b
UNIVERSITÄT
BERN

Encrypting Into the Future

Exploring Time-Lock Encryption

Bachelor Thesis

Lukas Leo Schacher

from

Escholzmatt-Marbach LU, Switzerland

Faculty of Science, University of Bern

18. August 2021

Prof. Christian Cachin
Dr. Giorgia Marson
Cryptology and Data Security Group
Institute of Computer Science
University of Bern, Switzerland

Abstract

Time-lock encryption (TLE) allows a sender to encrypt a message “into the future”. The decryption of such a message will only be possible after a certain deadline has passed. While the original idea was introduced over two decades ago, recently TLE schemes received more attention because of their power as building block in protocols to enforce delay in decentralized systems. For instance, many cryptocurrencies use TLE for their consensus algorithms to replace the resource-inefficient proof-of-work. This thesis provides an overview over TLE based on three selected constructions. Desirable properties for such schemes are identified and formalized. Subsequently, the investigated implementations are evaluated based on adequate criteria to provide an overview over the strengths and limitations. This evaluation might assist in the creation of future TLE schemes.

Acknowledgements

I would like to express my gratitude to my supervisor, Giorgia Mason, for her continuous guidance. She consistently allowed this thesis to be my own work but steered me in the right direction whenever she thought I needed it. I would also like to thank Prof. Christian Cachin for providing his expertise and insight. Last but not least, I wish to thank my family for their support and Nina Baumgartner, Lukas Hauck and Lars Hulliger for the discussions on the topic and for proofreading this thesis.

Contents

Chapter 1

Introduction

Encryption is the process of encoding information to ensure confidentiality [1]. Traditional encryption allows transmitting a message such that only the intended recipient, who holds a decryption key, can retrieve the message. This thesis explores encryption schemes where the goal is different: The thesis investigates a special class of cryptosystems that allow a sender to encrypt a message "into the future". These schemes are similar to traditional symmetric and asymmetric encryption, in the sense of providing confidentiality, however, they provide a guarantee about "at which time" an encrypted message can be decrypted. More specifically, that one can only decrypt a message after a specific time instance is reached. Academically, this kind of cryptosystems are called "Time-Lock", "Timed-Release", "Time-Specific", "Time-Sensitive" or "Time-Lapse" cryptographic systems [2, 3, 4, 5, 6, 7]. In this thesis, the term *time-lock encryption* (TLE) is used for the investigated encryption schemes.

TLE has many applications besides the basic use of sending a message "into the future" One idea is to use it for sealed-bid auctions: In this auction type, bidders want their bids to be secret even to the auctioneer during the auction [8]. A malicious auctioneer could decrypt some bids before the deadline and instruct favoured bidders about rivalling offers. These bidders could then adapt the prices accordingly to profit from the scheme. Said type of abuse could be prevented by using TLE. TLE could also be useful for data collection in clinical trials [5]. Collected data during such a study could be encrypted using TLE in order to preserve data integrity. Aforementioned trials are usually funded by companies who have a significant interest on the outcome, as they stand to make or lose significant amounts of money. That creates pressure on the people who carry out the trial to obtain a positive result. The use of TLE can neutralize this favourism without prematurely revealing confidential information about the trial. Further examples of applications include delayed digital cash payments, electronic voting, timed signatures for fair contract signing, randomness beacons and prevention of denial of service attacks [2, 5, 9, 10, 11].

Naïvely, a TLE scheme can be obtained by using a trusted third party to perform the decryption at a pre-defined time. Third parties usually represent vulnerabilities and with cryptographic protocols it is possible to omit their use [12].

Recently, TLE received increasing attention from both researchers and the public because of its power as cryptographic building block to construct resource-efficient blockchains [13]. Blockchains based on proof-of-work, e.g. Bitcoin, consume considerable amount of energy and do not provide long-term sustainability [14]. To mitigate this issue, recent cryptocurrencies experiment with a combination of TLE and proof protocols. Examples include the Chia network [15], which uses proof-of-space and verifiable delay functions, and Ethereum 2.0 [16], where it is intended to employ proof-of-stake together with verifiable delay functions.

With this thesis, we first and foremost aim to give an overview over TLE. We intend to define criteria to analyse TLE schemes. Additionally, we want to provide an evaluation of existing schemes, based on three selected constructions. Thus, the contributions of this thesis are threefold:

- Presenting and introducing selected existing implementations of TLE
- Formulating criteria for desirable properties of TLE designs
- Evaluating and discussing the presented constructions based on the identified criteria

In this thesis we focus on three selected implementations of TLE. Chapter 2 provides the necessary background information as well as an overview of the mathematical notations and conventions used in the realizations. In Chapter 3, we identify and discuss desirable properties for TLE schemes. Based on these properties we then formulate corresponding criteria. We then introduce the investigated schemes in Chapters 4-6. In Chapter 4, we present the first known construction of the field, the so-called "time-lock puzzle" (TLP) by Rivest et al. [2]. This publication from 1996 pioneered the field of time-lock encryption. Chapter 5 covers a more modern second implementation. We investigate the cryptographic primitive called "verifiable delay function" (VDF) [10]. Furthermore, we also discuss a specific implementation of VDFs that uses a TLP as baseline. Pietrzak [17] takes a TLP and extends it to be publicly verifiable and thus creates a VDF. In the following Chapter 6, we study the TLE construction by Liu et al. [18]. They propose a scheme which is fundamentally different from TLPs and VDFs. Next to a generic construction, which could be implemented using any blockchain, Liu et al. [18] also created a specific implementation based on the Bitcoin blockchain. Chapter 7 evaluates the three schemes based on the criteria and discusses the advantages and limitations of each implementation. In Chapter 8, we provide concluding thoughts about this thesis. Additionally, we discuss the limitations of this work and give an outlook for future research in the field of TLE.

Chapter 2

Background

2.1 Cryptographic Hash Functions

A cryptographic hash function is a deterministic algorithm that maps a digital input of arbitrary length to a unique binary output of fixed length, called *hash-value* or simply *hash* [19]. If an original input text is changed - even minimally - it is not possible to tell from the calculated hash value what has been changed. But one can say with certainty that the input has changed.

It is also very unlikely for any two different inputs to result in the same hash value. Consequently, the hash value is often referred to as a digital fingerprint. The reverse way of reproducing the input from the hash value is virtually impossible and can thus be seen as one-way function. Technically it is possible to reverse-hash a value, but the required computing power makes it infeasible.

Definition 1 (Cryptographic Hash Function)**.** A *cryptographic hash function* H maps an input $x \in X$ of arbitrary length to a binary output of fixed length ℓ.

$$H : X \to \{0,1\}^{\ell} \tag{2.1}$$

Additionally, cryptography handbooks [20] list the following three desired properties for such hash functions:

1. *Preimage Resistance*:

   For essentially all pre-specified outputs, it is computationally infeasible to find any input which hashes to that output, i.e., to find any preimage x such that $H(x) = y$ when given any y for which a corresponding input is not known

2. *2nd-Preimage Resistance*:

   It is computationally infeasible to find any second input which has the same output as any specified input, i.e., given x, to find a 2nd-preimage $x' \neq x$ such that $H(x) = H(x')$.

3. *Collision Resistance*:

   It is computationally infeasible to find any two distinct inputs x, x' which hash to the same output, i.e., such that $H(x) = H(x')$.

Hash functions are widely used for data integrity checks or to provide data protection to password storage. Checksums on data files can be generated by hash functions. To check the integrity of a file one compares the hashes of the original file and the possibly modified file. User passwords can be hashed and stored as hash values in a database rather than as plaintext. An intruder therefore can only view the hashes of the stored passwords, even with access to the database. Further uses include digital signatures, message authentication and other information-security applications.

Well-known hashing algorithms include MD5, RIPEMD, WHIRLPOOL and SHA-1, SHA-2 and SHA-3 (Security Hashing Algorithm). For example, SHA-256, an SHA variant which creates hash values with the length of 256 bits, is an integral part of the Bitcoin protocol [21], as it is part of the *proof-of-work* and also utilised in the creation of Bitcoin addresses.

2.2 Complexity Classes P and NP

The main task of complexity theory is to classify computational problems according to the resources required to solve them. Typically, the resource considered is time or memory. In particular, an attempt is made to distinguish the class of problems that can be solved efficiently from those that are computationally unreasonable. The theory of computational complexity uses decision problems to categorize problems. Decision problems are problems that can be answered either with "yes" or "no" (or "1" and "0" respectively). The standard work on cryptography by Menezes et al. [20] defines the complexity classes and its delimitations as follows:

Definition 2 (P)**.** The complexity class *P* contains all decision problems that are deterministic and solvable in polynomial time.

Definition 3 (NP)**.** The complexity class *NP* contains all decision problems for which a "yes" answer is verifiable in polynomial time, given some extra information, called a certificate.

Definition 4 (co-NP)**.** The complexity class *co-NP* contains all decision problems for which a "no" answer is verifiable with a corresponding certificate in polynomial time.

Since solving implicitly involves verifying, P $\subseteq$ NP and P $\subseteq$ co-NP hold. Equality would mean that all decision problems with efficiently verifiable solutions would also be efficiently deterministically solvable. Accordingly, inequality is suspected, but proof is still lacking. Other suspected inequalities, for which no proof has yet been found either, are NP $\neq$ co-NP and P $\subset$ (NP $\cap$ co-NP).

2.3 Witness Encryption

The notion *witness encryption* for NP-relations was introduced by Garg et al. [22]. They describe witness encryption as "the answer to the question whether one can encrypt a message so that it can only be opened by a recipient who knows a witness to an NP relation". A more direct application of such a scheme is a reward for solving some NP-hard puzzle [23].

For example (see Figure 2.1), a person, Alice, wants to pay someone to solve her favourite sudoku [24]. She wants to pay anyone who solves the sudoku first and she does not want to be present to hand over the money. She stores the money in a save and encrypts the password m from the safe using the sudoku x in such a way, that anyone who knows the solution of the sudoku w can decrypt the ciphertext c, obtain the password m and therefore be able to access the payment. This solution to the sudoku is called "witness".

A witness encryption scheme is defined for some NP language L with corresponding witness relation R. It allows a user to encrypt a message m to a particular instance x to generate a ciphertext c: $c \leftarrow \mathsf{WE.Enc}(x, m)$. If $x \in L$ and the recipient of the ciphertext knows a witness w with $R(x, w) = 1$ then a recipient can decrypt the message: $m = \mathsf{WE.Dec}(c, w)$. If, however, $x \notin L$, then no polynomial-time attacker can distinguish between any encryptions of messages of equal length. Intuitively, statement x can be seen as a "public key", so that any witness w in $(x, w) \in R$ can be used as complementary "private key".

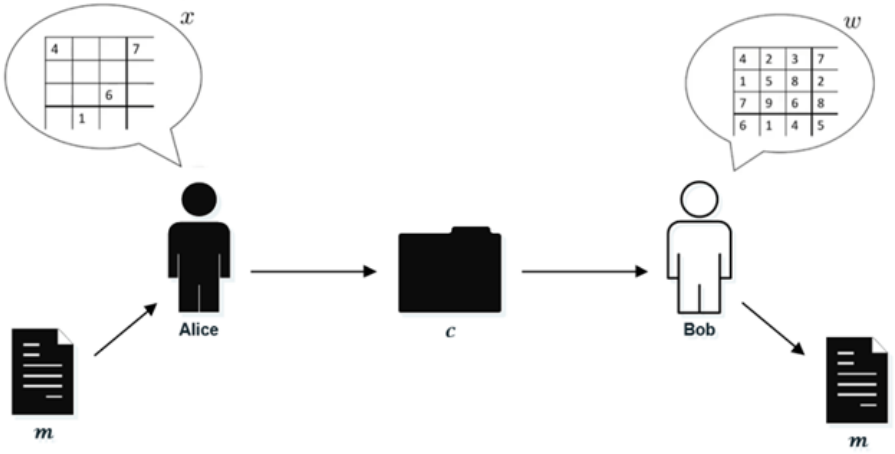

Figure 2.1. Witness Encryption: Sudoku Example [24]

Definition 5 (Witness encryption (WE) [22])**.** A *witness encryption scheme* for an NP relation R consists of the following two polynomial-time algorithms:

- $\mathsf{WE.Enc}(1^\lambda, x, m)$ is an encryption algorithm that takes as input a security parameter 1^λ, an unbounded-length string x, and a message $m \in M$, and outputs a ciphertext c.
- $\mathsf{WE.Dec}(c, w)$ is a decryption algorithm that takes as input a ciphertext c and a bit-vector w, and outputs a message m or the symbol $\perp$.

Correctness:
For any $(x, w) \in R$, we have that

$$Pr[\mathsf{WE.Dec}(\mathsf{WE.Enc}(1^\lambda, x, m), w) = m] = 1 \tag{2.2}$$

"Extractable" [25, 26] witness encryption schemes additionally guarantees that any decrypting party who successfully decrypts a ciphertext c must "know" a corresponding witness w. Put differently, an adversary cannot learn any non-trivial information about an encrypted message, unless it already has knowledge on a witness w with $R(x, w) = 1$.

Existing witness encryption constructions [22, 27, 26, 28] are based on multilinear maps [29, 30] or obfuscation [31, 32, 33].

2.4 Proof of Work

Proof of Work (PoW) [34], also proof-of-work, was introduced to enforce that a certain amount of energy was expended for doing some task in a publicly verifiable way [35]. The concept was invented as a way to prevent denial-of-service attacks, email spam and other service abuses on a network by forcing malicious users to accumulate a large computational workload. Most recently, PoWs were popularized by the cryptocurrency Bitcoin [21] as a means to reach consensus and to prevent double spending. PoWs eliminate the need for a trusted third party and enable the processing of transactions in a peer-to-peer network.

Within the Bitcoin network hash-based PoWs, called Hashcash [36], are used to confirm the authenticity of stored transactions for the generation of new blocks. For a block to be accepted to the blockchain a cryptographic puzzle must be solved and the solution, the block's hash (SHA-256 for Bitcoin), must

meet the network's difficulty target. This difficulty parameter is regularly adjusted depending on the computational power contribution by miners so that roughly every 10 minutes a new block can be added to the chain. This hash consists of the transaction data as well as of the hash of the previous block.

PoW is seen as the main reason why cryptocurrencies, e.g. Bitcoin, are consuming considerable amounts of energy [37]. Since finding a hash that matches the difficulty target is random, almost all calculations, more specifically all but the one finding the correct hash, are discarded. These calculations all consume energy and with the current scale of Bitcoin, this consumption is considerable.

2.5 Bitcoin

A cryptocurrency is the cryptographic equivalent of a regular currency. Bitcoin [21] differs from government regulated fiat currencies in that there exists no central authority within the network to verify transactions and prevent frauds and attacks [38]. Instead, Bitcoin relies on a highly replicated public ledger, secured by means of a hash chain and validated through community consensus [14]. Bitcoin is the first fully decentralized cryptocurrency. Decentralized cryptocurrencies like Bitcoin are in high demand and are therefore subject to many research approaches.

