure, content unawareness, and policy noncompliance. As
3.2. SECURITY AND PRIVACY ANALYSIS

Figure 3.3: A Data Flow Diagram of our System. As common in these diagrams rectangles denote entities, circles denote processes, two lines denote data stores, and arrows denote data flows.
argued in our assumptions above, for our scenario it suffices to consider only
linkability and identifiability as threats.

In the LINDDUN methodology entities are only considered to be at risk
if there is a linkable login information at a service provider. Since this is not
the case in our system, we do not have to map privacy threats to entities.
Further, due to our assumption of correctly implemented processes we do not
map privacy threats to the processes. Thus, we are left with checking linkability
and identifiability at the data flows and data stores.

<table>
<thead>
<tr>
<th>E</th>
<th>DF</th>
<th>DS</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linkability</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Identifiability</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
</tbody>
</table>

Table 3.2: Mapping of privacy threats to elements of the data flow diagram.
The symbol × denotes the mappings we have to check. Cells containing “—”
can be disregarded.

Identify threat scenarios

The next step in applying the LINDDUN methodology is to identify specific
threat scenarios and document them. For this, LINDDUN offers privacy threat
trees which need to be traversed in order to detect if there is a threat present.
In Table 3.3 this is done for each of the data flows. Table 3.4 provides the
identification of threat scenarios for data stores. We do not consider the
data flows DF4 and DF5, since these are locally in the miner and due to our
assumptions we consider them to be only susceptible to insider threats. Each
of the threats is documented in Figures 3.4 to 3.8. These templates constitute
misuse cases, i.e., use cases for the attacker.

<table>
<thead>
<tr>
<th>Linkability</th>
<th>Identifiability</th>
</tr>
</thead>
<tbody>
<tr>
<td>DF1</td>
<td>T1</td>
</tr>
<tr>
<td>DF2</td>
<td>T1</td>
</tr>
<tr>
<td>DF3</td>
<td>T1, T3, T4</td>
</tr>
</tbody>
</table>

Table 3.3: Threat Scenarios at Data Flows

<table>
<thead>
<tr>
<th>Linkability</th>
<th>Identifiability</th>
</tr>
</thead>
<tbody>
<tr>
<td>DS1</td>
<td>T1, T3, T4</td>
</tr>
<tr>
<td>DS2</td>
<td>—</td>
</tr>
</tbody>
</table>

Table 3.4: Threat Scenarios at Data Stores

In summary we have found five threat scenarios:

• T1: Linkability of senders and receivers
3.2. SECURITY AND PRIVACY ANALYSIS

Summary: Each transaction contains a sender and a receiver. Thus the transaction data itself provides a link between senders and receivers.

Consequence: Each participant in the system can link the senders of each transaction to their receivers. This allows to trace payment flows and thus violates privacy.

Threat Tree Nodes: L_df7, L_ds

Figure 3.4: Threat T1: Linkability of Senders and Receiver

Summary: Each transaction contains the senders and receivers in plain format. This allows their identification.

Consequence: Each participant in the system with access to the transaction data can identify the senders and receivers of each transaction. This allows to determine the balance of other participants.

Threat Tree Nodes: L_df5, L_ds

Figure 3.5: Threat T2: Identifiability of Senders and Receivers

Summary: Since each block contains the address of its miner in order to grant it a reward, blocks contain a link to their respective miners.

Consequence: Miners can be linked to their mined blocks and thus it is possible to trace the mining rewards for each miner.

Threat Tree Nodes: L_df5, L_ds

Figure 3.6: Threat T3: Linkability of miners and their mined blocks

- T2: Identifiability of senders and receivers
- T3: Linkability of miners and their mined blocks
- T4: Linkability of miners and their stored chunks
- T5: Identifiability of miners

The next step in the LINDDUN methodology is to prioritize the threats. However, since we consider all threats to be equally important we can skip this step.
Summary: Since each block contains a proof of storage and the address of its miner, blocks provide links between miners and their stored chunks.

Consequence: Miners can be linked to their stored chunks. This allows to determine which files have been stored by a miner. Note that this threat is only applicable if a miner has generated a block. Thus, not all stored chunks of a miner are revealed.

Threat Tree Nodes: $L_{df5}, L_{ds}$

<table>
<thead>
<tr>
<th>Summary:</th>
<th>Each block contains the public key of its miner in plain format, thus revealing its identity.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consequence:</td>
<td>The public identity of miners is revealed. This threat can be seen as a special case of T1—linkability of senders and receivers—as the miners are receivers of their mining rewards.</td>
</tr>
<tr>
<td>Threat Tree Nodes:</td>
<td>$L_{df5}, L_{ds}$</td>
</tr>
</tbody>
</table>

Figure 3.7: Threat T4: Linkability of miners and their stored chunks

Figure 3.8: Threat T5: Identifiability of Miners

Elicit Mitigation Strategies

Misuse cases describe the relevant threat scenarios for our system. Now that we know these, LINDDUN offers mitigation strategies based on the respective nodes in the privacy threat trees. In our case, the relevant nodes are $L_{df7}$, $L_{df5}$, $L_{ds}$, and $L_{ds}$. These basically mean that the data itself (not the metadata) is not sufficiently anonymized or hidden.

LINDDUN offers four mitigation strategies: Remove, Hide, Replace, and Generalize.

Using the strategy **Remove** implies to remove the senders and receivers from the transactions. Additionally, the information which miner mined a block is removed. This removes functionality from the system, since it is impossible to reward miners if any identifying information is removed from their blocks. Nevertheless, it is possible to remove the proofs of storage from the transactions, after the transactions are hidden sufficiently deep in the blockchain. Then, the proofs of storage can be considered valid, since otherwise they would not have been included in the blockchain in the first place. The validity of the proof of storage and not the proof of storage itself is the information which is necessary for checking the validity of blocks. Let us now take a look at senders and receivers in transactions. When senders and receivers are removed from

---

3Note that these are LINDDUN mitigation strategies and thus different from Hoepman’s privacy design strategies as discussed in Section 2.1
transactions, what remains is the amount of the transaction with no owner. This clearly decreases functionality.

Applying the strategy Hide, the links between senders and receivers are still there but hidden and only visible for the sender and the receiver of the respective transaction. The same can be done for miners and their blocks. We will apply the strategy Hide in the design of the next chapter. However, it is unclear how to hide the link between a miner and its stored chunk if a proof of storage is necessary for mining a block, since the proof of storage is always bound to a specific chunk. Further, these proofs of storage need to be publicly verifiable and thus the link cannot be hidden. Consequently, in the design in the next chapter we reintroduce the proof of work mining process in order to separate the link between a miner and its stored chunks.

The strategy Replace suggests to replace the data by other data which is not personally identifiable, but fulfills the same purpose. In our use case, the most important data contained in a transaction are the identity of the sender and receiver and the amount of the transaction. Replacing the sender and the receiver by different data does not make sense.

The last strategy Generalize aims to generalize the data. This can be in the form of aggregation or simply a coarser granularity of the data. In our case this means storing sets of senders and sets of receivers in the transactions. However, again this decreases functionality, since the transaction data cannot be validated.

Select corresponding PETs

Consequently, we employ the privacy threat mitigation strategy Hide in the next chapter in order to separate the link between senders and receivers in transactions. In particular, we hide the senders by using linkable ring signatures. These allow to conceal the true sender of a transaction in a set of other decoy senders. Anonymity of receivers is achieved by one-time payment addresses. Each participant has a long-term public and private key. If a sender wants to send a transaction, it derives a one-time public key from the long-term public key of the receiver. Only the receiver with its long-term public key is able to extract the according one-time private key and thus spend the transaction again. These techniques mitigate T1 (Linkability of senders and receivers), T2 (Identifiability of sender and receiver), and T5 (Identifiability of miners). Additionally, T3 (Linkability of miners and their mined blocks) is mitigated since the link between the miner and its mined block is a reward transaction. Consequently, achieving anonymity of receivers of transactions hides miners in their reward transactions, since miners are the recipients of the reward transactions. It remains to remedy T4 (Linkability of miners and their stored chunks). This is not done by employing the strategy Hide, but instead by separating the roles of miners and storage providers, as discussed in the next chapter.

Conclusion

As money is created by mining and not through payments from users, the link between storage and payment is separated. This results in immediate benefits for privacy and censorship resistance, since users cannot be linked through the
payment process to their stored chunks. However, the transactions in the system do not provide any privacy guarantees beyond Bitcoin, which are deemed to be insufficient as previously discussed.

The files are mostly unlinked to miners, except if a block is mined. In this case, the miner creates a proof of storage and thus proves which file it stores. So, each block reveals exactly one link between a miner and a stored file. The link between a user and its files is public, since it can be deduced from a store-transaction in the blockchain.

In summary, while our system provides a decentralized storage, it is not fully privacy-preserving and additional measures need to be taken to increase the privacy guarantees of the system, as will be discussed in the next chapter.

3.3 Conclusion

This chapter presented how to combine a decentralized token system with a peer-to-peer file storage which provides direct reward for participants contributing storage resources. It is based on the idea of a blockchain to manage files and ownership of the tokens. The mining process to maintain the network is realized by a proof of storage instead of a proof of work, thus providing incentives to store files in the system.

We outlined basic concepts and discussed benefits of our system in terms of tight integration of the file storage system with the token system as a reward mechanism. We did not focus on the underlying network layer, although for a secure operation of our system this should be considered as well.

While our system is a promising approach to implement a distributed peer-to-peer file storage system that offers incentives for participation, there need to be more mechanisms in place to achieve privacy properties. In particular, our LINDDUN privacy analysis showed the following privacy threats:

• T1: Linkability of senders and receivers
• T2: Identifiability of senders and receivers
• T3: Linkability of miners and their mined blocks
• T4: Linkability of miners and their stored chunks
• T5: Identifiability of miners

Additionally, our design suffers from several security flaws such as a nothing at stake attacks which leads to a failure of the consensus mechanism, since forks are not resolved. Further, it is possible to gain an unfair advantage by abusing the proof of storage, as explained in Section 3.2.1.

As such, the design presented in this chapter provides an intermediate step towards the privacy preserving storage system in the next chapter.
Privacy Enhanced Design

Solve et coagula!

Alchemical motto

In the last chapter we provided the design of a decentralized storage offering remuneration of its participants. However, the previous design does not provide strong privacy guarantees for its participants. In this chapter we modify the previous scheme using privacy technologies in order to guarantee anonymity of its participants. We combine a privacy-enhanced payment scheme on the basis of blockchain with a distributed storage system.

The design in the last chapter suffered from nothing at stake attacks, which allows miners to extend multiple ends of the blockchain in the case of a fork. As a consequence, no single longest chain emerges and thus the consensus protocol fails. As the attacks is due to the fact that the mining process uses only few resources, in this design we switch to proof of work, as in Bitcoin. Another reason why we switch to proof of work is that in the design of the last chapter, a malicious miner could gain an unfair advantage by uploading random data and only storing the seed of the random number generator. This attack is solved if we switch to proof of work, since miners do not gain an advantage by doing this. Consequently, in our new design the proof of work is done by miners and storage needs to be integrated by another mechanism.

Especially, in our new design miners and storage providers can be separate entities. We need to provide incentives not only for the miners, but for the storage providers as well. This is done as follows: When storing a file, a user creates a storage smart contract together with a specific storage provider to exchange amounts of a digital currency against storage of a file. Again, we call the currency units koppercoins. The storage smart contract is broadcast into the network and persisted in the blockchain if it is correct. It contains a payment, which can only be spent by the storage provider, if the storage provider can prove storage of the user’s data. Again, the proofs of storage are publicly verifiable and persisted in the blockchain in order to allow miners to check if the storage provider has adhered to the contract. If the storage provider behaves honestly, it gains a reward. Otherwise, if the storage provider has broken the contract clauses the user is refunded. Since these contracts provide a way to reward participants who contribute their storage resources to
Recall from the previous chapter, that our previous design suffers from five privacy threats.

- **T1**: Linkability of senders and receivers
- **T2**: Identifiability of senders and receivers
- **T3**: Linkability of miners and their mined blocks
- **T4**: Linkability of miners and their stored chunks
- **T5**: Identifiability of miners

Note that basically all privacy threats boil down to **T2**, the identifiability of senders and receivers.

For **T1**, senders and receivers of transactions are only linkable, since their identity is known and contained in a transaction.

Regarding **T3**, miners are linkable to their mined blocks, since the identity of the miner is contained in the block. This is necessary since the miner gains a reward for mining that particular block. However, the miner is a recipient of this reward transaction. Thus, if we remedy **T2** and achieve receiver anonymity, the link between miners and their mined blocks is separated.

After switching to proof of work, as explained above, **T4** is not about the linkability of miners and their stored chunks, but rather about linkability of storage providers and their stored chunks. This is due to the fact that the storage provider is the recipient of a payment in a storage contract. Thus, storage providers are linked to their stored chunks due to **T2**, the identifiability of receivers in transactions. Consequently, we have reduced all privacy threats to **T2**: Identifiability of senders and receivers of transactions.

As foreshadowed, to remedy these issues we employ the privacy design strategy **Hide**. In particular, we anonymize the sender and receiver of transactions.

In order to tackle receiver anonymity, we make use of linkable ring signatures as briefly described in Section 2.2.2. Each transaction input references multiple different transaction outputs, such that the real transaction input is hidden among these fake inputs. The ring signature then proves that one of the inputs is owned by the signer without revealing which one. Linkability of the ring signatures is needed to prevent double spending attacks by detecting if two transactions are signed by the same key. Variations of this are currently used by CryptoNote and Monero.

Sender anonymity is achieved by using one-time payment addresses. When sending a transaction, the sender derives a one-time public key from the long-term public key of the recipient. The transaction is then addressed to this one-time public key. The recipient can recover the one-time private key which corresponds to the one-time public key and spend the funds. Additionally, any ppt observer is unable to link two one-time public keys as belonging to the same long-term public key.

These solutions—linkable ring signatures for sender anonymity, and one-time keys for receiver anonymity—constitute privacy patterns, as they are general
4.1 Building Blocks

This section offers an elaborate description of linkable ring signatures which are used in our system to achieve sender anonymity, as well as one-time payment addresses which are used to achieve receiver anonymity.

4.1.1 Linkable Ring Signatures

Ring signatures are a special kind of digital signatures due to Rivest, Shamir and Tauman [198]. Recall that a ring signature proves that a document was signed by a member of a group of signers. Since the identity of the member is not revealed, this provides a level of privacy. To create a ring signature the signer needs its own private key, as well as the list of public keys of the other members in the group. In particular, no group setup procedure is necessary. For verification, the signature and the list of public keys of the members in the group are needed. In a nutshell, ring signatures are digital signatures where the signer is $k$-anonymous, i.e., indistinguishable from $k-1$ other possible signers.

We will now define linkable ring signatures and then go on to explain their usage in our system together with a concrete instantiation. Our definition is according to Liu et al. [144]. However, they define linkability only with respect to a fixed group of public keys. In contrast, in our definitions and constructions linkability needs to be achieved even across different sets of members in the ring, since otherwise double spending becomes possible.

**Definition 4.1.1 (Linkable Ring Signature ([144])).** A linkable ring signature scheme is a quadruple of algorithms, $\Sigma = (\text{Gen}, \text{Sign}, \text{Verify}, \text{Link})$ such that:
1. \((pk, sk) \leftarrow \text{Gen}(1^\lambda)\) is a ppt algorithm that takes as input a security parameter \(\lambda\) and outputs a public and secret key pair \((pk, sk)\).

2. \(\sigma \leftarrow \text{Sign}(m, sk, pk_{1,\ldots,n})\) is a ppt algorithm run by the signer to create a ring signature \(\sigma\). Its input is a message \(m\), a secret key \(sk\), as well as a list of public keys \(pk_{1,\ldots,n}\). In a usual invocation of the signing algorithm, \(sk\) corresponds to one public key of the list \(pk_{1,\ldots,n}\).

3. \(b := \text{Verify}(\sigma, m, pk_{1,\ldots,n})\) is a deterministic algorithm run by the verifier which returns a single bit \(b\), where '1' indicates acceptance and '0' indicates rejection. Its input is a message \(m\), a signature \(\sigma\), and a list of public keys \(pk_{1,\ldots,n}\).

4. \(\ell := \text{Link}(m, m', \sigma, \sigma')\) is a deterministic algorithm that takes as input two messages \(m\) and \(m'\), together with their signatures \(\sigma\) and \(\sigma'\) and returns one bit \(\ell\), where \(\ell = 0\) indicates that the signatures are linked, i.e. were signed by the same secret key \(sk\), and \(\ell = 1\) indicates that they are independent.

For a linkable ring signature we require the following properties:

**Correctness** This guarantees that a correctly signed message will be accepted by the verification. For all security parameters \(\lambda \in \mathbb{N}\), all \((pk_i, sk_i) \leftarrow \text{Gen}(1^\lambda)\) where \(i \in \{1, \ldots, n\}\) for some \(n \in \mathbb{N}\), all messages \(m\), all \(\sigma \leftarrow \text{Sign}(m, sk_1, pk_{1,\ldots,n})\) it holds that \(\text{Verify}(\sigma, m, pk_{1,\ldots,n}) = 1\).

**Linkability** Intuitively, linkability means that two messages that are signed by the same secret key will be spotted by the algorithm \(\text{Link}\). Conversely, Link will claim only with negligible probability that two signatures from different secret keys are linked. In the original definition of Liu et al. [14] the detection works only if the signature is created regarding the same set of public keys. Our definition in contrast assumes linkability even if the signatures have been created using different sets of public keys. For all security parameters \(\lambda \in \mathbb{N}\), all \((pk_i, sk_i) \leftarrow \text{Gen}(1^\lambda)\), where \(i \in \{1, \ldots, n\}\), all \((pk'_j, sk'_j) \leftarrow \text{Gen}(1^\lambda)\), where \(j \in \{1, \ldots, \ell\}\), and any messages \(m, m'\), the term
\[
P\left[\text{Link}(m, m', \text{Sign}(m, sk_1, pk_{1,\ldots,n}), \text{Sign}(m', sk_1', pk'_{1,\ldots,\ell})) = 0\right]
\]
is negligible in \(\lambda\). Intuitively this means that the algorithm \(\text{Link}\) has a negligible probability of marking two independent signatures as linked. Additionally for all security parameters \(\lambda \in \mathbb{N}\), all \((pk_i, sk_i) \leftarrow \text{Gen}(1^\lambda)\), where \(i \in \{1, \ldots, n\}\), all \((pk'_j, sk'_j) \leftarrow \text{Gen}(1^\lambda)\), where \(j \in \{1, \ldots, \ell\}\), and for one \(j\) it holds that \((pk'_j, sk'_j) = (pk_i, sk_i)\) for some \(i\), and any messages \(m, m'\), the term
\[
P\left[\text{Link}(m, m', \text{Sign}(m, sk_1, pk_{1,\ldots,n}), \text{Sign}(m', sk_1, pk'_{1,\ldots,\ell})) = 1\right]
\]
is negligible in \(\lambda\). This means that the algorithm \(\text{Link}\) has a negligible probability of marking two linked signatures as independent even if some ring members change.
Signer Ambiguity

Intuitively, signer ambiguity guarantees the anonymity of the real signer of the message among the other members of the ring. An algorithm $E$ without any access to secret keys that tries to guess the real signer has only a negligible advantage over randomly picking one of the members in the ring. However we have to take into account that some secret keys may be leaked and thus the guessing algorithm $E$ could have a proportionally higher chance of success, since it can verify if one of the leaked keys has been used in the ring. If that is the case, then $E$ knows the true signer. Formally, for any ppt algorithm $E$, on input of any message $m$, any lists of $n$ public keys $pk_1, \ldots, pk_n$, any set of $t$ corresponding secret keys $D_t = \{sk_1, \ldots, sk_t\}$, and any valid signature $\sigma \leftarrow \text{Sign}(m, sk_j, pk_1, \ldots, n)$ generated by the user with public key $pk_j$ it holds that

$$P[E(m, pk_1, \ldots, n, D_t, \sigma) = pk_j] = \begin{cases} \frac{1}{n} - \frac{1}{Q(\lambda)} + \frac{1}{n-t} + \frac{1}{Q(\lambda)}, & \text{if } sk_j \notin D_t \text{ and } t < n-1 \\ 1 - \frac{1}{Q(\lambda)}, & \text{otherwise} \end{cases}$$

for any polynomial $Q(\lambda)$.

Regarding unforgeability of the ring signature we are able to use the attacker model of Liu et al. [144].

**Definition 4.1.2** (Existential unforgeability against adaptive chosen-plain-text, adaptive chosen-public-key attackers [144]). Let $SO(m', pk'_1, \ldots, n')$ be a signing oracle that accepts as inputs a message $m'$ and a list of $n$ public keys $pk'_1, \ldots, n'$, and produces a signature $\sigma'$ such that $\text{Verify}(\sigma', m', pk'_1, \ldots, n') = 1$. A linkable ring signature scheme is called existentially unforgeable (against adaptive chosen-plain-text and adaptive chosen public-key attackers) if, for any ppt algorithm $A$ with access to a signing oracle $SO$ such that $(pk_1, \ldots, n, m, \sigma) \leftarrow A^SO(pk_1, \ldots, n)$ for a list $pk_1, \ldots, n = (pk_1, \ldots, pk_n)$ of $n$ public keys chosen by $A$, its output satisfies $\text{Verify}(\sigma', m', pk'_1, \ldots, n') = 1$ only with negligible probability. Note that $(pk_1, \ldots, n, m, \sigma)$ should not correspond to any query-response pair previously given to the signing oracle.

In the context of cryptocurrency systems, ring signatures can be used to provide sender anonymity, as explained in Section 2.2.2. Each transaction input references multiple transaction outputs of earlier transactions where one output is the real output and the others are used to hide the sender and thus increase privacy. The public keys used in the ring signature are those of the earlier outputs, which are referenced in the transaction input. The message which is signed are the transaction outputs of the new transaction. The signature is included in the input, as in other blockchain systems. As a result, to an observer the real spent output is indistinguishable from the other fake outputs.

If an attacker wants to double spend the same funds, it needs to create two transactions with one signature each for the same referenced transaction output. A linkable ring signature allows to link two signatures if the same secret key has been used for signing and thus these transactions can be dropped as invalid by the miners. Consequently double spending is prevented.

CryptoNote [202] and many of its derivatives like Monero or Bytecoin use a variation of the Fujisaki-Suzuki (FS) linkable ring signatures [88]. In contrast,
(pk, sk) ← Gen(1^k)

1. Choose a value \( x \in \{1, \ldots, \ell\} \) uniformly at random, where \( \ell \) is the group order of the elliptic curve \( G \) with generator \( G \). This is the secret key \( sk \).
2. Choose a point \( P \in G \) on the elliptic curve \( G \) with generator \( G \) uniformly at random.
3. Compute the public key \( pk \) as the point \( P = xG \) and return the pair \( (pk, sk) = (P, x) \).

\( \sigma \leftarrow \text{Sign}(m, sk, pk_{1, \ldots, n}) \)

1. Parse the points \( (P_1, \ldots, P_n) = pk_{1, \ldots, n} \).
2. W.l.o.g. assume that the secret key \( x = sk \) corresponds to the \( j \)-th public key \( P_j = pk_j \), i.e., \( pk_j = xG \).
3. Compute the key image as \( I = xH_p(P_j) \), where \( H_p \) is a hash function returning a point on the elliptic curve.
4. Choose \( \alpha \), and \( s_i \) for \( i = 1, \ldots, n \), \( i \neq j \) uniformly at random from the base field of the elliptic curve \( G \).
5. Compute the following values.
   \[
   L_j = \alpha G, \quad R_j = \alpha H_p(P_j), \quad c_{j+1} = H(m, L_j, R_j)
   \]
6. For all \( i \in \mathbb{Z}/n\mathbb{Z}, i \neq j \) compute the following.
   \[
   L_{i+1} = s_{i+1}G + c_{i+1}P_{i+1} \\
   R_{i+1} = s_{i+1}H_p(P_{i+1}) + c_{i+1}I \\
   c_{i+2} = H(m, L_{i+1}, R_{i+1})
   \]
7. Define \( s_j = \alpha - c_jx_j \mod \ell \), where \( \ell \) is the group order of the elliptic curve and close the ring by computing
   \[
   L_j = \alpha G = s_jG + c_jx_jG = s_jG + c_jP_j \\
   R_j = \alpha H_p(P_j) = s_jH_p(P_j) + c_jI \\
   c_{j+1} = H(m, L_j, R_j)
   \]
8. Return the signature \( \sigma = I, s_1, \ldots, s_n \).

\( b := \text{Verify}(\sigma, m, pk_{i\in I}) \)

1. Parse \( \sigma \) as \( (I, c_1, s_1, \ldots, s_n) \) and recompute \( L_i, R_i \) for \( i \in \{1, \ldots, n\} \), as well as \( c_1, \ldots, c_{n+1} \).
2. Return 1 if \( c_{n+1} = c_1 \).

\( \ell := \text{Link}(m, m', \sigma, \sigma') \)

1. Parse \( \sigma \) as \( (I, c_1, s_1, \ldots, s_n) \) and \( \sigma' \) as \( (I', c'_1, s'_1, \ldots, s'_n) \).
2. Return \( \ell = 0 \) if \( I = I' \). Otherwise return \( \ell = 1 \).

Figure 4.1: A variation of the LWW ring signature scheme [17]

our system uses a linkable variation of Liu-Wei-Wong (LWW) signatures [17] which has the advantage of being significantly shorter than FS signatures. In particular, the small variation to the LWW signatures allows for linking signatures even if the other public keys in the ring are different, in contrast to the original LWW signature scheme. We sketch the linkable variation of the LWW scheme in the following. The signature algorithm is shown in Figure 4.1

Let \( G \) be an elliptic curve with generator \( G \). Let \( P_i \in G, i = 1, \ldots, n \) be the
public keys of the members in the ring. Note, that previously these were denoted by \( pk_i \). However, since we are now discussing a concrete implementation of a linkable ring signature based on elliptic curves we use the common notation of denoting points on the curve in capital letters. Assume that for the \( j \)-th public key we know the corresponding private key \( x \) with \( P_j = xG \). Let \( I = xH_p(P_j) \) be the key image, where \( H_p \) is a hash function returning a point on the elliptic curve. The key image is included in the signature to enable linking of the signatures. Knowledge of the key image does not reveal the signer since \( x \) is private.

Let \( m \) be the data to sign and \( \mathcal{H} \) a hash function. Let \( \alpha \), and \( s_i \) for \( i = 1 \ldots, n, i \neq j \), be random values in the base field of the elliptic curve. For generation of the signature the following values need to be computed.

\[
L_j = \alpha G \\
R_j = \alpha H_p(P_j) \\
c_{j+1} = \mathcal{H}(m, L_j, R_j)
\]

For all \( i \in \mathbb{Z}/n\mathbb{Z}, i \neq j \) define \( L_i, R_i, \) and \( c_i \) successively as follows.

\[
L_{i+1} = s_{i+1}G + c_{i+1}P_{i+1} \\
R_{i+1} = s_{i+1}H_p(P_{i+1}) + c_{i+1}I \\
c_{i+2} = \mathcal{H}(m, L_{i+1}, R_{i+1})
\]

The last step is closing the ring by “stitching” the two ends together. Let \( s_j = \alpha - c_jx_j \mod \ell \), where \( \ell \) is the group order of the elliptic curve. Then define

\[
L_j = \alpha G = s_jG + c_jx_jG = s_jG + c_jP_j \\
R_j = \alpha H_p(P_j) = s_jH_p(P_j) + c_jI \\
c_{j+1} = \mathcal{H}(m, L_j, R_j)
\]

The signature consists of \( \sigma = (I, c_1, s_1, \cdots, s_n) \).

To check a signature the verifier recomputes the sequence \( c_1, \ldots, c_{n+1} \) and checks if \( c_{n+1} = c_1 \). If that is the case the signature is valid. Two signatures are linked, i.e., signed by the same party, if they have the same key image \( I \).

In the signature scheme used by CryptoNote [202] based on FS signatures [88] the \( c_i \) are chosen randomly and appended to the signature. The construction explained [17] in contrast uses the hash function as a PRNG, so the \( c_i \) are chosen pseudorandomly and can be reconstructed given \( c_1 \). Thus a signature in CryptoNote consists of \( (I, c_1, \ldots, c_n, s_1, \ldots, s_n) \) and is therefore larger than signatures in the scheme described. The construction by Liu et al. [144] works the same as the one shown, but in their scheme signatures are only linkable if they are created with respect to the same list of public keys, due to a different choice of the key image \( I \).

The proofs of security, linkability and signer ambiguity of the described scheme are simple modifications of the proofs found in the work of Liu et al. [144] and can be found in the work of Noether et al. [171, 172].

When using linkable ring signatures in a blockchain system to obscure the sender of transactions one remaining question is how to choose the other
transactions in the anonymity set. If the sender of a transaction chooses these in a bad way its privacy is likely to suffer. One naive though insecure approach is to choose the other transactions in the anonymity set uniformly at random from all transactions published in the blockchain. This poses a risk, since with this choice older transactions are used in more ring signatures as younger transactions in expectancy. Thus, younger transactions in the anonymity set have a higher probability of being the real spent output. While it is clear that a uniform distribution of transactions for the anonymity set is not optimal, it remains an open question which distribution is more appropriate. A publication of the Monero labs \[152\] deals with this issue in more depth, but also fails to provide an adequate solution. The impact of the choice of the anonymity set on the current Monero system is analyzed by Möser et al. \[166\]. The optimal choice of the other transactions in the ring signature is still an open research question.

In summary, linkable ring signatures enforce sender anonymity, also called untraceability \[202\] of transactions as all identities included in the ring have the same probability of being the real sender of the transaction. We use a variation of LWW signatures to allow for linking signatures even if the other public keys in the ring are different. The original LWW signature scheme does not allow for linking signatures if the other public keys in the ring are different, but only if the signatures are over the same sets of public keys. Further, LWW signatures are comparatively short and thus a good fit for our use case.