The aim of this thesis is not to describe the Bitcoin system completely. We only give a simplified description of the relevant features of Bitcoin for the investigated encryption schemes. Especially details about Bitcoin transactions are not discussed. For a full description one can refer to Nakamotos original introduction of the cryptocurrency [21]. The following section focusses on one central building block of Bitcoin, the so-called "Bitcoin blockchain".

2.5.1 The Bitcoin Blockchain

A block is a record in the blockchain that contains and confirms many waiting transactions [39]. Every block contains two sections, a header, and a body. The header part includes the block information, and the body part contains the transaction information. The frequency of the block generation is controlled by the difficulty parameter included in the hash-based PoW explained in Section 2.4. Successfully contributing to the generation of a new block rewards the miner with a certain amount of bitcoins.

The Bitcoin blockchain is a public record of Bitcoin transactions in chronological order [40]. It enables the protocol to work as a decentralized system without any trusted third parties. A blockchain is a distributed database acting as a public ledger that holds all processed transactions. It is used to prevent double spending and to verify the continuity of Bitcoin transactions. The Bitcoin blockchain is based on a distributed consensus [41].

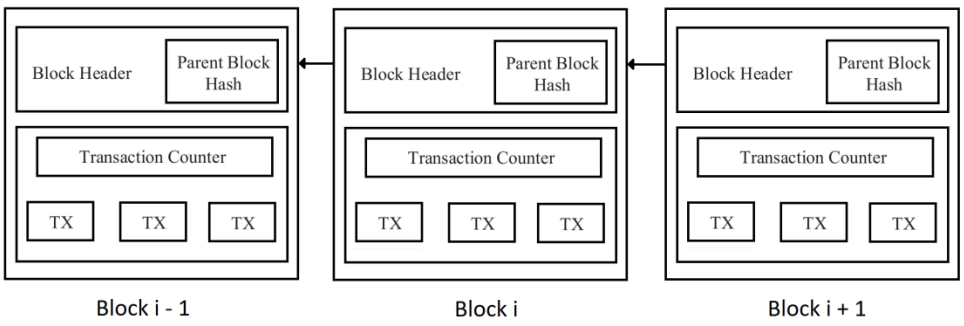

Figure 2.2. Simplified Blockchain Architecture [42]

This blockchain consists of blocks $B_1, ..., B_\tau$, each carrying a set of validated transactions, where B_1 represents the first block and B_τ the most recently attached block. Block B_0 is denoted as the *genesis*

block, which is the head of the blockchain. The genesis block is a distinguished value and does not reference any previous block, it is a specified constant within the cryptocurrency. Blocks are linked back-to-back, with each one referencing its previous block to form the complete blockchain [41].

Definition 6 (The Bitcoin blockchain [18])**.** The blockchain is a sequence of tuples

$$(T_1, r_1, D_1, B_1), ..., (T_s, r_s, D_s, B_s) \tag{2.3}$$

that satisfies

$$B_i := H(T_i, r_i, D_i, B_{i-1}) \tag{2.4}$$

where H is a cryptographic hash function based on SHA-256 and $B_1, ..., B_s$ are called blocks. The other parameters are defined as follows:

- T: a list of transactions
- r: a counter value
- D: a target hash generated from the difficulty parameter

More details about the parameters T_i, r_i, D_i follow hereafter.

The following description closely follows [18]. So-called *miners* contribute computational resources trying to find the next block B_{s+1} to append to the blockchain. This process is computationally expensive, yet feasible, as B_{s+1} must fulfil the criteria described above.

Miners keep a local copy of the blockchain and collect recent Bitcoin transactions of the network. Transactions deemed invalid are discarded, whereas T_{s+1} is the list of stored new transactions. These transactions will later function as auxiliary input for the computational reference clock (see Section 2.6). A miner tries to confirm the validity of these transactions to find a new block B_{s+1} which includes T_{s+1}. The miner increments the counter parameter $r_s + 1$, until the hash

$$B_{s+1} := H(T_{s+1}, r_{s+1}, D_{s+1}, B_s) \tag{2.5}$$

is found, where $B_{s+1} \leq D_{s+1}$ is satisfied. Parameter D_{s+1} is the *target hash*, the public system parameter which is used to determine the computational hardness of finding a new block. This target hash is related to the *difficulty* parameter of the Bitcoin protocol. Note that the difficulty parameter is set dynamically to match the desired rate of one new block every 10 minutes, as it is adapted in response to the current available mining resources.

Afterwards, the new block B_{s+1} including the related data $(T_{s+1}, r_{s+1}, D_{s+1}, B_s)$ is broadcast inside the Bitcoin system to all miners. The produced block B_{s+1} represents the *proof of work* for the contributed computational resources by the miner that found the new block. This miner gets rewarded with a certain number of bitcoins, as an incentive for all miners to keep searching for new blocks. Henceforth, miners use their resources to find B_{s+2}, the next block.

2.5.2 Forks

Inconsistencies inside the blockchain structure may occur, as the Bitcoin system operates over a global peer-to-peer network [43]. *Forks* divide blockchains into two or more variants, which either continue to run in parallel (soft fork) or develop completely independently of each other (hard fork) [18]. For example, it is possible that two or more miners independently find and broadcast a new valid block B_{s+1} at the same time. Consequently, such an event often leads to the emergence of multiple *branches* in the system. In that case, only the blocks that form the longest branch of the fork will be considered valid. The shorter branch is rejected by the network. Only miners who contribute to the longest branch are rewarded [38].

The aforementioned "length" of the branch is determined by the sum of the difficulty of the blocks in the chain, not by the number of blocks itself [18]. Otherwise, an adversary could append several blocks with low difficulty to fork the chain.

2.6 Computational Reference Clock

The issue of finding a physical time equivalent through computational models persists to this day. Liu et al. [18] conclude that currently there is no feasible way to achieve this notion of real-world time in a computational model without some form of a trusted third party. They introduce the concept of a "computational reference clock" which they describe as "a novel and very realistic method to *emulate* real-world time in a computation model" [18]. They use the block creation inside the blockchain of a cryptocurrency as reference point of passed physical time. The authors formalize this concept as follows.

Definition 7 (Computational reference clock (CRC) [18])**.** A *computational reference clock* (CRC) is a stateful probabilistic machine $\mathcal{C}(1^{\mathcal{K}})$ that takes input a difficulty parameter $\mathcal{K}$ and outputs an infinite sequence $w_1, w_2, ...$ in the following way. The initial state of $\mathcal{C}$ is w_0. It runs a probabilistic algorithm $f_{\mathcal{C}}$ which computes $w_\tau = f_{\mathcal{C}}(w_{\tau-1})$ and outputs w_τ.

We write $w_\tau \leftarrow \mathcal{C}(1^{\mathcal{K}}, \tau)$ for $\tau \in \mathbb{N}$ to abbreviate the process of executing the clock τ times in a row, starting from initial state w_0, and outputting the state w_τ of $\mathcal{C}$ after τ executions.

Intuitively, the machine $\mathcal{C}$ iteratively and publicly computes $f_{\mathcal{C}}$. Each $f_{\mathcal{C}}$ is calculated in super-polynomial time, meaning that the execution time is not bounded above by any polynomial. As the complete internal state of $\mathcal{C}$ is broadcast after each execution no hidden information can be stored inside $\mathcal{C}$. Algorithm $f_{\mathcal{C}}$ is public as well. After τ executions the machine outputs the current state of computation at "time" τ. In this setting $w_{\tau-1}$ serves as "witness" that the current time is "at least τ".

One can think of $\mathcal{C}$ as a highly performing machine that solves an infinite sequence of computational puzzles. Jumping ahead to the construction discussed in Chapter 6, the entirety of all contributing Bitcoin miner is used to instantiate $\mathcal{C}$. This corresponds to the pool of computational resources dedicated to appending new blocks to the public Bitcoin blockchain. After the τ-th execution, this machine returns the blockchain with a length of τ.

Chapter 3

Evaluation Criteria

In this chapter we identify and discuss criteria for formulating desirable properties for time-lock encryption schemes. We obtained these criteria by analysing existing TLE schemes and their core benefits. These criteria can not only be used for evaluating existing schemes, but also as a starting point for future implementations.

3.1 Categorisation

We first introduce two broader categories and then identify key properties of TLE designs. The first category *practicality* combines the two subcategories *ease of adoption* and *usability*. The first subcategory is about the ease of adoption of a construction. Adoption of TLE schemes is not only influenced by the underlying security and usability properties, but also by requirements inflicted by the underlying building blocks of a scheme. Some building blocks might demand considerable infrastructure and significant computational resources to ensure security. This might introduce issues when adopting the scheme in practise. Usability properties focus on the involved effort for human end users to utilize a TLE scheme securely and properly.

Security properties form the second category. This category contains properties that cover the hardness of a scheme and how vulnerabilities are avoided.

- Practicality
  - Ease of Adoption
  - Usability
- Security

Note that the order of the properties is determined alphabetically and does not represent the importance to the scheme.

3.2 Practicality

3.2.1 Accuracy of Time Frame

Although computerized physical clocks have become very accurate today (for example, atomic clocks), all abstract models of computation struggle to this date to capture real-world time exactly [44]. The pre-determined target time to decrypt a time-lock encrypted message should match the actual decryption timing of a TLE scheme as closely as possible. The pre-defined lifespan of the ciphertext must be accurate for a TLE scheme to be practical. Aligning real-world time with computational models is an ongoing quest of computer scientists worldwide [18]. The preciseness of the encryption duration is key to the practicality of TLE constructions in its entirety.

3.2.2 Ease to Extend to Multi-User Setting

TLE schemes are created to enable sending a message "into the future". These constructions usually disclose how to time-lock encrypt a single message at a time to a single receiver [45]. Such a scheme might allow a sender to target multiple receivers at once with no or only limited changes required to the proposed construction. Ideally, these multiple recipients should be able to decrypt the encrypted message at the same time. In order to meet the criterion *ease to extend to multi-user setting*, a construction should be adaptable to support multiple receivers with a synchronous decryption, with no or only minor modifications.

3.2.3 No Resource Requirements

This criterion closely follows the property *no resource restrictions* used by Liu et al. [18]. Decrypting parties should not require performing large computations to decrypt the message. As a result, the amount of available computational resources can be disregarded, and all decrypting parties can decrypt the ciphertext at essentially the same time. This property is provided if a scheme does not demand a certain amount of computational power to provide the intended functionality.

3.2.4 Practicality of Building Blocks

The theoretical foundation, in form of a proof of concept, is merely the first step to a working implementation in practise. Ideally, a TLE scheme should perform well on paper and in practice, while efficiently using the assigned available resources. For an encryption scheme to be useful, the used building blocks are required to be practical.

3.2.5 Transparency of Decryption Progress

TLE schemes allow the decryption of a message only after a certain amount of time has passed. These schemes calculate this time frame with the use of predictions about how much time specific computations approximately require [46]. The progress of this ongoing computation might be available to public, even to parties which are not performing decrypting calculations themselves [47]. In other words, the actual progress and therefore the actual passed "time" of an ongoing decryption may be accessible for anyone. Constructions with a transparent decryption progress provide this property.

3.3 Security

3.3.1 No "Trust" in Third Party

The use of a trusted third party (TTP) should not be required. The presence of a TTP usually represents a possible single point of failure for a system [2]. Ideally, a TLE scheme does not put any trust in third parties and therefore is not bound to the functionality and honesty of any external entity. Constructions providing this property do not put trust in third parties.

3.3.2 Parallelization Resistance

The decryption process might require performing a certain number of sequential steps before the ciphertext can be decrypted. The property *parallelization resistance* extends *no resource requirements* (see Section 3.2.3) in that the decryption computation may not be parallelizable, independently of the available computational resources [48]. This property is provided if the TLE scheme is resistant to parallel solvers.

3.3.3 Public Verifiability

Similar to the property *non-interactive* used by Liu et al. [18], *public verifiability* also states that the encryptor is not needed for the decryption and the verification of the found solution. Everyone should be able to evaluate the function on any input and produce a proof that the corresponding output is in fact correct. The production of this proof should also require only little computation and therefore be accessible to computationally limited parties [10]. A scheme provides this property if it is easily verifiable.

Later in this thesis we use the introduced criteria to evaluate existing TLE schemes. We analyse to what extent the selected constructions fulfil these properties and we discuss the strengths and weaknesses of each construction

Chapter 4

Time-Lock Puzzles

In this chapter, we present a *time-lock puzzle* [2], which pioneered the field of TLE.

4.1 Overview

The idea of sending a message into the future was introduced by Timothy May [49] in 1993. The first known implementation of time-lock encryption was proposed in 1996 by R. Rivest, A. Shamir and D. Wagner. In the paper "Time-Lock Puzzles and Timed-Release Crypto" [2] they introduce the notion of so-called *time-lock puzzles*. The goal is to encrypt a message so that nobody can obtain the message until a pre-determined amount of time has passed. The encrypted message cannot be decrypted by anyone before the expiration of the chosen time frame.

They introduce two ways to implement such a system:

1. Use computational problems which require a certain amount of time to be solved, called "time-lock puzzles". Such a puzzle requires a computer to be running continuously for at least a certain amount of time to solve the problem.

2. Use a trusted third party to keep the information safe and only reveal certain information until a specific time has passed.

May [49] initially suggested the approach based on a trusted third party. Rivest et al. [2] explore both options. The following section introduces the cryptographic primitive of time-lock puzzles, while the approach using trusted third parties is described in Section 4.3.

4.2 Time-Lock Puzzles

A time-lock puzzle (TLP) allows a sender to send messages "to the future". The sender publishes a puzzle where the message to be sent is the corresponding solution [50]. This system therefore hides the message until enough time has elapsed for the puzzle to be solved. A TLP takes an (approximately) pre-defined amount of time to solve.

This time frame is determined by a difficulty parameter. For longer encryption durations, this difficulty is to be chosen according to Moore's Law: Moore's Law refers to the prognosis of then Intel Corporation president Gordon Moore, that the number of active elements on a microchip doubles roughly every 18 months [51]. Moore predicted a dramatic increase in the power of data processing and a decrease in relative costs - on an exponential scale [52].

For a TLP to be useful, the generation of a puzzle should take significantly less time than solving it. Rivest et al. [2] achieve this by generating a puzzle in $\mathcal{O}(\log_2 T)$, while solving it requires $\Omega(T)$ time [53, 54]. Furthermore, parallelization should not be possible while computing the solution, so that even adversaries with considerable computational advantages cannot produce the solution much faster than

serial ones. Up to now, only one construction is known that is believed to satisfy these properties: the original instantiation of TLPs as proposed by Rivest et al. [2].

Solving a *time-lock puzzle* reveals the key which can be used to decrypt the message. The main difficulty is aligning physical time with CPU-time. To circumvent using parallel computing to solve the puzzle, it must be constructed so that only a strictly sequential approach leads to a solution. More precisely, the progress of computing the solution cannot be sped up by parallelization. Rivest et al. [2] use the comparison that two women cannot have a baby in 4.5 months. The second problem is that this method can only use an approximation of how much time will pass until the puzzle is solved as computing power is ever improving.