### 4.1.2 One-time Payment Addresses

The blockchain-based cryptocurrency construction kit CryptoNote \[202\] introduced so called one-time payment addresses \[173\] which increase privacy of the receiver by using different unlinkable recipient keys. Bitcoin later adopted this mechanism under the name stealth address\[^1\].

Instead of simply referencing the recipient directly by its long-term public key, the sender derives a new temporary public key per transaction output using a random nonce and the recipient’s long-term public key as explained below. The derived one-time public key, called destination key and the original long-term public key of the recipient are unlinkable without knowledge of the long-term private key. The recipient can recover the private key corresponding to the destination key by using his long-term private key and a transaction public key which is included in the transaction by the sender.

By providing a signature with the recovered private key, the recipient can prove that she was in fact the intended recipient. Thus, the recipient can spend the funds without revealing her identity or that the transaction belongs to her (long-term) public key.

More exactly, let $G$ be an elliptic curve with generator $G$. Let $d \in \{1, \ldots, \ell\}$ be the private key of the receiver, where $\ell$ denotes the group order of the elliptic curve. $Q = dG$ is the long-term public key of the recipient.

To send a transaction, the sender generates a one-time recipient address as follows. First, the sender computes $P = eG \in G$, where $e \in \{1, \ldots, \ell\}$ is chosen uniformly at random. The sender defines the shared Diffie-Hellman secret $c :=$
4.1. BUILDING BLOCKS

$H(eQ)$ and sends his transaction to the one-time public key $Q' = Q + cG \in \mathbb{G}$. The transaction needs to include the additional information $P$ which is needed for the recipient to recover the private key corresponding to the public key $Q'$. Since

$$Q' = Q + cG = dG + H(eQ)G = dG + H(edG)G = (d + H(dp))G$$

the recipient can recover the one-time private key $d + H(dp)$ corresponding to the public key $Q'$. This is due to the fact that the recipient knows its private key $d$, and the shared Diffie-Hellman secret $P$ included in the transaction.

The unlinkability of two one-time keys derives from the fact that $cG$ is essentially a random offset which is added to the long-term public key. The security of the scheme in the sense that only the recipient can recover the private key follows from the DL-assumption \[68\].

Unfortunately this scheme does not allow the delegation of the checking of incoming transactions since checking that a transaction belongs to a specific person requires knowledge of the long-term private key. However, with that long-term key, the money can be spent. In order to use the scheme on a low-power device, e.g., a smartphone, it would either have to download the whole blockchain, which is infeasible, or give the private key to a miner which checks the incoming transactions for the smartphone. However this miner would then also be able to spend the funds. Another scenario where delegation of transaction processing is desired is an audit by a third party, like a tax fraud investigation.

In order to allow for the delegation of transaction checking, there is the possibility of having a scan key pair and a spend key pair. The scan key pair is given to the auditor which can then check (but also link) the incoming payments and, e.g., only forward those to the low power device. However for spending the transactions the private spend key is still needed.

Let $d \in \{1, \ldots, \ell\}$, $Q = dG \in \mathbb{G}$ be the private and public scan key of the recipient, and $f \in \mathbb{Z}_\ell, R = fG \in \mathbb{G}$ the private and public spend key of the recipient. Here, $\ell$ is again the group order of the elliptic curve $\mathbb{G}$. To send a transaction the sender chooses $e \in \{1, \ldots, \ell\}$ at random as above and computes $P = eG \in \mathbb{G}$. We define the shared DH secret $c := H(eQ) = H(dp)$ as above. The sender now addresses his transaction to $R' = R + cG \in \mathbb{G}$. Again, $P$ needs to be included as an additional information in the transaction.

In order to decide if the transaction is addressed to the recipient only the private scan key $d$ is needed to check if $R' = R + H(dp)G \in \mathbb{G}$. However, in order to spend the transaction the private spend key $f$ is needed, since the private key corresponding to $R'$ is $(f + c)$, i.e., $R' = (f + c)G$. Thus the checking of incoming transactions can be delegated at the cost of privacy.

To summarize, we use one-time payment addresses as a building block to achieve unlinkable transactions \[202\], in the sense that an observer cannot prove that any two transactions were sent to the same user. Further, transaction processing can be delegated without delegating the authorization to spend transactions.
4.2 Roles

Every participant in the system needs to support the core functionality of being able to process cryptocurrency transactions. Beyond the core functionality the system is designed with the following three advanced roles in mind.

- **User**: A user is a node which joins the network to store and retrieve its files in the system. Users create storage contracts which are a pledge to pay a storage provider if the storage provider keeps a file for a certain amount of time. These storage contracts are broadcasted and eventually included in the blockchain.

- **Storage provider**: A node joining the network with the intention to earn koppercoins by providing storage space to users is called a storage provider. Storage providers are responsible for providing storage space. Additionally, they publish proofs of storage of files they store to prove their compliance with storage contracts.

- **Miner**: A miner earns rewards for supporting the system operation. Miners validate transactions and storage contracts for their correctness and aggregate them in blocks which they append to the blockchain. They are comparable to miners in Bitcoin.

Note that in contrast to the design proposed in the previous chapter, in this chapter the miner and storage provider are different roles. A node can take on a combination of these advanced roles at the same time, e.g., it can provide storage, as well as mine new blocks. These roles correspond to the core functionalities of the system: Mining, transferring koppercoins, and storing and retrieving files.

To participate as a miner in the system, one does not need to own any units of koppercoins. In order to serve as a storage provider or a user, the participant needs to own some currency units in the system which he can gain by mining or by buying it from an exchange. For the user the necessity of possessing koppercoins is obvious, since storing files at storage providers incurs a cost for the user. The storage provider needs to have koppercoins as a security deposit against cheating in our file retrieval process using MAD transactions, as explained in Section 3.1.4.

4.3 Design of our System

A simplified overview of our system is given in Figure 4.2.

Our system basically consists of three main components which will be explained in this section. One component is a mining process which is done by the miners in the system in order to maintain the blockchain. The miners append new blocks to the blockchain and receive compensation in the form of a mining reward. This is described in Section 4.3.1.

As a second component, our system supports anonymous payments by employing linkable ring signatures to support sender anonymity and one-time payment addresses for receiver anonymity. These are explained in Section 4.3.2.

Anonymous payments can be done by any participant in the system.

The third component of our system is the file handling. Users negotiate storage conditions with a storage provider and then publish a storage smart contract.
contract with a particular storage provider in the blockchain. Storage smart contracts are a special type of payment from a user to a storage provider and thus inherit the privacy properties of anonymous payments in our system. The storage provider can only spend the payment contained in a storage smart contract if it proves storage of data of the user. The file handling component is explained in Section 4.3.3.

4.3.1 Mining

Our mining process follows the successful design of the mining system employed in Bitcoin [167] as already explained in Section 2.2.1. Thus, we are using proof of work for mining and not proofs of storage, as in the previous chapter. The only difference between our mining process and the one of Bitcoin is, that the coinbase transactions, i.e., the mining rewards in our system are realized by a one-time payment address, as will be explained in the next section about anonymous payments.

As in Bitcoin, the process of mining fulfills two important functions: Maintenance of a valid distributed ledger by validating transactions and the initial distribution of money in our system, i.e., the creation of money as the system grows. Additionally, the blockchain provides a loose synchronization of time between the participants, such that the notion of an expiry date for storage
4.3.2 Anonymous Payments

In 1992, Okamoto and Ohta identified privacy as one among six properties of an ideal cash system [179]. Privacy in their interpretation means that the privacy of the user should be protected. Especially, the relationship between the user and his purchases must be unlinkable by anyone. In blockchain systems, this is even more important, since the transaction data is public and thus can be analyzed by anyone to allow for deep inferences about the actors [157, 200]. In the cryptocurrency construction kit CryptoNote [202], the requirement of privacy by Okamoto and Ohta, is further separated into untraceability and unlinkability. Untraceability is basically anonymity of the sender, whereas unlinkability is anonymity of the receiver. More exactly, untraceability means that for each incoming transaction there is a set of possible senders and all possible senders have the same probability of being the real sender. Unlinkability means that for any two transactions it is impossible to determine if they were sent to the same recipient. In our system, as in CryptoNote, untraceability is guaranteed by linkable ring signatures, whereas unlinkability is achieved through one-time payment addresses. Consequently, we combine standard mechanisms to achieve unlinkability and untraceability of the payments, as we explain in the following.

To transfer money, a user creates and publishes a transaction, which consists of a set \( \{in_1, \ldots, in_m\} \) of inputs and a set \( \{out_1, \ldots, out_n\} \) of outputs.

An output of a transaction consists of an amount of koppercoins and a one-time payment address of the recipient which can be created by the sender as explained later. The private key corresponding to the one-time address is used later to prove ownership of the output.

Each input \( in_i \) consists of a set of references to previous outputs \( \{out_{i(1)}, \ldots, out_{i(\ell(i))}\} \) together with a linkable ring signature over the outputs \( \{out_1, \ldots, out_n\} \) of the transaction containing this input. To create this ring signature the users’ secret key and all the public keys contained in the referenced outputs are used. This proves that the sender owns at least one of the referenced outputs \( \{out_{1}, \ldots, out_{\ell(i)}\} \). Further, the outputs of the transaction \( \{out_1, \ldots, out_n\} \) cannot be modified without invalidating the signature.

Of course, the amounts of the referenced transaction outputs \( \{out_{i(1)}, \ldots, out_{i(\ell(i))}\} \) need to be equal, since due to the ring signature the real transaction output cannot be distinguished from the other outputs in the anonymity set. Therefore, our system uses standardized denominations of koppercoins. This is similar to banknotes and coins where the set of possible values is fixed, and these values need to be combined to pay different amounts. The granularity of the amounts impacts scalability and privacy. A small set of possible amounts, e.g., only one in an extreme case, provides a large anonymity set, but increases transaction sizes as many outputs are needed to pay sums differing from the units provided. This can be compared to paying only with coins of a single unit. On the other hand, many possible denominations lead to smaller and thus more scalable transactions. But as a drawback, they provide smaller anonymity sets.

---

\(^2\) The other five are independence, security, off-line payment, transferability, and dividability.
since it is harder to find transaction outputs with the same amount suitable for
the anonymity set. A preliminary choice of the denominations is as powers of
two, however, a justified choice of this parameter needs further evaluation and
is still subject to research. The initial anonymity sets are given as the coinbase
transactions, i.e., the rewards to the miners for creating valid blocks.

Algorithm 3 Money Transfer

Input: sender long term secret key $sk_S$, sender long term public key $pk_S$,
recipient long term public key $pk_R$, value to transmit, size $n$ of anonymity
set

Output: a transaction transferring value koppercoins from the owner $S$ to the
recipient $R$

1: $(\text{outputs}_\text{own}, \text{change}) \leftarrow \text{find-own-outputs}(value, sk_S)$
\hspace{1em} \triangleright \text{Find own unspent outputs to represent value and remember change, the}
\hspace{1em} \text{difference between the sum of the values of the outputs and the actual}
\hspace{1em} \text{outputs to transmit.}

2: $ota_R \leftarrow \text{create-ota}(pk_R)$ \hspace{1em} \triangleright \text{Create one-time payment address}

3: $ota_S \leftarrow \text{create-ota}(pk_S)$ \hspace{1em} \triangleright \text{Create two one-time payment addresses: one for the recipient, the other}
\hspace{1em} \text{for sending change.}

4: $\text{targetoutput} \leftarrow \text{create-output}(ota_R, value)$

5: $\text{changeoutput} \leftarrow \text{create-output}(ota_S, change)$
\hspace{1em} \triangleright \text{Create the outputs of the new transaction.}

6: $\text{tospend} \leftarrow \{\text{targetoutput}, \text{changeoutput}\}$

7: $\text{in} \leftarrow \emptyset$

8: for output in $\text{outputs}_\text{own}$ do
\hspace{1em} \triangleright \text{Create the anonymity sets and the according signatures.}

9: $\text{anonymityset} \leftarrow \text{find-outputs(output, n)}$

10: $\text{anonymityset.pk}$s $\leftarrow \text{retrieve-keys(privacy set)}$

11: $\text{signature} \leftarrow \text{ringsign(tospend, \{sk_S, \text{anonymityset.pk}s\})}$

12: $\text{in} \leftarrow \text{in} \cup \{\{\text{output, \text{anonymityset}}, \text{signature}\}$

13: end for

14: return $(\text{in}, \text{tospend})$

The high level steps for sending a payment are given by Algorithm 3. In the
first line, the sender $S$ searches the blockchain for its own unspent transaction
outputs which will be spent in the new transaction. If the sum of amounts of the
unspent transaction outputs do not match the envisioned amount which should
be spent, the value of the change is stored. In Line 2 and 3 one-time addresses
are created as will be explained later to send money to the recipient and return
the change back to $S$. Lines 4 and 5 create outputs with their respective values
for the payment and for the change. In Line 6 an anonymity set of size $n$ is
retrieved from previous transactions in the blockchain. The public keys of
these decoy outputs are extracted since they are needed to create the ring
signature. The coinbase transactions from the mining reward act as an initial
anonymity set in the system. Thus, finding decoy outputs is unlikely to fail,
even if there are only few transactions stored in the blockchain. In Line 11 the
newly created outputs are signed with a ring signature using the keys from the
anonymity set and the users’ secret key. The whole transaction with inputs $in$
and signed outputs \textit{tospend} is returned and broadcast to the network in order
to be included in a block by the miners.

Consequently we achieve k-anonymity of senders, and anonymity of the
receivers in the sense that it cannot be detected if two different transactions
have been sent to the same recipient.

4.3.3 File Handling

In order to allow for storage of data, our system supports storage contracts
between users and storage providers. These contracts are realized as special
transaction types—storage smart contracts—where additional requirements
need to be fulfilled to spend the money, namely the storage of the data. A
storage smart contract locks some money of the user which can be spent by the
storage provider under the condition that the storage provider stores files of
the user.

In order to set up a storage smart contract, a user searches for a storage
provider by broadcasting a storage request as can be seen in Step 1 in Figure 4.3.
This storage request contains metadata like the file size and storage duration.
The storage providers answer with their respective prices they demand for
storing the data for the requested duration (Step 2 in Figure 4.3). The user
then chooses a storage provider, e.g., the cheapest one and sends the file to it
(Step 3a in Figure 4.3). The user creates and signs a storage contract between
itself and the storage provider. This storage contract containing the file identifier
and the agreed storage duration is then broadcast into the network by the user,
so that the miners can include it in the blockchain (Step 3b in Figure 4.3). The
data stored by the storage provider is not published in the blockchain.

The storage smart contract is a payment from the user to the storage provider
which can only be spent by the storage provider, if it is able to prove storage
of the data at the beginning and at expiry of the contract (Step 4a and 4b in
This is realized by a cryptographic mechanism known as proof of storage, as explained in Section 2.3. The proofs of storage are persisted in the blockchain, where they can be verified by the miners. If the storage provider is unable to provide these two necessary proofs of storage, it is assumed that the storage provider has not stored the data and thus is not eligible for payment. In this case the user can create a transaction spending the output of the contract, again using linkable ring signatures and one-time keys, and thus is refunded (Step 5 in Figure 4.3).

All payment rules—remuneration for the storage provider or refunding of the user—are enforced by the miners due to the consensus protocol of the blockchain. That means, if miners generate blocks containing transactions where the storage provider receives its payment despite not proving storage of the file these transactions are rejected by the blockchain as long as there is an honest majority of (hashing power of) miners.

File retrieval is handled with MAD transactions as in our previous design (cf. Section 3.1.4). However, in this design, MAD transactions incorporate linkable ring signatures and one-time payment addresses, and thus offer more privacy as in the last chapter.

In the next section we explain the life cycle of a storage contract and the integration of proofs of storage in smart contracts in more detail.

**Life cycle of Storage Smart Contracts**

To store a file, the user \( U \) searches a storage provider accepting a file of the requested size for the given storage period \( c \). The user creates a contract for the agreed price that can be spent by a one-time key of the storage provider. Next, the user generates a fresh public key pair \((pk, sk)\) and uses it to encode the file according to the proof of storage algorithm used. The encoding is necessary for being able to generate and validate the proofs of storage. The public key \( pk \) and a file identifier \( st \), the latter chosen uniformly at random, are included in the storage contract. The secret key \( sk \) which has been used in the encoding of the file needs to be deleted. Knowledge of the secret key can lead to an adversary creating valid proofs of storage without storing the file.

Afterwards the user publishes the final storage contract containing the public key \( pk \), the file identifier \( st \), a one-time public key \( ref \) of the user \( U \), as well as the storage duration and the payment data, so that it will eventually be persisted in the blockchain. Since the storage contract is publicly verifiable all miners are able to verify it. The storage provider can then see the contract in the blockchain and accept the file transfer accordingly.

The algorithm to store a file is given by Algorithm 4. A full overview of the life cycle of a contract is given in Figure 4.4.

To prevent false acceptance of contracts without accepting the file afterwards, we require the storage provider to create a proof of storage directly after the storage process. Otherwise the storage provider could accept the contract while dropping the file instantly, locking the money of the user without cost for themselves. To provide the proof of storage, the storage provider has to accept the file, with the one-time cost of bandwidth and storage space. Of course, the storage provider can still drop the file after accepting it and creating the proof of storage. However, the cost for this attack is higher than without requiring the first proof of storage.
Algorithm 4 Storing a File

**Input:** User secret key $sk_U$, anonymity set size $n$, file $f$, contract duration $c$, one-time public key $ref$ of the user

**Output:** a storage contract between the user $U$ and an anonymous storage provider, as well as a file transfer

1: $(pk, sk) \leftarrow \text{Gen}(1^k)$  
   \hspace{1em} $\triangleright$ Generate a key pair for the proof of storage

2: $(f', st) \leftarrow \text{Encode}_{sk}(f)$  
   \hspace{1em} $\triangleright$ Encode the file for the proof of storage

3: $(p, pk_P) \leftarrow \text{find-SP(size($f'$), $c$)}$  
   \hspace{1em} $\triangleright$ Find a storage provider and receive a price $p$ and its public key $pk_P$, given the storage period and the size of the file

4: $tx \leftarrow \text{money-transfer}(sk_U, pk_P, p, n)$  
   \hspace{1em} $\triangleright$ See Section 4.3.2

5: $\text{publish}(pk, st, ref, c, tx)$

6: $\text{transfer}(f')$  
   \hspace{1em} $\triangleright$ Transfer the encoded file to the storage provider

---

Figure 4.4: Visualization of the contract life cycle and involved payments over time. The user is on the upper line, whereas the storage provider is shown on the lower line. Starting on the left, a user commits to a payment to the storage provider and transmits its data. The storage provider provides a proof of storage which costs a small fee for the inclusion in the blockchain. This is repeated at contract expiry. To retrieve the data a MAD transaction is executed, as already explained in the previous design in Section 3.1.4.

A second proof of storage is required from the storage provider at contract expiry. This is necessary to show that the storage provider still has access to the data. That means that it must have had stored the data for the whole duration of the contract. The storage provider could not have produced the proof of storage in advance, since the challenge used to create each of the two proofs of storage is derived from the randomness of a block in some appropriately chosen window at that time and thus is unknown in advance. The window is necessary to deal with latency issues of the blockchain.

For the proof at the beginning and the end of the contract we assume a period of goodwill of length $\Delta$, wherein a storage provider needs to provide a proof of storage. The contract is stored in block $B_i$ and contains the storage duration $c$. We denote the start of the period, where the first proof of storage should have happened with $a$, which should be shortly after the contract, but
4.3. DESIGN OF OUR SYSTEM

not instantly, as the file transfer might take some time. This results in one of the blocks \( B_{i+a}, \ldots, B_{i+(a+\Delta)} \) for the first proof of storage. Using the Fiat-Shamir heuristic \[84\] yields a non-interactive proof of storage where the challenger is substituted by a hash function. Consequently, we can use the hash of \( B_{i+a-1} \) as the challenge for the generation of the first proof of storage. The hash of the block \( B_{i+a-1} \) is random and cannot be precomputed since the block itself cannot be precomputed. The Fiat-Shamir heuristic then guarantees security of our construction.

The start of the period, where the second and last proof of storage should have happened is denoted by \( c \) as it is defined by the contract length, resulting in one of the blocks \( B_{i+c}, \ldots, B_{i+(c+\Delta)} \) to contain the second and last proof of storage. The challenge for this proof of storage is the hash of \( B_{i+c-1} \).

The storage provider publishes both proofs of storage as transactions. Such a transaction includes a reference to the storage contract in the blockchain and the publicly verifiable proof. To incentivize miners to include these transactions in blocks the transactions are equipped with a small fee that can be spent by the miner of the block. Fraudulent proofs which cannot be verified are rejected by the miners.

The storage smart contract contains a one-time address \( \text{ref} \) of the user to refund the payment invested by the user in case the contract is broken by the storage provider. To spend the refund, the user proves the breach of the contract by referencing the broken storage contract and specifying if the proof of storage is missing at the beginning \( a \) or the end \( c \) of the contract. The specification by the client reduces validation cost by cutting the number of blocks that need to be checked in half. The miners then check, depending on the point in time specified by the user, \( a \) for example, the blocks \( B_{i+a}, \ldots, B_{i+(a+\Delta)} \) for a proof of storage referencing the contract. If these blocks do not contain a proof, the contract has been broken by the storage provider and the user addressed by \( \text{ref} \) is allowed to use the funds locked in the contract in a new transaction.

To reclaim storage space occupied by proofs of storage, the proofs themselves are not included in the computation of the hash of the transaction but only their hash value. If there are enough blocks on top of the block containing the proof such that it is very likely that no fork will overtake this chain, the proofs can be removed. Since only the hash of the proof is contained in the transaction, removal of proofs does not change the hash of the blocks. This is analogous to pruning spent transaction outputs in Bitcoin as explained in Section \[2.2.1\] only this time with proofs embedded in transactions, opposed to transactions embedded in a blockchain. If a proof of storage is deep inside the blockchain, its existence alone guarantees its correctness. To show this, assume that an incorrect proof of storage is added to the blockchain. Then miners do not continue mining on top of that chain, since they risk losing their mining rewards. Such a chain eventually is overtaken by a chain of honest miners. Thus, incorrect proofs of storage are not deep inside the blockchain if there are more than 50% honest miners in the network. Consequently, the removal of old proofs of storage does not affect the verifiability of storage contracts.

Theoretically, our design supports more complicated clauses for storage contracts than two proofs of storage. It is conceivable to design storage smart contracts where a storage provider needs to prove storage, e.g., at two out of five fixed points of time. However, since our design focusses on privacy, we decided against these more flexible contracts. The more unique the conditions
in the smart contracts, the smaller the size of the anonymity set. With very unique choices of contract clauses different smart contracts could be linked as belonging to the same user. Consequently we decided to use only two points in time where the storage provider needs to prove that it stored the data.

Note that the smart contracts, as well as the proofs of storage with their fee make use of one-time payment addresses and linkable ring signatures in order to provide anonymity of the user and the storage provider.

Additionally to being paid for storage of files, storage providers need to be remunerated for allowing users to retrieve their files. Analogous to our previous design, MAD transactions as already explained in Section 3.1.4 can be used for this purpose. Note that MAD transactions can be trivially enhanced with linkable ring signatures and one-time payment addresses to support anonymity.

4.4 Security and Privacy Analysis

In this section we discuss the privacy and security properties of our system at the abstraction layer of the blockchain. At some parts we require certain properties of the network layer to achieve our privacy goals. We will indicate this where necessary.

4.4.1 Privacy Properties

In the following our attacker is a passive observer with access to all transactions in the blockchain. We organize this section according to the different roles in the system.

In a nutshell, anonymity of senders and receivers in money transfers in provided. Only the amount of the transactions is public. Regarding storage, we hide the user, the storage provider, as well as the file. Of course, the identity of a file is linkable with its corresponding proofs of storage, but the content of the file is never revealed to a passive observer. We assume that the client application takes care of hiding the content of the file from a storage provider, e.g., by applying encryption.

Our system achieves that even two storage contracts with the same user, storage provider and file cannot be linked. Only the amount in the contract is known.

Sender Anonymity

With sender anonymity or untraceability (179, 202) we describe the property that the sender of a transaction is hidden.

Our scheme achieves this on the level of the blockchain, since linkable ring signatures are used when referencing previous transaction outputs. Thus, all identities included in the ring have the same probability of being the sender of the transaction (202). In practice one may choose a ring size between three and five since large ring sizes lead to larger transaction inputs and thus to higher transaction fees. Since the anonymity of a sender depends on the anonymity of other senders in the ring, care must be taken to disallow rings with only one member. Consider the sequence of transactions in Figure 4.5. Transaction $tx_1$ is controlled by $A$ and transaction $tx_2$ is controlled by $B$. $B$ spends $tx_2$ in the transaction $tx_3$ and uses $tx_1$ as member of the ring for the ring signature.
4.4 SECURITY AND PRIVACY ANALYSIS

Figure 4.5: Overview of an Attack on Sender Anonymity. Rectangles denote transactions, arrows denote inputs as members in the ring signature. If A uses the same one-time payment address for tx 4 as for tx 1, it can be deduced that tx 3 must spend tx 2 and thus the sender of tx 3 is B.

The receiver of tx 3 is C. At this point, it is publicly known that tx 3 spends either tx 1 or tx 2. However, A knows that A did not send tx 3. Thus, A can deduce that the originator of tx 3 is B. This allows A to reveal the originator of a transaction and is the reason why we do not allow ring sizes of two. Further, A can now send tx 4 to itself with a ring size of one. Note, that this cannot happen accidentally, since one-time payment addresses hide the fact that the receiver of tx 4 is A. We assume that A is malicious and sends tx 4 to the same one-time payment address as tx 1, since in this case tx 4 is detected as a double-spend. This is the reason why we disallow rings with only one member. If transactions containing ring signatures with one member are considered invalid, deanonymization of other participants as described cannot occur. A statistical analysis of this attack if rings with one member are allowed is given by Noether et al. [174]. Möser et al. [166] investigated the impact of this attack on the Monero system.

In case of a double-spend in our system, both transactions become linked and the second is discarded by the miners. The identity of the sender is not revealed thereby, if the same sets of keys are used. Thus, sender anonymity is achieved. If the sender double-spend with two disjoint sets of other public keys in the rings, an observer learns the one-time key which is used in the double-spend. However, since this is only one-time key and even though sender anonymity is broken, the long-term identity of the sender is not revealed.

To provide anonymity of senders of transactions we require that the network hides the senders of broadcast messages. Clearly, linking a sender of a transaction to the participant broadcasting the transaction in the network violates our privacy requirements.

Receiver Anonymity

The notion of receiver anonymity means that the identity of the receiver of a transaction is hidden. Our scheme provides unlinkable transactions, i.e., an observer cannot decide if any two transactions were sent to the same recipient.
We accomplish this through the use of one-time addresses as explained in Section 4.3.2. If two transactions are sent to the same receiver the senders derive two different unlinkable one-time payment addresses from the long-time key of the receiver \([173, 202]\). Only the receiver is able to check if both transactions belong to him, and recover the private key which is needed to spend the transactions.

User Anonymity

The anonymity of the storage user \(U\) could potentially be violated through his actions on the blockchain, as well as through the file handling of the system. When a user wants to store a file, a storage contract is issued on the blockchain. This contract includes the payment, i.e., references to sets of previous transactions outputs, a public key which is used for the proof of storage, as well as the address of the storage provider \(P\), and details under which conditions \(P\) receives the payment.

As explained above, the public key of the sender of the payment, in this case the user \(U\), is not revealed in a storage contract, because of the use of linkable ring signatures.

The public key used for the proof of storage is different from the other keys of the user and unique per file. It is needed to enable public verifiability of the proof of storage. The corresponding private key is needed only once to run the algorithm \texttt{Encode} of the proof of storage prior to the upload and can then be deleted. Since the key pair is unique per file it cannot be used to identify the user or link different files of the same user.

Unlinkability between the user and his files is a consequence of the anonymity of the user in the storage contract due to the linkable ring signatures. Thus if a user uploads multiple files, an observer cannot decide if they originated from the same user.

If we allow unique conditions on the storage contracts apart from the two necessary proofs of storage as described above, such as five necessary proofs of storage or two out of five proofs, we run into problems with user anonymity. Multiple storage contracts with very unique conditions on the storage provider could be linked as corresponding to the same user. This is the reason why we provide only one standard format for storage contracts.

Since a MAD transaction is just a multi signature transaction and does not contain any references to the file, the privacy of the user is protected through the same mechanisms as in a normal transfer of funds.

To provide our strong privacy guarantees on the network layer, we require a mechanism to hide the sender of messages when uploading the file. When the user downloads the file the network needs to hide the receiver of messages. If the network additionally provides unlinkability of message senders and receivers, users and their storage providers are unlinkable on the network layer.

Storage Provider Anonymity

As with user anonymity, we need to check the storage contracts and the MAD transactions to check the anonymity of the storage provider. Regarding the published proofs of storage it is easy to see that these do not identify the storage
provider, since they only contain a reference to the storage contract in the blockchain.

In a storage contract the identity of the sender is not revealed, since a one-time payment address is used. Our system even achieves unlinkability of two storage contracts with the same storage provider, since with high probability different one-time payment addresses are derived from the long-term public key of the storage provider.

In a MAD transaction the anonymity of the storage provider is also preserved, again since one-time addresses are used and the security deposits are done with ring signatures.

Again we require the network to provide unlinkability of the communicating parties.

File Traceability

With file traceability we mean the identifiability of the file throughout its life cycle.

From the storage contract an observer can learn an identifier for the file, which is later used in the proofs of storage. This is the only time these identifiers are used in the blockchain.

In particular an attacker cannot link a storage contract concerning a specific file and a corresponding MAD transaction, since a MAD transaction contains no reference to the file. Hence, an observer cannot learn how often a specific file is requested in the network.

A passive attacker can neither learn the identity of a user, nor of a storage provider of a specific file, since the identities in the storage contract are hidden. The attacker can only learn the price for the storage contract on which the participants agreed.

When the same file is stored a second time later, a different file identifier is used. Thus, the same file is unlinkable through different uploads. However, when the file is stored at the same storage provider as the first time, then the storage provider can link both uploads. However, we do not consider this to be a privacy risk.

The identifiability of a file in a network when a download request occurs depends heavily on the concrete network employed. If the packets sent in the network are kept confidential and sender and receiver are unlinkable, a passive attacker is not able to trace the file. For a non-global passive attacker a mixing system like Tor \cite{69} could be used.

4.4.2 Security Analysis

This section discusses the security properties of our scheme. We discuss only attackers specific to our scenario and will omit general issues concerning blockchain architectures, such as the necessary conditions for double-spending or assumptions on the attackers to obtain a secure consensus algorithm. For a security discussion of these general problems we refer the reader to previous literature on the subject (see, e.g., \cite{79, 90, 107, 115, 133, 169}).
CHAPTER 4. PRIVACY ENHANCED DESIGN

Denial of Service Attack

The most simple type of attack is a denial of service attack.

Concerning transactions, this means to repeatedly transfer money to other addresses of the same participant. Like in Bitcoin, the attacker needs to pay transaction fees to get its transactions included in the blockchain. This means that such an attack has a financial cost associated with it which provides an upper limit for the attacking capability of the attacker.

Concerning storage contracts the same reasoning holds.

An attacker can also try to include many proofs of storage in the blockchain, thereby increasing the size of the blockchain. This attack is prevented by the miners checking the proofs of storage and only including them in the blockchain if these are valid and required to fulfill a storage contract.

Malicious Storage Providers

Let us take a look at the potential actions of a malicious storage provider.

A malicious storage provider can offer to store a file of the user, but instead modify or delete it. This way the data of the user is lost.

When modifying or deleting the data of the user before the storage contract has expired the storage provider is unable to compute a valid proof of storage anymore, since the challenge is not known in advance. This way, a malicious storage provider loses a potential reward from the storage contract, as well as from potential file retrievals by MAD transactions. The same reasoning can be applied to a storage provider demanding unreasonably high prices for the retrieval of files in the MAD-transaction. If a storage provider asks for a high remuneration to allow the user to retrieve its chunk, the user is unable to agree and consequently does not pay. Thus, the storage provider misses a potential profit.

A solution against the loss of data on the side of the user is to split the file in multiple chunks and apply an erasure code before uploading it to multiple storage providers. Then the data can be recovered if only a sufficiently small fraction of chunks cannot be retrieved. This remedies the problem of storage providers demanding unrealistically high prices for file retrieval, since the user can simply choose to retrieve a different chunk from a different storage provider instead. Though, currently there is no mechanism in our scheme to ensure that the chunks will be stored at different storage providers.

Concluding, a malicious storage provider misses a financial profit by cheating and thus will not cheat under the assumption of rational behavior.

Malicious Users

On the side of the user there is few potential for malicious actions, since the user always has to pay for using storage resources of the network.

As discussed in Section 3.1.4, in the MAD transaction the user cannot take the security deposit of the storage provider hostage. This is due to the fact, that in our system the storage provider, and not the user, sends the second multi signature transaction in a MAD transaction.
4.5 Additional Considerations

In previous sections we discussed many aspects of our system focusing on the abstraction layer of the blockchain such as the transactions and the storage contract handling. However, focusing on this layer is not sufficient, due to possible attacks on other layers [30]. This section discusses additional considerations on other layers for our system to be used in practice. However, these are not the primary focus of this chapter.

4.5.1 Files

A truly private file storage needs to ensure that files can only be read by authorized entities. To reach this goal, a strong file encryption scheme is needed, as in our design any party can initialize a MAD transaction and thus pay for the retrieval of files. Access control through file encryption can be provided by publicly available libraries, e.g., NaCl [32] or dedicated software such as miniLock [4], which also supports sharing of files. Sharing the file between different participants with public keys $pk_1, \ldots, pk_n$ is done by first encrypting the file with a symmetric key $sk$ and then prepending the secret key $sk$ encrypted with the public keys $pk_1, \ldots, pk_n$ to the file. In summary, the file which is uploaded is $\text{enc}_{pk_1}(sk), \ldots, \text{enc}_{pk_n}(sk), \text{enc}_{sk}(f)$, where $f$ denotes the original file and commas denote concatenation. When the owner of $pk_i$ for $i \in \{1, \ldots, n\}$ receives the file, it tries to decrypt all of the prepended encrypted keys until it has recovered $sk$. Then, $sk$ can be used to decrypt the file. Participants which are not in possession of at least one $pk_i$ for $i \in \{1, \ldots, n\}$ are unable to recover the file if a secure encryption mechanism is used.

Our core system is agnostic to such systems but it is not recommended to use our system without such an addition, since otherwise the storage providers, and everyone willing to pay for the download of the file, can read their contents.

4.5.2 Blockchain Transmissions

Interactions with the blockchain, such as creating transactions or disseminating new blocks typically function over a peer-to-peer network. This directly affects the privacy of users of the system: A well connected peer, or a collusion of multiple peers, might log the first appearance of a transaction, allowing them to identify the originating IP address [29]. While the public key of the sender is protected by linkable ring signatures, as described in Section 4.3.2, IP addresses can be considered personally identifiable information and can be used to track activities of users.

To prevent this identification of users, a privacy preserving broadcast mechanism is required. There are multiple solutions available for this problem: Information theoretically secure systems such as Dissent [228] can be adapted, but provide rather slow distribution. Dandelion [37] is inspired by adaptive diffusion [81] and created especially for existing blockchains to provide more privacy for transaction dissemination. However, Dandelion’s privacy guarantees are fairly low. We contributed to the design of a scheme [164, 165] which provides a middle ground between the fast but less private dissemination of...
Dandelion, and the slow, but information theoretically secure message dissemination of Dissent. However, it is not the main focus of our work and instead refer the reader to the original publications [164].

All these three anonymous broadcast systems—Dissent, Dandelion, and our approach—provide different performance and privacy profiles. Consequently, in order to use our system in practice, we need to consider these profiles and implement one anonymous broadcast mechanism to provide true unlinkability. Otherwise the unlinkability of transmissions on the blockchain layer can be circumvented by attacking the network.

4.5.3 File Transmissions

Contracts within the blockchain use the presented privacy mechanisms, but actual file transmission needs to traverse the network. However, direct communication between the participants reveals the sender and receiver and thus violates the privacy of the user and storage provider. Indirect communication might leak the same information to more participants, since the file traverses more hops.

To preserve the anonymity, a mechanism for privacy preserving file transmission is required. There are multiple approaches to solve this problem. One possible solution is to augment contract negotiations with information for a Tor [69] rendezvous point for a hidden service. This creates Tor communication tunnels for both participants and creates sender and receiver anonymity, subject to the limitations of the Tor attacker model.

A more integrated solution to the network could be modeled after decentralized storage solutions without financial incentives, e.g., GNUnet [25] or Freenet [54]. These also provide sender and receiver anonymity and users can additionally use Tor for a defense in depth strategy.

4.5.4 Summary

All considerations for our storage system presented in this chapter have many possible solutions which are already applied in real world scenarios. However, each of these solutions comes with their own advantages and drawbacks that need to be carefully evaluated: On the one hand, privacy needs to be strong enough to support the described privacy mechanisms for anonymous transactions and file handling. On the other hand, enhanced privacy can lead to less usability, as, e.g., the system may become very slow, thus thwarting adoption of the system. Depending on the envisioned use case, these considerations need to be taken into account and balanced against each other.

4.6 Conclusion

This chapter improved our basic design of a distributed storage system with mechanisms to enhance privacy. We augmented the blockchain based transactions by ring signatures to ensure anonymity of the sender, and one-time payment addresses to guarantee anonymity of the receiver. By relying on these primitives our scheme provides public verifiability of transactions and privacy at the same time. As discussed, this design leads to strong security and privacy
guarantees. Further, these mechanisms are of sufficient generality to be reused in different blockchains and thus can be used as privacy patterns.

Additionally, we introduced storage smart contracts, which allow storage providers to be paid only after they cryptographically prove compliance, i.e., that they indeed provided the promised storage. If a storage provider is unable to prove compliance, the user is refunded.

Although our design hides the sender and the receiver of transactions, the amounts of the transactions are public. More current proposals [171, 172] hide even the transaction amounts by introducing homomorphic commitments on the transaction amounts to prove that the sum of the input amounts equals the sum of the output amounts. Additionally, for hiding the amounts in transactions, range proofs are needed to prove that all amounts in a transaction are positive and thus no additional money is created, as explained in Section 2.2.2.

Privacy and security are often perceived as conflicting requirements, but as we have shown in our design, they are not. There was no necessity to achieve a trade-off between privacy and security, but rather between security and privacy on the one hand and usability and scalability on the other hand. As we have shown, privacy and security can well be integrated into a coherent design without compromising on functionality.
A mathematician is a conjurer who gives away his secrets.

John H. Conway

A Comparison of Publicly Verifiable Proofs of Storage

Up to now we used proofs of storage schemes as a black box, but in this chapter we take a deeper look at them. Our previous designs for decentralized storage systems with payment used publicly verifiable proofs of storage to allow the storage provider to prove that it has stored a file without tampering or deleting it. In our first design in Chapter 3 this is necessary for finding blocks in the mining process. In our second design in Chapter 4, proofs of storage are used by storage providers in order to prove that they complied with a storage contract and thus are eligible to spend the money contained in a storage contract.

For both of our designs we have three requirements on the proof of storage:

**Small Size:** It is desirable if the size of the proofs is small, such that they do not burden the blockchain much. Note that in both designs the proofs of storage which have been verified can be pruned from the blockchain if there are sufficiently many blocks on top of them. In this case the proofs of storage can be assumed to be correct and do not need to be stored anymore. Nevertheless, the current proofs of storage need to be distributed to all miners for verification. Thus, small proofs of storage are desirable.

**Public Verifiability:** In our first design the proofs of storage are parts of the blocks in the blockchain, whereas in the second design they are appended as transactions. In both systems however, each miner needs to be able to verify the proofs of storage. Since the participants in our system are dynamic, miners cannot be assumed to have secret knowledge. Consequently we require the proofs of storage to be publicly verifiable in order for the miners to be able to validate them.
Static Data: We require the proofs of storage schemes in our system to be over static data. This means that our proof of storage does not need to account for updates to the data or even version control.

Unlimited number of Verifications: In the first design we require a proof of storage scheme that supports an unlimited number of verifications over the same file. This is required, since the proofs of storage are used for mining, and the same chunk can be challenged multiple times. For our second system, we only need two verifications. One is done at the beginning of the storage smart contract and one is done at the end. However, most proof of storage schemes automatically support an unlimited number of verifications, thus this requirement is very weak.

Ateniese et al. [16] described a transformation in the random oracle model from any homomorphic identification protocol to a publicly verifiable proof of storage protocol where the communication complexity is independent of the file size. In particular, the resulting proof of storage has a constant size for a given security parameter. This transformation yields a proof of storage scheme with the properties we require.

The transformation of Ateniese et al. consists of two parts. First the identification protocol is transformed into a homomorphic linear authenticator in the random oracle model. In a second step, the homomorphic linear authenticator is transformed to a publicly verifiable proof of storage. For this last part, the random oracle model is not needed.

An identification protocol is a three-move-protocol which allows a prover to prove its identity to a verifier. The verifier $V$ in turn is unable to convince someone else that $V$ is the prover [84]. A homomorphic identification protocol additionally allows aggregation of multiple transcripts of different runs of the protocol without sacrificing security. Despite lots of research in identification protocols [84, 103, 177, 204, 212] there has not been much effort in applying these findings to construct publicly verifiable proofs of storage. A modification of the Shoup protocol [212] is used by Ateniese et al. [16] to construct a publicly verifiable proof of storage. There is also a publicly verifiable scheme by Shacham and Waters [207] which was proposed independently of the transformation but can be modified to fit in the framework of Ateniese et al., as we describe later. As a by-product of fitting the Shacham-Waters proof of storage in the framework of Ateniese et al., we obtain a novel unforgeable homomorphic identification protocol which may be of independent interest. To our knowledge these two schemes have been the only published publicly verifiable proofs of storage with constant size previous to our publication.

In order to obtain a new publicly verifiable proof of storage we modify the Guillou-Quisquater protocol [103] to be secure in a stronger attacker model than the original one and apply the transformation by Ateniese et al. on it. Modifying the Guillou-Quisquater protocol is necessary, since the transformation of Ateniese et al. is only proven secure for identification protocols which are secure in the stronger attacker model. The result is a novel secure publicly verifiable proof of storage.

One major shortcoming of the current state of proofs of storage is that practical performance is rarely investigated. There are complexity discussions of privately verifiable proofs of storage by Xu and Chang [231] and Ateniese...
et al. [14], and of publicly verifiable proofs of storage [232]. But these are only asymptotic comparisons which do not show the real-world performance. Practical evaluations of proofs of storage [195, 226] only benchmark one scheme or variations of one scheme and thus are often incomparable due to differences in the hardware and the programming language used. Further, the schemes themselves are often incomparable, since they support different features like dynamic data or confidentiality of data with respect to the external auditor. In line with our requirements above, we implement and compare only publicly verifiable static proofs of storage. In particular, we examine our novel proof of storage scheme, together with the proof of storage based on the Shoup protocol [16], and the scheme by Shacham and Waters [207].

Parts of this chapter have been published at ICICS 2018 [127].

**Contribution** Our contributions are as follows:

- We provide a modification of the Guillou-Quisquater (GQ) identification protocol and prove its unforgeability in a strictly stronger attacker model than the original GQ protocol. This is necessary, since the transformation of Ateniese et al. is only proven secure for identification protocols which are secure in the stronger attacker model. Thus, using the unmodified GQ protocol in the transformation of Ateniese et al. leads to a proof of storage scheme whose security is not proven.

- Based on our modified GQ identification protocol we propose a novel efficient publicly verifiable proof of storage scheme.

- We propose a novel homomorphic identification protocol based on the Diffie-Hellman assumption by reconstruction from a slightly modified version of the proof of storage by Shacham and Waters [207]. Additionally, a proof of security of the identification protocol is given. As a lemma we receive a proof of security of our modifications to the scheme of Shacham and Waters.

- We implement the Shacham-Waters proof of storage [207], the proof of storage instantiated from the Shoup identification protocol as mentioned by Ateniese [16], and our novel proof of storage scheme.

- We provide comprehensive performance measurements for the three schemes we have implemented.

- We describe how to combine sublinear challenging with an erasure code in order to speed up a proof of storage scheme at the cost of retrieval guarantees. This technique was previously known [14] but offered only private retrievability.

- Further we explore the effects of allowing a preprocessing phase in the verification of proofs of storage.
CHAPTER 5. A COMPARISON OF PROOFS OF STORAGE

Roadmap  In the next section we are going to provide the formal definitions of identification protocols and proofs of storage. In Section 5.2 we investigate identification protocols and their corresponding proofs of storage. This section also introduces our novel proof of storage based on the GQ identification protocol. Section 5.3 provides a performance evaluation of the presented schemes. In Section 5.4 we explain how to speed up our proof of storage construction even more at the cost of retrieval guarantees. Future work is discussed in Section 5.5. We conclude this chapter with Section 5.6.

5.1 Preliminaries

In this section we introduce the necessary definitions for homomorphic identification protocols and proofs of storage.

5.1.1 Identification Protocol

The purpose of an identification protocol is that a prover $P$ who is in possession of a secret key $sk$ can prove its identity to a verifier $V$ who knows the corresponding public key $pk$. The security property of an identification protocol is that a verifier $V$ is not able to prove to someone else that $V$ is the prover $P$, i.e., that $V$ knows the private key $sk$.

\[
\begin{array}{c}
\mathcal{P} \quad V \\
(pk, sk) \leftarrow \text{Setup}(1^k) \\
\alpha \leftarrow \text{Commit}(pk; r) \\
\text{Choose } \beta \text{ uniformly at random} \\
\gamma \leftarrow \text{Response}(pk, sk, r, \beta) \\
\gamma \leftarrow \text{Verify}(pk, \alpha, \beta, \gamma) \\
b := \text{Verify}(pk, \alpha, \beta, \gamma)
\end{array}
\]

Table 5.1: Usual Sequence of Messages in an Identification Protocol

An overview of an identification protocol is given in Table 5.1. The prover generates the first message $\alpha$ using his public key $pk$ and a random value $r$. The verifier $V$ chooses a random challenge $\beta$ in the challenge space of the identification protocol and gives it to $\mathcal{P}$. The prover $\mathcal{P}$ computes a response $\gamma$ by using the challenge $\beta$ together with the private key $sk$ and the randomness $r$ used in generating the first message $\alpha$. To verify the interaction, $V$ needs to know the transcript $\alpha, \beta, \gamma$, as well as the public key $pk$. We repeat the formal definitions of a homomorphic identification protocol by Ateniese et al. [16] in Definitions 5.1.1 and 5.1.2.

Definition 5.1.1 (Identification Protocol [16 Definition 3]). An identification protocol is a three-move-protocol between a ppt prover $\mathcal{P}$ and a ppt verifier $V$. 

5.1. PRELIMINARIES

Such a protocol consists of four polynomial-time algorithms (Setup, Commit, Response, Verify) such that:

1. \((pk, sk) \leftarrow \text{Setup}(1^k)\) is a probabilistic algorithm that takes as input the security parameter \(k\) and outputs a public and private key pair \((pk, sk)\).

2. \(\alpha \leftarrow \text{Commit}(pk; r)\) is a probabilistic algorithm run by the prover \(P\) to generate the first message. It takes as input the public key and random coins \(r\), and outputs an initial message \(\alpha\). We stress that there is no need for the secret key \(sk\).

3. \(\gamma \leftarrow \text{Response}(pk, sk, r, \beta)\) is a probabilistic algorithm that is run by the prover \(P\) to generate the third message. It takes as input the public key \(pk\), the secret key \(sk\), a random string \(r\), and a challenge \(\beta\) from some associated challenge space, and outputs a response \(\gamma\).

4. \(b := \text{Verify}(pk, \alpha, \beta, \gamma)\) is a deterministic algorithm run by the verifier \(V\) to decide whether to accept the interaction. It takes as input the public key \(pk\), an initial message \(\alpha\), a challenge \(\beta\), and a response \(\gamma\). It outputs a single bit \(b\), where ‘1’ indicates acceptance and ‘0’ indicates rejection.

We call an identification protocol correct if for all \(k \in \mathbb{N}\), all \((pk, sk)\) output by \(\text{Setup}(1^k)\), all random coins \(r\), and all \(\beta\) in the appropriate challenge space, it holds that

\[
\text{Verify}(pk, \text{Commit}(pk; r), \beta, \text{Response}(pk, sk, r, \beta)) = 1.
\]

For applying the transformation by Ateniese et al. on a proof of storage \([16]\) we need the identification protocol to be homomorphic. That means that transcripts of several runs of the identification protocol can be aggregated. A verifier can then verify the aggregated transcripts. This amounts to batch verification of the transcripts of different runs of the protocol without sacrificing security.

Definition 5.1.2 (Homomorphic Identification Protocol \([16, \text{Definition 4}]\)). An identification protocol \(\Sigma = (\text{Setup}, \text{Commit}, \text{Response}, \text{Verify})\) is homomorphic if efficient functions \(\text{Combine1}, \text{Combine3}\) exist such that:

Completeness For all \((pk, sk)\) output by \(\text{Setup}(1^k)\) and all coefficient vectors \(s \in \mathbb{Z}_{2^n}\), if transcripts \(\{(\alpha_i, \beta_i, \gamma_i)\}_{1 \leq i \leq n}\) are such that \(\text{Verify}(pk, \alpha_i, \beta_i, \gamma_i) = 1\) for all \(i\), then:

\[
\text{Verify}(pk, \text{Combine1}(s, \alpha), \sum_{i=1}^{n} s_i \beta_i, \text{Combine3}(s, \gamma)) = 1
\]

Unforgeability We define unforgeability by defining success of an attacker. If the success probability of the attacker is negligible, we call the homomorphic identification protocol unforgeable. Consider the following experiment involving a challenger and an adversary \(\mathcal{A}\):

1. The challenger computes \((pk, sk) \leftarrow \text{Setup}(1^k)\) and gives \(pk\) to \(\mathcal{A}\).
2. The following is repeated a polynomial number of times:
• \( \mathcal{A} \) outputs \( \beta' \) in the challenge space. The challenger chooses a random \( r \), computes \( \gamma \leftarrow \text{Response}(pk, sk, r, \beta') \) and gives \((r, \gamma)\) to \( \mathcal{A} \).

3. The adversary outputs an \( n \)-vector of challenges \( \beta \). Then for each \( i \) the challenger chooses \( r_i \) at random, sets \( \alpha_i \leftarrow \text{Commit}(pk; r_i) \) and \( \gamma_i \leftarrow \text{Response}(pk, sk, r_i, \beta_i) \), and gives \((r_i, \gamma_i)\) to \( A \).

4. \( A \) outputs a triple \( (s, \mu', \gamma') \), where \( s \in \mathbb{Z}_{n^2}^k \). The adversary succeeds if (1) \( \mu' \neq \sum_i s_i \beta_i \) and (2) \( \text{Verify}(pk, \text{Combine}_1(s, \alpha), \mu', \gamma') = 1 \).

Note that although the second and third step look the same, they are not. The second step is the preprocessing phase, where the attacker is allowed to try polynomially many \( \beta' \) and receive responses from the challenger. Intuitively, this gives the attacker the chance to learn something about the internal state of the challenger. In the third phase the attacker chooses the \( n \)-vector of challenges \( \beta \) for which it has to forge the proof of storage in the fourth step.

**Remark 5.1.1.** The attacker has access to the randomness \( r \) underlying the messages, since this is given by the challenger. Thus, the attacker has access to the messages of the identification protocol itself since he can reconstruct \( \alpha = \text{Commit}(pk; r) \). This is a strictly stronger attacker model compared to an attacker who has access only to the messages and not to the underlying randomness.

As an example, the well-known Schnorr identification protocol [204] as shown in Figure 5.1 is insecure in this attacker model but secure if the attacker has only access to the messages. In the Schnorr protocol the commit message consists of \( g^r \) and thus the attacker is unable to know \( r \) only from the commit due to the hardness of computing the discrete logarithm. Knowledge of the underlying randomness \( r \) however, allows the attacker to reconstruct the secret key \( sk \) using subsequent messages, as \( sk = (\gamma - r) \cdot \beta^{-1} \in \mathbb{Z}/p\mathbb{Z} \). Thus, the Schnorr identification protocol is insecure in the stronger attacker model of Ateniese et al.

Since the transformation of Ateniese et al. from an identification protocol to a proof of storage scheme is only secure in the stronger attacker model, the Schnorr identification protocol cannot be used to construct a secure proof of storage scheme. More exactly, when using an identification protocol in a proof of storage the randomness is chosen as a hash of the file identifier and an index and thus is public. That is the reason why the stricter security model is needed for the transformation to be secure. When using the Schnorr identification protocol in a proof of storage scheme it is possible to reconstruct the secret key, similar to our computation above. The same holds true for the Okamoto identification protocol [177]. It is difficult to find homomorphic identification protocols that satisfy this strictly stronger notion of unforgeability and consequently can be transformed to proof of storage schemes.

### 5.1.2 Proof of Storage

Recall from Section 2.3 that a proof of storage is a mechanism used by a prover \( P \) to convince a verifier \( V \) that \( P \) stores a specific file \( f \). We will now refine this definition. In particular in this chapter we will not use the term proof of
5.1. PRELIMINARIES

- \((pk, sk) \leftarrow \text{Setup}(1^k)\)
  1. Choose a prime \(p \in \mathbb{P}\) big enough and \(g\) as a generator of \(\mathbb{Z}/p\mathbb{Z}^*\).
  2. Choose the secret key \(sk \in \mathbb{R}\{1, \ldots, p\}\) uniformly at random and define
     the public key as \(pk = g^{sk}\).
  3. Return \((pk, sk)\).

- \(\alpha \leftarrow \text{Commit}(pk; r)\)
  1. Return \(g^r \in \mathbb{Z}/p\mathbb{Z}^*\).

- \(\gamma \leftarrow \text{Response}(pk, sk, r, \beta)\)
  1. Return \(r + \beta \cdot sk \in \mathbb{Z}/p\mathbb{Z}^*\).

- \(b := \text{Verify}(pk, \alpha, \beta, \gamma)\)
  1. Check if \(g^\gamma = \alpha pk^\beta \in \mathbb{Z}/p\mathbb{Z}\).

- \(\text{Combine}_1(s, \alpha) := \prod_i \alpha_i^{s_i}\)
- \(\text{Combine}_3(s, \gamma) := \sum_i s_i \cdot \gamma_i\)

\(\text{Figure 5.1: Identification protocol due to Schnorr}\)

storage as an umbrella term as in the rest of this thesis, but rather for proofs of storage with an efficient knowledge extractor as defined below.

Since we consider only publicly verifiable proofs of storage, the verifier \(V\) and the user \(U\) who is uploading the file \(f\) may be distinct. First, \(U\) encodes its file using its private key \(sk\) and sends the encoded file \(f'\) together with a state \(st\) which serves as an identifier for the file to the prover or storage provider \(P\). As soon as the verifier \(V\) wants to know if \(P\) stores the file, \(V\) generates a random challenge \(c\) and sends it to \(P\). \(P\) generates a proof \(\pi\) by using the encoded file \(f'\) as well as the challenge \(c\). To verify the proof the public key \(pk\), the identifier of the file \(st\), the challenge \(c\), and the proof \(\pi\) itself is needed. Since the private key is not needed for the verification, the proof is called publicly verifiable.

**Definition 5.1.3 (Proof of Storage [16, Definition 5]).** A publicly-verifiable **proof of storage** is a tuple of four ppt algorithms \((\text{Gen}, \text{Encode}, \text{Prove}, \text{Verify})\) such that:

1. \((pk, sk) \leftarrow \text{Gen}(1^k)\) is a probabilistic algorithm that is run by the user \(U\) to set up the scheme. It takes as input a security parameter \(k\), and outputs a public and private key pair \((pk, sk)\). We assume that \(pk\) implicitly defines a positive integer \(B\) which determines the size of the file chunks.

2. \((f', st) \leftarrow \text{Encode}_{sk}(f)\) is a probabilistic algorithm that is run by the user in order to encode the file. It takes as input the secret key \(sk\), and a file \(f \in (\mathbb{Z}/B\mathbb{Z})^\ell\) viewed as an \(\ell\)-dimensional vector of chunks with size \(B\). It outputs an encoded file \(f'\) and state information \(st\).

3. \(\pi := \text{Prove}(pk, f', c)\) is a deterministic algorithm that takes as input the public key \(pk\), an encoded file \(f'\), and a challenge \(c \in \{0, 1\}^k\) viewed as a
bit string. It outputs a proof $\pi$.

4. $b := \text{Verify}(pk, st, c, \pi)$ is a deterministic algorithm that takes as input the public key $pk$, the state $st$, a challenge $c \in \{0, 1\}^k$, and a proof $\pi$. It outputs a bit, where '1' indicates acceptance and '0' indicates rejection.

We require that for all $k \in \mathbb{N}$, all $(pk, sk)$ output by $\text{Gen}(1^k)$, all $f \in (\mathbb{Z}/B\mathbb{Z})^\ell$, all $(f', st)$ output by $\text{Encode}_{sk}(f)$, and all $c \in \{0, 1\}^k$, it holds that

$$\text{Verify}(pk, st, c, \text{Prove}(pk, f', c)) = 1.$$  

By using the transformation of Ateniese et al. [16] which transforms any homomorphic identification protocol to a proof of storage we can describe the intuition behind the algorithms more clearly. In a first step the user generates a public and secret key pair. In $\text{Encode}_{sk}(f)$ the user $U$ splits the file into $\ell$ chunks, and signs each of them with a homomorphic signature scheme generated from the homomorphic identification protocol. As before, will refer to these signatures as authenticators. The encoded file $f'$ consists of the original file $f$ and the authenticators. In the algorithm $\text{Prove}$ the challenge $c$ is expanded to a vector $\text{chal}$ of dimension $\ell$ by setting $\text{chal}_i = H_c(i)$, where $H_c$ is a pseudorandom function keyed with the challenge $c$. Each coefficient in the vector $\text{chal}$ corresponds to a chunk of the file. The proof consists of a linear combination $\tau$ of the chunks and a linear combination $\mu$ of the authenticators where the coefficients are from the expanded challenge $\text{chal}$. This is the reason that the size of the proof is constant and independent of the size of the file, since $\tau$ and $\mu$ are elements in finite groups. Validation is done by checking if the authenticators validate. This is straightforward since the authenticators are homomorphically aggregated signatures. Moreover, it needs to be checked if the same linear combination of chunks $\text{chal}$ is used. This is done by regenerating the expanded challenge from $c$ in the validation step. The state $st$ guarantees that the linear combination is computed over the correct file.

After we have defined a proof of storage scheme and its correctness property we now turn to its security property. We use the security definition by Ateniese et al. [16]. There, soundness is formalized using a knowledge extractor [21,83] as in [13,208]. In particular our definition uses the paradigm of “witness extended emulation” [142].

**Definition 5.1.4** (Security of a publicly verifiable proof of storage [16 Definition 6]). Let $\Pi = (\text{Gen, Encode, Prove, Verify})$ be a publicly verifiable proof of storage. We say that $\Pi$ is secure if there is an expected polynomial time knowledge extractor $K$ such that for any ppt adversary $A$ we have that:

1. The distributions

$$
\left\{ \begin{array}{l}
(pk, sk) \leftarrow \text{Gen}(1^k); (f, st, A) \leftarrow A^{\text{Encode}_{st}(\cdot)}(pk); \\
(f', st) \leftarrow \text{Encode}_{st}(f); c \in_R \{0, 1\}^k : \\
(c, A(st, f', st, c))
\end{array} \right.
\right\}
$$


1In the original definition by Ateniese et al. [16] Definition 5) the challenge $c$ is chosen from $\mathbb{Z}/p\mathbb{Z}^\ell$ though in the remainder of their paper $c$ is chosen as an element in $\{0, 1\}^k$. This is an important difference since in the first case the size of the challenge depends on the size of the file $\ell$. We always choose $c \in \{0, 1\}^k$. 

and

\[
\begin{aligned}
(pk, sk) &\gets \text{Gen}(1^k); 
(f, st_A) &\gets A^{\text{Encode}_A}(); (pk); \\
(f', st) &\gets \text{Encode}_{sk}(f); \\
A^{\text{K}_1} &\left(st_A, f', st, ·\right)(pk, st)
\end{aligned}
\]

are identical. \(A^{\text{K}_1}\) denotes the first output of \(A\).