4.2.1 Construction

The approach by Rivest et al. [2] is based on the assumption that exponentiation modulo an RSA integer is an "inherently sequential" computation [55]. While the authors did not formally define TLPs, more recent academic publications did. For example, Bitansky et al. [55] proposed the following formal definition of a time-lock puzzle in 2016:

Definition 8 (Time-Lock Puzzle [55])**.** A *time-lock puzzle* is a pair of two algorithms $(\mathsf{TLP.Gen}, \mathsf{TLP.Solve})$ satisfying the following requirements.

- *Syntax*:
  - $Z \leftarrow \mathsf{TLP.Gen}(T, s)$ is a probabilistic algorithm that takes as input a difficulty-parameter T and a solution $s \in \{0,1\}^\lambda$, where λ is a security parameter, and outputs a puzzle Z.
  - $s \leftarrow \mathsf{TLP.Solve}(Z)$ is a deterministic algorithm that takes as input a puzzle Z and outputs a solution s.
- *Completeness*:

  For every security parameter λ, difficulty parameter T, solution $s \in \{0,1\}^\lambda$ and puzzle Z in the support of $\mathsf{TLP.Gen}(T, s)$, $\mathsf{TLP.Solve}(Z)$ outputs s.
- *Efficiency*:
  - $z \leftarrow \mathsf{TLP.Gen}(t, s)$ can be computed in time $\mathsf{poly}(log_2 t, \lambda)$
  - $\mathsf{TLP.Solve}(Z)$ can be computed in time $t \cdot \mathsf{poly}(\lambda)$.

In a TLP, we require that the parallel time required to solve a puzzle is proportional to the time it takes to solve the puzzle honestly, up to some fixed polynomial loss.

Rivest et al. [2] originally encrypted the message using the RC5 [56] encryption algorithm. The proposed construction is generic; hence any (secure) encryption algorithm may be used to implement TLPs. The generic approach is depicted as Algorithm 1. In the following, we introduce the procedure of [2] in more detail.

Suppose Alice has a message M that she wants to encrypt for a period of t seconds with a time-lock puzzle so that Bob can only open it afterwards.

- She generates an RSA modulus
$$N = p \cdot q, \tag{4.1}$$
where p and q are two large randomly chosen primes, such that $|N| > 2048$. She also computes
$$\phi(N) = (p-1)(q-1). \tag{4.2}$$

Algorithm 1 Time-Lock Puzzle Generation

Input

message: M
encryption duration in seconds: t

Algorithm

1: $p, q \leftarrow$ random large prime
2: $N \leftarrow p \cdot q$
3: $\phi(N) \leftarrow (p-1)(q-1)$
4: $S \leftarrow$ possible squarings (modulo N) with current hardware
5: $T \leftarrow t \cdot S$
6: $K \leftarrow$ random encryption key
7: $C_M \leftarrow \mathsf{Enc}(K, M)$
8: $x \leftarrow$ random integer in $[2, N-1]$
9: $C_K \leftarrow K + x^{2^T} \pmod N$
10: **erase** $p, q, \phi(N)$
11: **output** N, x, T, C_K, C_M

- She calculates

$$T = tS, \tag{4.3}$$

  where S is the number of squarings per second (modulo N) that can be performed by the solver.

- She generates a random key K for an encryption algorithm of choice, such as RC5, which was used by Rivest et al. [2] in the original paper. This key is long enough (originally > 160 bits, today more likely > 256 bits for modern symmetric-key algorithms, i.e. AES-256 [57]) that directly searching for it is infeasible, even with the advances in computing power expected during the lifetime of the puzzle.

- She encrypts M with key K and encryption algorithm RC5, to obtain the ciphertext

$$C_M = \mathsf{RC5.Enc}(K, M). \tag{4.4}$$

- She picks a random x modulo N, where $1 < x < N$, and encrypts K as

$$C_K = K + x^{2^T} \pmod N. \tag{4.5}$$

  To do this efficiently, Alice uses $\phi(N)$ to reduce the complexity and she first computes

$$e = 2^T \pmod{\phi(N)}, \tag{4.6}$$

  and then uses e to compute

$$y = x^e \pmod N. \tag{4.7}$$

- Alice produces as output the time-lock puzzle (N, x, T, C_K, C_M) and erases any other variables (such as $p,\ q$ and $\phi(N)$) created during this computation. She sends the TLP to Bob to solve.

Steps 4.6 and 4.7 form the shortcut for an efficient creation of the puzzle. They are not included in the generic Algorithm 1. We discuss the efficiency later in Section 4.2.3.

4.2.2 Security

Rivest et al. [2] do not formally discuss the properties *correctness* and *soundness* for their invention, time-lock puzzles. The style of said publication differs not only in description but also in the extent of the technical presentation of the encryption scheme to more modern publications. Formal properties for TLPs have been introduced in more recent literature, e.g. by Malavolta and Thyagarajan [44]. In this publication the authors put forward the construct of *homomorphic time-lock puzzles*. In this process they formally define correctness and security for TLP as presented by Rivest et al. [2].

Trying to solve the puzzle shows why this construction still holds up to this day. There are several approaches to obtain the solution of the puzzle.

1. Trying to guess or searching for the encryption key K directly is infeasible by design, if K is chosen large enough.

2. If $\phi(N)$ is known, then one can reduce the formula using Equation (4.6) so that y can be calculated efficiently by substituting to obtain

$$y = x^e \pmod{N}. \tag{4.8}$$

   Note, however, that computing $\phi(N)$ from N is provably as hard as factoring N, assuming that the primes p and q are chosen large enough. Hence, this approach is also infeasible.

3. Therefore, the fastest known way to solve the puzzle is for Bob to somehow determine

$$y = x^{2^T} \pmod{N}. \tag{4.9}$$

   - Assuming that Alice destroys the variables p, q and $\phi(N)$ the fastest way to calculate y is to start with x and perform T sequential squarings

$$\underbrace{x \to x^{2^1} \to x^{2^2} \to x^{2^3} \to ... \to x^{2^T}}_{T \text{ steps}} \pmod{N}. \tag{4.10}$$

   - With y Bob can then recover the original key K by calculating

$$K = C_K - y \pmod{N}. \tag{4.11}$$

   - Finally, he uses the decryption algorithm of RC5 to reverse the encryption and decrypts C_M to obtain M:

$$M = \mathsf{RC5.Dec}(K, C_M) \tag{4.12}$$

Algorithm 2 Time-Lock Puzzle Honest Computation

Input

Time-lock puzzle: (N, x, T, C_K, C_M)

Algorithm

$y \leftarrow x^{2^T} \pmod{N}$
$K \leftarrow C_K - y \pmod{N}$
$M \leftarrow \mathsf{Dec}(K, C_M)$
output M

4.2.3 Efficiency of TLP

Generator Efficiency

Generating a TLP requires the creator to compute x^{2^T} to obtain C_K (see Equation 4.5). However, the shortcut shown in steps 4.6 and 4.7 is possible because the creator knows the factorization of $N = p \cdot q$. With this shortcut the complexity of the puzzle creation is significantly reduced to require only $\mathcal{O}(\log_2 T)$ squarings [53].

Solver Efficiency

Rivest et al. [2] describe the computation of a TLP as an "intrinsically sequential" process. Honestly solving a TLP as shown in Algorithm 2 is considerably more computationally expensive than generating it. Even with large parallelism is it necessary to complete T sequential steps (see Equation 4.10 to compute $x^{2^T} \pmod N$). Hence, solving a TLP is only possible in $\Omega(T)$ sequential time [54].

4.2.4 TLP in Practice: LCS35 Time Capsule Crypto-Puzzle

To show how TLPs perform in practise it is worth looking at LCS35, a cryptographic challenge and TLP set by Ronald Rivest in 1999 [58]. LCS35 was created to honour the 35th anniversary of the MIT Laboratory for Computer Science, now part of MIT Computer Science and Artificial Intelligence Laboratory (CSAIL) [59]. Rivest constructed a time capsule in form of time-lock puzzle, which is based on the before mentioned 1996 paper by Rivest et al. [2].

Rivest intended the puzzle to last 35 years (1999-2034). In other words, if one started calculating the result in 1999, it was expected to take you around 35 years of continuous computation to obtain the solution. These approximated 35 years equate to roughly 80 trillion (10^{12}) successive squarings.

The challenge was solved twenty years later, in 2019, by self-taught programmer Bernard Fabrot [60]. The Belgian therefore produced the result 15 years earlier than MIT scientists expected. It took Fabrot almost 3.5 years of continuous computation on an ordinary Intel Core i7-6700 CPU core to finish the challenge. Another team finished LCS35 only weeks later, but, in contrast, it took them only around 2 months of computation. The embedded message of the TLP contained the string *"!!! Happy Birthday LCS !!!"* in numeric values.

Rivest was quoted to have underestimated the hardware and software advances. "The puzzle's fundamental challenge of doing roughly 80 trillion squarings remains unbroken, but the resources required to do a single squaring have been reduced by much more than I predicted." [60]

4.3 Further Constructions: Trusted Third Party

Using trusted agents was the alternative idea behind time-lock encryption and also brought up by May in 1993 [49]. His idea was based on using a pool of trusted third parties. He wanted to employ a secret sharing scheme by splitting the message M into several pieces $m_1, m_2, ..., m_n$ and distributing the pieces as sealed messages to escrow agents. By sealing the message and not telling the agents about the contents May wanted to reduce the temptation to peek inside the hidden message.

Each agent would get an agreed sum of money if they stored the message the agreed amount of time. After the time passed, they would release the information to the public. This would create a network of pieces of information that is being made public over time. May himself admitted that this idea was not fleshed out yet.

Rivest et al. [2] expanded on this basic secret sharing scheme and introduced an approach where a trusted third party creates and distributes public and private keys at appropriate times. The construction

also uses agents. The main task of an agent is to periodically publish shares of the secret key K. Let s_{it} denote the secret published by agent i at time t. The secondary task of an agent is to perform encryptions.

The agent receives a request with some y and t and is expected to return $\mathsf{Enc}(s_{it}, y)$, the encryption of y under the secret key s_{it} that will be published by the agent at time t. They produce the digitally signed response

$$(i, t, t_0, \mathsf{Enc}(s_{it}, y)) \tag{4.13}$$

where i is the agent's index, t the requested publishing time, t_0 the current time and $\mathsf{Enc}(s_{it}, y)$ the requested ciphertext. Note that the response is encrypted with the public key of the requestor and signed with the private key of the agent. The published sequences of secret values of each agent are independent.

Additionally, the secret an agent reveals at time t enables the computation of all previously released secrets. Thus, it is always possible to reconstruct the decryption key k once time t is reached and the agents released their corresponding secrets.

A message M which is to be released at time t is encrypted with some key K to yield the ciphertext $C_M = \mathsf{Enc}(K, M)$. The user may choose a threshold θ (where $0 < \theta \leq d$) so that one can reconstruct K given θ or more time-released shares and the corresponding agent's secrets for time t. To use the construction the user picks some number d of agents $i_1, i_2, ..., i_d$ and splits K into d shares $y_1, y_2, ..., y_d$ according to some secret sharing construction with threshold θ. Then they ask agent i_j (for $1 \leq j \leq d$) to produce the "time-released share"

$$r_j = \mathsf{Enc}(s_{i_j t}, y_j), \tag{4.14}$$

the encryption of share y_j of K with secret $s_{i_j t}$ of agent i_j that will be revealed at time t. The user then publishes

$$(C, i_1, i_2, ..., i_d, r_1, r_2, ..., r_d) \tag{4.15}$$

where the time-released shares $r_1, r_2, ..., r_d$ allow the reconstruction of K once time t is reached.

In summary, as long as θ agents are still around at time t the message M will be reconstructible at time t and at any later time. If fewer than θ agents have been corrupted, the message M will not be revealed before time t.

Chapter 5

Verifiable Delay Functions

In this chapter we first present the basics of the cryptographic primitive VDF. In the subsequent section we cover the TLP-based VDF variant by Pietrzak [17] and discuss his scheme in more detail.

5.1 Overview

The cryptographic primitive of *verifiable delay functions* was introduced by Boneh et al. [10] in 2018. A verifiable delay function (VDF) is a pseudorandom function that can only be evaluated after a specified number of steps and then produces a unique output as well as a verifiable proof that these steps have been performed to produce the output. VDFs also guarantee, that it takes a party at least a certain amount of time to evaluate a certain number of steps. This prescribed computing time is also required on a parallel computer. Further, the proof corresponding to the produced output is efficiently and publicly verifiable. In other words, VDFs take the core concept of TLPs [2] and extend it to be efficiently and publicly verifiable, while still requiring to perform a certain amount of sequential steps by the computing party [6].

In the paper "Simple Verifiable Delay Functions", Pietrzak [17] models a concrete instantiation of a VDF using TLPs. He starts with a TLP by Rivest et al. [2] and displays how a TLP can be made publicly verifiable and thus constructing a novel VDF candidate. The author achieves this by modelling a sound public-coin protocol that verifies the proof of a TLP and subsequently using the Fiat-Shamir heuristic to make this protocol non-interactive and hence creating a VDF.

5.2 VDF

A VDF is a function $\mathcal{X} \to \mathcal{Y}$ that requires a specified time to compute and is quickly verifiable by anyone after the output is produced. Also, every input $x \in \mathcal{X}$ must have a unique valid output $y \in \mathcal{Y}$. We refer to the (sequential) time needed for computation on a single processor as *total time*. The running time for VDFs is measured with regards to the number of sequential steps required to calculate the output of a function. In the following work by Pietrzak [17] a "sequential step" is the $\circ$ operation in the group of (signed) quadratic residues, which essentially is a multiplication modulo an RSA modulus.

5.2.1 Construction

The general description of VDFs above is formally defined in the following way by the authors:

Definition 9 (Verifiable Delay Function (VDF) [10, 61])**.** A *verifiable delay function* (VDF) $V = (\mathsf{VDF.Setup}, \mathsf{VDF.Eval}, \mathsf{VDF.Verify})$ implements a function $f : \mathcal{X} \to \mathcal{Y}$ that consist of the following triple of algorithms:

- $\mathsf{VDF.Setup}(\lambda, T) \rightarrow pp = (ek, vk)$ is a randomized algorithm that takes a security parameter λ and a desired time bound T, and outputs public parameters pp that consists of an evaluation key ek and a verification key vk.
- $\mathsf{VDF.Eval}(ek, x) \rightarrow (y, \pi)$ takes an input $x \in \mathcal{X}$ and outputs a $y \in \mathcal{Y}$ and a (possibly empty) proof π.
- $\mathsf{VDF.Verify}(vk, x, y, \pi) \rightarrow \{\mathsf{accept}, \mathsf{reject}\}$ is a deterministic algorithm takes an input $x \in \mathcal{X}$, output $y \in \mathcal{Y}$ and proof π and outputs accept if y is the correct evaluation of the VDF on input x and reject if not.

The domain $\mathcal{X}$ and codomain $\mathcal{Y}$ of f are defined in the public parameters pp. The authors assume that $\mathcal{X}$ is efficiently sampleable. Boneh et al. [10] define that $\mathsf{VDF.Setup}(\lambda, T)$ must run in polynomial-time in λ.