2. The following is negligible.

\[
P
\begin{aligned}
(pk, sk) &\gets \text{Gen}(1^k); 
(f, st_A) &\gets A^{\text{Encode}_A}(); (pk); \\
(f', st) &\gets \text{Encode}_{sk}(f); \\
((c, \pi), f^*) &\gets \text{K}_1^{A(st_A, f', st, ·)}(pk, st); \\
\text{Verify}(pk, st, c, \pi) &\equiv 1 \land f^* \neq f
\end{aligned}
\]

Informally, witness extended emulation means that given an adversary that produces a valid proof with some probability, there exist an emulator that produces a similar proof with the same probability and at the same time provides a witness, i.e., the file \(f'\).

The first property is actually a restriction on the output of the knowledge extractor and guarantees that the knowledge extractor produces the same proofs as the adversary with the same probability. The second property guarantees that the knowledge extractor can provide the witness if the proof is correct, i.e., if the proof is accepted by the verifier then the prover indeed has access to the original file.

Remark 5.1.2. The proof of storage resulting from the original transformation by Ateniese et al. \cite{16} requires that \(\ell\), the number of chunks in the file is public, since in the verification step the challenge \(c\) is expanded to coefficients for the file chunks \(\text{chal}_i = \mathcal{H}_c(i)\) for \(i = 1, \ldots, \ell\). In practice, public knowledge of the length of the file is often undesired. We can remedy this by including the number of chunks \(\ell\) in the state \(st\) as \(st' = (st, \ell)\) and use \(st'\) instead of \(st\) as the key for the pseudorandom function used for computing the challenges for the authenticators. This way the verifier knows the number of chunks from \(st\) and is able to verify the proof without further knowledge. By including \(\ell\) in the state \(st\), the length \(\ell\) cannot be modified since otherwise the authenticators on the chunks could not be verified.

A different approach to remedy that the number of chunks \(\ell\) of the file needs to be public is to augment the proof of storage with a signature scheme. With this modification the algorithm \(\text{Gen}(1^k)\) additionally outputs a signing key pair. In the encoding step the number of chunks is signed by the user and included in the information needed to verify the proof of storage. In the verification step the signature is checked, in addition to the verification of the proof. If the signature does not validate it is rejected, since the information of the number of chunks \(\ell\) is not correct. Thus, tampering with the file size is prevented. This technique is used in the publicly verifiable proof of storage by Shacham and Waters \cite{207} as described later..
• \((pk, sk) \leftarrow \text{Setup}(1^k)\)
  1. Choose primes \(p, q \in \mathbb{P}\) big enough. Define \(n = pq\) and choose \(e\) and \(d\) s. t. \(ed = 1 \mod \varphi(n)\), where \(\varphi\) is Euler’s totient function.
  2. Choose \(J \in \mathbb{Z}/n\mathbb{Z}\) at random. The public key is \(pk = (J, e)\). The secret key is \(sk = J^{-d}\).
  3. Return \((pk, sk)\).

• \(\alpha \leftarrow \text{Commit}(pk; r)\)
  1. Parse \(pk\) as \((J, e)\).
  2. Return \(r^e \in \mathbb{Z}/n\mathbb{Z}\).

• \(\gamma \leftarrow \text{Response}(pk, sk, r, \beta)\)
  1. Return \(r \cdot sk^\beta \in \mathbb{Z}/n\mathbb{Z}\).

• \(b := \text{Verify}(pk, \alpha, \beta, \gamma)\)
  1. Parse \(pk\) as \((J, e)\).
  2. Check if \(\alpha = J^\beta \cdot \gamma^e\).

• \(\text{Combine}_1(s, \alpha) := \prod_i \alpha_i^{s_i}\)
• \(\text{Combine}_3(s, \gamma) := \prod_i \gamma_i^{s_i}\)

Figure 5.2: Identification protocol due to Guillou-Quisquater

5.2 Proofs of Storage

In this section we describe three proof of storage schemes: (i) our novel Guillou-Quisquater-based proof of storage, (ii) the Shoup proof of storage, and (iii) the Shacham-Waters proof of storage.

First, we modify the Guillou-Quisquater identification protocol to be secure in the attacker model defined above such that it can be used to construct a novel proof of storage. Next, we describe a proof of storage based on the Shoup protocol. Its existence was stated by Ateniese et al. [16] but the scheme was not explicitly described. We present this scheme to which we will refer as Shoup proof of storage in Section 5.2.2. The third and last scheme we explain is the scheme of Shacham and Waters [207] along with some modifications. These modifications enable a better comparison and allow us to reconstruct the underlying identification protocol which to our knowledge has not been described yet in literature and might be of independent interest.

5.2.1 Proof of Storage from Guillou-Quisquater

In 1988, Guillou and Quisquater (GQ) described an identification protocol based on the RSA problem [103] as shown in Figure 5.2.

This identification protocol is proven secure under an attacker that has access to the messages but not to the underlying randomness \(r\). In our attacker model from Definition 5.1.2 the protocol is clearly insecure as we describe below.
5.2. PROOFS OF STORAGE

Thus, the original GQ identification protocol cannot be used to construct a secure proof of storage using the transformation of Ateniese et al.

Remark 5.2.1 (Why the GQ protocol is insecure in the attacker model of Ateniese et al.). In the second step of the unforgeability game of Definition 5.1.2 the attacker can choose two $\beta$, $\beta'$ which are coprime. Due to Bézout’s lemma there are $s$ and $t$, such that $s\beta + t\beta' = 1$ which can be found trivially by the extended Euclidean algorithm. The challenger then chooses random $r$, $r'$ and computes $\gamma \leftarrow \text{Response}(pk, sk, r, \beta) = r \cdot sk^\beta$ and $\gamma' \leftarrow \text{Response}(pk, sk, r', \beta') = r' \cdot sk'^\beta$.

The attacker $A$ receives $(r, \gamma = r \cdot sk^\beta)$ and $(r', \gamma' = r' \cdot sk'^\beta)$ from the challenger. Dividing by $r$ and $r'$ respectively yields $sk^\beta$ and $sk'^\beta$ from which the attacker can compute the secret key $sk = (sk^\beta)^s \cdot (sk'^\beta)^t$ using $s$ and $t$ from Bézout’s identity.

In the third phase of the game the attacker can choose any $n$-vector of challenges $\beta$ for the challenger. The challenger computes $\alpha_i \leftarrow \text{Commit}(pk; r) = r^e \in \mathbb{Z}/n\mathbb{Z}$ and $\gamma_i \leftarrow \text{Response}(pk, sk, r, \beta) = r \cdot sk^\beta \in \mathbb{Z}/n\mathbb{Z}$ and sends $(r, \gamma)$ to the attacker.

In the fourth phase of the unforgeability game, to succeed in the attack the attacker can choose $\mu'$ and $s$ arbitrarily and compute $\gamma' = r_i \cdot sk_i^{\beta}$ since he knows the secret key $sk$.

We have modified the original GQ protocol to be secure in our attacker model. The main idea is to substitute the random $r$ from the original protocol by $r^e$. Hence, the commit in our modified scheme is only the randomness $r$ instead of $r^e \in \mathbb{Z}/n\mathbb{Z}$ as in the original protocol. The resulting scheme is shown in Figure 5.3. Its proof of security is given below.

Theorem 5.2.1. Our modified GQ identification protocol as described in Figure 5.3 is an unforgeable homomorphic identification protocol under the RSA-assumption.

Proof. Completeness is clear. To prove unforgeability we construct an algorithm $B$ that can solve the RSA problem given access to an attacker $A$. Let $n = pq$ be an RSA modulus with unknown primes $p$ and $q$.

- $B$ is given the composite $n$, as well as an integer $J \in \mathbb{Z}/n\mathbb{Z}$ and $e \in \mathbb{Z}$ which is coprime to $n$. We construct $B$ to return $sk^{-1} = J^e$, where $de = 1 \mod \varphi(n)$, i.e., the $e$-th root of $J$. $B$ gives the public key $pk = (J, e)$ as an input to the algorithm $A$.

- Whenever $A$ outputs $\beta'$ in the challenge space, $B$ chooses a random $\gamma \in \mathbb{Z}/n\mathbb{Z}$ and sets $r = J^{e^c} \gamma^e$. The algorithm $B$ then invokes $A$ on the input $(r, \gamma)$.

- When algorithm $A$ outputs an $n$-vector of challenges $\beta$, for each $i$ the algorithm $B$ computes $(r, \gamma)$ as in the previous step, sets $\alpha_i \leftarrow \text{Commit}(pk; r) = r$ and gives the vectors $(r, \gamma)$ to the algorithm $A$.

- If $A$ outputs $(s, \mu', \gamma')$ where $s \in \mathbb{Z}_{2^n}$, and
  1. $\mu' \neq \sum_i s_i \beta_i$, and
  2. Verify($pk$, Combine1($s, \alpha$), $\mu', \gamma') = 1$
• \((pk, sk) \leftarrow \text{Setup}(1^k)\)
  1. Choose primes \(p, q \in \mathbb{P}\) big enough. Define \(n = pq\) and choose \(e\) and \(d\) s.t. \(ed = 1 \mod \varphi(n)\), where \(\varphi\) is Euler’s totient function.
  2. Choose \(J \in \mathbb{Z}/n\mathbb{Z}\) at random. The public key is \(pk = (J, e)\). The secret key is \(sk = (J^{-d}, d)\).
  3. Return \((pk, sk)\).

• \(\alpha \leftarrow \text{Commit}(pk; r)\)
  1. Return \(r \in \mathbb{Z}/n\mathbb{Z}\).

• \(\gamma \leftarrow \text{Response}(pk, sk, r, \beta)\)
  1. Parse \(sk\) as \((B, d)\).
  2. Return \(r^d \cdot B^\beta \in \mathbb{Z}/n\mathbb{Z}\).

• \(b := \text{Verify}(pk, \alpha, \beta, \gamma)\)
  1. Parse \(pk\) as \((J, e)\).
  2. Check if \(\alpha = J^\beta \cdot \gamma^e\).

Combine1\((s, \alpha) := \prod_i \alpha_i^{s_i}\)

Combine3\((s, \gamma) := \prod_i \gamma_i^{s_i}\)

Figure 5.3: Our modified Guillou-Quisquater identification protocol

we compute the \(e\)-th root of \(J\), namely \(sk^{-1}\), as follows:

For ease of notation define \(\alpha^* := \text{Combine1}(s, r)\), \(\gamma^* := \text{Combine3}(s, \gamma)\), and \(\mu^* := \sum_i s_i \beta_i\). We know that

\[
J^{\mu^*} (\gamma^*)^e = \alpha^* = J^{\mu'} (\gamma')^e.
\]

where the first equality follows from completeness and the second equality follows from the condition on the output of \(A\). Thus

\[
J^{\mu^* - \mu'} (\gamma^*/\gamma')^e = 1.
\]

Since \(\gcd(e, \mu^* - \mu') = 1\) by the choice of \(e\), using the Euclidean algorithm we can find coefficients \(s\) and \(t\) such that \(s \cdot e + t \cdot (\mu^* - \mu') = 1\). The \(e\)-th root of \(J\) is now \((\gamma'/\gamma^*)^t \cdot J^s\), since

\[
((\gamma'/\gamma^*)^t \cdot J^s)^e = (\gamma'/\gamma^*)^{te} \cdot J^{se}
= (\gamma'/\gamma^*)^{te} \cdot J^{1-t(\mu^* - \mu')}
= J \left( (\gamma'/\gamma^*)^e \cdot J^{-t(\mu^* - \mu')^t} \right)
= J.
\]

Since \(B\) solves the RSA problem if \(A\) succeeds we conclude that the success probability of \(A\) is negligible. \(\square\)
5.2. PROOFS OF STORAGE

\[ (pk, sk) \leftarrow \text{Gen}(1^k) \]

1. Choose primes \( p, q \in \mathbb{P} \) big enough. Define \( n = pq \) and choose \( e \) and \( d \) s. t. \( ed \equiv 1 \pmod{\varphi(n)} \), where \( \varphi \) is Euler’s totient function.
2. Choose \( J \in \mathbb{Z}/n\mathbb{Z} \) at random. The public key is \( pk = (J, e) \). The secret key is \( sk = (J^{-d}, d) \).
3. Return \((pk, sk)\).

\[ (f', st) \leftarrow \text{Encode}_{sk}(f) \]

1. Parse \( f \) as \( f_1, \ldots, f_\ell \) with \( f_i \in \mathbb{Z}/n\mathbb{Z} \) for all \( i = 1, \ldots, \ell \).
2. Parse \( sk\) as \((J^{-d}, d)\).
3. Define the state \( st = (s, \ell) \), where \( s \in \mathbb{R}\{0,1\}^k \) is chosen at random.
4. Compute the authenticators for each chunk as \( \sigma_i = \text{H}_{st}(i)^d \cdot (J^{-d})^{f_i} \in \mathbb{Z}/n\mathbb{Z} \) for \( i = 1, \ldots, \ell \), where \( \text{H}_{st}(i) \) is a pseudorandom function keyed with key \( st \).
5. Return \( f' = (f, \sigma) \) and \( st \).

\[ \pi := \text{Prove}(pk, f', c) \]

1. Expand the challenge \( c \) as \( \text{chal}_i = \text{H}_{c}(i) \) for \( i = 1, \ldots, \ell \).
2. Parse \( f' \) as \((f, \sigma)\) and \( f \) as \( f_1, \ldots, f_\ell \).
3. Aggregate the authenticators as \( \tau = \prod_{i=1}^{\ell} \sigma_i^{\text{chal}_i} \).
4. Aggregate the chunks as \( \mu = \sum_{i=1}^{\ell} f_i \cdot \text{chal}_i \).
5. Return \( \pi = (\tau, \mu) \).

\[ b := \text{Verify}(pk, st, c, \pi) \]

1. Recover the number of chunks \( \ell \) in the file from \( st \).
2. Parse the public key \( pk \) as \((J, e)\).
3. Parse \( \pi \) as \((\tau, \mu)\).
4. Check if \( J^\mu \cdot \tau^e = \prod_{i=1}^{\ell} \text{H}_{st}(i)^{\text{H}_{c}(i)} \).

Figure 5.4: Our GQ Proof of Storage

With the modified GQ identification protocol we are now able to construct a proof of storage using the transformation of Ateniese et al. [16]. The resulting scheme is depicted in Figure 5.4.

Lemma 5.2.1. The GQ proof of storage shown in Figure 5.4 is secure under the RSA-assumption if \( \text{H} \) is modelled as a random oracle.

Proof. Follows directly from the security of the transformation of a homomorphic identification protocol to a proof of storage (Theorem 1 and 2 in [16]) and the security of the modified GQ scheme in Theorem 5.2.1.

---

Actually, only \( \text{H}_{st}(i) \) needs to be modelled as a random oracle due to internal workings of the transformation. For \( \text{H}_{c}(i) \) it suffices to be a pseudorandom function.
CHAPTER 5. A COMPARISON OF PROOFS OF STORAGE

• \((pk, sk) \leftarrow \text{Gen}(1^k)\)
  1. Choose primes \(p, q \in \mathbb{R}\) with \(p, q = 3 \mod 4\) big enough. Define \(n = pq\).
  2. Choose a quadratic residue \(y \in \mathbb{QR}_n\) at random. The public key is \(pk = (y, n)\). The secret key is the factorization \(sk = (p, q)\).
  3. Return \((pk, sk)\).

• \((f', st) \leftarrow \text{Encode}_s(f)\)
  1. Parse \(f\) as \(f_1, \ldots, f_\ell\) with \(f_i \in \mathbb{Z}/n\mathbb{Z}\) for all \(i = 1, \ldots, \ell\).
  2. Define the state \(st = (s, \ell)\), where \(s \in \mathbb{R}\{0, 1\}\) is chosen at random.
  3. Compute the authenticators \(\sigma_i\) for each chunk such that \(\sigma_i^{2^m} = \pm \mathcal{H}_{st}(i)^2 \cdot y^{f_i} \in \mathbb{Z}/n\mathbb{Z}\) for \(i = 1, \ldots, \ell\), where \(\mathcal{H}_{st}(i)\) is a pseudorandom function mapping to \(\mathbb{Z}/n\mathbb{Z}\) keyed with key \(st\), and \(m\) is a fixed integer. The sign is chosen such that \(\sigma\) exists.
  4. Return \(f' = (f, \sigma)\) and \(st\).

• \(\pi \leftarrow \text{Prove}(pk, f', c)\)
  1. Expand the challenge \(c\) as \(\text{chal}_i = \mathcal{H}_c(i)\) for \(i = 1, \ldots, \ell\).
  2. Parse \(f'\) as \((f, \sigma)\) and \(f\) as \(f_1, \ldots, f_\ell\).
  3. Aggregate the authenticators as \(\tau = \prod_{i=1}^\ell \sigma_i^{\text{chal}_i}\).
  4. Aggregate the chunks as \(\mu = \sum_{i=1}^\ell f_i \cdot \text{chal}_i\).
  5. Return \(\pi = (\tau, \mu)\).

• \(b \leftarrow \text{Verify}(pk, st, c, \pi)\)
  1. Recover the number of chunks \(\ell\) in the file from \(st\).
  2. Parse \(\pi\) as \((\tau, \mu)\).
  3. Check if \(\tau^{2^m} = \pm y^\mu \prod_{i=1}^\ell \mathcal{H}_{st}(i)^{2\mathcal{H}_c(i)}\).

Figure 5.5: Shoup’s Proof of Storage

5.2.2 Shoup Proof of Storage

The Shoup identification protocol [212] is an unforgeable homomorphic identification protocol based on the factoring assumption for Blum integers. It is similar to the GQ protocol with the main difference that the secret key is a \(2^{km}\)-th root instead of an \(e\)-th root, where \(e\) is chosen such that it is invertible modulo \(\varphi(n)\).

The proof of unforgeability in the homomorphic case is Theorem 3 in the paper of Ateniese et al. [16]. There, it is shown that this protocol can be used to obtain a proof of storage based on the factoring assumption. Since the resulting proof of storage from Shoup’s identification protocol is not stated explicitly in their paper we present it in Figure 5.5.
5.2. PROOFS OF STORAGE

5.2.1 Proofs of Storage

- \((pk, sk) \leftarrow \text{Gen}(1^k)\)
  1. Take a group \(G\) of order \(p\) such that an efficient bilinear pairing \(e : G \times G \rightarrow G_T\) exists. Let \(g\) be a generator of \(G\).
  2. Generate a random signing key pair \((ssk, spk)\) using a signature algorithm.
  3. Choose a random integer \(x \in \mathbb{Z}\{1, \ldots, p-1\}\).
  4. The public key is \(pk = (g^x, spk)\), the secret key is \(sk = (x, ssk)\).
  5. Return \((pk, sk)\).

- \((f', st) \leftarrow \text{Encode}_{sk}(f)\)
  1. Parse \(f\) into \(n\) blocks, each \(s\) sectors long: \(\{f_{ij}\}_{1 \leq i \leq n, 1 \leq j \leq s}\).
  2. For defining the state \(st\) choose a random filename \(name\) from some sufficiently large domain. Choose \(s\) random elements \(u_1, \ldots, u_s \in \mathbb{G}\).
  3. The state \(st\) is \(st_0 = (name, n, u_1, \ldots, u_s)\) together with a signature of \(st_0\) using the secret signing key \(ssk\).
  4. Compute the authenticators \(\sigma_i = \left[\mathcal{H}(name, i) \cdot \prod_{j=1}^s u_j^{f_{ij}}\right]^x \in G\) for \(i = 1, \ldots, n\), where \(\mathcal{H}\) is a pseudorandom function mapping to \(G\).
  5. Return \(f' = (f, \sigma)\) and \(st\).

- \(\pi := \text{Prove}(pk, f', Q)\)
  1. The challenge \(Q\) is given as a set \(\{(i, \nu_i), i \in \{1, \ldots, n\}, \nu_i \in \mathbb{Z}/p\mathbb{Z}\}\) of indices and coefficients.
  2. Parse \(f'\) as \((f, \sigma)\) and \(f\) as \(\{f_{ij}\}_{1 \leq i \leq n, 1 \leq j \leq s}\).
  3. Aggregate the authenticators as \(\tau = \prod_{(i, \nu_i) \in Q} \sigma_i^{\nu_i}\).
  4. Aggregate the chunks as \(\mu_j = \sum_{(i, \nu_i) \in Q} f_{ij} \cdot \nu_i \in \mathbb{Z}/p\mathbb{Z}\) for \(1 \leq j \leq s\).
  5. Return \(\pi = (\tau, \mu_1, \ldots, \mu_s)\).

- \(b := \text{Verify}(pk, st, Q, \pi)\)
  1. Parse \(pk\) as \((v, spk)\).
  2. Use the public signing key \(spk\) to verify the signature on \(st\). If the signature is invalid reject the proof as invalid.
  3. Otherwise parse \(st\) and recover \(name\), the number of blocks \(n\), and \(u_1, \ldots, u_s\).
  4. Parse \(\pi\) as \((\tau, \mu_1, \ldots, \mu_s)\).
  5. Check if \(e(\tau, y) = e\left(\prod_{(i, \nu_i) \in Q} \mathcal{H}(name, i)^{\nu_i} \cdot \prod_{j=1}^s u_j^{\mu_j}, v\right)\).

Figure 5.6: The original Shacham-Waters Proof of Storage [207]

5.2.3 Shacham-Waters Proof of Storage

Shacham and Waters (SW) proposed a proof of storage [207] where the proofs are of constant size and publicly verifiable. Their original scheme is shown in Figure 5.6. Its proof of security can be found in their paper [207].

The SW scheme supports a tuning parameter \(s\) which subdivides the chunks.
into smaller subchunks. In the terminology of Shacham and Waters these are called blocks and sectors. Each of the $n$ blocks of the file $f$ is $s$ sectors long. If $\ell$ is the number of sectors, then $\ell = n \cdot s$. A larger value for $s$ increases the size of the proofs by a factor of $s$. On the other hand this decreases the number of authenticators $\sigma_i$, the prover $P$ needs to store by a factor of $s$.

The SW scheme was presented independently of the transformation from an identification protocol [16]. However as hinted by Ateniese et al. [16] it can be slightly modified to be a result of this transformation. We modified the original scheme in order to increase the comparability with the other schemes described. As a side effect, we are able to reconstruct the underlying identification protocol based on the Diffie-Hellman assumption. In particular we made the following changes to the original protocol:

- In the SW scheme, a second key pair is generated and used to sign the number of chunks in the file such that it cannot be tampered. Instead we include the number of chunks in the state $st$ as in the other schemes.

- SW introduce a tuning parameter $s$ which subdivides the chunks into smaller subchunks. A larger value for $s$ increases the size of the proofs linearly but on the other hand decreases the number of authenticators $\sigma_i$, the prover $P$ needs to store, linearly. We set $s = 1$ to not subdivide the chunks, so that the number of authenticators is the same as in the other schemes.

- In the SW scheme, there are public values $u_1, \ldots, u_s \in \mathbb{G}$ chosen at random from an elliptic curve $\mathbb{G}$. Since we set $s = 1$, this is only one value $u$, which we include in the public key, rather than choosing it per file.

- The challenge in the SW scheme is a set $Q = \{ (i, \nu_i) \text{, where } 1 \leq i \leq \ell, 1 \leq \nu_i \leq p \}$ of coefficients. We generate the challenge from a pseudorandom function $\nu_i = H_c(i)$ for $i = 1, \ldots, \ell$, mapping to $\mathbb{Z}/p\mathbb{Z}$ and thus the size of the challenge is the same as in the other schemes we described.

- Further, we adapted the notation to match our earlier expositions.

A full description of the modified protocol is given in Figure 5.7. These modifications allow us to extract the underlying identification protocol which is shown in Figure 5.8 and might be of independent interest.

It can be shown that our modifications to the SW proof of storage scheme are secure by proving that the underlying identification protocol is indeed an unforgeable homomorphic identification protocol in our attacker model from Definition 5.1.2. Security of the proof of our modified SW proof of storage then follows by the security of the transformation from an identification protocol to a proof of storage.

**Theorem 5.2.2.** Our reconstructed identification protocol shown in Figure 5.8 is an unforgeable homomorphic identification protocol under the assumption that the computational Diffie-Hellman problem in $\mathbb{G}$ is intractable.

**Proof.** Completeness is immediate. To prove unforgeability we construct a ppt algorithm $B$ that can solve the Diffie-Hellman problem, given access to an attacker algorithm $A$. 
5.2. PROOFS OF STORAGE

- $(pk, sk) \leftarrow \text{Gen}(1^k)$
  1. Take a group $G$ of order $p$ such that an efficient bilinear pairing $e : G \times G \to G_T$ exists. Let $g$ be a generator of $G$.
  2. Choose a random secret key $sk \in \mathbb{R} \{1, \ldots, p-1\}$ and compute the public key as $pk = (g^{sk}, u)$, where $u$ is another generator of $G$.
  3. Return $(pk, sk)$.

- $(f', st) \leftarrow \text{Encode}_{sk}(f)$
  1. Parse $f$ as $f_1, \ldots, f_\ell$ with $f_i \in \mathbb{Z}/p\mathbb{Z}$ for all $i = 1, \ldots, \ell$.
  2. Define the state $st = (s, \ell)$, where $s \in \mathbb{R} \{0, 1\}$ is chosen at random.
  3. Compute the authenticators $\sigma_i = [H_{st}(i) \cdot u^f]^{sk} \in G$ for $i = 1, \ldots, \ell$, where $H_{st}$ is a pseudorandom function mapping to $G$ keyed with key $st$.
  4. Return $f' = (f, \sigma)$ and $st$.

- $\pi := \text{Prove}(pk, f', c)$
  1. Expand the challenge $c$ as $\text{chal}_i = H_c(i)$ for $i = 1, \ldots, \ell$, where $H_c$ is a pseudorandom function mapping to $\mathbb{Z}/p\mathbb{Z}$ keyed with key $c$.
  2. Parse $f'$ as $(f, \sigma)$ and $f$ as $f_1, \ldots, f_\ell$.
  3. Aggregate the authenticators as $\tau = \prod_{i=1}^{\ell} \sigma_i^{\text{chal}_i}$.
  4. Aggregate the chunks as $\mu = \sum_{i=1}^{\ell} f_i \cdot \text{chal}_i$.
  5. Return $\pi = (\tau, \mu)$.

- $b := \text{Verify}(pk, st, c, \pi)$
  1. Parse $pk$ as $(v, u)$.
  2. Recover the number of chunks $\ell$ in the file from $st$.
  3. Parse $\pi$ as $(\tau, \mu)$.
  4. Check if $e(\tau, g) = e\left(\prod_{i=1}^{\ell} H_{st}(i)^{H_c(i)} \cdot u^\mu, v\right)$.

**Figure 5.7:** Our modified Shacham-Waters Proof of Storage

- $B$ is given an element $g \in G$ as well as $g^{sk}$ for some $sk \in \{1, \ldots, p-1\}$ and an element $h \in G$. To solve the Diffie-Hellman problem $B$ needs to compute $h^{sk}$. It chooses random coefficients $s, t \neq 0$ and sets $u = g^s h^t$. Thus, the public key is $pk = (g^{sk}, u)$. Afterwards the algorithm $B$ executes $A$.

- Whenever $A$ outputs $\beta'$ in the challenge space $B$ chooses a random $r \in \mathbb{R} \{1, \ldots, p-1\}$ and computes $\gamma = pk^r$. One can easily verify that this is indeed a valid transcript. $B$ gives $(r, \gamma)$ to $A$.

- When $A$ outputs an $n$-vector of challenges $\beta$, then for each $i$ the algorithm $B$ computes $(r, \gamma)$ as in the previous step and sets $\alpha = g^r u^{-\beta}$. It gives the vectors $(r, \gamma)$ to the algorithm $A$.

- If $A$ outputs $(s, \mu', \gamma')$ where $s \in \mathbb{Z}_{2^k}^n$, and
• \((pk, sk) \leftarrow \text{Setup}(1^k)\)
  1. Let \(G\) be a group of prime order \(p \in \mathbb{F}\) such that an efficient bilinear pairing \(e : G \times G \rightarrow G_T\) exists. Let \(g\) be a generator of \(G\).
  2. Choose a random secret key \(sk \in \{1, \ldots, p-1\}\) and compute the public key as \(pk = (g^s, u)\), where \(u\) is another generator of \(G\).
  3. Return \((pk, sk)\).
• \(\alpha \leftarrow \text{Commit}(pk; r)\)
  1. Return \(r\).
• \(\gamma \leftarrow \text{Response}(pk, sk, r, \beta)\)
  1. Return \((r \cdot u^\beta)^s\).
• \(b \leftarrow \text{Verify}(pk, \alpha, \beta, \gamma)\)
  1. Check if \(e(\gamma, g) = e(r \cdot u^\beta, pk)\).
• Combine1\((s, \alpha) := \prod \alpha_i^s\)
• Combine3\((s, \gamma) := \prod \gamma_i^s\)

Figure 5.8: Our Shacham-Waters identification protocol

1. \(\mu' \neq \sum s_i \beta_i\), and
2. \(\text{Verify}(pk, \text{Combine1}(s, \alpha), \mu', \gamma') = 1\)

we solve the discrete logarithm problem by computing \(h^s\) as follows:

Define \(\alpha^* := \text{Combine1}(s, r)\), \(\gamma^* := \text{Combine3}(s, \gamma)\), and \(\mu^* := \sum s_i \beta_i\).

Because of the second condition on \((s, \mu', \gamma')\) we know that

\[ e(\gamma', g) = e(\alpha^* u^{\mu'}, g^s) . \]

Since \(\alpha^*, \mu^*, \gamma^*\) is also a valid transcript we know that

\[ e(\gamma^*, g) = e(\alpha^* u^{\mu'}, g^s) . \]

Dividing the two equations yields

\[ e(\gamma' / \gamma^*, g) = e(u^{\mu' - \mu^*}, g^s) . \]

Substituting \(u = g^s h^t\) results in

\[ e(\gamma' / \gamma^*, g) = e(g^{s(\mu' - \mu^*)} h^t(\mu' - \mu^*), g^s) . \]

We can now rearrange the terms as

\[ e(\gamma' / \gamma^*, g) = e(h, g^{sk}) t(\mu' - \mu^*) . \]

and solve the discrete logarithm problem by computing \(h^s\) as follows:

\[ h^s = \left(\gamma' / \gamma^* \cdot g^{s(\mu' - \mu^*)}\right)^{\frac{1}{\mu' - \mu^*}} . \]

This fails if and only if the denominator is zero. But this will not happen, as we have chosen \(t\) to be nonzero and \(\mu' - \mu^* \neq 0\) holds due to the first condition on the output of the attacker.
5.3. EVALUATION

Since \( B \) solves the discrete logarithm problem whenever \( A \) succeeds we conclude that the success probability of the attacker \( A \) is negligible.

Lemma 5.2.2. The modified SW proof of storage shown in Figure 5.7 is secure under the Diffie-Hellman assumption if \( H \) is modelled as a random oracle.

Proof. Follows directly from the security of the transformation of a homomorphic identification protocol to a proof of storage (Theorem 1 and 2 in [16]) and the security of the SW identification scheme in Theorem 5.2.2.

5.3 Evaluation

In the following we provide benchmarks for the three proof of storage schemes we described and implemented.

5.3.1 Method

We implemented the three discussed proof of storage schemes in Python 3.5.2 and evaluated their performance on a quad core Intel Xeon CPU with 3.10 GHz running Ubuntu 16.04.1. We did not use any parallelization though most of the algorithms are easily parallelizable. The implementations were written by the same developer, hence the coding style and quality is similar.

The choice of parameters is summarized in Table 5.3. We use the symbol \(|\cdot|\) to denote the bit length.

For the GQ and Shoup scheme we chose the order of the finite group to be 2048 bits. For these schemes, we used gmpy v1.17, which is a Python binding of the GNU multiple precision library GMPlib [95]. In the Shoup protocol we chose the parameter \( k = 5 \) thus the security depends on the inability to compute \( 2^{15} \)-th roots in a 2048 bit group which according to NIST [20, Section 5.6.1] is equivalent to a security level of 112 bits and is considered secure.

The pairing operations in the SW scheme were implemented with the Python bindings of the PBC Library v0.5.14 [149]. We decided to use a type F pairing for the implementation. On the one hand, type F pairing operations are slower than in other types of pairings, but the algorithm in the SW scheme only needs to compute one of them. On the other hand, elements in type F pairings are smaller than in other pairing types of comparable security, so the many exponentiations in our implementation are faster. The pairing was generated with the script \texttt{genfparam} bundled with libPBC. For a security level of 112 bits the size of the curve needs to be around 224–255 bits according to NIST [20, Section 5.6.1]. However, this does not take into account the recent improvements in computing discrete logarithms in finite fields by Kim and Barbulescu [120] which halve the security parameter. Thus, we chose to use a curve of size 448–510 bits. Table 5.2 shows the exact parameters used in our evaluation for the type F pairing in libPBC. To be secure we need that generic discrete logarithms are infeasible in groups of order \( r \) and finite field discrete logarithms are infeasible in finite fields of order \( q^{12} \). With our choice of \( r \) and \( q \) we get \(|r| = 445\) and \(|q^{12}| = 5342\). Hence, the security of the three implemented schemes is comparable.

\[^3\]Again, only \( H_{st}(i) \) needs to be modelled as a random oracle due to internal workings of the transformation. For \( H_{c}(i) \) it suffices to be a pseudorandom function.
CHAPTER 5. A COMPARISON OF PROOFS OF STORAGE

Table 5.2: Parameters for the bilinear pairing used in the SW scheme

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our GQ PoS</td>
<td>(</td>
</tr>
<tr>
<td>Shoup PoS</td>
<td>(</td>
</tr>
<tr>
<td>SW PoS</td>
<td>(</td>
</tr>
</tbody>
</table>

Table 5.3: Overview of the Parameters

An overview of the choice of all parameters can be found in Table 5.3.

For \(H_{st}\), the pseudorandom function keyed with the state, where the state is a pair, we convert the state into JSON format and use the standard HMAC with SHA256 algorithm as described by RFC 2104 [134]. Note in particular that if the JSON representation of the state is longer than 256 bits, then it is first hashed using SHA256 and the result is used as key.

We measured the duration needed for encoding the file \(T_{Encode}\), generating the proof of storage \(T_{Prove}\), and verification time \(T_{Verify}\) for files up to 4000 kB in steps of 250 kB. Each measurement is the mean of the duration of 10 runs of the algorithm. For the time needed to generate the keys \(T_{Gen}\) we computed the mean over 160 runs.

5.3.2 Results and Discussion

As expected all durations were linearly dependent on the size of the file. The interpolated processing bandwidth for the different algorithms, as well as the
5.3. EVALUATION

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$T_{Gen}$</th>
<th>$B_{Encode}$</th>
<th>$B_{Prove}$</th>
<th>$B_{Verify}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our GQ PoS</td>
<td>0.111s</td>
<td>56.1 kB/s</td>
<td>414 kB/s</td>
<td>62.7 kB/s</td>
</tr>
<tr>
<td>Shoup PoS [16]</td>
<td>0.187s</td>
<td>4.37 kB/s</td>
<td>416 kB/s</td>
<td>64.8 kB/s</td>
</tr>
<tr>
<td>SW PoS [207]</td>
<td>0.124s</td>
<td>5.20 kB/s</td>
<td>54.4 kB/s</td>
<td>16.9 kB/s</td>
</tr>
</tbody>
</table>

Table 5.4: Overview of the Results

time used for key generation can be found in Table 5.4. The interpolation was done by ordinary least squares regression.

When encoding the file, the Shoup proof of storage was slower than our GQ proof of storage. This performance difference can be traced back to a performance difference in the underlying identification protocols. The main difference between the Shoup and the GQ identification protocol is that in the Shoup identification protocol the secret key is a $2^{3m}$-th root instead of an $e$-th root, where $e$ is chosen such that it is invertible modulo $\varphi(n)$. Thus the Shoup protocol has to be slower, since when computing an $e$-root in the GQ identification protocol we can simply exponentiate with the inverse $d$, where $de = 1 \mod \varphi(n)$. For a $2^{3m}$-th root in the Shoup protocol this does not work, since $2 \mid \gcd(2^{3m}, \varphi(n))$ and thus we cannot compute an inverse $d$. Instead we have to compute roots modulo the primes $p$ and $q$ by successively raising to the $p^{-1}$-th power, respectively the $q^{-1}$-th power and recompose the solution modulo $n$ with the chinese remainder theorem. The performance of the algorithm $Encode$ in the SW protocol was faster than the same procedure in the Shoup protocol, but slower than in the GQ protocol.

The Shoup and GQ scheme needed exactly the same time for generation of the proof, since the algorithm for generating the proof is the same. Generation of the proof in the SW protocol had the worst performance. We found out that this is due to expanding the challenge by computing $\text{chal}_i = H_c(i)$ for $i = 1, \ldots, \ell$. This was implemented by using the function $\text{element from hash}$ of libPBC since the element needed to be a point on the elliptic curve. The exact algorithm can be found on page 19 in Lynn’s thesis [148].

In the verification procedure the Shoup protocol was insignificantly faster than our GQ protocol. As in generation of the proof, the SW proof of storage performed worst in verifying.

An interesting observation is that generation of the key in the SW scheme was fastest. This is explained by the fact that we used a type F pairing, so our underlying group is small and thus exponentiations are fast, as explained previously.

The full performance measurements of the schemes are given in Figures 5.9a to 5.9d. The duration of the algorithms in seconds is shown on the y-axis. The size of the file of which storage was proven is shown on the x-axis.

5.3.3 Verification Preprocessing

In some scenarios it is possible to speed up the verification process by precomputation. The idea behind verification preprocessing is to partition the verification into two phases. In the first phase the verifier has access to everything needed for the verification except the proof $\pi$ itself, i.e., the public key $pk$, state $st$,
and challenge $c$. Only in the second phase, the verifier obtains the proof $\pi$. The crucial point is that the verifier may precompute parts of the formula needed for the verification in the first phase, such that the second phase is significantly faster.

Take as an example our GQ proof of storage from Section 5.2.1. There, for the verification one needs to check if $J^\mu \cdot \tau^e = \prod_{i=1}^\ell H_{st}(i)^{H_c(i)}$. Without access to the proof $\pi = (\tau, \mu)$ the verifier can compute the right side $\prod_{i=1}^\ell H_{st}(i)^{H_c(i)}$, which is the main part of the computation in advance. After obtaining the proof the verifier needs to compute $J^\mu \cdot \tau^e$, which are only two exponentiations, one multiplication, and one comparison, to verify the proof of storage. Thus, if the verifier knows the challenge in advance the verification time can be decreased significantly.

For the other schemes described, similar arguments can be made. An overview of the necessary processing steps for the verification after a preprocessing phase can be found in Table 5.5. It can be seen that there are very few steps to verify a proof of storage after allowing a preprocessing phase, independent of the proof of storage scheme.
5.4 Modifications for Use in our Storage System

Our evaluations have shown that directly using a proof of storage in the decentralized storage system of this thesis only works for small data. Otherwise, the algorithms for encoding the data, creating the proof of storage, and verifying the proof of storage are too inefficient. Consequently, for storing large amounts of data in our scheme we need to find a feasible alternative.

The main efficiency drawback of proofs of storage as described, stems from the fact that each block needs to be challenged in order to provide a deterministic storage guarantee. When only some chunks are challenged, there is instead just a probabilistic storage guarantee, but the algorithms involved are faster. This is called a proof of data possession [13, 14, 15].

Proofs of data possession are not deterministic proofs that the data is stored, but instead allow statements like “If the storage provider modifies a fraction of $\epsilon$ of the data, this will be detected with a probability of $1 - \delta$”. While these statements are harder for end-users to interpret, the overall usability may still increase, since the runtime decreases significantly.

In particular, proofs of data possession need only sublinearly many chunks to be challenged in order to create a proof. To detect a loss of 1% of a file with 90% probability the client needs to challenge less than 90% of the blocks of the file, since the challenges are chosen uniformly at random from the whole file. As an example, let $f$ be a file with 10,000 blocks. Then, if the storage provider has corrupted 1% of the blocks checking only 460 blocks can detect misbehavior with probability greater than 99% [14, Section 4.2].

This sublinear challenging can be applied in combination with an erasure code which increases the size of a file $f = f_1, \ldots, f_\ell$ by a factor of $\delta$, but any $\ell$ blocks of the erasure encoded file suffice to recover the original file. Combining the sublinear challenging and the erasure code, less than the whole file needs to be processed to prove recoverability with high probability of an erasure encoded file. This improves the running time of the generation and verification of the proof.

An explanation of this process for speeding up private proofs of data possession is given by Ateniese et al. [14]. The resulting provable data possession scheme of Ateniese et al. achieves performance which is only bounded by disk I/O and not by cryptographic operations. In particular, Ateniese et al. [14, Section 4.2] give a description of the choice of parameters for the erasure code and the number of chunks to challenge which we will not repeat here. Instead, we will explain that their modifications cannot be used directly for public proofs.
of data possession, since they do not offer public retrievability of the file. We go on to modify the scheme to allow retrievability by a fixed set of participants. An overview of proofs of storage as previously discussed, the provable data possession scheme of Ateniese et al. [14], and our modifications proposed in this chapter can be found in Table 5.6.

<table>
<thead>
<tr>
<th>Storage Guarantees</th>
<th>Proofs of Storage</th>
<th>PDP [14]</th>
<th>Our modifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verifiability</td>
<td>Deterministic</td>
<td>Probabilistic</td>
<td>Probabilistic</td>
</tr>
<tr>
<td>Retrievability</td>
<td>Public</td>
<td>Private</td>
<td>Public</td>
</tr>
<tr>
<td>Bottleneck</td>
<td>Computations</td>
<td>Disk I/O</td>
<td>Disk I/O</td>
</tr>
</tbody>
</table>

Table 5.6: An overview of the properties of Proofs of Storage as explained before, provable data possession (PDP) of Ateniese et al. and our modifications in Section 5.4

Naively combining the sublinear challenging with the erasure code results in an insecure protocol due to the fact that an erasure code induces linear relations on the encoded file. When generating the homomorphic linear authenticators, i.e., when executing the Encode algorithm from the proof of storage, these linear relations are preserved. A malicious storage provider could use these relations in order to prove storage of the erasure encoded file, but only storing the unencoded file and regenerating the erasure encoded authenticators on demand. In order to hide the relations between the authenticators from a malicious storage provider, a secret permutation $\pi_{skp}$ needs to be applied to the check blocks from the erasure code to yield a secure scheme. Here, $skp$ denotes the secret key of the permutation. If parts of the file are lost, to recover the original file using the erasure code knowledge of the secret permutation, i.e., of the secret key $skp$ is needed. Thus, using this approach in our context results in a publicly verifiable proof, where recoverability can only be achieved privately. This is undesirable for many applications where public verifiability is needed—in the context of our storage system this prohibits sharing of data. As discussed in Section 4.5.1, sharing of a file $f$ between participants with public keys $pk_1, \ldots, pk_n$ is done by encrypting the file with a symmetric key $sk$ and then uploading $\text{enc}_{pk_1}(sk), \ldots, \text{enc}_{pk_n}(sk), \text{enc}_{sk}(f)$. To support the technique with the erasure encoding and the secret permutation, we instead encode the file $f = (f_1, \ldots, f_\ell)$ as follows:

- Using the erasure code, compute check blocks $c_1, \ldots, c_\delta\ell$ which provide the necessary redundancy to recover the file.
- Permute the check blocks using $\pi_{skp}$ for some secret key $skp$. This yields $\pi_{skp}(c_1, \ldots, c_\delta\ell)$.
- Choose some secret key $sk$ and encrypt the file together with the check blocks as $\text{enc}_{sk} (f_1, \ldots, f_\ell, \pi_{skp}(c_1, \ldots, c_\delta\ell))$.
- Finally, encrypt the secret keys $sk$ and $skp$ which are necessary for recovering the file with the public keys $pk_1, \ldots, pk_n$. This yields $\text{enc}_{pk_i}(sk, skp)$.
5.5. FUTURE WORK

In this section we review some open questions and possible improvements to our work.
5.5.1 Anonymous Proofs of Storage

A second line of future work for our scenario is to construct a k-anonymous proof of storage, where the client does not encode its file with its private key but rather with its private key together with a set of other public keys. In order to verify the proof of storage, the k public keys are needed and it cannot be determined to which public key the real private key used in the encoding process belongs. This enables the client to hide in a set of k public keys without revealing its identity.

We tried to construct a transformation from a homomorphic group identification protocol to such a k-anonymous proof of storage. We defined a homomorphic group identification protocol as an interactive protocol that allows a participant to prove that it is in possession of a private key corresponding to a public key in a set of public keys. Analogous to Ateniese et al. [16], who transform a homomorphic identification protocol first to a homomorphic linear authenticator and then to a proof of storage, we tried to transform the homomorphic group identification protocol to a homomorphic ring linear authenticator and then to a k-anonymous proof of storage. Since we are unaware of any homomorphic group identification protocols, we constructed one through the or-composition of k identification protocols. However, in the classical or-composition of such sigma protocols the secret knowledge which public key is the correct one is needed in the commitment phase. This lead to the impossibility of applying a simple generalized version of the transformation of Ateniese et al. to the group identification protocol. However, there may be a different way to achieve a k-anonymous proof of storage, which is unknown to us and may be of independent interest for this research field.

5.5.2 Post-quantum Security

A third line of work is related to post quantum security. The proofs of storage discussed in this chapter are secure under the RSA assumption (GQ proof of storage), the factoring assumption (Shoup proof of storage), or the computational Diffie-Hellman assumption (SW proof of storage). All of these problems can be solved efficiently in polynomial time if the attacker has access to a quantum computer using the well-known algorithms by Shor [21]. Thus, none of these schemes is secure under an attacker with access to a quantum computer. We identified the following main post-quantum identification protocols in order to transform them to a publicly verifiable proof of storage secure under an attacker with access to a quantum computer.

**Lyubashevsky identification protocol [15]:** This scheme provides an identification protocol based on a hash function [15] which is collision resistant based on some lattice assumption. However, the identification protocol aborts if some size property on the last message is not given and consequently has to be restarted. The protocol is homomorphic but since it aborts sometimes, the generation of some tags in the encoding step of the resulting proof of storage may be undefined.

**Stern protocol [21]:** The Stern identification protocol is based on syndrome decoding, a special kind of error-correcting code. However, it is not homomorphic, since the challenges determine branches in the computation
of the response and thus cannot be used to construct a publicly verifiable proof of storage.

**Vérón identification protocol** [225]: Vérons scheme is in some sense a dual version to the Stern identification protocol. Thus, as the Stern protocol, it is based on syndrome decoding. Since the challenges are not from any group, the protocol is not homomorphic and cannot be used to construct a publicly verifiable proof of storage.

**Cayrel-Véron-El Yousfi identification protocol** [49]: This identification protocol is based on the $q$-ary syndrome decoding problem. However, it has five rounds and thus does not fit our definition of an identification protocol. Further, the challenge is a single bit and consequently the scheme is not homomorphic and cannot be used to construct a publicly verifiable proof of storage.

**Sakumoto-Shirai-Hiwatari identification protocol** [203] The security of this protocol is based on solving a system of multivariate quadratic polynomials over a finite field. Again, it is not homomorphic, since the challenge is not a member of a group and thus does not yield a publicly verifiable proof of storage.

Consequently, we have been unable to find a post-quantum secure identification protocol which satisfies our requirements.

The most promising approach was Lyubashevsky's identification protocol, since the used lattices are ideal lattices and thus possess additional algebraic structure which is needed for the homomorphicity of the identification protocol. All other schemes interpreted the challenge as a bit sequence instead of an integer in a certain algebraic domain.

As a side effect, these protocols require more rounds than number theoretic identification protocols, like the Guillou-Quisquater or the Shoup identification protocol for the same security level. This is due to the fact, that the proof of security of an identification protocol shows that these schemes are zero-knowledge proofs of knowledge of some secret value, namely the private key. Thus, a successful prover actually knows the secret, without revealing any part of it, since the protocol is zero-knowledge. In order for the identification protocol to have negligible soundness error, it needs to be repeated for a polynomial number of rounds. This needs to happen sequentially to maintain security, since zero-knowledge is not preserved under parallel repetition [97].

There is a lattice based publicly verifiable proof of storage protocol by Xu et al. [233] which is secure under the shortest integer solution problem. However, their construction does not start at an identification protocol, but with homomorphic verifiable tags. Hence, they only use the second half of the transformation by Ateniese.

Therefore, an open question is to construct a homomorphic post-quantum secure identification protocol which can be used to construct a publicly verifiable proof of storage protocol. Due to the additional algebraic structure, this identification protocol would probably be faster than current post-quantum secure identification protocols (except the Lyubashevsky protocol), since it does not require parallel repetition of many rounds.
5.6 Conclusion

In this chapter we modified the GQ identification protocol such that it is secure in a stronger attacker model. This allowed us to construct a novel proof of storage based on the RSA assumption. Additionally we introduced a novel identification protocol based on bilinear pairings. We carefully compared the performance of our scheme to other state of the art publicly verifiable proof of storage schemes. In our evaluation our new scheme outperforms existing schemes with similar security level in the time taken to encode the file. For the time taken to generate the proof of storage or verify the proof of storage its performance is comparable to the fastest scheme respectively.

All examined proof of storage schemes in this chapter can be used in our distributed storage system. Although the performance of the implemented schemes is not sufficient to share large files in our system, it should be noted that our goal was to provide a fair comparison between the schemes instead of optimizing the schemes for throughput. For using our scheme in our decentralized storage system, we suggest to turn it into a proof of data possession as explained in Section 5.4.
uMine: A Mining Algorithm based on Human-Work

In our first design of a decentralized storage system with incentives in Chapter 3, we substituted the proof of work mining process from regular blockchains through proofs of storage. However, as this introduces a tight coupling between the mining process and the storage mechanics, our second design in Chapter 4 separates these layers and introduces storage smart contracts in order to improve privacy. Mining in our privacy enhanced design is done by classical proofs of work. However, mining through proofs of work suffers from high power consumption.

The power consumption of the mining process in the Bitcoin blockchain alone is estimated to be comparable to the electricity consumption of Ireland [176] which constitutes a serious liability to the widespread adoption of blockchain technology. More recent sources [1] estimate Bitcoins energy consumption at 56.89 TWh which corresponds to 0.25% of the world’s electricity consumption. This is due to the fact that the mining process currently is controlled by very few large mining companies using special mining equipment, contrary to the original vision of Nakamoto [167] that each participant has approximately the same power in the mining process. With the widespread adoption of blockchain based cryptocurrencies their energy consumption will likely continue to rise further. We identify this problem to be one of the main challenges of scaling blockchains and allowing for their widespread adoption.

In this chapter we tackle the issue of the huge energy consumption of blockchains by introducing uMine, a mining algorithm based on a novel proof of human-work construction. Proofs of human-work are cryptographic mechanisms where a prover can convince a verifier that it has spent some amount of human work. In particular, proof of human-work puzzles can only be solved by humans and not by computers under the hardness assumption of some underlying AI problem. This allows us to lower the energy consumption of the blockchain by exchanging the costly proof of work mining algorithm by a proof of human-work which can only be provided by humans.

Next to our instantiation there is only one other instantiation known [32] which relies on indistinguishability obfuscation, a cryptographic primitive whose
existence is only conjectured and where no realization is known. In contrast, our
construction is based on the cryptographic principle of multiparty computation
(which we use in a black box manner) where multiple feasible instantiations are
known (cf. [24, 60, 61, 116, 170]). Thus, our instantiation is the first known
feasible proof of human-work scheme.

Beyond usage in a mining algorithm, proofs of human-work can be used
to construct password authentication schemes which provably protect users
against offline attacks [32]. Thus, other applications beyond mining may benefit
from our construction.

Large parts of this chapter have been previously published at ICICS 2018 [128, 129].

Contribution Our contributions can be summarized as follows:

• We provide a novel instantiation of a proof of human-work which does
  not rely on indistinguishability obfuscation but instead uses multiparty
  computation as a black box.

• We prove the security of our proof of human-work given a secure captcha.

• We use our proof of human-work to construct uMine, a novel energy
  efficient mining algorithm where the mining is performed by human
  miners creating proofs of human-work.

• We introduce protocol which allows to uncover collusions in a multiparty
  computation protocol by rewarding traitors of a collusion.

Roadmap The next section introduces the basic primitives used in the re-
mainder of this chapter. Section 6.2 provides a thorough explanation of our
novel instantiation of a proof of human-work and its use as a mining algorithm.
Section 6.3 describes considerations when implementing our proof of human-
work scheme. In Section 6.3 we analyze the security properties of our solution.
Related work is discussed in Section 6.5. Finally, Section 6.6 concludes this
chapter.

6.1 Building Blocks

This section introduces the various building blocks used in our construction.

6.1.1 Slow Hash Functions

Slow hash functions are a special kind of hash function. While usual hash
functions $H$ are designed to be easy to compute, the evaluation of a slow hash
function $H$ in contrast is computationally costly. Normally the evaluation of a
slow hash function like bcrypt [193] or scrypt [185] is on the order of several
hundred milliseconds, thus slowing down brute-force attacks significantly. The
intuition behind slow hash functions is that an authorized user needs to evaluate
them only once, and thus the overhead is negligible, while an attacker needs to
evaluate them many times and thus is slowed down significantly. Throughout
this chapter, we will denote slow hash functions by the symbol $H$. 
6.1.2 Captchas

Captcha is an acronym for a Completely Automated Public Turing test to tell Computers and Humans Apart. They are challenge response tests to determine if the user is a human or a program. One major application is to prevent automated registrations of accounts in web services. The most common form of a captcha puzzle consists of a set of warped letters, where the user is requested to recognize the letters, a task which is supposedly hard for computers and easy for humans. There are also other forms like audio-based captchas where the user is challenged to recognize speech data. To enable automatic verification of a given solution without human assistance the service provider has usually stored a secret set of puzzle-solutions pairs. These pairs are generated by computing a puzzle from a known solution. For verification, access to these puzzle-solution pairs is needed and hence captchas are in general not publicly verifiable.

Since captchas are based on the assumption that some fundamental AI problem is hard to solve, the need to model the human solver as an entity distinct from an algorithm arises. Sometimes this is done in the form of a (yet) unknown algorithm. Since we prefer giving a clearer exposition to giving a philosophically correct one we simply model the human as an oracle that can provide the solutions to a captcha puzzle along the lines of Blocki and Zhou [32].

Definition 6.1.1 (Captcha [32, Definition 1]). A Captcha \( \text{CAPT} \) is a quintuple of algorithms (\( \text{Setup}, W, G, \Sigma_{\text{human}}, \text{Verify} \)) with the following properties:

1. \( \text{PP} \leftarrow \text{CAPT}.\text{Setup}(1^\lambda) \) is the generation of the public parameters \( \text{PP} \) given a security parameter \( \lambda \).

2. \( \sigma \leftarrow \text{CAPT}.W(\text{PP}) \) is a randomized algorithm sampling a solution \( \sigma \) given the public parameters.

3. \( Z \leftarrow \text{CAPT}.G(\text{PP}, \sigma) \) generates a captcha-puzzle \( Z \) with solution \( \sigma \). We write \( \text{CAPT}.G(\text{PP}, \sigma; r) \) if we fix the randomness \( r \), i.e., if we consider \( \text{CAPT}.G \) as a deterministic function.

4. \( \sigma \leftarrow \text{CAPT}.\Sigma_{\text{human}}(\text{PP}, Z) \) is a solution finding algorithm that takes as input the public parameters and a puzzle \( Z \) and outputs a solution \( \sigma \). It has internal access to a human oracle.

5. \( b := \text{CAPT}.\text{Verify}(\text{PP}, Z, \sigma) \) outputs a single bit which is 1 whenever there is a random \( r \), such that \( \text{CAPT}.G(\text{PP}, \sigma; r) = Z \).

The original definition of a captcha by Blocki and Zhou [32] additionally uses a tag which is generated together with the puzzle and needed for the verification of a solution \( \sigma \). We stress that our construction later also works with the definition of Blocki and Zhou, where the tag is set as undefined. However, the tags are not necessary in our construction and thus we decided to present our work using a simpler definition to aid in the understanding.

If the randomness \( r \) which was used to generate the captcha in the algorithm \( \text{CAPT}.G \) is known it may be possible to invert \( \text{CAPT}.G \). In the case of image based captchas (see Figure 6.1) \( r \) determines the chosen transformations, e.g., rotation, addition of noise, and their parameters applied on the solution to yield a puzzle [2]. Knowledge of these may allow an attacker to invert the used transformations and thus recover the solution \( \sigma \) from a puzzle \( Z \) without
the use of human work. Consequently the security of a captcha puzzle \( Z = \text{CAPT}.G(PP, \sigma; r) \) is usually based on the secrecy of the random value \( r \), which was used to generate the puzzle [2, Section Who knows What?].

**Additional Requirements for our Construction** In contrast to the original definition by Blocki and Zhou [32] we require the generation of the puzzles \( \text{CAPT}.G(PP, \sigma; r) \) to be collision-free, i.e., injective, in its randomness \( r \) and in its solutions \( \sigma \). Regarding the solutions \( \sigma \), it is natural to assume that there can be no two different solutions to the same puzzle. Regular image based captchas do have this property. Regarding injectivity in the randomness \( r \) we can assume that it serves as an enumeration of the puzzle space for a given captcha solution. Consider the case of image based captchas (see Figure 6.1) where the randomness determines the type of transformations. Different transformations with different parameters yield different puzzles and thus collision freeness can be assumed. An example of two captcha puzzles generated with different randomness from the same solution can be seen in Figure 6.1.

**Use in our Instantiation** In our construction of a proof of human-work the randomness \( r \) used in the puzzle generation \( \text{CAPT}.G \) is set to a deterministic value containing a hash of the solution \( H(\sigma) \), which was computed by a slow hash function. This way it is easy to verify a solution publicly, given a puzzle, since one only needs to regenerate the puzzle from the solution \( \sigma \) using the same randomness and check if the given puzzle equals the computed one. The use of a slow hash function is necessary to prevent bruteforcing of the solution using the verification algorithm.

**Security Properties** We require that any captcha should be solvable by a human. We use the term human-work unit to denote the effort needed to solve a single instance of a captcha. Although the time needed to solve a captcha may depend on the human and its abilities, we expect these differences to be small and similar to the differences in performance of different computer hardware. Of course, for a small set of specially designed captchas which, e.g., can only be solved by non-colorblind people, this assumption does not hold. Also note that the differences in performance of computer hardware are very large and thus we tolerate large variation of the time a human needs to solve a captcha. Our thought process is analogous to the introduction of Turing machines in computer science. Turing machines allow to abstract from special computation devices and thus allow to reason about run time and memory consumption...
of algorithms in a way that is independent of the concrete architecture of the processor. Our aim is to formalize human work by introducing human work units, as opposed to computational work which is usually measured in steps of a Turing machine.