On the other hand, algorithm $\mathsf{VDF.Eval}(ek, x)$ must run in parallel time T with $\mathsf{polylog}(\log_2 T, \lambda)$ processors for all pp generated by $\mathsf{VDF.Setup}(\lambda, T)$ and all $x \in \mathcal{X}$.

VDF.Verify must in turn run in total time polynomial in $\log_2 T$ and λ. We note that VDF.Verify runs considerably quicker than VDF.Eval.

5.2.2 Security

VDF.Verify must accept every output of VDF.Eval. The authors guarantee the uniqueness of every output y on input x, because the algorithm VDF.Eval evaluates a deterministic function on $\mathcal{X}$. Interestingly, the proof π must not be unique, but is required to be sound and that a different output y' cannot be accepted by the verifier as correct output. Also, a VDF V is required to be *correct* and must provide that the proof is *sound*. This allows to define to define correctness as follows.

Definition 10 (VDF Correctness [10, 61])**.** A VDF V is *correct* if for all λ, T, parameters $(ek, vk) \leftarrow \mathsf{VDF.Setup}(\lambda, T)$, and all $x \in \mathcal{X}$, if $(y, \pi) \leftarrow \mathsf{VDF.Eval}(ek, x)$ then $\mathsf{VDF.Verify}(vk, x, y, \pi) = \mathsf{accept}$.

For a VDF to be secure, even an adversary with a large number of parallel processors cannot compute an output for VDF.Eval in time less than T and pre-computation should not accelerate this outcome. The authors specify a time bound for the allowed parallelism for an adversary in the *sequentiality* property. A VDF V as introduced above satisfies the following security properties [61]:

- *ϵ-Evaluation time*:

  $\mathsf{VDF.Eval}(ek, x)$ runs in time at most $(1 + \epsilon)T$, for all $x \in \mathcal{X}$ and for all pp produced by $\mathsf{VDF.Setup}(\lambda, T)$.

- *Sequentiality*:

  A parallel algorithm $\mathcal{A}$, using at most $\mathsf{polylog}(\lambda)$ processors, that runs in time less than T cannot compute the function. Specially, for a random $x \in \mathcal{X}$ and pp output by $\mathsf{VDF.Setup}(\lambda, T)$, if $(y, \pi) \leftarrow \mathsf{VDF.Eval}(pp, x)$ then $Pr[\mathcal{A}(pp, x) = y]$ is negligible.

- *Uniqueness*:

  For an input $x \in \mathcal{X}$, exactly one $y \in \mathcal{Y}$ will be accepted by VDF.Verify. Specifically, let $\mathcal{A}$ be an efficient algorithm that given pp as input, outputs (x, y, π) such that $\mathsf{VDF.Verify}(vk, x, y, \pi) =$ accept. Then $Pr[\mathsf{VDF.Eval}(pp, x) \neq y]$ is negligible.

5.3 From TLP to VDF

In order to transform a TLP by Rivest et al. [2] to a VDF it has to be modified to be publicly verifiable. Pietrzak [17] uses the following slightly simplified model of TLPs as starting point:

Definition 11 (Pietrzak's Time-Lock Puzzle (P-TLP) [17])**.** A *Pietrzak Time-Lock Puzzle* (P-TLP) consists of two elements.

- *The puzzle*

  The puzzle is of a tuple (N, x, T) where $N = p \cdot q$ is an RSA-modulus of two primes p and q, $x \in \mathbb{Z}_N^*$ is a random positive integer and $T \in \mathbb{N}$ is a time parameter.

- *The solution*

  The solution of the puzzle is $y = x^{2^T} \pmod N$. It can be computed making two exponentiations by the party who generates the puzzle (and thus knows the group order $\phi(N) = (p-1)(q-1)$) as

  $$e = 2^T \pmod{\phi(N)}, \tag{5.1}$$

  and then

  $$y = x^e \pmod N. \tag{5.2}$$

  But requires T sequential squarings if the group order (or equivalently the factorization of N) is not known.

Pietrzak omits details regarding the encryption and decryption of a message, as well as the message itself. P-TLPs focus on the hardness of producing an honest computation of a solution without knowing N or the factorization of N.

The main challenge is to efficiently verify $y \stackrel{?}{=} x^{2^T} \pmod N$. In order to accomplish this, one needs to have knowledge on the group order of $x \in \mathbb{Z}_N^*$ or alternatively know the factors of N, p and q. However, the factorization cannot be made public because this would enable the computation of y easy and therefore nullify the difficulty.

5.3.1 Algebraic Setting

Pietrzak modifies the algebraic setting from the original TLP by Rivest et al. [2]. Firstly, he uses random *safe primes* p and q to calculate N, instead of random regular primes. A prime p is *safe* if $p = 2q + 1$, where q is also prime [62]. Secondly, his construction executes the computations in the group of *signed quadratic residues* [63, 64], denoted QR_N^+, instead of $(\mathbb{Z}_N^*, \cdot)$. More specifically, he uses a random quadratic residue $x \in QR_N$, which is a generator of QR_N:

$$QR_N^+ = \{z^2 \pmod N : z \in \mathbb{Z}_N^*\} \tag{5.3}$$

where the group operation is defined as:

$$a \circ b = a \cdot b \pmod N. \tag{5.4}$$

Additionally, by executing the computations in the group of signed quadratic residues QR_N^+ allows one to efficiently decide if an element is in QR_N^+ or not, while QR_N does not provide this property. Furthermore, using QR_N^+ results in a unique proof for the construction.

Both changes are implemented to obtain a proof for statistical soundness and more efficiency in this construction. Pietrzak shows that using $(QR_N^+, \circ)$ instead of $(\mathbb{Z}_N^*, \cdot)$ does not make computing x^{2^T}, with $x^2 \stackrel{\text{def}}{=} x \circ x$ for $x \in QR_N^+$, significantly easier.

5.3.2 Protocol

Pietrzak introduces an elegant solution involving a prover $\mathcal{P}$ that can convince a verifier $\mathcal{V}$ that it correctly computed $y = x^{2^T} \pmod N$, where neither prover nor verifier know the factorization of N. This public-coin protocol is then made non-interactive using the Fiat-Shamir transformation, so that one can create a VDF. The protocol goes as follows.

1. $\mathcal{P}$ and $\mathcal{V}$ have as common input a P-TLP (N, x, T) and a security parameter λ with $T \in \mathbb{N}$, p, q safe primes, $N := p \cdot q$ and $x \in QR_N^+$.
2. $\mathcal{P}$ performs T sequential squarings in $(QR_N^+, \circ)$ and computes $y = x^{2^T}$. $\mathcal{P}$ then sends y to $\mathcal{V}$.
3. $\mathcal{P}$ and $\mathcal{V}$ execute the halving subprotocol shown in Figure 5.1. They iterate the subprotocol on common initial input (N, x, T, y) until it stops with output accept or reject. The subprotocol uses the preliminary outputs $(n, x', \lceil T/2 \rceil, y')$ on line 21 or 23 as input for the next iteration and is executed until a final output accept or reject is reached.

The key idea behind the proof is straightforward. We have a prover $\mathcal{P}$ that wants to convince a verifier $\mathcal{V}$ that the tuple (x, y) satisfies $y = x^{2^T}$. To accomplish this, $\mathcal{P}$ starts with sending $\mu = x^{2^{T/2}}$ to $\mathcal{V}$. We now have $y = \mu^{2^{T/2}}$ together with $\mu = x^{2^{T/2}}$ that imply $y = x^{2^T}$.

Then $\mathcal{V}$ can merge these two statements in a randomized way into a single statement $(x', y') = (x^r \cdot \mu, \mu^r \cdot y)$. If the original statement $y = x^{2^T}$ was correct (and $\mathcal{P}$ sends the correct μ) then $y' = x'^{2^{T/2}}$. However, we almost certainly get $y' \neq x'^{2^{T/2}}$ (over the choice of the random exponent r) if the original statement was wrong, independent of which μ a malicious prover sent.

This subprotocol halves the time parameter T each time and is iterated $\log_2 T$ times until $T = 1$. By then $\mathcal{V}$ can efficiently verify the correctness of the claim itself.

5.3.3 Transformation to VDF

We first need to make minor adjustment to the basic definition of VDFs displayed in Definition 9. In short, we divide VDF.Setup into two separate algorithms P-VDF.Setup and P-VDF.Gen, so that x is being randomly sampled and include the time parameter T as input for P-VDF.Eval and P-VDF.Verify. This is required so that we can incorporate the variables from the TLP into the construction.

Definition 12 (Pietrzak Verifiable Delay Function (P-VDF) [17])**.** A P-VDF is defined by a four-tuple of algorithms:

- P-VDF.Setup$(1^\lambda) \to pp$ takes a statistical security parameter 1^λ and outputs public parameters pp.
- P-VDF.Gen$(pp, T) \to (x, T)$ takes as input the public parameters pp and a time parameter $T \in \mathbb{N}$ and samples a random x.
- P-VDF.Eval$(pp, (x, T)) \to (y, \pi)$ on input (x, T) outputs the tuple (y, π), where π is the proof that y has been computed correctly.
- P-VDF.Verify$(pp, (x, T), (y, \pi)) \to \{$accept, reject$\}$ takes input/output tuples (x, T) and (y, π) and outputs either accept or reject.

Next, we transform the protocol from Section 5.3.2 into a VDF. For simplicity, we assume that $T = 2^t$ is an exponent of two. However, the protocol can be modified to handle any T. We instantiate the algorithms specified in Definition 12 in the following way:

Halving Subprotocol

1 :	**Verifier** $\mathcal{V}$	**Prover** $\mathcal{P}$
2 :	*common input*	
3 :	input $\leftarrow (N, x, T, y)$	
	....	
4 :	**if** $T = 1$ **then**	
5 :	**if** $y \geq x^{2^T}$ **then**	
6 :	output **accept**	
7 :	**else**	
8 :	output **reject**	
	....	
9 :		**if** $T > 1$ **then**
10 :		$\mu \leftarrow x^{2^{T/2}}$
11 :	send over μ $\longleftarrow$	
	....	
12 :	**if** $\mu \notin QR_N^+$ **then**	
13 :	output **reject**	
14 :	**else**	
15 :	$r \leftarrow\!\!\$\, \mathbb{Z}_{2^\lambda}$	
16 :	send over r $\longrightarrow$	
	....	
17 :	*common instructions*	
18 :	$x' \leftarrow x^r \cdot \mu \Big(= x^{r+2^{T/2}} \Big)$	
19 :	$y' \leftarrow \mu^r \cdot y \Big(= x^{r+2^{T/2}+2^T} \Big)$	
20 :	**if** $T/2$ is even **then**	
21 :	Halving Subprotocol$(N, x', T/2, y')$	
22 :	**elseif** $T/2$ is odd **then**	
23 :	Halving Subprotocol$(N, x', (T+1)/2, y'^2)$	

Figure 5.1. Halving Subprotocol

Definition 13 (Pietrzak's VDF from TLP (TLP-VDF) [17])**.** A VDF from a TLPs consists of the four algorithms:

- $\mathsf{TLP\text{-}VDF.Setup}(1^\lambda) \to N$ the statistical security parameter λ defines another security parameter λ_{RSA} specifying the bitlength of an RSA modulus, where λ_{RSA} should be at least as large so that an λ_{RSA}-bit RSA modulus offers λ bits of security (e.g. $\lambda = 100$ and $\lambda_{\mathsf{RSA}} = 2048$).

  $\mathsf{TLP\text{-}VDF.Setup}$ samples two random $\lambda_{\mathsf{RSA}}/2$-bit safe primes p, q and outputs as public parameter the RSA modulus $N := p \cdot q$.

- $\mathsf{TLP\text{-}VDF.Gen}(N, T) \to (x, T)$ samples a random $x \in QR_N^+$ and outputs (x, T).

- $\mathsf{TLP\text{-}VDF.Eval}(N, (x, T)) \to (y, \pi)$ outputs the tuple (y, π), where $y = x^{2^T}$ is the solution to the TLP (but over $(QR_N^+, \circ)$ not $(\mathbb{Z}_N^*, \cdot)$) and $\pi = \{\mu_i\}_{i\in[t]}$ the corresponding proof that y has been computed correctly.

  We obtain π from the protocol in 5.3.2 by applying the Fiat-Shamir heuristic and consequently making the protocol non-interactive. In this heuristic the public-coin challenges $r_i \in \mathbb{Z}_{2^\lambda}$ of the verifier are replaced with a hash of the last prover message. Concretely we use a hash function $H : \mathbb{Z} \times \mathbb{Z}_N^4 \to \mathbb{Z}_{2^\lambda}$ and proceed as depicted in Algorithm 3.

Algorithm 3 Fiat-Shamir Heuristic in $\mathsf{TLP\text{-}VDF.Eval}$

$(x_1, y_1) \leftarrow (x, y)$
for $i = 1, ..., t$ **do**
    $\mu_i \leftarrow x_i^{2^{T/2^i}} \in QR_N^+$
    $r_i \leftarrow H((x_i, T/2^{i-1}, y_i)\mu_i)$
    $x_{i+1} \leftarrow x_i^{r_i} \circ \mu_i$
    $y_{i+1} \leftarrow \mu_i^{r_i} \circ y_i$

- $\mathsf{TLP\text{-}VDF.Verify}(N, (x, T), (y, \pi)) \to \{\mathsf{accept}, \mathsf{reject}\}$ parses $\pi = \{\mu_i\}_{i\in[t]}$ and proceeds as depicted in Algorithm 4 until the proof is either verified with output accept or refuted with output reject.

Algorithm 4 $\mathsf{TLP\text{-}VDF.Verify}$

if $x, y \notin QR_N^+$ or $\exists \mu_i \notin QR_N^+$ **then**
    output reject
else
    $(x_1, y_1) \leftarrow (x, y)$
    for $i = 1, ..., t$ **do**
        $r_i \leftarrow H((x_i, T/2^{i-1}, y_i)\mu_i) \in \mathbb{Z}_{2^\lambda}$
        $x_{i+1} \leftarrow x_i^{r_i} \circ \mu_i$
        $y_{i+1} \leftarrow \mu_i^{r_i} \circ y_i$
    if $y_{t+1} = x_{t+1}^2$ **then**
        output accept
    else
        output reject

We now briefly discuss how this VDF candidate construction provides the required properties (see Section 5.2.2) of a VDF.

- *ϵ-Evaluation time*:

  The prover efficiency of running TLP-VDF.Eval will be discussed in more detail in the next section. Note, however, that TLP-VDF.Eval requires $\mathcal{O}(1+\frac{2}{\sqrt{T}}T)$ and therefore provides the required time of at most $\mathcal{O}(1+\epsilon)T$.

- *Sequentiality*:

  The sequentiality of TLP-VDFs relies on the same assumption as Rivest et al. [2] for TLPs. An honest solver must compute y in less than T sequential steps to break the sequentiality. The authors state that such a shortcut does not exist, unless $p, q, \phi(N)$ are known. Pietrzak shows that working over $(QR_N^+, \circ)$ instead of $(\mathbb{Z}_N^*, \cdot)$ does make the assumption weaker, but it still holds. Also, using *safe* primes for p, q has no impact the assumption.