**Definition 6.1.2 (Honest Human Solvability [32, Definition 2]).** We say that a human-machine solver \( \text{CAPT}.\Sigma^{\text{human}} \) controls \( m \) human-work units if it can query its human oracle at least \( m \) times. We say that a captcha system \( \text{CAPT} = (\text{Setup}, W, G, \Sigma^{\text{human}}, \text{Verify}) \) is honest human solvable if for every polynomial \( m = m(\lambda) \) and for any human \( \text{CAPT}.\Sigma^{\text{human}} \) controlling \( m \) human-work units, it holds that

\[
P \geq 1 - \text{negl}(\lambda)
\]

Intuitively, this definition means that a human-machine solver with access to \( m \) human-work units can solve \( m \) captchas, except for a negligible probability of failure. If this is not the case the captcha is not honest human solvable.

Finally we require that captchas are hard for computers to solve without access to a human oracle. Intuitively, an attacker with \( m \) human-work units succeeds in breaking a captcha system if it is able to solve more than \( m \) captchas with non-negligible probability.

**Definition 6.1.3 (Captcha Break [32, Definition 3, 4]).** We say that a ppt adversary \( A \) who has at most \( m \) human-work units breaks security of a captcha system \( \text{CAPT} = (\text{Setup}, W, G, \Sigma^{\text{human}}, \text{Verify}) \) if there exist polynomials \( m = m(\lambda), n = \text{poly}(\lambda) \) and \( \mu(\lambda) \) such that if \( A \) controls at most \( m \) human-work units it holds that

\[
P \geq \frac{1}{\mu(\lambda)}
\]

It is debatable, whether in AI research the concept of a security parameter applies [2]. AI research does not deal with asymptotics and thus it can be argued that problem classes are either solvable or unsolvable, independent of the concrete problem in the problem class. This is in contrast to classical cryptography where it may be feasible to solve certain “small” instances of problems without solving all, e.g., factorization of small integers may be possible, without being able to factorize all integers. Thus, if captchas are either solvable or unsolvable in the real world our definitions can be made even stronger by setting the negligible term in the definition of honest human solvability to zero. Without a tunable security parameter, a captcha is called broken if the attacker has a success probability of 1 of finding solutions without access to a human oracle.
CHAPTER 6. A MINING ALGORITHM BASED ON HUMAN WORK

6.1.3 Proof of Human-work Puzzles

Proof of human-work puzzles (PoH) were first introduced by Blocki and Zhou \[32\]. Their goal was to construct a publicly verifiable proof that some amount of human work has been exercised. Their construction relies on indistinguishability obfuscation \[91\] and thus is currently infeasible.

A proof of human-work in contrast to a captcha has a tunable difficulty parameter and is publicly verifiable. That means that no secret knowledge is needed neither to generate nor to verify a proof of human-work. Especially, the solution does not need to be known beforehand to generate the puzzle as is the case with captchas. The difficulty parameter is necessary to enable its use as a mining algorithm in a blockchain.

**Definition 6.1.4** (Proof of Human-work Puzzle \[32, Definition 6\]). A proof of human-work puzzle system $\text{POH}$ consists of four algorithms ($\text{Setup}, G, \Sigma^{\text{human}}, V$) where:

1. $PP \leftarrow \text{POH.\text{Setup}}(1^\lambda, 1^\omega)$ is a randomized system setup algorithm that takes as input a security parameter $\lambda$ and a difficulty parameter $\omega$ and outputs public parameters of the system $PP$.

2. $x \leftarrow \text{POH.\text{G}}(PP)$ is a randomized algorithm that takes as input the public parameters $PP$ and outputs a puzzle $x$.

3. $a \leftarrow \text{POH.\Sigma^{\text{human}}}(PP, x)$ is a solution finding algorithm that has access to a human oracle. It takes as input the public parameters and a puzzle $x$ and outputs a solution $\sigma$ to the puzzle.

4. $b := \text{POH.\text{V}}(PP, x, a)$ is a deterministic verification algorithm that takes as input the public parameters $PP$, together with a puzzle $x$ and a solution $a$ and outputs a bit $b$ where $b = 1$ if and only if $a$ is a valid solution to the puzzle $x$.

Similar to a captcha we require from PoHs that they are solvable by a human, with a success probability depending on the difficulty parameter $\omega$. Later, this difficulty parameter is used additionally as the difficulty of mining new blocks. Following the notation of Blocki and Zhou \[32\] and Miller et al. \[163\] we define $\zeta(m, \omega) := 1 − (1 − 2^{−\omega})^m$. This describes the probability of finding a valid solution using $m$ queries to the human oracle.

**Definition 6.1.5** (Honest Human Solvability \[32, Definition 7\]). We say that a PoH system $\text{POH} = (\text{Setup}, G, \Sigma^{\text{human}}, \text{V})$ is honest human solvable if for every polynomial $m = m(\lambda)$, and for any honest human-machine solver $\text{POH.\Sigma^{\text{human}}}$ who controls $m$ human-work units, it holds that

$$\Pr \left[ \forall PP \leftarrow \text{POH.\text{Setup}}(1^\lambda, 1^\omega); \ x^* \leftarrow \text{POH.\text{G}}(PP); \ a^* \leftarrow \text{POH.\Sigma^{\text{human}}}(PP, x^*); \ \text{POH.\text{V}}(PP, x^*, a^*) = 1 \right] \geq \zeta(m, \omega) − \text{negl}(\lambda)$$

Further, we require that any adversary that controls too few human-work units succeeds in solving a proof of human-work only with negligible probability.
6.1. BUILDING BLOCKS

Definition 6.1.6 (Adversarial Human Unsolvability [32, Definition 8]). We say that a ppt algorithm $A$ breaks security of the PoH system $\text{POH} = (\text{Setup}, G, \Sigma^{\text{human}}, \mathcal{V})$ if for some polynomials $m = m(\lambda)$ and $\mu(\lambda)$ when $A$ controls at most $m$ human-work units, it holds that

$$P \left[ \forall PP \leftarrow \text{POH.Setup}(1^{\lambda}, 1^{\omega}); \ x^* \leftarrow \text{POH.G}(PP); \ a^* \leftarrow A(PP, x^*); \ \text{POH.V}(PP, x^*, a^*) = 1 \right] \geq \zeta(m + 1, \omega) + \frac{1}{\mu(\lambda)}$$

6.1.4 Multiparty Computation Protocol

As already touched on in Section 2.2.2, multiparty computation protocols (MPC) are cryptographic protocols that allow a set of mutually distrusting parties to collaboratively compute a function with private input values. For example the parties may decide to evaluate some $f(y_1, \ldots, y_n)$, where the input $y_i$ is only known to party $i$. The participants in the protocol do not learn anything beyond their own inputs and the solution $f(y_1, \ldots, y_n)$.

While traditional schemes suffered from severe performance issues, over the last few years, multiple practical solutions that can deal with arbitrary computable functions $f$ have emerged [24, 60, 61, 116, 170]. In our case we require a secure multiparty protocol with $k$ different parties, where $k - 1$ participants can be controlled by an active attacker. An active (malicious) attacker can arbitrarily deviate from any protocol execution in an attempt to cheat. This is in contrast to passive (semi-honest) attackers who try to gather as much information about the underlying inputs and (intermediate) outputs but honestly follow the prescribed steps in the given protocol.

We use MPC as a black box in this chapter, having secret sharing based MPC protocols in mind (such as SPDZ [60]). For this we define an MPC protocol $\text{MPC}$ as a triple of ppt algorithms $(\text{Setup}, \text{Share}, \text{Reveal})$ where:

1. $PP \leftarrow \text{MPC.Setup}(1^{\lambda})$ is a randomized algorithm that takes a security parameter $\lambda$ as input and sets up the protocol by distributing the keys and parameters. It outputs the public parameters for the system.

2. $\langle y \rangle \leftarrow \text{MPC.Share}(PP, y)$ shares the value $y$ among the $k$ participants using a secret sharing scheme such that each of the $k$ participants receives one share. We use $\langle y \rangle$ to denote the vector of secret shares of $y$. Note that the Share algorithm can be executed by one of the participants or any other external party with access to the parameters $PP$.

3. $y \leftarrow \text{MPC.Reveal}(PP, \langle y \rangle)$ reconstructs the value $y$ from its secret shares $\langle y \rangle$. The MPC participants send their secret shares $\langle y \rangle$ to an external party who can then execute $\text{MPC.Reveal}$ and learn the value $y$; consequently being the only party knowing $y$ in the clear.

By abuse of notation, we apply computable functions on secret shares to denote the computation of the secret shares of the result of the function applied to the clear values, i.e., we denote $f(y_1, \ldots, y_n)$ by $f(\langle y_1 \rangle, \ldots, \langle y_n \rangle)$. The clear values are not revealed by this operation. Knowledge of the public parameters may be needed for this computation, but is left out to simplify our notation.
One possible MPC framework for our use is SPDZ \[60\], which consists of a preprocessing and an online phase. The preprocessing phase is independent of the function to be computed as well as of the inputs. In the online phase the actual function is evaluated. The online phase has a total computational and communication complexity linear in the number of participants \(k\). The work done by each participant in SPDZ is only a small constant factor larger than what would be required to compute the function in the clear. Thus, SPDZ provides an efficient framework which satisfies our requirements.

6.2 Our Construction

This section provides a comprehensive description of our construction of a mining algorithm based on proof of human-work. First a short overview is given. Next we construct a novel proof of human-work scheme and then show how it can be applied to substitute proof of work in a blockchain.

6.2.1 Overview

On a high level we are interested in exchanging the proof of work by a proof of human-work. The parties involved in our system are human miners who try to solve the proof of human-work puzzles in order to gain the block rewards, as well as a consortium of \(k\) puzzle generators. To mine a new block, each human miner requests a puzzle for a proof of human-work from the puzzle generators. The puzzle is linked to the transactions the human miner wants to persist, as well as to the current block in the blockchain. Throughout the generation of the puzzle, the solution is unknown to any single party, in contrast to regular captchas. If the human miner does not succeed in solving the puzzle, it can request a new puzzle from the captcha generators. If the human miner succeeds however, it can publish the new block containing the captcha puzzle, its solution, and the transactions. A node which receives a new block can check the transactions and the proof of human-work for validity. It accepts the block if all of these are correct and mining continues on top of the new block.

6.2.2 Our Proof of Human-Work

We give a new instantiation of a proof of human-work puzzle which does not rely on indistinguishability obfuscation \[91\] like the work of Blocki and Zhou \[32\], but instead on MPC. Our construction shown in Figure 6.3 is the first proof of human-work which is feasible and does not involve a trusted third party. In contrast to the work of Blocki and Zhou \[32\], computing the algorithm \(\text{POH}_G\) in our construction needs interaction with a set of captcha generators \(\{C_1, \ldots, C_k\}\). This set can be a fixed consortium of \(k\) parties. However, in Subsection 6.2.4 we elaborate on alternatives for choosing the captcha generators. The assumption of interaction poses no problem for our use case since for mining on a blockchain the miners are required to be online anyway to receive the latest blocks and transactions.

The intuition behind our construction is that the captcha generators collaboratively compute a captcha puzzle using multiparty computation. This computation is done in such a way that each captcha generator has access to
6.2. OUR CONSTRUCTION

Figure 6.2: Simplified Overview of our Proof of Human-work Construction

only a secret share of the solution, but not to the solution itself. Consequently, the solution is unknown to any single party.

Nevertheless, since it is a captcha, the solution can be found by querying the human oracle. If the hash of the solution $\sigma$ of the captcha puzzle is above a difficulty parameter the solution is deemed invalid for the proof of human-work. Thus, for creating a valid proof of human work, one may need to solve multiple captchas depending on the difficulty, until a captcha solution with a small hash is found. This captcha solution then constitutes a proof of human-work.

In order to achieve public verifiability, remember that the generation of a captcha puzzle is a probabilistic algorithm using the solution. If the randomness used in the captcha generation is known it is possible to regenerate the captcha puzzle from the solution. This allows public verification of the solution since the recomputed puzzle can be compared to the given puzzle. In our construction the randomness in the captcha generation is derived from the captcha solution itself.

A standard workflow is shown in Figure 6.2. As a first step the captcha generators $\{C_1, \ldots, C_k\}$ are initialized by executing $\text{MPC.Setup}$ with an appropriate security parameter. Now, suppose a human $M_H$ wants to compute a proof of human-work.

First, the human sets up the public parameters $PP$ for the proof of human-work by executing $\text{POH.Setup}$. The public parameters consist of the public keys of the captcha generators, a difficulty parameter $\omega$, and a security parameter $\lambda$.

Next, the human queries the captcha generators to obtain a captcha by executing $\text{POH.G}$. To this end, he computes public parameters $\text{CAPT.PP}$ for the captcha by running the setup algorithm of the captcha with security parameter $\lambda$. The public parameters $\text{CAPT.PP}$ are distributed to the captcha generators $\{C_1, \ldots, C_k\}$ (Step 1 in Figure 6.2). Each captcha generator $C_j$ chooses a random value $y_j$ and shares it among the other captcha generators, according to the multiparty computation protocol (Step 2 in Figure 6.2). Together, the captcha generators sample a solution $\sigma$ of the captcha by using the sum of their chosen randomness $y_1 \oplus \cdots \oplus y_k$ in the sampling algorithm $\text{CAPT.W}$ together with the public parameters of the captcha $\text{CAPT.PP}$. This solution $\sigma$ is not revealed, but rather stays secret shared between the captcha generators. Thus, no captcha generator knows the solution. From the shared solution $\langle \sigma \rangle$ the shares of the captcha puzzle $\langle Z \rangle$ are computed by the captcha generators using multiparty computation as $\langle Z \rangle \leftarrow \text{CAPT.G (CAPT.PP, } \langle \sigma \rangle; \overline{H}(\langle \sigma \rangle))$ (Step 3 in Figure 6.2).

Here, $\overline{H}$ denotes a slow hash function. Each captcha generator signs its share of the puzzle and sends it to the human $M_H$ (Step 4 in Figure 6.2). We call these signed shares $\tau$. The human then reveals the puzzle $Z$ by executing the
MPC.Reveal algorithm of the multiparty computation protocol. \( M^H \) is now the only person knowing the captcha puzzle \( Z \), and the solution \( \sigma \) is unknown to any single party. The puzzle to the proof of human-work is \( x = (\text{CAPT}.PP, Z, \tau) \).

In order to create the proof of human-work, the human solves the captcha puzzle \( Z \) by executing its captcha solving algorithm. This yields a solution \( \sigma \) to the captcha. If \( H(\sigma) < T_\omega \) then this constitutes a valid proof of human-work. Here, \( H \) is a hash function and \( T_\omega = 2^{n-\omega} \) analogous to Blocki and Zhou \[32\], where \( \omega \) is the difficulty parameter and \( n \) is the bit size of the output of \( H \). If this is not the case, i.e., if the hash of the solution is not small enough, the human has to start again by querying the captcha generators for a new puzzle until he succeeds in solving a captcha with a small solution. The proof of human-work consists of the solution to the captcha puzzle \( a = \sigma \).

To verify the proof of human-work, i.e., to execute POH.V, the public parameters \( PP \), the puzzle \( x \), and its solution \( a \) are needed. The verifier first needs to check if the puzzle has been computed in a correct way, i.e., by the captcha generators. This can be done by checking \( \tau \), the signatures on the shares of the solution which are included in the puzzle \( x \). Next, the verifier checks that the hash of the solution is small enough, i.e., if \( H(\sigma) < T_\omega \). As a final step, the verifier checks that the solution is a correct solution to the captcha. This can be done by simply regenerating a puzzle from the solution and checking equality between the recomputed puzzle and the original puzzle. I.e., it needs to be checked if \( \text{CAPT}.G(\text{CAPT}.PP, \sigma; \overline{H}(\sigma)) = Z \). If any of these three steps fails, the proof of human-work is rejected. Otherwise it is considered valid.

**Construction 6.2.1.** Let \( \text{CAPT} \) be a secure human solvable captcha and \( \text{MPC} \) be a secure MPC scheme initialized with public parameters \( \text{MPC}.PP \leftarrow \text{MPC.Setup}(1^\lambda) \). Let \( H : \{0,1\}^* \rightarrow \{0,1\}^n \) be a hash function. We define \( T_\omega = 2^{n-\omega} \) analogous to Blocki and Zhou \[32\]. We use \( T_\omega \) to scale our difficulty parameter \( \omega \), since \( P(H(\tau) < T_\omega) = 2^{-\omega} \) for a random \( \tau \). We now construct a proof of human-work by defining the following operations.

- \( PP \leftarrow \text{POH.Setup}(1^\lambda, 1^\omega) \) outputs the parameters \( PP \) containing \( \lambda \), \( \omega \), and the public keys of the captcha generators \( C_1, \ldots, C_k \).

- \( x \leftarrow \text{POH.G}(PP) \) is computed by interacting with the set of captcha generators \( \{C_1, \ldots, C_k\} \). First we parse \( \lambda \) and \( \omega \) from \( PP \) locally and compute the public parameters for the captcha as \( \text{CAPT}.PP \leftarrow \text{CAPT.Setup}(1^\lambda) \). These are then given to the captcha generators. Each captcha generator \( C_j \) chooses a secret random value \( y_j \) and uses \( \text{MPC.Share(\text{MPC}.PP, y_j)} \) to distribute shares of its value \( y_j \) among the \( k \) captcha generators. In a next step the captcha generators compute \( \langle \sigma \rangle = \text{CAPT.W(\text{CAPT}.PP; } \langle y_1 \rangle \oplus \cdots \oplus \langle y_k \rangle \rangle \) using \( \text{MPC} \) such that each of the captcha generators now possesses a secret share of the solution \( \sigma \) to the captcha. The solution \( \sigma \) is not revealed but stays in the secret shared domain. Next, the captcha generators compute the puzzle \( \langle Z \rangle \leftarrow \text{CAPT.G(\text{CAPT}.PP, } \langle \sigma \rangle; \overline{H}(\langle \sigma \rangle)) \). The captcha generators each sign their shares \( \langle Z \rangle \) of the puzzle as \( \tau \) which later guarantees that each of the captcha generators \( C_j \) has participated in the protocol.

Finally the captcha puzzle \( Z \) is revealed by computing \( \text{MPC.Reveal(\text{MPC}.PP, } \langle Z \rangle \rangle \) and the proof of human-work puzzle \( x = (\text{CAPT}.PP, Z, \tau) \) is output.
6.2. OUR CONSTRUCTION

- \( PP \leftarrow \text{POH.Setup}(1^\lambda, 1^\omega) \)
  1. Return the parameters \( PP \) containing \( \lambda, \omega \), and the public keys of the captcha generators \( C_1, \ldots, C_k \).

- \( x \leftarrow \text{POH.G}(PP) \)
  1. Parse \( \lambda \) and \( \omega \) from \( PP \).
  2. Compute the public parameters for the captcha as \( \text{CAPT}.PP \leftarrow \text{CAPT.Setup}(1^\lambda) \).
  3. Each captcha generator \( C_i \) chooses a secret random value \( y_i \) and distributes the secret shares \( \text{MPC}.\text{Share}(\text{MPC}.PP, y_i) \).
  4. The captcha generators sample a secret shared solution to a captcha \( \langle \sigma \rangle = \text{CAPT}.W(\text{CAPT}.PP; \langle y_1 \rangle \oplus \cdots \oplus \langle y_k \rangle) \) using MPC.
  5. The captcha generators compute the captcha puzzle corresponding to their solution as \( \langle Z \rangle \leftarrow \text{CAPT}.G(\text{CAPT}.PP, \langle \sigma \rangle; \overline{H(\langle \sigma \rangle)}) \) and \( \tau \) as a signature on their share \( \langle Z \rangle \).
  6. Reveal the captcha puzzle \( Z \) by executing \( Z = \text{MPC.Reveal}(\text{MPC}.PP, \langle Z \rangle) \).
  7. Return the proof of human-work puzzle \( x = (\text{CAPT}.PP, Z, \tau) \).

- \( a \leftarrow \text{POH.}\Sigma_{\text{human}}(PP, x) \)
  1. Parse \( \lambda \) and \( \omega \) from \( PP \).
  2. Parse the puzzle \( x \) as \( (\text{CAPT}.PP, Z, \tau) = x \).
  3. Execute \( \sigma \leftarrow \text{CAPT.}\Sigma_{\text{human}}(\text{CAPT}.PP, Z) \).
  4. If \( H(\sigma) < T_\omega \) return the solution \( a = \sigma \).
  5. Otherwise return \( a = \bot \).

- \( b := \text{POH.V}(PP, x, a) \)
  1. Parse \( \sigma = a \)
  2. If \( H(\sigma) \geq T_\omega \) return \( b = 0 \).
  3. Parse \( (\text{CAPT}.PP, Z, \tau) = x \) and check the signatures and the final shares of the puzzle \( \tau \) for correctness
  4. If \( \tau \) is invalid return \( b = 0 \).
  5. If \( \text{CAPT.G}(\text{CAPT}.PP, \sigma; \overline{H(\langle \sigma \rangle)}) = Z \) return \( b = 1 \).
  6. Return \( b = 0 \).

Figure 6.3: Our novel instantiation of a proof of human-work.
• \( a \leftarrow \text{POH} \Sigma^\text{human}(PP, x) \) computes a solution \( a \) to a proof of human-work as follows. First, the parameters \( \lambda, \omega \) are parsed from \( PP \). The puzzle is parsed as \( (\text{CAPT}.PP, Z, \tau) = x \). Then the captcha solving algorithm is queried \( \sigma \leftarrow \text{CAPT}.\Sigma^\text{human}(\text{CAPT}.PP, Z) \). If \( \mathcal{H}(\sigma) < T_\omega \) we return the solution \( a = \sigma \). Otherwise \( a = \bot \) is returned.

• \( b := \text{POH}.V(PP, x, a) \) first parses \( \sigma = a \) and checks if \( \mathcal{H}(\sigma) < T_\omega \). If that is not the case \( b = 0 \) is returned. Otherwise we parse \( (\text{CAPT}.PP, Z, \tau) = x \) and check the signatures and the final shares of the puzzle \( \tau \) to ensure that the puzzle has been generated in a correct way, i.e., by the captcha generators \( \mathcal{C}_i \), and not by anyone else. This is possible, since the public keys of the captcha generators needed for the verification of the signatures are contained in \( PP \). If \( \tau \) is invalid, we return \( b = 0 \). As a third step we need to ensure that the solution \( \sigma \) is a valid solution to the captcha. This can be done by checking if \( \text{CAPT}.G(\text{CAPT}.PP, \sigma; \Pi(\sigma)) = Z \). If that is the case, return \( b = 1 \), otherwise return \( b = 0 \).

**Theorem 6.2.1.** If our construction is instantiated with a secure and honest human solvable captcha, then the resulting proof of human-work is honestly human solvable and adversarial human unsolvable under the assumption that at least one of the \( k \) captcha generators \( \mathcal{C}_1, \ldots, \mathcal{C}_k \) is honest.

**Proof.** The basic idea of the proof is to reduce the security of the proof of human-work to the security of the underlying building blocks.

Honest human solvability of the proof of human-work follows from our definition of \( T_\omega \) and since the captcha is honest human solvable. Further, the puzzle has been generated correctly, since due to our assumptions that at least one of the \( k \) captcha generators is honest our MPC protocol will compute the puzzle correctly. The signed shares of the puzzle \( \tau \) guarantee that the captcha was computed by the correct parties.

To show adversarial human unsolvability, we need to construct an attacker \( \mathcal{A}_{\text{CAPT}} \) on the captcha mechanism, given an attacker \( \mathcal{A}_{\text{POH}} \) on the proof of human-work. We can assume that if at least one of the \( k \) captcha generators is honest, the solution to a captcha does not leak in the generation, due to the security properties of our MPC protocol. Whenever \( \mathcal{A}_{\text{CAPT}}(\text{CAPT}.PP, Z_1, \ldots, Z_n) \) is called it can create a set of proof of human-work puzzles whose solutions \( a \) correspond to solutions \( \sigma \) of the captcha, whenever \( \mathcal{H}(\sigma) < T_\omega \). For this \( \mathcal{A}_{\text{CAPT}} \) needs to set up the public parameters \( PP \) of the proof of human-work and construct \( \tau_i \) by secret sharing the puzzles \( Z_i \), thereby simulating the captcha generators. This can be done, since the public keys of the captcha generators are contained in the public parameters \( PP \) which the attacker can set accordingly. Thus the generated \( \tau_i \) and the \( Z_i \) are valid. The attacker \( \mathcal{A}_{\text{CAPT}} \) can then execute \( \mathcal{A}_{\text{POH}}(PP, x_i) \) on the proof of human-work, where \( x_i = (\text{CAPT}.PP, Z_i, \tau_i) \).

Since for a random value \( r \) we know that \( P(\mathcal{H}(r) < T_\omega) = 2^{-\omega} \), we can conclude that if the success probability of \( \mathcal{A}_{\text{POH}} \) is larger than \( \zeta(m + 1, \omega) + \frac{1}{\mu(M)} \), then the success probability of \( \mathcal{A}_{\text{CAPT}} \) is non-negligible.

### 6.2.3 Block Generation

In this section we describe a design of a blockchain which is based on proofs of human-work. We call the resulting mining process uMine. In order to mine
6.2. OUR CONSTRUCTION

a new block $B_i$, a human miner $\mathcal{M}^H$ needs access to the previous block $B_{i-1}$. Further it needs to have a set of transactions $Tx_i$, which it wants to persist in the new block $B_i$.

To mine a new block, first, the algorithm $\text{POH.Setup}(1^\lambda,1^\omega)$ is run in order to generate the public parameters for the proof of human-work. The security parameter $\lambda$ is globally fixed but the difficulty parameter $\omega$ needs to be adjusted dynamically to ensure a stable block creation rate. As explained in Section 2.2.1 in Bitcoin the difficulty parameter is adjusted every 2016 blocks such that the expected block generation interval is 10 minutes, assuming no changes in the global mining power. Although it is unknown if these parameters are optimal, there is insufficient research covering the choice of parameters and thus we see no reason to deviate from them.

The captcha generators $\{C_1, \ldots, C_k\}$ are initialized by computing $\text{MPC}.PP \leftarrow \text{MPC.Setup}(1^\lambda)$. After generating the public parameters for the proof of human-work the human miner $\mathcal{M}^H$ contacts the set of captcha generators to receive a proof of human-work puzzle $x_i$ for the new block $B_i$ as we will explain in the following.

The human miner splits the hash of the transactions $\mathcal{H}(Tx_i)$, as well as the hash of the current block $h_{i-1} = \mathcal{H}(B_{i-1})$ into secret shares $\langle \mathcal{H}(Tx_i) \rangle$, $\langle h_{i-1} \rangle$ which are distributed to the captcha generators. Each captcha generator computes the captcha parameters as $PP \leftarrow \text{CAPT.Setup}(1^\lambda)$. Together they compute a random captcha solution in the secret shared domain as follows.

First, each captcha generator $C_j$ chooses a secret input $y_j$ uniformly at random. This secret randomness is shared among the captcha generators by computing $\langle y_j \rangle = \text{MPC.Share}(\text{MPC}.PP, y_j)$. The shared randomness is used to compute the secret shared random captcha solution as $\langle \sigma_i \rangle = \text{CAPT.W}(PP; \langle y_1 \rangle \oplus \cdots \oplus \langle y_k \rangle)$. This way, none of the captcha generators knows the solution $\sigma_i$.

To be able to use our proof of human-work construction from above in a blockchain we need to include a reference to the previous block $h_{i-1} = \mathcal{H}(B_{i-1})$ and the new transactions $Tx_i$ in the puzzle. Otherwise, if an already persisted transactions in the blockchain is modified the proof of human-work is still valid, and thus integrity of persisted transactions cannot be guaranteed. In order to connect the hash of the previous block and the transactions with the puzzle, the captcha generators compute their secret shares of the captcha puzzle $Z_i$ given their shares of the solution $\langle \sigma_i \rangle$ by $\langle Z_i \rangle = \text{CAPT.G}(PP, \langle \sigma_i \rangle; \mathcal{H}(h_{i-1}, \mathcal{H}(Tx_i), \langle \sigma_i \rangle))$. Note that here the hash of the previous block $h_{i-1}$ and the current transactions $\mathcal{H}(Tx_i)$ are included in the randomness of the puzzle generation in contrast to our more general construction of a proof of human-work above. At this stage, each captcha generator has a share of a captcha puzzle $\langle Z_i \rangle$ where the solution is effectively unknown to any single party.

Next, the captcha generators send their signed final shares of the captcha puzzle to the human miner $\mathcal{M}^H$ who assembles them as a proof of human-work $x_i = (\text{CAPT}.PP, Z_i, \tau_i)$, where $Z_i = \text{MPC.Reveal}(\text{MPC}.PP, \langle Z_i \rangle)$ is the revealed captcha puzzle. Here, $\tau_i$ is the set of the final signed shares of the puzzle from the multiparty protocol run. It is used to prove that each captcha generator

---

We explain our protocol using classical secret sharing based MPC, where a dealer distributes shares of the input and a set of nodes computes on these shares. We hope this makes our explanations more clear. In a practical implementation we suggest the use of SPDZ which is a highly optimized variant thereof.
participated in the protocol and thus guarantees that the puzzle has been generated in a correct way.

The human can now try to solve its proof of human-work $x_i$. If the hash of the solution to the encapsulated captcha is too big, that is, when $H(\sigma) \geq T_\omega$, the human requests another proof of human-work puzzle from the captcha generators. If the human eventually succeeds to find a solution $\sigma'_i$ to the proof of human-work, it can locally verify its solution, by checking if $Z_i = \text{CAPT.G}(\text{CAPT.PP}, \sigma'_i; H(h_i-1, H(Tx_i), \sigma'_i))$. If that is the case, it can publish the new block $B_i$ containing the captcha puzzle $x_i$, its solution $a_i = \sigma'_i$, as well as the transactions $Tx_i$, and a reference to the previous block in form of a hash $h_i$ as can be seen in Figure 6.4.

Each receiving node verifies the solution to the captcha, by running $\text{POH.V}(\text{PP}, x_i, a_i)$. More specifically, the captcha puzzle $Z_i$ is recomputed from its solution and it is examined if this leads to the same $Z_i$, i.e., if $Z_i = \text{CAPT.G}(\text{CAPT.PP}, \sigma_i; H(h_i-1, H(Tx_i), \sigma_i))$. Additionally the signed shares of the puzzle $\tau_i$ are checked for correctness of the signature to guarantee that the puzzle was created by the correct parties. Further it is examined if $H(\sigma) < T_\omega$ holds.

Beyond these steps of verification of the proof of human work for usage in a blockchain, the difficulty parameter $\omega$ and the validity of the transactions in the new block is checked, as in Bitcoin.

If any of these checks fails, the new block is discarded and mining continues on top of the old block $B_{i-1}$. Otherwise the human miners can continue to generate blocks on top of $B_i$.

If one of the captcha generators is malicious it may abort the generation of the puzzle to the proof of human-work, thus preventing that new blocks can be mined by a proof of human-work. To remedy this situation we additionally allow blocks to be mined by proof of work as in Bitcoin. However, in order to keep the advantages of the mining with human work, we use a distinct difficulty parameter from the proof of human-work. The difficulty parameter of the proof of work is chosen in such a way that mining a block using proof of work is significantly harder than mining with proofs of human-work. This ensures that the mining process is dominated by proof of human-work and proof of work is only used as a fallback mechanism.

### 6.2.4 Choosing the Captcha Generators

One important design consideration is the choice of the captcha generators. We provide two alternatives, namely a static consortium and a dynamic choice of the captcha generators based on the randomness contained in the blockchain.
Static Consortium

In the most simple case we can assume a consortium of $k$ fixed entities. If some of them are not online, no proof of work puzzles will be generated. In this case proof of work can be used as a fallback mechanism as explained above. Thus, if the captcha generators are offline, our system collapses to proof of work mining. Due to our use of SPDZ\cite{60} we can tolerate up to $k-1$ cheaters. However, if all $k$ parties collude, they may be able to generate captchas where they already know the solution and thus mine faster than any human miner, achieving a significant financial gain. We can remedy this situation by providing incentives for captcha generators to expose collisions and then punish the colluding parties and reward the traitor (see next paragraph). Intuitively this provides incentives for the traitor to reveal collusion, thus preventing the formation of collisions in the first place. For this to work the captcha generators additionally need to publish a signed commit on their shares of the solution $\sigma$. These can be included in the information used to verify that the puzzle was generated by the correct parties $\tau$ which consists of the signed shares of the puzzle.

The Traitor Reward Protocol  The traitor reward protocol has two rounds. In a first round any captcha generator can claim that the captcha generators colluded by publishing the particular shares of the puzzle solution of each captcha generator. If collusion influenced the creation of the current block, at least the miner of the block knows this information.

Other parties are allowed to chime in with their claims of collusion by also publishing commits to the respective shares of the captcha generators. After a fixed timespan the first round ends and the second round starts.

In the second round the captcha generators have to reveal their commitments on their shares of the solution. If none of them complies within a fixed time period, collusion can be assumed. The claims of the supposed traitors are handled in the order of their arrival. Note that since we are in a distributed setting there is no global time. But we only want to reward some traitor to deter collusion and not necessarily the first traitor, so this poses no problem. The claimed shares of the solutions are compared with the real shares of the captcha generators and if they coincide, collusion has occurred. In this case, the witness of collusion can be persisted in the blockchain as a regular transaction and the block reward of the fraudulent block is granted to the traitor.

The time periods for the traitor reward protocol need to be chosen appropriately and the block reward needs to be locked for a fixed amount of time to prevent that it is already spent before collusion claims can be handled.

Consequently each captcha generator can choose to either collude or not collude and orthogonally to betray the other nodes or refrain from doing so. The incentives need to be designed such that not colluding and not claiming to betray has to be the strictly dominant strategy in a game-theoretic sense, because this is the behavior we want to support in the captcha generators. Colluding and

\footnote{It may be the case that the $k$ colluding parties decide to reveal the solution to the proof of human-work by MPC, such that no one knows the partial solutions of the other captcha generators. Even then $k-1$ nodes can collude to reveal their particular shares, recompute the missing share of the last captcha generator, and claim betrayal. This increases their reward in contrast to not betraying the last captcha generator. For our cases it is irrelevant if a subset of captcha generators or only a single one claims betrayal. Though for the sake of simplicity we assume a single traitor.}
not betraying the others needs to be a strictly dominated strategy, such that colluding nodes gain a profit from betraying the other conspirators. However the profit needs to be smaller than if there would have been no collusion at all. Otherwise it may be rational to stage betrayal and share the traitor reward with the other nodes.

It is interesting to note that under the assumption of rational actors the traitor reward protocol will never be executed. Thus, collusions are prevented by the existence of the traitor reward protocol and not by its execution.

Dynamic Consortium

The second alternative is to choose the $k$ miners dynamically. We choose the $k$ last miners who found a block to serve as captcha generators. Since this choice is essentially random, they collude only with small probability. The captcha generators can be financially rewarded for staying online and generating captchas in the form of an additional block reward in the newly mined blocks.

A choice of $k$ among the last $n$ miners who found a block as captcha generators is also conceivable. This decreases the probability that they are offline, but increases the probability of collusion.

So the two remaining problems we face in the scenario with a dynamic choice of captcha generators are that some of the captcha generators are offline and hence no new proof of human-work puzzles are generated, or that all $k$ of them collude.

One naive approach to the case where some of the captcha generators are offline is to choose miners from older blocks as captcha generators. However, this opens the door to attacks on miners after mining a block, to achieve that other captcha generators are chosen. Consequently this approach of choosing different captcha generators does not work immediately. We decided for a different solution, namely if some of the captcha generators are offline and no new puzzles can be generated we allow the mining of blocks with a proof of work using a high difficulty parameter as fallback option as explained before in our construction. The difficulty of finding a block with a proof of work should be significantly higher than solving a proof of human-work, such that it is only used as a fallback option. With this design choice we can guarantee liveness of the chain even if all captcha generators are offline.

Concerning the second problem, when all $k$ captcha generators collude, they can share the solutions and quickly mine blocks without proof of human-work, thereby splitting the block rewards among themselves. As in the case of a static consortium, traitors are incentivized by a reward to expose the collusion and the colluding parties are financially punished. The traitor reward protocol is the same as in the case of a static consortium explained above. Again, the captcha generators need to publish a signed commit on their shares of the solution $\sigma$ for the traitor reward protocol to allow for verification of the witness of collusion.

6.3 Security

In the following we show how our modifications to the mining algorithm affect the security properties of the blockchain.

Additionally to the trust assumptions in usual blockchain systems, we need that at least one of the $k$ captcha generators is honest due to our use of
6.3. SECURITY

SPDZ [60]. For the security of our proof of human-work scheme we require that it is instantiated with a secure captcha system. If this is not the case and the captcha can be solved without human work, our uMine construction will not lose its functionality but instead degrade to a form of proof of work. This is the case, since the captcha can then be solved by computers alone. The difficulty parameter of the captcha adjusts, and mining continues without human intervention. Other than the use of a secure captcha we do not impose any additional trust assumptions.

Since our main focus is to substitute the proof of work by an environmentally friendly alternative, some of the attacks in Bitcoin also affect our scheme. In particular our construction is vulnerable to 51% attacks and eclipse attacks [107]. Although, to successfully pull off a 51% attack an attacker needs to be in charge of more than 50% of the human work units in the system instead of more than 50% of the computational resources, as in Bitcoin. However, we do not introduce any new security vulnerabilities under our assumptions.

In the following we discuss the infeasibility of selected attacks.

**History Rewriting** If old transactions are changed in the blockchain, the solution to the captcha is invalidated, since the transactions are also used in the generation of the puzzle $Z_i$ from the solution $\sigma_i$ as follows:

$$Z_i = \text{CAPT.G}(PP, \sigma_i; H(B_{i-1}, H(Tx_i), \sigma_i)).$$

Finding two sets of transactions $Tx_i \neq Tx'_i$ which yield the same puzzle $Z_i$ for the solution $\sigma_i$, implies that $H(B_{i-1}, H(Tx_i), \sigma_i) = H(B_{i-1}, H(Tx'_i), \sigma_i)$, since $\text{CAPT.G}$ is collision-free in its randomness by assumption as explained in Section 6.1.2. So, an attacker would have to find a collision in the slow hash function $H$ to successfully change the old transactions which is assumed to be infeasible. Note that changing the solutions $\sigma_i$ does not yield an attack, since they are referenced in the next block.

Thus, the only way left to change transactions already persisted in the blockchain would be to split the chain after this block and redo the human work. This is only possible if an attacker controls more than 50% of the human resources in the network.

**Transaction Denial Attack** In a transaction denial attack, the attacker tries to prevent a transaction from being confirmed. If the attacker is a human miner, it can only succeed if his chain grows faster than the chain containing the transaction it wants to censor. This is exactly the case if it has more than 50% of the human power in the system. As soon as that is not the case anymore, the transaction will be included in the blockchain.

If the attacker is one of the captcha generators instead, it cannot prevent inclusion of the transaction in the chain, by not serving a captcha to the miners which want to include that specific transactions. That is due to the fact that the captcha generators do not see the transactions but only their hash. Identifying if a transaction is included in a set of transactions, given only the hash of the set is infeasible.

Thus, an attacker is unable to target specific transactions for denial.
Bruteforcing of Solutions Since practical captchas normally do not have much entropy—image based captchas consists of up to 12 characters—an attacker $A$ may have the idea to simply brute-force the solution $\sigma_i$ to a puzzle $Z_i$. This would possibly allow $A$ to mine a block without spending human labor on it.

While we can almost never fully prevent brute-forcing, our use of a slow hash function impedes the attempts of the attacker. To brute-force a solution, an attacker needs to guess a $\sigma'_i$, compute $Z'_i = \text{CAPT}.G(PP, \sigma'_i; \overline{H}(B_{i-1}, H(Tx_i), \sigma'_j))$ and then check if $Z'_i = Z_i$. I.e., $A$ needs to evaluate a slow hash function for each guess, which is expensive.

6.4 Notes on Implementing our Scheme

While in theory it is possible to evaluate any computable function in a multiparty fashion, in practice this is not as straightforward. Most multiparty computation frameworks allow the evaluation of gates of a circuit only. Consequently, we need to represent a secure captcha generation algorithm as a circuit in order to implement our proof of human-work scheme.

We suggest to use SPDZ due to its high performance. SPDZ allows to describe the function for the multiparty computation protocol in a language which is heavily inspired by Python. In particular, it supports the basic data types and syntax of Python. The SPDZ framework can compile this high level language to virtual machine code for the SPDZ virtual machine. The virtual machine code is further compiled to an arithmetic circuit before it is evaluated. Thus, in order to implement our proof of human-work scheme, instead of representing a secure captcha generation algorithm as a circuit it suffices to represent it in a subset of Python.

We tried to implement our proof of human work scheme using the Python package captcha version 0.2.4 which can be used to create visual captchas. However, we did not succeed in implementing our scheme, since the captcha package uses many Python language constructs which are not supported in SPDZ. In particular, the font parsing, as well as the image processing is not done in Python, but in a C library and thus is unsupported by SPDZ.

In the following, we describe how our scheme can be implemented. Our description of a secure captcha is based on the Python captcha package.

As described in Figure 6.3, in order to generate proof of human work, i.e., to execute $\text{POH}.G(PP)$, the scheme needs to be set up first. This is straightforward. The public parameters may contain the length of the captcha, as well as the set of allowed characters, and the typeface used. Next, each captcha generator $C_j$ chooses a secret random value $y_j$ and distributes the secret shares $\text{MPC}.\text{Share}(\text{MPC}.PP, y_j)$, which is fairly trivial and can be executed directly with SPDZ. After the secret shares are distributed, the captcha generators sample a secret shared solution to a captcha as $\langle \sigma \rangle = \text{CAPT}.W(\text{CAPT}.PP; \langle y_1 \rangle \oplus \cdots \oplus \langle y_k \rangle)$. The algorithm $\text{CAPT}.W$ returns as output a solution to a captcha, i.e., a string whose length depends on the length contained in the public parameters. The randomness for this sampling process depends on $\langle y_1 \rangle \oplus \cdots \oplus \langle y_k \rangle$. In order to implement $W$, a secure pseudo random number generator is used whose seed is set to $\langle y_1 \rangle \oplus \cdots \oplus \langle y_k \rangle$. This is hard to do in
6.5. RELATED WORK

practice, since it is not clear how to represent a secure pseudo random number generator as an arithmetic circuit.

As a next step, a captcha puzzle $Z$ with the solution string $\sigma$ is computed as $(Z) \leftarrow \text{CAPT}.G(\text{CAPT}.PP, (\sigma); \overline{H((\sigma)))}$. Representing the slow hash function $\overline{H}$ as an arithmetic circuit should be easy. However, computing $\text{CAPT}.G$ is difficult. In practice $\text{CAPT}.G$ takes the string $\sigma$ and represents it as an image in the typeface given in $\text{CAPT}.PP$. Next, various transformations are applied on the resulting image such as rotation, warping, adding noise curves and noise dots. These transformations and their parameters are determined by $\overline{H((\sigma))}$. While these steps are trivial, implementing them as an arithmetic circuit is not. SPDZ currently has support for neither parsing TrueType fonts nor image processing. In order for our proof of human-work scheme to be implemented, it is necessary to implement these features as an arithmetic circuit.

For convenience it is possible to use a lookup table for individual letters of a fixed font instead of determining the typeface through the public parameters of the captcha. Images can be represented as vectors of numbers. However, for their transformations a separate library is desirable, as multiparty computation on images constitutes a reusable building block for other protocols. As the parameters for the transformation are determined by a pseudo random number generator whose seed is set to $\overline{H((\sigma))}$, again we need a pseudo random number generator represented as arithmetic circuit.

The further steps in creating the proof of human-work, i.e., signing the shares and revealing the puzzle are trivial.

In summary the problems concerning the implementation of our proof of human-work scheme are restricted to the representation of the captcha generation algorithm as an arithmetic circuit. After tackling this step, the SPDZ protocol guarantees execution as a multiparty protocol with only a constant overhead as opposed to normal execution.

6.5 Related Work

There is a series of related work which suggests an alternative to Bitcoin’s wasteful proof of work as already discussed in Section 2.2.2. The most famous among these approaches is probably proof of stake [26, 122], where the scarcity used to power the blockchain is the underlying currency itself. In proof of stake the miner of the next block is chosen pseudorandomly among the set of all miners. The probability of a miner being chosen to create a new block is dependent on its wealth which can lead to an undesirable “rich get richer” scenario.

Other protocols such as Algorand [94] do not provide incentives for the participants.

A spiritual predecessor to our work is HumanCoin [32] who introduced the notion of a proof of human-work. However, HumanCoin is based on a proof of human-work based on indistinguishability obfuscation [91], a cryptographic principle where no construction is known yet. Thus, HumanCoin is currently infeasible.

Their construction of a proof of human-work, and consequently HumanCoin requires a trusted setup phase for the generation of the obfuscated programs which are used to generate the captchas without revealing the solutions.
If the unobfuscated programs are known, miners can generate puzzle-solution pairs to the proof of human-work without spending any human work by running the puzzle-generation in the clear. In contrast, in our work we are able to verify that the puzzles have been created in such a way that no-one knows the solution, by publishing and verifying the signed final shares of the captcha generators $\tau$.

In HumanCoin, collision-freeness of the captcha puzzle generation algorithm is not stated, but if this is not assumed their scheme is trivially insecure. Their PoHs are generated by $x_i = \text{CAPT.G}(PP; \mathcal{H}(Tx_i, h_{i-1}))$, where $h_{i-1}$ is the hash of the previous block. If there is no collision resistance in the randomness, old transactions can be changed without changing the puzzle, thus invalidating the integrity of the transactions stored in blockchain.

HumanCoin does not implement any countermeasures against brute-forcing the solution to the proof of human-work from its verification function in contrast to our use of a slow hash function.

In contrast to HumanCoin our PoH puzzle generation phase is online and requires $k$ captcha generators. However, this poses no problem, since to mine new blocks a miner has to receive new transactions and blocks and thus is required to be online anyway.

6.6 Conclusion

In this chapter we have introduced uMine, an energy-efficient alternative to proof of work mining which utilizes human workers. Thereby, we provide an alternative approach to mitigating the effects of the mining process on global energy consumption.

Our construction is based on a novel instantiation of proofs of human-work which relies on MPC, thereby answering an open question of Blocki and Zhou [32] whether proofs of human work without indistinguishability obfuscation are possible. Our instantiation of a proof of human-work scheme is introduced as a separate building block and thus may find applications beyond cryptocurrencies.

We think that our game theoretic approach to preventing collusions in multiparty computation settings by rewarding traitors may be interesting on its own. Future work could answer the question if this building block can be generalized and applied to other multiparty computation scenarios to uncover collusion.

Another obvious question raised by our work is the social and economical impact on cryptocurrency mining. Our uMine system may share similarities with early manual accounting systems, whose bookkeepers were financially compensated. Additionally, our system decentralizes the ledger and provides anyone with the opportunity to be an accountant, provided it accepts the remuneration. We leave the social and economical implications of our work as an open question.
The Economical Impact on Routing in Payment Channel Networks

A different scalability problem of blockchains than the energy consumption discussed in the previous chapter is the transaction throughput. Current blockchain systems maintain their global state by high levels of data replication. As each transaction needs to be stored at each peer, current blockchains do not scale with respect to transaction throughput, due to the coordination and communication overhead for replication. As discussed in the section about scalability in Section 2.2.2, Bitcoin can only process up to 7 transactions per second\(^\text{[57]}\) globally which is clearly not sufficient for a global payment network. Other blockchains do not achieve significantly higher figures. While our decentralized storage system does not need the transaction throughputs of mainstream payments processors, we can envision that 7 transactions per second are still too slow.

Off-chain state channels\(^\text{[156]}\) are supposed to overcome these technical scalability limitations by employing a special overlay network with fast payment confirmation and only sporadic settlement of netted transactions on the blockchain. The approach builds upon the idea of setting up bilateral connections, so-called channels, between pairs of nodes of a blockchain network. The blockchain is only used to open and close the channels and resolve disputes about the final state of the sequence of transactions.

Off-chain state channels are useful in the context of bilateral payments that can take place within such a channel. These state channels concerning payments are called payment channels. However, states in state channels are not limited to payments but can include arbitrary states, only limited by the expressiveness of the blockchain. In our case, it is possible to settle storage contracts in state channels.
Considering payment channels, signed transactions are sent through the channel between the parties in both directions and a bilateral ledger is updated with every transaction. Only the final state, i.e., the result of netting all transactions of that bilateral ledger, is persisted in the blockchain if one of the participants decides to close the channel. Payment channels lead to a higher transaction throughput, since only netted transactions are written to the blockchain instead of intermediate states.

Payments between nodes that have no direct bilateral connection can be routed through the network of bilateral channels between other nodes. In previous literature [64, 101, 187, 188, 222], payment channels were considered mostly from a technical perspective in terms of saving bandwidth, preventing blockchain bloat, and enabling cheating resistance. However, the influence of the economic perspective on usability and feasibility of payment channels was at best considered as an afterthought. Routing in payment channels is subject to economic routing constraints that limit decentralized scalability and are currently not well understood. This is similar to the case of IPv6 where development focussed on engineering aspects instead of how to incentivize protocol adoption, leading to the situation that IPv6 is not widely used in practice [140].

All current payment channel protocol proposals require an initial deposit by both parties who set up a bilateral channel. This technical requirement results in capital binding at the opening of a payment channel for the duration of the lifetime of the channel. Capital binding economically implies the necessity for remuneration. Additionally, participants in the network are likely to set individual routing fees for their forwarding of transactions. Consequently, all channel capacities along the route need to support the transaction amount, together with routing fees. Currently, the dynamics emerging from these economic constraints are not taken into account in payment channel design and it is unclear how such a network will evolve.

In this chapter, we model the economic incentives for participants in a generic payment channel network. Note that this provides a slight digression from our previous focus on a storage network. The main system of this thesis deals with a storage network and its tokens are primarily designed for remuneration of storage providers, and not as a global currency system. However, in order to evaluate the economic impact on routing decisions, we need data on spending habits. Thus, it makes more sense to use a distribution of amounts from real public transaction data [93], in contrast to guessing the spending habits for our storage network, where the frequency and amounts of transactions are largely unknown. With this in mind, we provide the first formal model of payment channel economics and analyze how the cheapest path can be found. Additionally, our simulation assesses the cost of routing in a payment channel network.

We find that even for small routing fees, sometimes it is cheaper to settle the transaction directly on the blockchain. Consequently, we argue that in future designs of payment channel networks the economic incentives should receive major attention, since they greatly influence adoption and the topology of the network.

Since currencies are subject to various regulations regarding know your customer, anti-money laundering, and counter-terrorist financing it makes sense to discuss a payment channel scenario without privacy protections. Note however,
7.1 BACKGROUND

that there are proposals of anonymous payment channels such as Bolt \cite{101}, Z-channels \cite{238} or the work of Malavolta et al. \cite{153} as previously discussed in the subsection about scalability in Section \ref{section:scalability}.

Large parts of this chapter have been previously published at SERIAL \cite{77}. This was a joint work between the authors and my part of this publication was mainly the formalization of the economic constraints, together with the design of the linear program for routing. Nevertheless, there has been much collaboration and it is hard to strictly separate the contributions of the different authors.

Contribution Our contributions are as follows:

- We show the impact and the resulting challenges of economic considerations on the routing process in a payment network.
- Two approaches for computing a cheapest path for a transaction are given, assuming a global view of the network. These consist of a linear program and a modification of Dijkstra’s algorithm.
- We implement a simulator for payment channel networks and perform measurements to show the magnitude of the economical impact on transaction routing.

Roadmap The remainder of this chapter is structured as follows. Section \ref{section:background} explains the necessary background of payment channels. Section \ref{section:challenges} highlights the main challenges of routing in payment channel networks. We discuss ways to compute the cheapest route in Section \ref{section:cheapest_path}. In Section \ref{section:economic_implications} we evaluate the economic implications of routing mechanics in payment channels relying on a simulation. Section \ref{section:conclusion} concludes this chapter.

7.1 Background

For Bitcoin, two main proposals of bidirectional payment channels currently exist: duplex micropayment channels (DMC) \cite{64} and the Lightning network \cite{187}. For Ethereum there is a proposal for a payment channel network called the Raiden network \cite{222}. We do not focus on their technical details, as we are concerned with routing in these networks. A good overview of the technical details of payment channels is given by McCorry et al. \cite{156}.

These mechanisms consist of three phases:

Setup Phase: In the first phase, two parties set up a bilateral payment channel between them. Either party needs to allocate an amount of coins to prove that it is able to pay the subsequent transactions. This amount is called the capacity of the channel. It can differ in the two directions and is the maximum total amount that can be routed in each direction.

Trading Phase: In the second phase, each of the two parties can send multiple signed transactions to the other party, where subsequent transactions have priority over the previous ones. This enables to retain the strong security properties of the blockchain since no party can cheat by committing an
CHAPTER 7. ROUTING IN PAYMENT CHANNEL NETWORKS

older transaction to the ledger. In DMC, priority of later transactions is achieved by a timelocking mechanism which permits the old transaction to be persisted in the blockchain before the newer transaction. In Lightning, the priority of transactions is implemented by continuously revoking the old transactions. If one party includes a revoked transaction in the blockchain, a preimage of a hash is revealed. This preimage enables the other party to issue a penalty to claim all bitcoins in the channel.

Settlement Phase: The third phase of current payment channel proposals consists of closing the channel. This can be necessary if the channel is not needed anymore or if the initial capacity of the channel is depleted, i.e., one participant has received all the money the other party has initially deposited and thus no further transactions are possible in this direction. Depending on the implementation, capacity can be replenished instead of closing the channel and opening it again.

DMC and Lightning, are running on an unspent transaction output (UTXO) based blockchain like the Bitcoin blockchain. In contrast, the Raiden Network relies on the account state-based Ethereum blockchain. The three general phases described above also occur in Raiden, however, the technical implementation differs significantly. In UTXO based blockchains like Bitcoin, smart contracts are special output conditions specified in program code which is attached to transactions. In Raiden, a channel is a smart contract that contains the channel setup properties and is represented by a separate account which is created when the smart contract code is deployed on the Ethereum blockchain. Nonetheless, the functionality provided by Raiden is comparable to DMC and Lightning, at least if compared on the abstract level of investigation in this work.

Raiden additionally implements auxiliary smart contracts which are deployed to the public Ethereum blockchain to support, for example, the tracking of existing channels and hence facilitate the routing of a payment over multiple hops. Raiden provides additional features which facilitate recovering the state of off-chain channels when restarting a node. It is worth noting here, that nodes involved in a transaction must be online during the time the channel is open. If this is not the case, one party can cheat by releasing an old transaction, thereby closing the channel. The counterparty cannot react to this by releasing a later transaction with priority over the first transaction, since the counterparty is offline. The closing of channels can, however, be delegated to third parties who monitor the closing of channels on behalf of the party that is offline when the opposite party closes the channel.

All three—Raiden, DMC, and Lightning—support routing of transactions over multiple hops. A user A can send a payment via a node B to C if payment channels are set up between A and B, and between B and C. To achieve this, A sends its payment to B and then B sends the corresponding amount to C, possibly deducting a routing fee. These two transactions use a mechanism called hashed timelocks (HTLC) to enforce their dependency. Thus, either none or both of the payments are processed.

Since each node needs to have sufficient capacity for its payments, routing a payment of size $X$ along each of the $n-1$ edges between $n$ nodes requires each channel to have a capacity of at least $X$ units of the underlying (crypto)currency.
Consequently, in total $O(Xn)$ capital needs to be locked to route a transaction of size $X$ over $n$ hops. In addition to the capacity constraints, economic routing considerations arise, since intermediate nodes may demand a premium for their cost of capital binding and operating a node in the network. Note however, that the premium cannot be set arbitrarily high, since if the routing fee exceeds the transaction fee on the blockchain, a rational sender uses the blockchain for settlement instead of the payment channel. Nevertheless, senders aim to minimize their cost of transferring their transactions instead of the number of hops as in classical routing algorithms.

To our best knowledge, optimal routing algorithms in payment channel networks are currently a neglected problem and demand further research. While the work of Prihodko et al. [188] provides a first step in this direction by proposing a routing algorithm for establishing routes between transaction partners, it fails to take the full financial dimension into account.

### 7.2 Challenges

Beyond problems of traditional routing scenarios, like changing topologies, payment channel networks need to deal with additional economically induced challenges.

**Dynamic Channel Capacities:** First, the state of a payment channel is changing with every transfer due to updated balances. Hence, the capacities of the network graph’s edges are highly dynamic as a result of the design of payment channels. This affects the transaction amount which can be routed over a channel. One consequence of dynamic capacities is that after a first transaction has been routed, the same path may become invalid for a second, identical transaction due to channel updates. A common assumption which remedies this problem is that there will be roughly equal amounts of payment in both directions at every node, thus leading to an equilibrium in the channels. However, this assumption is unrealistic, as we do not know of any real world scenario where this assumption holds.

**Channel Depletion:** Dynamic channel capacities can lead to the capacity of a channel being exhausted in one direction. In this case, no more transactions can be routed along the channel in this direction. Channels get depleted if too many transactions are routed through a channel in the same direction and the channel has not been equipped with enough funding at setup time.

One possible solution to the problem of channel depletion could be similar to multipath TCP where transactions, which cannot be routed along a single path, are split into multiple smaller transactions which take different routes to their recipient. Another possible solution could be the adjustment of routing fees in such a way that the fees cover the cost of the capacity imbalance, and thus lead to a balancing of the channel by market mechanisms. However, current implementations do not explicitly incentivize or provide decision support for setting routing fees appropriately. None of these two solutions is sufficiently researched.
Dynamic Routing Fees: Every node along the route of a transaction is remunerated by taking a fee for every forwarded transfer. These fees can change over time at the discretion of every node. Hence, not only the capacities are changing over time, but also the costs per node. Increasing fees at a node may lead to paths becoming invalid, since the capacity is not sufficient anymore.

Unstructured Overlay Network: Additionally, in all three proposed systems, DMC, Lightning, and Raiden, nodes are allowed to open channels arbitrarily. This results in an unstructured network where the identifier of a node is not related to its location in the network. That is, there is no data structure or node-ID assignment mechanism (like finger tables based on data hashes in P2P networks) to guide the transaction hierarchically towards its recipient. On the other hand, however, a full view of the graph of active channels can be obtained either by gathering all opening and closing transactions (in UTXO systems) or by inspecting the account of the smart contract that is tracking open channels (in account-based systems). Hence, we can assume that a full view of the graph is available. Note that this is not the case in envisioned privacy preserving systems and needs to be achieved by separate mechanisms there.

In combination, these properties of current payment channel proposals make routing in payment channels more complicated than routing in usual P2P networks. Nevertheless, if the routing of a transaction fails, a last option is to send the transaction as a regular blockchain transaction without any involvement of the payment channel.

7.3 Our Approach

In this section we introduce our formal notation, and suggest an improved solution to the computation of the cheapest path assuming that transactions are not split but routed as a whole.

7.3.1 Notation

A payment channel network is composed of nodes \( v_i \in V \) that are linked to each other via bi-directional payment channels. We write \( G = (V, E) \) for the directed graph spanned by the channels between nodes.

A channel, i.e., an edge, between node \( v_i \) and node \( v_j \) is denoted by \( e_{ij} \) with \( e_{ij} \in E \). Since the graph is directed, the index is ordered, i.e., \( e_{ij} \neq e_{ji} \) for all \( i \neq j \). The capacity in the channel \( e_{ij} \) in the channel from node \( v_i \) to \( v_j \) is denoted by \( \omega_{ij} \). This is the amount of the maximal payment that currently can be sent from node \( v_i \) to \( v_j \) along the edge \( e_{ij} \). Note that \( \omega_{ij} \) may be different from the capacity in the other direction \( \omega_{ji} \). Over time the capacities in the channels change according to the payments made in the channel as described previously.