- *Uniqueness*:

  For every x there is only a single output $y = x^{2^T}$ with corresponding proof π that produces accept from TLP-VDF.Verify. Further, the proof π produced in TLP-VDF.Eval is unique as well, because we use $(QR_N^+, \circ)$ in place of $(\mathbb{Z}_N^*, \cdot)$.

5.3.4 Efficiency of TLP-VDF

Prover Efficiency

The cost of computing the proof $(y, \pi) \leftarrow \mathsf{TLP\text{-}VDF.Eval}(N, (x, T)))$ consists of two steps. First, the prover has to perform T sequential squarings on x to solve the embedded TLP $y = x^{2^T}$. The fact that this computation is performed in $(QR_N^+, \circ)$ instead of $(\mathbb{Z}_N^*, \cdot)$ and that we use *safe* primes does not break the security assumption of TLPs. Secondly, one must compute the corresponding proof $\pi = \{\mu_i\}_{i\in[t]}$ where $\mu_i = x_i^{2^{T/2^i}}$ to obtain a VDF. We still work with the assumption that $T = 2^t$ is a exponent of 2.

Note that the prover is required to compute μ_i for each corresponding recursion. If naïvely implemented, computing μ_1 requires $T/2$ squarings, $T/2^2$ for μ_2, $T/2^3$ for μ_3, etc. This adds up to a total of $T \approx T/2 + T/4 + T/8 + ... + 1$ sequential steps.

However, it is not required to compute $\mu_1 = x^{2^{T/2}}$ from scratch, as the prover already calculated $y = x^{2^T}$ by repeated squaring. Consequently, the remaining $\mu_2, \mu_3, ...$ can be calculated using stored values. This becomes increasingly costly. For example, assuming that we stored $x^{2^{T/8}}, x^{2^{3T/8}}, x^{2^{5T/8}}$ and $x^{2^{7T/8}}$, we can use these values to compute μ_3:

$$\begin{aligned}
\mu_3 &= x_2^{2^{T/8}} \\
&= (x_1^{r_2} \cdot \mu_2)^{2^{T/8}} \\
&= (x^{r_1 \cdot r_2} \cdot \mu_1^{r_2} \cdot \mu_2)^{2^{T/8}} \\
&= (x^{2^{T/8}})^{r_1 \cdot r_2} \cdot (x^{2^{3T/8}})^{r_1} \cdot (x^{2^{5T/8}})^{r_2} \cdot (x^{2^{7T/8}}).
\end{aligned} \tag{5.5}$$

For some $s \in [t]$, the prover requires to store 2^s values $\{x^{2^{T\cdot i/2^s}}\}_{i\in[2^s]}$ and then to perform 2^s exponentiations to compute $\mu_1, \mu_2, ..., \mu_s$. By doing so we obtain that the proof contains $\log_2 T$ elements [61]. Overall, TLP-VDF.Eval requires $\mathcal{O}(1 + \frac{2}{\sqrt{T}}T)$ to calculate the VDF output and the corresponding proof π [65].

To put this in perspective, Pietrzak [17] outlines that it is $\approx 2^{13}$ times faster to obtain π than to compute y for typical values such as $t = 40$, $\lambda = 100$. For example, if it takes 1 hour to obtain y, then computing the corresponding π takes only about half a second additionally.

Verifier Efficiency

Each iteration of TLP-VDF.Verify requires the verifier to perform two small exponentiations to calculate x_{i+1} and x_{i+1} respectively for the next repetition. Thus, the verification of the proof takes around $2 \cdot log_2 T$ group exponentiations and consequently requires $\mathcal{O}(\log_2 T)$.

5.4 Further Constructions: Trustless Setup

It is essential for the security of TLP-VDFs that the prover does not know the factors p and q of the public parameter N. If they are known, the prover can simply use the shortcut of the TLP to efficiently compute $y = x^{2^T}$ as shown in Equation 5.1 of Definition 11 and the scheme is broken.

For this construction, one would therefore ideally use a trustless RSA group selection protocol, that is where no party knows the factorization of N. Pietrzak discusses either involving a trusted party or using multi-party computation to sample $N = p \cdot q$. It is stated that this has been done before but there are also random-oracle based proof of sequential work applications [66] which function without a setup procedure.

The alternative is to entirely avoid a trusted setup. This can be achieved by replacing the RSA group with the class group of an imaginary quadratic number field [6, 61]. These groups of unknown order can be obliviously samples. Simply put, choosing a large negative prime generates a group whose order is computationally infeasible to determine, even for parties knowing the prime. However, this class group lacks research and it is believed to be difficult to find low-order elements in class groups of quadratic number fields [67]. RSA groups, in contrast, do not suffer from this drawback.

Chapter 6

Blockchain-Based TLE

In this chapter, we first discuss a generic blockchain-based approach to TLE using computational reference clocks. In a second step, we present the specific implementation based on the Bitcoin blockchain.

6.1 Overview

Liu et al. [18] propose a new time-lock encryption scheme which is fundamentally different from the original implementation by Rivest et al. [2]. The construction does not rely on *trusted* third parties nor do they involve a computational overhead (e.g. time-lock puzzles) for the decryption. The authors suggest a combination of a witness encryption scheme and a computational reference clock, which is a primitive introduced by themselves. The scheme consists of four major parts:

- Introducing the notion of *computational reference clocks*
- Obtaining an NP-relation from the Bitcoin blockchain
- Constructing a witness encryption scheme from this NP-relation
- Combining the computational reference clock and the witness encryption scheme to implement time-lock encryption

The main idea for the construction of the time-lock encryption scheme by Liu et al. [18] is the combination of the two primitives *computational reference clocks* and witness encryption schemes. Note that witness encryption in itself is not associated with time-lock encryption. The authors enable a feasible way of constructing time-lock encryption from a blockchain by introducing the notion of computational reference clocks. The authors first propose a generic construction using the blockchain of any decentralized cryptocurrency and introduce in a second step a more concrete proposal using Bitcoin. They state that the generic construction can be adapted to use any large-scale, public, iterative computation. The introduced CRCs are simply designed to match the mechanism of a blockchain, but the underlying principle remains feasible for other computations. In this thesis we mainly discuss the more concrete implementation using the Bitcoin blockchain. Liu et al. [18] chose to instantiate Bitcoin in particular because of their wide adoption and the considerable number of computational resources contributed to the Bitcoin network.

The Bitcoin blockchain enables to define an instance for which there will only be a corresponding witness in the future [68]. The authors therefore use the cryptocurrency Bitcoin to instantiate a computational reference clock for the construction of time-lock encryption.

An NP-relation R is defined [18] such that

1. For $x \in \mathbb{N}$, statements have the form 1^x, that is, x in unary representation.
2. Any valid blockchain $w = (B_1, ..., B_x)$ of length $\geq x$ is a witness for statement 1^x, that is $(1^x, w) \in R$.

One can assume that a witness encryption scheme $(\mathsf{WE.Enc}, \mathsf{WE.Dec})$ for this relation R exists. Let the current state of the Bitcoin blockchain be $(B_1, ..., B_\tau)$. Then a witness w for $(1^x, w) \in R$ exists for all $x \leq \tau$ inside the blockchain. As the Bitcoin blockchain is public, anyone can immediately decrypt any ciphertext

$$c \leftarrow \mathsf{WE.Enc}(1^x, w) \text{ with } x \leq \tau,$$

by using the witness from the public blockchain. This witness functions as "decryption key".

This construction is not bound to the genesis block of the blockchain. The initial state w_0 of the "clock" can be at any point in the blockchain and older blocks will thus be ignored.

The authors also state, that not all transaction data has to be included on the witness. The hash chain of the block headers suffices for decryption, a single block header contains 80-byte of data.

It is worth pointing out that this construction of Liu et al. [18] specifically does not put any trust in the Bitcoin miners. They do not store any secret information and their computations can be completely public.

The authors state that the scheme guarantees the following properties simultaneously:

- Non-interactivity: The decryption does not require the encryptor
- No trusted setup: No trusted third parties are needed
- No resource restrictions: Decrypting parties do not require to perform large computations (with the assumption that the Bitcoin blockchain does not undergo any major changes)

6.2 Building Time-Lock Encryption

In this section we expand the basic concepts introduced in Chapter 2 which establish the core of this time-lock encryption scheme. Throughout this chapter we mainly use definitions from the paper by Liu et al. [18]. Note that this thesis covers only selected definitions and explanations, which form the essence of the constructions. For a deeper understanding and more extensive technical details, we advise to read the discussed and the referenced literature.

With the definition of the complexity class NP in general from Section 2.2 in place, we can turn to the specifics used by Liu et al. [18] for the scheme. Before introducing the two main building blocks of this construction one must understand the basics of NP-relations. The authors use the following definition.

Definition 14 (NP-Relation [18])**.** Let R be a relation. We say that R is an *NP-Relation*, if there exists a deterministic polynomial-time (in $|x|$) algorithm that, on input (x, w), outputs 1 if and only if $(x, w) \in R$.

Remark on the used vocabulary

In the following there are two different notions of "time". Firstly, there is the running time of an algorithm which is usually measured in the number of executed computational steps. Additionally, the authors introduce an approach for a computational equivalent of physical time, a so-called "computational reference clock", which quantifies physical time in some abstract discrete time unit. Therefore, we use the word "time" to describe the computational equivalent of physical time.

Similarly, we use the "number of operations" to specify the running time of an algorithm. We assume that all algorithms are performed by universal Turing machines. For example, we write "algorithm $\mathcal{A}$ performed t operations" instead of "algorithm $\mathcal{A}$ ran in time t".

6.2.1 Witness Encryption

The notion of witness encryption for NP-relations was introduced by Garg et al. [22] and explained in Section 2.3 of this thesis. In witness encryption, an instance x of some NP-language is encrypted to produce some ciphertext c. The decryption of a ciphertext is possible by using any witness w that can confirm that $x \in L$ [69].

The investigated construction uses a variant of witness encryption which is based on multilinear maps. They introduce a novel extractable witness encryption scheme built on the Subset-Sum-problem. They perform a reduction from CNF-SAT to obtain a scheme which enables the encryption with any NP language. This witness encryption construction is said to be the first without utilizing obfuscation, while still producing extractable security.

6.2.2 Computational Reference Clock

Liu et al. [18] propose the notion of computational reference clocks as computational model to measure real-world time. CRCs are introduced in the background chapter in Section 2.6. CRCs utilise the block creation inside the blockchain of a cryptocurrency as reference point of passed physical time. More specifically, the authors opt to use the popular cryptocurrency Bitcoin [21] (see also Section 2.5) as a reference for passed physical time for the encryption scheme. So-called miners contribute significant computational resources to the steady extension of the Bitcoin blockchain. This blockchain consists of an array of hash values $B_1, ..., B_\tau$ that satisfy various specifications. These specifications determine the *mining difficulty*, the difficulty of finding a new block $B_{\tau+1}$ in the chain. The difficulty is adjusted based on the available computational resources contributing to the Bitcoin network.

Remember that on average a new block is introduced to the blockchain every 10 minutes in expectation [39]. Therefore, the length τ of the blockchain is utilised as reference clock, where τ indicates the current "time" and each subsequent block represents 10 minutes of passed time. Intuitively, this can be seen like a clock that ticks forward every 10 minutes.

With the established notion of CRCs, it is possible to construct a general time-lock encryption scheme for CRCs.

Definition 15 (Time-lock encryption scheme using a CRC [18])**.** A *time-lock encryption scheme* for computational reference clock $\mathcal{C}(1^\mathcal{K})$ with message space $\mathcal{M}$ consists of two polynomial-time algorithms $(\mathsf{TL.Enc}, \mathsf{TL.Dec})$.

Encryption	The encryption algorithm $c \leftarrow \mathsf{TL.Enc}(1^\lambda, \tau, m)$ takes as input the security parameter λ, an integer $\tau \in \mathbb{N}$, and a message $m \in \mathcal{M}$. It computes and outputs a ciphertext c.
Decryption	The decryption algorithm $\mathsf{TL.Dec}(w, c)$ takes as input $w \in 0, 1^*$ and ciphertext c, and outputs a message $m \in \mathcal{M}$ or a distinguished error symbol $\perp$.

6.2.3 Time-Lock Encryption Based on Hash Blockchains

With these building blocks in place, it is possible to construct a time-lock encryption scheme using witness encryption. Jumping ahead, we then use witness encryption to verify the validity of a found chain of blocks in the blockchain. In other words, witness encryption is used to confirm that the blockchain reached the target length and that therefore the intended time has passed.

Definition 16 (Time-lock encryption from witness encryption [18])**.** Let $\mathcal{C}$ be a computational reference clock and let R be an NP-relation, such that R is associated to $\mathcal{C}$. Let $(\mathsf{WE.Enc}, \mathsf{WE.Dec})$ be a witness encryption scheme for R. Define algorithms $(\mathsf{TL.Enc}, \mathsf{TL.Dec})$ of a time-lock encryption scheme as

$$\mathsf{TL.Enc}(1^\lambda, \tau, m) := \mathsf{WE.Enc}(1^\lambda, 1^\tau, m) \tag{6.1}$$

and

$$\mathsf{TL.Dec}(w, c) := \mathsf{WE.Dec}(w, c) \tag{6.2}$$

Liu et al. [18] show with the concrete example of Bitcoin how the abstract concept of a computational reference clock can be instantiated with the use of decentralized cryptocurrencies. They emphasize that any cryptocurrency could be used, but the implementation would have to be adapted to the different block times and hash rates of other cryptocurrencies.

The authors first describe a general NP-relation based on any hash blockchain. They call β the *starting block* and δ the *target bound function*. The following definition allows us to use the blockchain as NP-relation which in turn enables a corresponding witness encryption scheme to exist.

Definition 17 (NP-relation based on hash blockchains [18])**.** Let $H : 0,1^* \to 0,1^d$ be a hash function for some constant d. Let $\beta \in 0,1^d$, and let $\delta : \mathbb{N} \to [0, 2^d - 1]$ be a function with polynomially-bounded description. Let $R_{\beta,\delta}$ be the relation where $(1^x, w) \in R_{\beta,\delta}$ if and only if

$$w = ((T_1, r_1, D_1, B_1), ..., (T_\tau, r_\tau, D_\tau, B_\tau)) \tag{6.3}$$

and w satisfies all the following properties:

- $|T_i|$ and $|r_i|$ are polynomially bounded, and $D_i \in [0, 2^d - 1]$
- w contains at least x tuples (T_i, r_i, D_i, B_i)
- $B_1 = H(T_1, r_1, D_1, \beta)$
- $B_i = H(T_i, r_i, D_i, B_{i-1}$ for all $i \in [2, x]$
- $\delta(i) \geq B_i$ for all $i \in [1, x]$, where we interpret bit strings $B_1, ..., B_x$ canonically as integers

Note that the task of finding a valid blockchain w of length x with respect to (β, δ) corresponds to the problem of identifying a witness w for a given statement 1^x. The difficulty of this computation decreases with an increasing δ, as a higher value for the target represents a longer target hash value and therefore less steps in the calculation.