Each node \( v_i \) which routes a transaction \( tx \) to its next hop \( v_j \) demands a financial reward \( \rho(e_{ij}, tx) \) depending on the edge \( e_{ij} \) and attributes of the transaction \( tx \). In practice it will most likely depend on the size of the transaction in bytes, since this corresponds to used bandwidth, or it will depend on the
transaction amount \( \alpha(tx) \), since the higher the transaction amount, the bigger the imbalance which is created in the channel.

Routing a transaction \( tx \) along \( e_{ij} \) decreases \( \omega_{ij} \) by the transaction amount \( \alpha(tx) \) and increases the inverse capacity \( \omega_{ji} \) by the same transaction amount. However, the sum \( \omega_{ij} + \omega_{ji} \) is always constant.

In order to minimize routing costs for a single transaction, agents need to send a transaction \( tx \) via the cheapest path through the network. We define the cheapest path \( \text{path}(tx) \) of the transaction \( tx \) with sender \( s \) and receiver \( r \) as the directed path \( P \) from \( s \) to \( r \), where \( \sum_{e_{ij} \in P, v_i \neq s} \rho(e_{ij}, tx) \) is minimized. Note that we exclude the fees of the sender in the sum, since the sender will not charge itself for sending its own transaction.

### 7.3.2 Cheapest Path as a Linear Program

In the following we describe a linear program whose result yields the cheapest path \( \text{path}(tx) \) for a single transaction \( tx \). Note that this describes the cheapest path in an algorithm-agnostic way.

We introduce the variables \( x_{ij} \) with the following meaning

\[
x_{ij} := \begin{cases} 
1 & \text{if } tx \text{ is routed along } e_{ij} \\
0 & \text{else}
\end{cases}
\]

Regarding the constraints we have preservation constraints as well as capacity constraints. The preservation constraints assure that there is a path between the sender \( s \) and the receiver \( r \). For each node with an incoming path edge, there is also an outgoing path edge, except if that node is the sender or the receiver. That is, for all nodes \( v_i \) we require that

\[
\sum_j x_{ij} - \sum_j x_{ji} = \begin{cases} 
1 & \text{if } v_i \text{ is the sender } s \\
-1 & \text{if } v_i \text{ is the receiver } r \\
0 & \text{else}
\end{cases}
\]

The capacity constraints enforce that the capacities along the path suffice to route the transaction. For all subsets \( S \) of the nodes which include the sender, the sum of the capacities of the outgoing channels need to support the routing fee outside the set \( S \), and the transaction amount for each outgoing edge. Formally, for all \( S \subset V \) with \( s \in S \) we require that

\[
\sum_{i : v_i \in S, j : v_j \notin S} \omega_{ij} x_{ij} \geq \alpha(tx) + \sum_{i : v_i \notin S, j : v_j \notin S} \sum_{i : v_i \notin S, j : v_j \notin S} \rho(e_{ij}, tx) x_{ij},
\]

where again the symbol \( \alpha(tx) \) denotes the amount of the transaction. If there is exactly one outgoing edge and no incoming edge to the subset \( S \), the bound is tight.

In summary, we receive the following linear program to compute a cheapest route.

\[
\min \sum_i \sum_j \rho(e_{ij}, tx) \cdot x_{ij}
\]
Chapter 7. Routing in Payment Channel Networks

140

\( \sum_{i} x_{ij} - \sum_{j} x_{ji} = \begin{cases} 
1 & \text{if } v_i \text{ is the sender} \\
-1 & \text{if } v_i \text{ is the receiver} \\
0 & \text{else}
\end{cases} \text{ for all } i \)

\[ \sum_{i: v_i \in S} \sum_{j: v_j \notin S} \omega_{ij} x_{ij} \geq \alpha(tx) + \sum_{i: v_i \notin S} \sum_{j: v_j \notin S} \rho(e_{ij}, tx) x_{ij} \]

for all \( S \subset V \) with \( s \in S \)

As with classical shortest path routing the representation as a linear program is mostly of theoretical interest, as it shows, e.g., that the set of feasible solutions forms a convex polytope. Solving the linear program with standard simplex methods, however, is computationally difficult, due to the high number of constraints. With classical shortest path routing, instead of solving the linear program the well-known Dijkstra algorithm is used to compute solutions. Our investigation of cheapest paths is analogous to this classical scenario in the sense that we can formulate the problem as a linear problem. But due to the computational difficulty in solving it, we decided to follow a different approach for the practical computation as is outlined next.

### 7.3.3 Practical Computation of the Cheapest Path

In this section we describe our routing algorithm for finding the cheapest, i.e., cost minimal route in the network for a given transaction. In order to tackle this problem we make the following assumptions.

We assume that there is a global view of the network of currently open channels. This is in line with our argument that the current channel connectivity can be obtained from the public state of the blockchain. However, only the connection and the initial capacity are publicly visible. Neither capacities nor channel fees are available in the blockchain. Our approach is therefore intended for research on economic properties of payment channel networks. Note that for envisioned privacy preserving payment channels, this property of a global view may not hold anymore and different mechanisms are needed there.

We further assume that there is at most one payment channel between each pair of nodes. While this may not always be the case, we are able to abstract multiple channels into a single channel of higher capacity.

Lastly, we assume that transactions are not split into multiple smaller transactions, i.e., always the full amount is routed along one path.

A common approach to solve shortest-path problems is the well-known Dijkstra algorithm. However, the algorithm does not work in our use-case, since it only yields an optimal solution if the visited part of the network does not change during the calculation. Since we include routing fees, by extending the path we increase the amount that needs to be routed at each node on the path, thereby destroying the matroid property. Hence already visited edges may have insufficient capacity for the transaction and its routing fees depending on the remaining route and thus are unable to route the transaction. Consequently, a path with sufficient capacity cannot be calculated by removing all edges with a capacity smaller than the transaction amount and applying the Dijkstra algorithm.

To keep the matroid property and be able to compute a cheapest route efficiently, we reverse the graph so that edges change direction but keep their
capacity. We try to find a path by following the route from the receiver to the sender and add the routing fees to each path. Already visited nodes and edges will not be removed because of capacity violations as they all have an admissible path towards the receiver. When starting with the required amount at the receiver, the routing fees add up towards the sender, who learns how much more has to be transmitted to cover the routing fees. This is similar to a Dijkstra algorithm, where we start at the receiver and take care of the channel capacities.

Overall, this approach enables us to find the cheapest path while facing the outlined challenges. The resulting algorithm is shown in Algorithm 5. Consequently, we can obtain insights about the economic applicability of payment channels in general, without having to worry about the routing complexities in detail.

Algorithm 5 Our algorithm for finding a cheapest spanning tree from the receiver

**Input:** the graph \((V, E)\), recipient \(r\), transaction \(tx\).

**Output:** a cheapest spanning tree \(T\) for transactions to \(r\).

1: \(Q \leftarrow V\)
2: \(T \leftarrow \emptyset\)
3: \(cost(r) \leftarrow 0\), \(cost(v) \leftarrow \infty\) for all \(v \neq r\)
4: while \(Q \neq \emptyset\) do
5: \(v_i \leftarrow \text{argmin}\{cost(v), v \in Q\}\)
6: \(Q \leftarrow Q \setminus \{v_i\}\)
7: for all \(e_{ji} \in E\) do
8: if \(cost(v_j) + \rho(e_{ji}, tx) + \alpha(tx) \leq \omega_{ji}\) then
9: if \(cost(v_i) + \rho(e_{ji}, tx) < cost(v_j)\) then
10: \(cost(v_j) \leftarrow cost(v_i) + \rho(e_{ji}, tx)\)
11: \(path(v_j) \leftarrow e_{ji}\)
12: end if
13: end if
14: end for
15: end while
16: \(T \leftarrow \emptyset\)
17: for all \(v \in V\) do \(T \leftarrow T \cup \{path(v)\}\)
18: end for
19: return \(T\)

7.4 Evaluation

In this chapter we present our measurements to quantify the economical impact on routing.

7.4.1 Method

We implemented a simulator to obtain insights on the performance and behavior of a payment channel network at larger scale. This has various benefits over using a real payment channel implementation, such as speed due to elimination of cryptographic overhead and ease of statistical analysis.
The capabilities of our simulator comprise nodes which can open and close channels with a specified initial volume to another node, as well as the accounting of the channels.

As the characteristics of the routing fee have a strong influence on the network, we allow for each node to define its own fee structure in our network simulator. Initially we use a 0.5% fee of the routed transaction amount and adapt the fee reciprocal to the imbalance in the channel.

As our goal is to evaluate the performance and applicability of payment channels for a replacement of current means of payment, we based our measurements on real world statistics. To generate our transaction volumes, we fitted a lognormal distribution to the transaction statistics that were compiled in 2012 and published by the US Federal Reserve [93]. According to the data we set a mean of 2.95 and a standard deviation of 1.2 for a single payment’s volume. These volumes are then transmitted between two nodes chosen uniformly at random. This assumption is in favor of the implicit assumption of currently proposed systems that enough payments are routed between individuals such that the transfers cancel out—at least on average. The assumption is implicit in current proposals such as Flare [188], as there are no mechanisms incorporated that address channel exhaustion. Translated into a real world scenario, this assumes that banks do not exist and transactions take place directly between service providers and consumers in everyday life. Otherwise, channels would have to be replenished frequently due to the channel capacity exhaustion from routed transfers or own transactions.

The topology of channels is created by a random graph model known to yield node degree distributions that resemble real world social networks. We choose the Barabási-Albert graph generation algorithm [3] to account for the power law distribution of node degrees which is common in social networks. An often mentioned application scenario of cryptocurrencies like Bitcoin is “banking the underbanked” which refers to rural areas in developing countries. In this scenario, for example, these social structures can be assumed to be realistic even in the context of retail payments.

In the context of our decentralized storage system, it is unclear how the structure will evolve. Consequently we do not know if there will be few big banks or exchanges, leading to a star topology or rather a truly decentralized network with equal participants.

Apart from the initial channel network and the transactions’ participants and volumes, the simulation has multiple other degrees of freedom. The initial balance of the nodes, if too low, can influence the simulation as it might not provide enough funding to open a channel or handle a transaction directly on the blockchain. We avoid these effects by always setting a high enough initial balance. The capacities of the channels themselves can lead to constraint failures. At the beginning of our simulation, all channels are funded with an equal amount allocated in both directions. By design, a transaction larger than the sum of both balances can never be routed along an edge. Therefore, all larger transactions need to be settled on the blockchain. For interactions with the blockchain, i.e., channel openings, closings, and direct transactions, we charge a constant fee of 0.41, as this is the Bitcoin transaction fee in U. S. dollars at the time of writing.
7.4. EVALUATION

Figure 7.1: Transactions histogram and methods of settlement for 100 000 transactions, 1 000 000 initial node balance, 1000 nodes connected according to a Barabási-Albert graph with parameter $m = 2$ and 1000 funding from each side in the channels. The routing fees are 0.5% with an imbalance factor. A total of 99 922 transaction have been routed, among them 1453 where a route exists but is more expensive than a direct transaction. 78 transactions were settled directly on the blockchain, because there was no route with enough capacity.

7.4.2 Results and Discussion

In the transaction histogram shown in Figure 7.1 the number of blockchain transactions is nearly constant from 500 up to 1000 currency units. These blockchain transactions are from nodes with channels where the channel capacity is not sufficient for the transaction. Above 1000 currency units, blockchain transactions dominate as a nearly balanced channel with a funding of 1000 is unable to route these amounts. The majority of transactions with an amount smaller than 27 units are routed through channels because with an average route length of 4 hops (3 fees) the total routing fees are on average lower than the fee for a transaction on the blockchain. Transactions with a higher amount than 82 currency units are only profitable using direct channels without routing, except when they help to balance a channel and therefore have a reduced routing fee. This can be seen in the peak in transaction routes which are possible but more expensive than a blockchain transaction.

Regarding the transaction path length distribution shown in Figure 7.2 one hop transactions are always cheaper than blockchain transactions, as the sender
itself charges no fee. The too expensive routes of low length stem from high volume transactions where the proportional fee surpasses the constant fee of a blockchain transaction. For paths with more than 7 hops, the transaction amount has to be lower than 9 units to be routed profitably, so a larger share of valid routes is more expensive than the constant fee.

As shown in Figure 7.3, the earned routing fees of a node increase roughly exponentially with the number of channels and therefore with locked capital. This means that the revenue from routing is exponentially proportional to the invested capital. To handle all 100,000 transactions, the 1000 nodes together spent 5576 units to transfer a total of over 3.9 million units whereas without the payment channels, they would have spent 41,000 units. The cost of transactions reduces only slightly for well connected nodes as they have more direct neighbours to exchange high volume transactions with.

7.5 Conclusion & Future Work

Routing in payment channels is a central problem for solving the scalability issue of blockchains by a state channel approach. We showed that routing in payment channels is more complex than classical routing problems due to the connection of economic and technical constraints. First, capacity constraints limit the size of transactions that can be routed and induce capital binding. Second, channel depletion intensifies capacity constraints and is not addressed by
current proposals. Third, current constraints increase costs of usage, especially if common blockchain transactions are used as fallback option. A naive solution is centralization in form of few routing hubs. However, this would introduce a system that is orthogonal to decentralization—the intention and sole purpose of using blockchain systems in the first place. Additionally privacy is affected negatively by centralized routing hubs.

We envision three main lines of future work:

**Routing in Private Payment Networks** At the time of writing there are proposals which theoretically allow the routing of transactions in private payment networks. Among them, the work of Malavolta et al. [153] provides unlinkability between multiple channels of a route. However, there are no available mechanisms for computing the appropriate route in the first place, apart from naively flooding the payment channel network. These floods, however, can again lead to privacy violations, as broadcasts travel along a star topology and thus the sender at the center is easy to find [29]. For the anonymous blockchain based currency Zerocash, there are two very similar payment channel approaches called Bolt [101] and Z-Channels [238]. However, none of them supports routing. Thus, it is

![Figure 7.3: Routing fee earnings histogram for the same parameters as in Figure 7.1 and two simulation runs. The left axis shows the amount a node with the indicated number of open channels earned and to demonstrate how many nodes of this degree appeared in the graph, the right axis shows their distribution.](image-url)
an open problem to design privacy-preserving routing algorithms on a private payment channel network. Moreover, in a truly private payment channel network, even the topology may be hidden, leading to further problems of route discovery.

**Dynamic Topology Construction** In this paper we made use of a static topology. However, in a real world scenario users can arbitrarily open new bilateral channels. Thus, it may be interesting to model the topology evolution as an autonomous stochastic process with rational agents setting dynamic routing fees to maximize their expected return. This should lead to new insights on the centralization pressure in the system and how incentives need to be aligned to maintain a decentralized network, even in an adversarial setting. Note that a result of this could be that it is impossible to maintain decentralization when introducing payment channels with routing fees, thus necessitating research in different solutions for the throughput scalability problem.

**Exploring the Parameter Space** There are many parameters in payment channel networks and their interrelation is only slightly understood. To contribute to a better understanding of payment channels, future work should clarify the impact the parameters have on the successfully routed transactions, or in the case of a dynamic topology even on the resulting network. As an example, we can say that assuming all else being equal, the higher the routing fee, the more centralized the network will become, due to the expense of routing along multiple hops. However, as of now, there is no quantitative measure of this observation.

Additionally, there are further parameters which we did not account for in our analysis, but which should be taken into account. For example, maintaining a payment channel requires the participants to lock money in that respective channel. This implies the necessity of remuneration in the form of interest or a fee, since the availability of the locked money is restricted. Structurally, this remuneration quantifies the cost of immediacy of money, and thus is dependent on the economy as a whole. The impact on payment channel networks is as follows: The higher the cost of locked capital, the more sense it makes to not use payment channels at all and instead settle transactions directly on the blockchain. This is especially true for low-volume channels, where the cost of locked capital dominates the expensive transaction fees on the blockchain.

Our examinations in this chapter provide a first step towards a treatment of the economical implications for payment channels and show that these cannot be neglected. It is clear that payment channels are not yet ready to be used in our privacy-preserving storage system, due to the high number of open questions. In particular, economic considerations may imply that payment channels provide high centralization pressure, as it is more favorable to set up a channel with a node which already has many connections. Thus, when using payment channels in our storage system, these channels may deteriorate the privacy guarantees of our system due to the increase in centralization. In total, it is unclear if payment channels are desired in our system.
We hope that our analysis in this chapter leads to a more holistic treatment of the network effects in payment channel networks, integrating both economical and technical perspectives.
8.1 Summary of Contributions

In this work we have shown that a privacy-preserving storage system with anonymous payment is feasible. Thus, we have answered our main research question.

How can we design a decentralized privacy-preserving storage system with integrated anonymous payment?

We have designed two systems for decentralized storage with payment. Our goal was to increase the contribution in such a system beyond that of current peer-to-peer storage systems like GNUnet and Freenet by offering incentives to participants in order to increase storage capacity. In our systems, storage providers can offer storage to other participants in the network and get remunerated proportionally to their contributed resources. Further, we require no coordination by any trusted third party. Our remunerations are based on decentralized payments in a blockchain.

Our first design in Chapter 3 exchanges the proof of work mining process of regular blockchains by publicly verifiable proofs of storage. These are cryptographic proofs which show that the prover has stored some data chunks. Thereby, participants are able to publicly prove that they spent storage resources and thus are eligible for payment.

As this first system offered little privacy guarantees and suffered from some attacks on security we enhanced it by additional anonymity measures and changed parts of the architecture resulting in a second design which was explained in Chapter 4. Using ring signatures and one-time payment addresses our system achieves anonymity of senders and receivers of transactions. Storage is managed by storage smart contracts, i.e., special transactions on the blockchain which can be redeemed if and only if the storage provider is able to provide the appropriate proofs of storage. Thus, storage is treated as a special form of transaction and benefits from our privacy-preserving mechanisms for transactions.

After proposing our two designs, we examined various building blocks of our system in more detail.
In Chapter 5, we surveyed publicly verifiable proofs of storage. We proposed a novel proof of storage scheme based on a modification of the Guillou-Quisquater identification protocol and proved its security. Moreover, we implemented all discussed schemes and measured their practical performance. We provide a real-world comparison of these algorithms instead of a theoretical asymptotical comparison. In our evaluation our new scheme outperforms existing schemes with similar security level in the processing time taken to encode the file. For the processing time taken to generate the proof of storage or verify the proof of storage its performance is comparable to the fastest scheme respectively. As a side effect from our observations on the Shacham Waters proof of storage, we receive a novel homomorphic identification protocol based on the Diffie-Hellman assumption.

Next, we focussed our attention on the mining process in Chapter 6. In our second design we use the same proof of work mining mechanism as in Bitcoin which is known to lead to very high energy consumption resulting in a negative impact on scalability. We propose an alternative mining process which is based on proving that some amount of human work units have been performed instead of partially brute-forcing a hash function. Human work units are units of work that cannot be automated and can only have been performed by a human. Our construction is the first known feasible instantiation of a proof of human work. The only other instantiation currently known [32] is based on indistinguishability obfuscation, which is only conjectured to exist and where no general construction is currently known.

Chapter 7 considers payment channel networks. These are overlay networks on top of a blockchain which provide scalability of the transaction throughput. The main idea is to not persist all transactions in the blockchain. Instead, multiple transactions between the same parties can be netted and settled together. Further, payment channel networks allow routing of transactions over multiple channels. Current routing protocols do not take into account the economic dimensions, thus hindering their adoption. Routing fees emerging from the need to lock capital in channels leads to considering the cheapest route instead of the shortest route as in regular networks. Further, each channel has a capacity which provides an upper bound on the transacted amount. To our knowledge, we are the first to perform measurements in order to show the magnitude of the impact of these economical factors on transaction routing.

8.2 Outlook

We envision three main future research directions which could stem from our expositions.

One possible future project is to focus on the network layer of our decentralized storage network. Secondly, as we have already touched on in Chapter 6, our project could be carried on by constructing a privacy-preserving overlay network using state channels. This would allow to settle contracts much faster. Apart from looking at upper and lower layers, a third research direction could embed this work in a bigger context. Recall, that we constructed a protocol such that its users have incentives to participate. This method of providing incentives to participants could be generalized and applied to other protocols, thereby answering the research question how to design a protocol to ensure its
widespread use.

Anonymous Network Layer  While we focussed our efforts on upper layers, privacy could be attacked when insufficient precautions on other layers are used. This is especially the case for the network layer. If regular broadcast communication is used and the attacker controls many nodes in the network, it is possible to reveal the sender of the broadcast [29].

Tor, the usual solution for providing anonymity on the network layer, does not work in our case, since it does not support anonymous broadcasts. Thus, a user would have to create a secure channel to each other participant which is clearly infeasible.

There are two main current mechanisms for anonymous broadcasts. On the one hand there are topological methods like Dandelion or adaptive diffusion. Since senders can be revealed due to the propagation symmetry of a broadcast, topological methods introduce asymmetries in the message propagation. Dandelion [57, 80] routes a message first along a line and then proceeds with a second phase, where a regular flood and prune broadcast is used. Adaptive diffusion [81] uses a virtual source token to mark a synthetic sender of the broadcast. While these methods are very efficient, they can be broken by sufficiently powerful adversaries.

On the other hand there are information theoretic approaches to achieve an anonymous broadcast. A main building block of these mechanisms is the dining cryptographers (DC) protocol [51], where all participants exchange messages pairwise. Due to a clever choice of the messages, the xor operation of all messages reveals a single bit which was chosen by one of the participants without revealing the true originator. Thus, it can be used for anonymously broadcasting a single bit. Later work focusses on transmitting more bits in parallel, or dealing with collisions.

While DC-nets offer not only computational anonymity, but even information theoretical anonymity, i.e., even an adversary with unbounded computational power cannot break the scheme, in practice they are very inefficient. This is mainly due to the high number of messages. In order to broadcast a single message, all participants pairwise need to transfer messages, leading to $O(n^2)$ messages in total. Nevertheless, it is used in current systems such as Dissent [228].

So, in summary there are efficient but less secure schemes on the one hand, and very secure but impractical schemes on the other hand. There lacks a satisfying middle ground combining these two approaches. The PriCloud project supported by the Baden-Württemberg Stiftung which lead to this thesis funded David Mödinger, a second PhD student to find this middle ground. His thesis is envisioned to cover an anonymous broadcast mechanism.

One idea for a promising scheme is a protocol consisting of three phases. A DC-net is used for a first phase, where small groups of up to five participants exchange a message using the dining cryptographers protocol. In a second phase, one participant of the group takes part in a topological anonymous message diffusion protocol. The third phase is a regular flood and prune broadcast to ensure that all participants receive the message. Together with my colleague David Mödinger, we contributed to research in this area and refer to our publications for a more thorough discussion of this approach [164, 165]. Future
research should measure latency and anonymity guarantees of this approach, as well as provide protocols for joining and leaving such a network. These are envisioned to be covered by the dissertation of David Mödinger.

**State Channels** Payment channels [64, 156, 187, 222] provide a way to scale the throughput of blockchains by usage of an overlay network of bilateral channels. Each bilateral channel keeps track of the balance of its participants. Only when opening or closing a channel, a state update is written to the blockchain. Thus, there can be multiple payments in the same channel which are not all written to the blockchain. Only the final balances are persisted. Nevertheless, payment channels are designed in such a way that no party can cheat.

A generalization of payment channels are state channels, where instead of a balance, a common state is updated by the participants. This is envisioned for Ethereum [156, 222], whose blockchain does not simply track balances of users, but rather state transitions of a virtual machine. These state channels allow to execute Turing-complete smart contracts between two parties instead of persisting each state update in the blockchain. For example, it is possible to settle our storage smart contracts in a state channel, though that would lessen privacy.

Since payment channels are the only known way to fundamentally increase the throughput of a blockchain, there is much interest in their properties. Nevertheless, there are many open questions, in particular, if messages are routed over multiple channels. We already touched on the economic aspects of routing in Chapter 7. However, payment channels are a very young field and there are many open questions left. Among them, the privacy of state channels should be examined and mechanisms should be developed to hide senders and receivers. State channels suffer from concurrency issues, since the routing fees can change depending on the order of routed transactions. Routing mechanisms need to be able to cope with these problems. Further, there needs to be a way to combat the centralization pressure introduced by state channels, since otherwise the network evolves to a topology comparable to current banking systems. This means, there are only few central routing hubs. Users have an account at their routing hub and need to pay fees for maintaining their account. This subverts the original idea of Bitcoin as a purely peer-to-peer version of electronic cash.

**Incentives for Protocol Adoption** A third research direction is parallel to our approach. Recall, we implemented incentives in a decentralized storage system in order to increase participation.

A more general problem is that of designing a protocol to ensure its adoption. Currently, proposed protocols for the internet take a very long time before deployment, if they are deployed at all. Protocols such as DNSSEC [76] or IPv6 [65] are still not widely deployed twenty years after their specification. This phenomenon—that the technology powering the internet is subject to less and less change with only few improvements and enhancements—is called the ossification of the internet.

There is some research on how to design a protocol, such that it will be adopted and that subsequent transitions are possible [140, 220]. However,
research on these factors is still preliminary and insufficient. RFC 5218 \cite{221} gives a list of case studies of protocols together with their success factors.

Bitcoin on the other hand had many factors which were detrimental to its adoption and succeeded nevertheless \cite{36}. Bitcoin had to face many dominant rivals like PayPal and credit card companies. Further, it was often associated with crime. There have been many factors of uncertainty in the adoption of Bitcoin, like a looming government intervention. Despite these factors, Bitcoin achieved widespread success and its underlying blockchain architecture is copied in many other systems.

One main driving force behind this adoption is the built-in reward system for early adopters, in the form of a mining reward which decreases exponentially over time. Another factor is the interpretation of Bitcoin as money.

Our system is based on the blockchain architecture of Bitcoin in order to provide incentives for its participants. However, a more thorough and quantitative study of factors in protocol adoption would be desirable in order to identify alternative factors contributing to a widespread protocol adoption. A novel understanding of costs in protocol adoption could overcome the challenges faced in combatting the ossification of the internet.

In summary, while blockchain technology is basically a combination of established mechanisms, its main novelty is the achievement of uncloneability in a fully decentralized system. Thus, research directions are to design systems which require the property of uncloneability on the one hand, and how to improve blockchain technology by modern mechanisms which are better suited for the task on the other hand. Our dissertation goes a step in both directions, and it is our hope that future work provides even more innovation by building on our mechanisms.
### List of Symbols

- $\mathcal{A}$: An attacker of the system
- $\mathbb{G}$: The group of points on an elliptic curve
- $\mathcal{H}$: A cryptographic hash function
- $\mathcal{M}$: A miner of the blockchain
- $\mathbb{N}$: Natural numbers
- $\omega_{ij}$: The capacity of the payment channel from node $v_i$ to $v_j$
- $pk$: A public key
- $\mathbb{P}$: The set of prime numbers
- $\mathcal{P}$: The prover in a cryptographic protocol
- $\mathcal{QR}_n$: The set of quadratic residues in $\mathbb{Z}/n\mathbb{Z}$
- $sk$: A secret key
- $\mathcal{U}$: A user of the system
- $\mathcal{V}$: A verifier in a cryptographic protocol
- $\mathbb{Z}$: Ring of integers
- $\mathbb{Z}/p\mathbb{Z}$: The ring of residue classes modulo $p$
- $e_{ij}$: The directed edge from node $v_i$ to $v_j$
Author’s Publications


77 Felix Engelmann, Henning Kopp, Frank Kargl, Florian Glaser, and Christof Weinhardt. “Towards an Economic Analysis of Routing in Payment Chan-
nel Networks.” In: Scalable and Resilient Infrastructures for distributed Ledgers – SERIAL. ACM, Dec. 2017


[89] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Pearson Education, 1994 (cit. on p. 11).


[95] GNU Project. GMPlib. URL: https://gmplib.org/ (cit. on p. 103).


BIBLIOGRAPHY


[115] Ghassan O. Karame, Elli Androulaki, Marc Roeschlin, Arthur Ger-
vais, and Srđjan Čapkun. “Misbehavior in Bitcoin: A Study of Double-
Spending and Accountability.” In: *ACM Transactions on Information
and System Security – TISSEC* 18.1 (May 2015), 2:1–2:32. ISSN: 1094-
9224. DOI: 10.1145/2732196. URL: http://doi.acm.org/10.1145/
2732196 (cit. on p. 79).

Malicious Arithmetic Secure Computation with Oblivious Transfer.” In:
(cit. on pp. 114, 119).


“Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol.” In:
pp. 20, 21).

[119] Aggelos Kiayias, Hong-Sheng Zhou, and Vassilis Zikas. “Fair and Ro-
bust Multi-Party Computation using a Global Transaction Ledger.” In:
(cit. on p. 20).

[120] Taechan Kim and Razvan Barbulescu. “Extended Tower Number Field
Sieve: A New Complexity for the Medium Prime Case.” In: *Advances in
662-53018-4. DOI: 10.1007/978-3-662-53018-4_20. URL: https://
doi.org/10.1007/978-3-662-53018-4_20 (cit. on p. 103).

[121] Sunny King. *Primecoin: Cryptocurrency with Prime Number Proof-of-
(cit. on p. 21).

[122] Sunny King and Scott Nadal. *PPCoin: Peer-to-Peer Crypto-Currency
with Proof-of-Stake*. 2012. URL: https://peercoin.net/whitepaper
(cit. on pp. 20, 131).

[123] Stephan Klieber, Rens W. van der Heijden, Henning Kopp, and Frank
Kargl. “Terrorist Fraud Resistance of Distance Bounding Protocols Em-
on p. 158).

Message Syntax Reverse Engineering by Analysis of the Intrinsic Structure
of Individual Messages.” In: *Workshop on Offensive Technologies, WOOT.*

[125] Gillat Kol and Moni Naor. “Cryptography and Game Theory: Designing
Protocols for Exchanging Information.” In: *Theory of Cryptography


BIBLIOGRAPHY