6.2.4 Time-Lock Encryption Based on Bitcoin

As described, these building blocks can now be coupled with witness encryption. Let $\mathcal{C}_{btc}$ be the Bitcoin-based CRC and $R_{\beta,\delta}$ represents the correlating relation.

Definition 18 (NP-relation based on the Bitcoin blockchain [18])**.** Let $H : \{0,1\}^* \to \{0,1\}^{256}$ be the hash function used in Bitcoin. Let $R_{\beta,\delta}$ be the relation as specified in Definition 17, instantiated with H and the following parameters β and δ.

- Set $\beta := B_o$, where B_0 is the Bitcoin genesis block
- Fix a target bound function $\delta : \mathbb{N} \to [0, 2^{256} - 1]$.

Definition 19 (Bitcoin-based computational reference clock [18])**.** Let f_{btc} be the function that expands the Bitcoin blockchain. That is, on input

$$w_{\tau-1} = (T_1, r_1, D_1, B_1), ..., (T_{\tau-1}, r_{\tau-1}, D_{\tau-1}, B_{\tau-1}) \tag{6.4}$$

and auxiliary input $\mathsf{aux} = T_\tau$, it computes and outputs the new state w_τ such that

$$w_\tau = (T_1, r_1, D_1, B_1), ..., (T_\tau, r_\tau, D_\tau, B_\tau) \tag{6.5}$$

with

$$H(T_\tau, r_\tau, D_\tau, B\tau - 1) = B_\tau \leq D_\tau, \tag{6.6}$$

where D_τ is the current Bitcoin target.

This definition completes the scheme. The function f_{btc} is computed by the CRC denoted as $\mathcal{C}_{btc}$. At "time" τ, $\mathcal{C}_{btc}$ contains of the first τ tuples of the Bitcoin blockchain. As on average roughly every 10 minutes a new block is appended to the chain and assuming that the current length of the blockchain is τ, we can expect the blockchain to contain approximately $\tau + x$ tuples after $\tau \cdot 10$ minutes. Hence, $\mathcal{C}_{btc}$ enables a sender to approximately choose the encryption timing of a message, as long as the block rate of the Bitcoin network remains constant.

Note that *choosing δ carefully* is crucial for this construction to behave as intended. The reason being that the NP-relation $R_{\beta,\delta}$ as specified in Definition 18 is related to $\mathcal{C}_{btc}$ if and only if $\beta = B_0$ and δ satisfies $\delta(i) \geq B_i$ for all $i \in \mathbb{N}$. The last-mentioned property is not guaranteed, because it depends on the size of the Bitcoin target as well as on the choice of the target bound function δ.

Assume a target time x in the future, that is with $x > \tau$. Currently the Bitcoin blockchain contains no witness w for $(1^x, w) \in R_{\beta,\delta}$. Note, however, that any sequence

$$w = ((T_1, r_1, D_1, B_1), ..., (T_x, r_x, D_x, B_x) \tag{6.7}$$

found in the future in the Bitcoin blockchain is only a *potential* witness for $(1^x, w) \in R_{\beta,\delta}$, because $R_{\beta,\delta}$ depends on δ. Definition 17 specifies that w is exclusively a valid witness for $(1^x, w) \in R_{\beta,\delta}$, if and only if $\delta(i) \geq B_i$ for all $i \in [1, x]$. Furthermore, it is possible that at some future point $\gamma \in [\tau + 1, x]$ the Bitcoin target value is increased to a value D_γ such that $D_\gamma > \delta(\gamma)$. Consequently, we would have $(1^x, w) \notin R_{\beta,\delta}$ and the witness w would not be deemed valid.

This can be overcome by choosing δ carefully. Ideally, we chose δ so that it closely matches the Bitcoin target parameter D_i for all future $i > \tau$, that is $\delta(i) = D_i$. This would require predicting the extend of the available contributing computational resources in the Bitcoin network, as D_i is adjusted based on this amount. Since an exact prediction of D_i is infeasible, the authors try to create an approximation $\delta(i) = D_i - \epsilon_i$ with small values ϵ_i.

Liu et al. [18] argue that the "right" choice of δ ultimately depends on the *preference of the sender*. There are generally two options to choose from:

- Choosing δ too small might render it impossible for a witness w to appear in the public Bitcoin blockchain. Consequently, it would not be possible to decrypt the encrypted message. This might happen if the Bitcoin difficulty decreases faster than the encrypting party expected. Therefore, it might be infeasible to invest enough resources to decrypt the ciphertext within reasonable time. Nobody would learn the encrypted message at all or at least not close to the intended deadline.
- Choosing δ too large might allow an adversary to decrypt the ciphertext before the desired deadline, by performing many computations within a short amount of time. This is possible even with significantly less available computational resources than the Bitcoin miners.

The authors hence propose to choose δ either *application-dependent* or possibly even message-dependent, individually for each encrypted message, because no option above is preferable in general. The considerable amount of currently contributing computational resources of the Bitcoin network allows a generous error margin for δ. If an encrypted message must remain secret at the very least until time x and possibly even considerably longer, then a small target bound function δ is desired. However, if one prefers to have the message decrypted rather earlier than possibly never, then a large δ is advised.

6.2.5 Security

Assuming the security of the witness encryption scheme, an adversary has only two possibilities to learn any non-trivial information about the encrypted message m from this construction [22]. Suppose $c = \mathsf{WE.Enc}(1^x, m)$ is a ciphertext with $x > \tau$.

1. The adversary waits until the blockchain reaches length x and therefore waits until the time put on the time-lock has expired. Then the adversary can access a witness w for $(1^x, w) \in R$ in the blockchain, which allows the immediate decryption. However, after length x is reached everybody is able to read w from the public Bitcoin blockchain and to calculate $m = \mathsf{WE.Dec}(c, w)$. This is intended as the encryption scheme is designed to behave in such a way.

2. The adversary tries to compute the missing blocks $B_{\tau+1}, ..., B_x$ faster than the Bitcoin miners. The adversary therefore tries to "advance" the computational reference clock. To achieve that the adversary would have to outperform the computational network of all Bitcoin miners, which currently (April 2021) executes 1.8 x $10^{20} \approx 2^{67}$ hash computations per second [70]. One can assume that an adversary is not able to access enough computational power to achieve this performance.

The actual construction offers more complexity than the simplified description of the construction above. Here details about the "difficulty" parameter within the Bitcoin blockchain are omitted, but the underlying principle is the same.

Additionally, there is a possibility of the occurrence of forks within the blockchain. The authors work with the assumption that the longest chain always corresponds to the chain with the largest sum of difficulties. Hence, an adversary cannot fork the chain by adding a large sum of low-difficulty blocks.

6.3 Further Constructions: SNARK

The proposed implementation of time-lock encryption offers a linear complexity between the length of the ciphertext and the running time for both encryption and decryption of witness encryption regarding the size of the witness [18]. Usually, linearity is natural and efficient. The authors use applications of witness encryption which are based on multilinear maps [29]. Furthermore, the authors state that current implementations of multilinear maps [71, 72, 30] are not yet very practical. More importantly, the size of the group elements used in these multilinear maps is polynomial.

To reduce this problem the authors propose a different construction of TLE by using SNARKs (*Succinct Non-interactive ARgument of Knowledge* [73, 74, 75, 76]) coupled with witness encryption to reduce the complexity of the proposed time-lock encryption scheme.

Definition 20 (SNARKs [73, 74, 75, 76])**.** A SNARK for a relation R is a triple of polynomial-time algorithms ($\mathsf{SNARK.Gen}, \mathsf{SNARK.Prove}, \mathsf{SNARK.Verify}$):

- $\mathsf{SNARK.Gen}(1^\lambda, x) \rightarrow (ev, vk)$ takes as input a security parameter λ and an instance x. It computes and outputs a public evaluation key ek and public verification key vk.

- $\mathsf{SNARK.Prove}(ek, x, w) \rightarrow \pi$ on input $(x, w) \in R$, the prover outputs a non-interactive proof π

- $\mathsf{SNARK.Verify}(vk, x, \pi) \rightarrow b$ on input a verification key vk, an input x, and a proof π, the verifier outputs $b = 1$ if he is convinced that $(x, w) \in R$.

This expanded construction uses the verification procedure of SNARKs to encrypt the statement $(x, w) \in R$, instead of directly encrypting with an instance x in witness encryption. This enables the multilinearity level to be constant regardless of the witness w and instance x while achieving a short ciphertext on witness encryption. The authors note that one can use a third party to set up the SNARK parameters, but it is not required for the implementation. The sender of the message can generate the parameters itself. One can assume this party to be trusted because it knows the plaintext and produces the ciphertext.

Chapter 7

Evaluation Results

In this chapter, we evaluate the explored TLE schemes based on the previously determined criteria. We discuss the properties for each construction and state advantages and limitations of each scheme afterwards. Encryption schemes usually utilize existing cryptographic primitives such as hashing, witness encryption and digital signatures. When assessing the security features of a construction, we assume that these cryptographic building blocks are implemented correctly and deliver the expected security.

7.1 Overview

Table I visualizes the evaluation of the approaches. Columns in the table represent the identified properties and are grouped according to the chosen categories in 3.1. Rows are named after the explored constructions. Each row is evaluated per property, showing if the scheme provides said property, marked with ●, or not, marked with ○. A scheme might only partially provide a property, indicated with ◐. In these cases, we provide an explanation why and to what extend the scheme meets the property.

Scheme	Practicality					Security		
	Accuracy of Time Frame	Multi-User Setting	No Resource Requirements	Practicality	Transparency	No "Trust" in Third Party	Parallelization Resistance	Verifiability
Time-Lock Puzzles	◐	○	○	●	○	●	●	○
Verifiable Delay Functions	◐	○	○	●	○	●	●	●
Blockchain-Based TLE	●	●	●	○	●	◐	●	●

● = provides property; ◐ = partially provides property; ○ = does not provide property

TABLE I. Comparison Overview

7.2 TLP

7.2.1 Practicality

Accuracy of Time Frame

The solution "time" of TLPs is only approximately controllable when generating the puzzle. Rivest et al. [2] choose the difficulty parameter according to Moore's law. Industry experts have questioned Moore's law for many years and still have not reached consensus on exactly when it will cease to apply [77, 78]. Semiconductor improvement has slowed down industry-wide since around 2010 below the pace predicted by Moore's law. However, in 2018 leading manufacturers have claimed that current mass production advancements match the pace predicted by Moore's law [79]. Still, it is possible to give relatively precise estimates for shorter time frames, i.e. where the hardware used for computing the solution is constant.

The difficulty of achieving an accurate timing is especially challenging for long durations, as seen with the LCS35 time capsule in Section 4.2.4. Moore's law is no exact science as the improvement of hardware is hard to predict. The LCS35 time capsule took only two months to solve for the team that started 20 years later. And by doing so they were still 15 years ahead of the original 35-year deadline. Therefore, we argue that the encryption time frame of a TLP is not very accurate over very long periods and conclude that *accuracy of time frame* is only partially provided.

Ease to Extend to Multi-User Setting

There are two possibilities to target multiple receivers at once with the same TLP-encrypted message:

- One can send the same TLP to multiple recipients, who then compute the solution to the TLP individually. Decryption synchronicity between the receivers is challenging, as it would require for all the receivers to use exactly the same hardware and to start decrypting at exactly the same time. Even then an identical progress is not guaranteed, and decryption times among users are likely to differ.
- The TLP is sent to a single party which then computes the result. Once the message is decrypted, the computing party then shares the message among the intended receivers. Thus, there is only little delay between the availability of the message for the receivers.

While it is theoretically possible to expand TLPs to a multi-user setting with the first variant, a synchronized decryption is only possible under a specific setup and highly unlikely. The alternative includes using one of the receivers as a trusted third party, which is exactly what TLPs are designed to avoid. Therefore, we argue that TLPs do not provide this property under realistic circumstances.

No Resource Requirements

The property *no resource requirements* directly opposes the core functionality of time-lock puzzles. TLPs are designed to take a certain number of computational steps to calculate the result. While generating the puzzle is less computationally expensive than solving it, it still requires a considerable amount of resources. The difficulty of a puzzle is set in regard to the expected change in computing power until it's decryption. Again, recalling LCS35 (see Section 4.2.4), the time capsule was designed to take 35 years of continuous computation from the day of creation. The improvement in computing power makes the calculation less expensive the more time has passed, but there still is a resource requirement. This property is therefore not provided for TLPs.

Practicality of Building Blocks

TLPs do not consist of exotic cryptographic primitives or building blocks. Without the factorization of N, decrypting a TLP-encrypted message solely requires a solver to perform T sequential squarings, as shown within Equation 4.10. Generating the puzzle is possible with a shortcut by knowing $\phi(N)$, which reduces the complexity of the calculation even more. The *practicality of building blocks* is therefore provided as long as standard cryptographic assumptions (such as the hardness of RSA) hold.

Transparency of Decryption Progress

There is no possibility for an observer of knowing the actual decryption progress of an unsolved TLP, unless intermediate results are voluntarily shared by the decryptor. This is also the case even if the puzzle itself is made public. When computing the solution of a TLP only the calculating party has access to the actual advancement of the decryption calculations. Even if the computing party decides to share its progress it is not possible to verify that it is correct. It is also possible that an error in computation happened and there is no actual (correct) progress.

Other parties, including the puzzle creator, can only estimate the progress based on the timing approximation used by the puzzle creators. For TLPs with longer time frames, i.e., over multiple years, the actual improvement of hardware can also be used to estimate the possible progress made. This is only guesswork and TLPs thus do not provide any *transparency of the decryption progress*.

7.2.2 Security

No "Trust" in Third Party

Time-lock puzzles by Rivest, Shamir and Wagner [2] are designed to be independent of any third party. The process of creating and solving a TLP exclusively involves a sender (the puzzle creator) and a receiver (the puzzle solver). The puzzle can be sent directly to the receiver without the need of any third parties. Therefore, TLPs provide the first property of *no "trust" in third parties*.

Parallelization Resistance

TLPs are built to be inherently sequential. An honest solver has access to the TLP which includes the variables N, x, T, C_K and C_M. However, the solver does not know any of the variables p, q or $\phi(N)$ which would allow the use of the shortcut to compute the result using Equation 4.7. This forces the solver to perform the computation in sequential steps and consequently makes TLPs highly resistant to parallelization [3]. The computation of the puzzle requires T sequential squarings. Rivest et al. [2] state that only within each squaring some parallelization is possible, but not with the computation as a whole. Hence, the property of *parallelization resistance* is provided.

Public Verifiability

The construction introduced by Rivest et al. [2] solely allows the creator of the puzzle to verify the correctness of the puzzle solution, the reason being that the verifier requires the secret state to produce the puzzle in order to verify the result [80]. This is possible by using the original plaintext message M and the secret encryption key K which are not available to the solving party. Furthermore, even for the sender the puzzle creation and therefore the puzzle verification takes $\log_2 T$ steps. Especially with large T, i.e. for long encryption time spans, and large values of N, this also requires a considerable number of computational resources. *Public verifiability* is therefore not provided.

7.2.3 Limitations

Rivest et al. [2] created the first construction that allows to securely send a message "into the future". TLPs use a straightforward principle and provide an ease of use. They achieve this without having to put trust into a third party and while accomplishing a resistance to parallel solvers. The implementation provides a high accuracy of the encryption duration for shorter time frames. Additionally, TLPs are used as starting point for many TLE applications that followed.

However, there are several disadvantages due to the simplicity of the construction. LCS35 perfectly illustrates the missing timing accuracy over longer periods. Furthermore, TLPs also are not universally verifiable nor do non-computing parties know the progress of an active decryption process. This computation puts a significant computational overhead on the receiver and a synchronized decryption of a single message intended for multiple receivers is realistically not possible without the use of a trusted third party. Last but not least, TLPs are vulnerable to advances in factoring techniques and quantum attacks [44].

One has to keep in mind with this evaluation that TLPs were introduced in 1996. They form the fundamental idea for a whole category of encryption schemes. Many of the shortcomings are solved in advanced implementations which are based on TLPs. Variations and extensions of TLPs allow universal verifiability [81, 82]. There are more recent constructions involving random oracles that are somewhat quantum resistant [83, 84]. Additionally, an improvement for long time frames is made when Moore's law could be replaced once it is possible to determine the improvement of hardware more precisely.

7.3 VDF

In the following we evaluate Pietrzaks VDF candidate [17] which we described in Definition 13 as TLP-VDF. This evaluation will have some similarities to the one of TLPs, as TLP-VDFs use TLPs in the core functionality. The construction therefore shares the main characteristics of TLPs.

7.3.1 Practicality

Accuracy of Time Frame

As the computation performed by the prover is in essence a TLP, similar restrictions as for TLPs apply. TLP-VDFs perform with high timing accuracy over short time frames where the hardware is constant and therefore its capabilities are well documented. Implementations with long encryption durations over several years do not exist yet because the cryptographic primitive of VDF was only introduced in 2018. However, one can argue that the core problem of TLPs remains the same, as the progression of computing power remains hard to predict. Whether one uses Moore's law or not to predict hardware advancement is irrelevant.

Furthermore, the prover must produce a corresponding proof π in addition to the solution $y = x^{2^T}$ of the TLP. This, however, does not significantly impact the duration of the procedure. TLP-VDF.Eval is said to run in $\mathcal{O}(1 + \frac{2}{\sqrt{T}}T)$ and Pietrzak [17] demonstrated that the proof is calculated $\approx 2^{13}$ times faster than the solution itself. We therefore argue that the proof can be disregarded when evaluating the accuracy of the time frame. Hence, we arrive at the same conclusion as with TLPs and claim that *accuracy of time frame* is only partially provided for TLP-VDFs.

Ease to Extend to Multi-User Setting

A VDF by itself is no encryption scheme, but it can be used to construct one. As TLP-VDFs build on TLPs, they share the same core characteristics. The challenge of a synchronized decryption of a single message for multiple receivers without a trusted third party prevails for TLP-VDFs. We thus consider the same result for this property as for TLPs, that is that *ease to extend to multi-user setting* is not provided.

No Resource Requirements

Pietrzak's VDF candidate [17] requires the prover to solve the puzzle $y = x^{2^T}$ in T sequential steps and also to produce the corresponding proof π. Only parties that calculate the solution are able to decrypt the ciphertext and obtain the message. This puts a significant computational overhead on the prover. It is intended to continuously compute this solution, which is especially demanding for messages which are to be encrypted for long durations, i.e. multiple years. Thus, TLP-VDFs do not provide *no resource requirements*.

Practicality of Building Blocks

Even though TLP-VDFs extend TLP with additional requirements and functionality, the core principle and hence the same considerations on practicality remain the same. The four algorithms TLP-VDF.Setup, TLP-VDF.Gen, TLP-VDF.Eval and TLP-VDF.Verify are rather straightforward and do not contain special building blocks. Pietrzak [17] shows that performing the computation over $(QR_N^+, \circ)$ instead of $(\mathbb{Z}_N^*, \cdot)$ does not impact the scheme negatively and is feasible. We thus conclude that *practicality of building blocks* is provided.

Transparency of Decryption Progress

TLP-VDFs do not provide more insight in the actual decryption progress of the prover than TLPs. The prover is required to voluntary share intermediate results for others to see the advancement of decryption. VDFs from TLPs allow the verification of intermediate results, if the prover also shares the preliminary proofs. This is in contrast to pure TLPs, where intermediate results do not give any indication on the correctness of the computation so far. While this is related to the transparency, it does not impact the evaluation of this property as we discuss that fact in *public verifiability*. Hence, we declare *transparency of decryption progress* as not provided, for the same reasons as for TLPs.

7.3.2 Security

No "Trust" in Third Party

Again, the evaluation of this property closely follows the one for TLP. The initial setup of the RSA parameters requires trust, but the puzzle generator is the sender itself and no external party is involved. The prover who solves the puzzle and produces the proof is at the same time the receiver of the encrypted message. No middleman is required for the process. In Section 5.4, we mention how a trustless setup can be achieved. This, however, bears some disadvantages in terms of practicality. Since we evaluate the main construction of TLP-VDFs, we conclude that TLP-VDFs do not put any trust in third parties.

Parallelization Resistance

As specified in Definition 9, VDFs are required to provide the properties *sequentiality* and ϵ*-evaluation time* [10]. *Sequentiality* states that it is not possible for a parallel algorithm using at most $\mathsf{polylog}(\lambda)$ to compute the solution of the puzzle in time less than T. Additionally, ϵ*-evaluation time* guarantees that TLP-VDF.Eval runs in time at most $(1 + \epsilon)T$. TLP-VDFs provide both these properties. As a consequence, they are resistant to parallel solvers by design.

Public Verifiability

VDFs in general excel when it comes to verifiability, as they are built to be efficiently and publicly verifiable. TLP-VDF.Eval produces both the solution to the puzzle y and the corresponding proof π that the puzzle was computed correctly. This allows anyone to run TLP-VDF.Verify, which is an algorithm that determines weather the obtained solution is in fact correct.

Additionally, it is possible to verify intermediate results of the TLP-VDF, as the proof $\pi = \{\mu_i\}_{i\in[t]}$ is generated in iterations [85]. For example, to verify the preliminary result after z iterations one can extract all obtained $\mu_1, ..., \mu_z$ so far and use them together with y_z as inputs for the verification algorithm. The verification process can be executed by anyone, as the additional inputs for TLP-VDF.Verify besides y and π are public parameters. TLP-VDFs provide *public verifiability*.

7.3.3 Limitations

TLP-VDFs use TLPs as basis but provide more of the desirable properties of TLE. They provide the possibility of encrypting a message over a chosen duration and this duration is especially accurate over short periods. Furthermore, VDFs are designed to be resistant to parallel solvers. They omit the use of a trusted third party to accomplish said properties. The main advantage over TLPs is the possibility of an efficient public verifiability, even for intermediate results. Because of the advantages, VDFs are also used as building blocks for other constructions, for example to construct resource efficient blockchains [13].

Nevertheless, Pietrzak's VDF variant [17] based on TLPs shares some weaknesses with the puzzle itself. The difficulty of providing an exact estimation for the timing parameter T prevails for long durations and there is the burden of the overhead on the prover. Parties other than the prover itself can also only estimate the decryption progress, unless the computing party decides to share intermediate results with the public. Another bottleneck is the reliance on structured algebraic assumptions (e.g. groups of unknown order), which we briefly discussed in Section 5.4 [46]. Advances in factoring techniques and quantum attacks would also break the underlying TLP [44].

The aspect of public verifiability for the sequentially computed steps is what enables VDFs in general to be used as building blocks. Wesolowski [6] independently also introduced a VDF candidate which uses the principles of repeated squaring in groups of unknown order and therefore TLPs. This implementation is currently being used by the Chia Network to generate proofs for their blockchain [86]. Ethereum 2.0 also plans to use VDFs coupled with proof-of-stake to obtain consensus [16].

7.4 Blockchain-Based TLE

The TLE scheme by Liu et al. [18] uses the blockchain growth of cryptocurrencies, e.g. Bitcoin, for the measuring of elapsed time. Since the Bitcoin blockchain is used to instantiate a CRC and therefore is a core part of the construction, large amounts of the following evaluation are influenced by the workings of the cryptocurrency itself. Note that the following discussion mainly assumes the existence of a working cryptocurrency with a PoW-based blockchain and assesses the properties based on the implementation with Bitcoin as a CRC. Apart from the first criterion *no "trust" in third party*, general restrictions are separately discussed later.

7.4.1 Practicality

Accuracy of Time Frame

The accuracy of the construction relies on the accuracy of the Bitcoin network. The Bitcoin protocol adjusts the difficulty of finding a new block regularly by modifying the size of the target, to match the goal of a new block (on average) every 10 minutes. Note, however, that this only happens gradually, that is every 2016 blocks, so roughly every 14 days [87]. While there is a possibility of deviations in a short time horizon, on a large scale the average of 10 minutes per block are relatively accurate due to the frequent adjustments. Considerable short-term changes in available computing power are needed to significantly alter this average [43]. Consequently, it is feasible to rely on Bitcoin as real-world timing instance and the construction provides the property *accuracy of time frame*.

Ease to Extend to Multi-User Setting

This implementation uses the growth of the Bitcoin blockchain as tool for decryption. As discussed in *public verifiability*, the public nature of the blockchain enables everyone to use this blockchain to verify a found solution. This implementation is not restricted to a single receiver, as it does not require the recipient itself to perform a computation. The Bitcoin miners perform this computation, regardless of weather there are multiple targets for the message or not. Hence, multiple intended receivers can synchronously receive and verify the same message without any modifications to the implementation as well. Therefore, the construction is *expandable to a multi-user setting*.

No Resource Requirements

The authors designed the implementation with the core idea of *no resource requirements*. Neither encryption nor decryption require an involved party to have access to considerable computational power, as described in 6.2.5. The actual computing is done by the Bitcoin miners, which are used to instantiate a CRC. While encryptor and decryptor could be miners themselves, it is not required for the implementation to function. An adversary could also try to outperform the entirety of the Bitcoin network by itself. This would in turn require such considerable amount of computing power, that this is highly unlikely. Regardless of their involvement in the mining process, the blockchain will be extended and thus at some point the length required for decryption will be reached. The construction provides *no resource requirements*.

Practicality of Building Blocks

Except for the blockchain component, the used building blocks are rather abstract. While general witness encryption is already impractical, the proposed variant is even more exotic and therefore not practical [71, 72, 30]. Current encryption and decryption times using witness encryption are estimated to take around 2^{100} seconds and, hence, described as astronomical [88]. To mitigate this issue, Liu et al. [18] propose a second variation using SNARKs in their paper, as briefly presented in 6.3. The authors succeed in reducing the multilinearity level of WE based on multilinear maps, but the overall impracticality is not solved by that [88, 43]. The authors themselves acknowledge that the paper is more of a proof of concept than concrete proposal. In summary, the proposed encryption scheme is not practical at all and does not provide *practicality of building blocks*.

Transparency of Decryption Progress

Liu et al. [18] use the growth of the blockchain to measure the elapsed time. The Bitcoin blockchain is a public ledger: each time a new block is appended to the blockchain, said block is broadcast to the entire network and made available to all nodes [89]. This event functions as "tick" by the computational reference clock, or in other words, it indicates that 10 minutes passed. Since it is known when the time-lock expires, this publicity of the blockchain makes the decryption progress transparent. Hence, everyone has insight to the current decryption progress of a time-lock encrypted message and the property *transparency of decryption progress* is provided.

7.4.2 Security

No "Trust" in Third Party

The authors stress that their approach does not put any trust in a third party. While it is true that the construction does not require to trust, i.e. the miners of a cryptocurrency directly, it still relies on Bitcoin or any PoW-powered cryptocurrency. The authors also state that the generic construction can also be adapted to use any large-scale, public, iterative computation. The introduced CRCs are simply designed to match the workings of a blockchain, but the underlying principle remains feasible for other computations. As they use some form of a third party for the measurement of time, the property *no "trust" in third parties* is rated as partially provided. Without such a computing third party, the scheme does not work.

Parallelization Resistance

The proposed construction by Liu et al. [18] leverages the entirety or all Bitcoin miners to perform the required computation. Bitcoin uses PoW to guarantee the authenticity of a new block before it is appended to the blockchain. PoW is designed to enforce that a certain amount of work was carried out. The process of mining a new block requires miners to compute a hash which is equal or lower than the current target hash. Each time this is not the case a new computation is required. The difficulty of this computation is determined by the difficulty parameter of the Bitcoin protocol, which in turn is adjusted regularly so that on average every 10 minutes a new block is found.

It is possible for a miner to mine on more than just one instance at a time. However, this does not impact the proposed construction, because the difficulty parameter is adjusted according to the currently contributing computational resources. Consequently, even if a miner uses multiple devices to parallelise the mining process, a new block is still found roughly every 10 minutes. As a result, while parallelization is possible within the Bitcoin network, it does not affect the implementation and the property *parallelization resistance* is provided.

Public Verifiability

The Bitcoin blockchain is a public ledger, where newly appended blocks are available to everyone. The authors use CRCs combined with witness encryption to verify the elapsed time. Once the intended decryption time x for a time-lock encrypted message is reached, there should exist a witness, more specifically a valid chain of blocks $(B_1, ..., B_\tau, B_\tau + 1, ..., B_x)$, in the blockchain. The message can only be reconstructed after time x is reached with the help of said witness. As the blockchain is public the validity of the witness can be verified by everyone, although a local copy of the whole chain is required. The implementation by Liu et al. [18] provides *public verifiability*.

7.4.3 Limitations

Liu et al. [18] solve many limitations that the early implementations of TLE such as TLPs presented. The construction is very secure, albeit strictly theoretical. An adversary has to exploit the Bitcoin network altogether to break the scheme. By using a cryptocurrency such as Bitcoin, the regularly adjusted difficulty of the protocol enables an accurate estimation of the decryption timing. The publicity of Bitcoin also enables the scheme to be transparent and publicly verifiable. Another advantage of using a computing third party is that the sender and the receiver are not burdened with a computational overhead.

However, the scheme has some drawbacks. The major one is the theoretical nature of the approach. The construction is more of a proof of concept than a model ready for implementation. The impracticality of the building blocks, particularly witness encryption, is significant. Also, some form of trust is needed. While the scheme does not need to trust a single third party, it relies on the secure functionality of the computing instance altogether, i.e. that Bitcoin stays stable and that the mining of new blocks continues.

On the other hand, these limitations can be viewed as a chance. For example, advances of the practicality of WE would widely improve the scheme. Further, the authors not only introduce a blueprint for any blockchain related cryptocurrency, but the concept can be adapted for any large-scale, iterative, public computation as CRC. This leaves the scheme in its current form with potential for future constructions of TLE.

Chapter 8

Conclusion

In this thesis we explored time-lock encryption based on three selected constructions. We studied these three implementations that allow to send a message "into the future". This thesis identified criteria for formulating desirable properties for TLE schemes and evaluated the investigated constructions.

Rivest et al. [2] introduced the notion of time-lock puzzles back in 1996 and thereby pioneered the field of TLE. The key idea consists of creating a computation that is hard to solve, but easy to create. Computing the solution requires a certain number of sequential steps so that parallelism should not be possible. This is achieved by repeatedly squaring in an RSA group. The importance of TLP for the field is considerable, even though it seemingly performed the worst in our evaluation. The core assumption that exponentiation modulo an RSA integer requires a sequential computation is still widely used to this day. The functionality of TLPs can be extended significantly by adding universal verifiability and an exact way of predicting future hardware advancements.

Over 20 years after TLPs, Boneh et al. [10] formalized the concept of VDFs. This primitive is related to TLPs in that they also involve computing an inherently sequential function. However, in the VDF setting a prover can convince a verifier that a certain amount of computation was performed [17]. They enable the efficient universal verification of computed outputs produced on any corresponding input. Pietrzak [17] then proposed a VDF candidate that uses a TLP in its core. VDFs scored slightly better in our evaluation than TLPs. They still bear some disadvantages of TLPs, but excel particularly with universal and efficient verifiability. This property enables VDFs to be used as powerful building block for novel blockchain approaches without the use of proof-of-work [13].

In the same year, Liu et al. [18] constructed a TLE scheme that is fundamentally different. They introduce a new cryptographic primitive called "computational reference clock" to use the continuous growth of cryptocurrency blockchains as a mean to measure real-world time. Our evaluation suggests that this scheme best meets the desired properties out of the three. The concrete instantiation of Bitcoin as computational reference clock is not without drawbacks, as the used cryptographic primitive witness encryption remains too impractical. Nevertheless, this execution laid the foundation to use any blockchain of a cryptocurrency to implement TLE. It would be interesting to see a concrete instantiation using a blockchain based on VDFs or an adapted variant that does not require witness encryption at all.

Note, that this evaluation does not include weighting of the various criteria. For example, the third scheme by Liu et al. [18] provides the most properties, but is heavily impractical. In the current state, the construction might even be useless in practice. But nevertheless strengths and limitations are identified by this evaluation and desirable properties provide a starting point for possible improvements.

All investigated implementations leave room for improvement. Future work might include identifying more desirable properties and formulating new criteria and include weighting. There exist many more interesting constructions to be evaluated. Such an evaluation may be useful for future constructions of TLE.

Bibliography

[1] J. Katz and Y. Lindell, *Introduction to Modern Cryptography, Second Edition*. CRC Press, 2014.

[2] R. L. Rivest, A. Shamir, and D. A. Wagner, "Time-lock puzzles and timed-release crypto," tech. rep., Massachusetts Institute of Technology, USA, 1996.

[3] M. Mahmoody, T. Moran, and S. P. Vadhan, "Time-lock puzzles in the random oracle model," in *CRYPTO*, vol. 6841 of *Lecture Notes in Computer Science*, pp. 39–50, Springer, 2011.

[4] K. G. Paterson and E. A. Quaglia, "Time-specific encryption," *IACR Cryptol. ePrint Arch.*, vol. 2010, p. 347, 2010.

[5] M. O. Rabin and C. Thorpe, "Time-lapse cryptography," tech. rep., Harvard University, 2006.

[6] B. Wesolowski, "Efficient verifiable delay functions," *IACR Cryptol. ePrint Arch.*, vol. 2018, p. 623, 2018.

[7] J. Cathalo, B. Libert, and J. Quisquater, "Efficient and non-interactive timed-release encryption," in *ICICS*, vol. 3783 of *Lecture Notes in Computer Science*, pp. 291–303, Springer, 2005.

[8] Z. Brakerski, N. Döttling, S. Garg, and G. Malavolta, "Leveraging linear decryption: Rate-1 fully-homomorphic encryption and time-lock puzzles," *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 720, 2019.

[9] D. Boneh and M. Naor, "Timed commitments," in *CRYPTO*, vol. 1880 of *Lecture Notes in Computer Science*, pp. 236–254, Springer, 2000.

[10] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch, "Verifiable delay functions," in *CRYPTO (1)*, vol. 10991 of *Lecture Notes in Computer Science*, pp. 757–788, Springer, 2018.

[11] V. Attias, L. Vigneri, and V. Dimitrov, "Preventing denial of service attacks in iot networks through verifiable delay functions," in *GLOBECOM*, pp. 1–6, IEEE, 2020.

[12] Gwern.net, "Time-lock encryption." `https://www.gwern.net/Self-decrypting-files`, 2015". [Online; accessed 23.02.2021].

[13] C. Baum, B. David, R. Dowsley, J. B. Nielsen, and S. Oechsner, "CRAFT: composable randomness and almost fairness from time," *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 784, 2020.

[14] Y. Song and T. Aste, "The cost of bitcoin mining has never really increased," *Frontiers Blockchain*, vol. 3, p. 565497, 2020.

[15] B. Cohen and K. Pietrzak, "The chia network blockchain." `https://www.chia.net/assets/ChiaGreenPaper.pdf`, 2019". [Online; accessed 16.07.2021].

[16] E. Foundation, "Ethereum 2.0 specifications." `https://github.com/ethereum/eth2.0-specs`, 2021". [Online; accessed 28.07.2021].

[17] K. Pietrzak, "Simple verifiable delay functions," in *ITCS*, vol. 124 of *LIPIcs*, pp. 60:1–60:15, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[18] J. Liu, T. Jager, S. A. Kakvi, and B. Warinschi, "How to build time-lock encryption," *Des. Codes Cryptogr.*, vol. 86, no. 11, pp. 2549–2586, 2018.

[19] P. Gauravaram and L. R. Knudsen, "Cryptographic hash functions," in *Encyclopedia of Information Assurance*, Taylor & Francis, 2011.

[20] A. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1996.

[21] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2009.

[22] S. Garg, C. Gentry, A. Sahai, and B. Waters, "Witness encryption and its applications," *IACR Cryptol. ePrint Arch.*, vol. 2013, p. 258, 2013.

[23] P. Chvojka, T. Jager, and S. A. Kakvi, "Offline witness encryption with semi-adaptive security," in *ACNS (1)*, vol. 12146 of *Lecture Notes in Computer Science*, pp. 231–250, Springer, 2020.

[24] P. Chvojka, "Talk on the paper ´´offline witness encryption with semi-adaptive security" by peter chvojka, tibor jager and saqib a. kakvi." `https://www.youtube.com/watch?v=c3oPSfFScqo`, 2020". [Online; accessed 15.05.2021].

[25] S. Goldwasser, Y. T. Kalai, R. A. Popa, V. Vaikuntanathan, and N. Zeldovich, "How to run turing machines on encrypted data," in *CRYPTO (2)*, vol. 8043 of *Lecture Notes in Computer Science*, pp. 536–553, Springer, 2013.

[26] E. Boyle, K. Chung, and R. Pass, "On extractability obfuscation," *IACR Cryptol. ePrint Arch.*, vol. 2013, p. 650, 2013.

[27] M. Bellare and V. T. Hoang, "Adaptive witness encryption and asymmetric password-based cryptography," in *Public Key Cryptography*, vol. 9020 of *Lecture Notes in Computer Science*, pp. 308–331, Springer, 2015.

[28] C. Gentry, A. B. Lewko, and B. Waters, "Witness encryption from instance independent assumptions," *IACR Cryptol. ePrint Arch.*, vol. 2014, p. 273, 2014.

[29] D. Boneh and A. Silverberg, "Applications of multilinear forms to cryptography," *IACR Cryptol. ePrint Arch.*, vol. 2002, p. 80, 2002.

[30] S. Garg, C. Gentry, and S. Halevi, "Candidate multilinear maps from ideal lattices and applications," *IACR Cryptol. ePrint Arch.*, vol. 2012, p. 610, 2012.

[31] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. P. Vadhan, and K. Yang, "On the (im)possibility of obfuscating programs," in *CRYPTO*, vol. 2139 of *Lecture Notes in Computer Science*, pp. 1–18, Springer, 2001.

[32] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. P. Vadhan, and K. Yang, "On the (im)possibility of obfuscating programs," *J. ACM*, vol. 59, no. 2, pp. 6:1–6:48, 2012.

[33] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters, "Candidate indistinguishability obfuscation and functional encryption for all circuits," in *FOCS*, pp. 40–49, IEEE Computer Society, 2013.

[34] C. Dwork and M. Naor, "Pricing via processing or combatting junk mail," in *CRYPTO*, vol. 740 of *Lecture Notes in Computer Science*, pp. 139–147, Springer, 1992.

[35] M. Ball, A. Rosen, M. Sabin, and P. N. Vasudevan, "Proofs of useful work," *IACR Cryptol. ePrint Arch.*, vol. 2017, p. 203, 2017.

[36] A. Back, "Hashcash." `http://www.cypherspace.org/hashcash/`, 1997". [Online; accessed 04.05.2021].

[37] N. Lasla, L. Alsahan, M. Abdallah, and M. F. Younis, "Green-pow: An energy-efficient blockchain proof-of-work consensus algorithm," *CoRR*, vol. abs/2007.04086, 2020.

[38] E. Sin and L. Wang, "Bitcoin price prediction using ensembles of neural networks," in *ICNC-FSKD*, pp. 666–671, IEEE, 2017.

[39] "Bitcoin vocabulary: Block." `https://bitcoin.org/en/vocabulary#block`, 2021". [Online; accessed 03.04.2021].

[40] "Bitcoin vocabulary: Blockchain." `https://bitcoin.org/en/vocabulary#block-chain`, 2021". [Online; accessed 03.05.2021].

[41] E. Zaghloul, T. Li, M. Mutka, and J. Ren, "Bitcoin and blockchain: Security and privacy," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 10288–10313, 2020.

[42] M. Maso, "Study and implementation of blockchain compression," 2020.

[43] I. Puddu, A. Dmitrienko, and S. Capkun, "μchain: How to forget without hard forks," *IACR Cryptol. ePrint Arch.*, vol. 2017, p. 106, 2017.

[44] G. Malavolta and S. A. K. Thyagarajan, "Homomorphic time-lock puzzles and applications," in *CRYPTO (1)*, vol. 11692 of *Lecture Notes in Computer Science*, pp. 620–649, Springer, 2019.

[45] N. Bitansky, S. Garg, H. Lin, R. Pass, and S. Telang, "Succinct randomized encodings and their applications," in *STOC*, pp. 439–448, ACM, 2015.

[46] M. Mahmoody, C. Smith, and D. J. Wu, "Can verifiable delay functions be based on random oracles?," in *ICALP*, vol. 168 of *LIPIcs*, pp. 83:1–83:17, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[47] L. Marekova, "E-cclesia: Self-tallying e-voting protocol in the uc framework," *University of Edinburgh*, 2018.

[48] G. Malavolta, *Cryptographic Clocks and Applications*. PhD thesis, University of Erlangen-Nuremberg, Germany, 2019.

[49] T. C. May, "Timed-release crypto." `http://cypherpunks.venona.com/date/1993/02/msg00129.html`, 1993". [Online; accessed 20.02.2021].

[50] W. Lai, C. Hsueh, and J. Wu, "A fully decentralized time-lock encryption system on blockchain," in *Blockchain*, pp. 302–307, IEEE, 2019.

[51] D. A. Grier, "Forgetting moore's law [body of knowledge]," *Computer*, vol. 54, no. 6, pp. 46–48, 2021.

[52] C. Edwards, "Moore's law: what comes next?," *Commun. ACM*, vol. 64, no. 2, pp. 12–14, 2021.

[53] C. Kamath, "Presentation about time-lock puzzles, think and drink ist austria." `https://pub.ist.ac.at/~ckamath/Resources/Documents/TLP.pdf`, 2018. [Online; accessed 01.07.2021].

[54] A. R. Choudhuri, P. Hubácek, C. Kamath, K. Pietrzak, A. Rosen, and G. N. Rothblum, "Ppad-hardness via iterated squaring modulo a composite," *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 667, 2019.

[55] N. Bitansky, S. Goldwasser, A. Jain, O. Paneth, V. Vaikuntanathan, and B. Waters, "Time-lock puzzles from randomized encodings," in *ITCS*, pp. 345–356, ACM, 2016.

[56] R. L. Rivest, "The RC5 encryption algorithm," in *FSE*, vol. 1008 of *Lecture Notes in Computer Science*, pp. 86–96, Springer, 1994.

[57] V. Rijmen and J. Daemen, "Advanced encryption standard," *Proceedings of Federal Information Processing Standards Publications, National Institute of Standards and Technology*, pp. 19–22, 2001.

[58] R. L. Rivest, "Description of the lcs35 time capsule crypto-puzzle." `https://people.csail.mit.edu/rivest/lcs35-puzzle-description.txt`, 1999". [Online; accessed 01.06.2021].

[59] "Lcs celebrates 35 years." `http://web.archive.org/web/19991006230454/http://www.lcs.mit.edu/news/events/lcs35announce`, 1999". [Online; accessed 01.06.2021].

[60] "Programmers solve mit's 20-year-old cryptographic puzzle." `https://www.csail.mit.edu/news/programmers-solve-mits-20-year-old-cryptographic-puzzle`, 2019". [Online; accessed 01.06.2021].

[61] D. Boneh, B. Bünz, and B. Fisch, "A survey of two verifiable delay functions," *IACR Cryptol. ePrint Arch.*, vol. 2018, p. 712, 2018.

[62] A. Stiglic, *Safe Prime*, pp. 541–541. Boston, MA: Springer US, 2005.

[63] R. Fischlin and C. Schnorr, "Stronger security proofs for RSA and rabin bits," *J. Cryptol.*, vol. 13, no. 2, pp. 221–244, 2000.

[64] D. Hofheinz and E. Kiltz, "The group of signed quadratic residues and applications," in *CRYPTO*, vol. 5677 of *Lecture Notes in Computer Science*, pp. 637–653, Springer, 2009.

[65] Z. Yang, B. Qin, Q. Wu, W. Shi, and B. Liang, "Experimental comparisons of verifiable delay functions," in *ICICS*, vol. 12282 of *Lecture Notes in Computer Science*, pp. 510–527, Springer, 2020.

[66] M. Mahmoody, T. Moran, and S. P. Vadhan, "Publicly verifiable proofs of sequential work," in *ITCS*, pp. 373–388, ACM, 2013.

[67] U. Vollmer, "Asymptotically fast discrete logarithms in quadratic number fields," in *ANTS*, vol. 1838 of *Lecture Notes in Computer Science*, pp. 581–594, Springer, 2000.

[68] W. Smith and M. Arapinis, "Towards timelock encryption," *University of Edinburgh*, 2019.

[69] J. Bartusek, Y. Ishai, A. Jain, F. Ma, A. Sahai, and M. Zhandry, "Affine determinant programs: A framework for obfuscation and witness encryption," in *ITCS*, vol. 151 of *LIPIcs*, pp. 82:1–82:39, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[70] "Bitcoin hashrate." `https://charts.bitcoin.com/btc/chart/hash-rate#5ma4`, 2021". [Online; accessed 08.04.2021].

[71] J. Coron, T. Lepoint, and M. Tibouchi, "New multilinear maps over the integers," *IACR Cryptol. ePrint Arch.*, vol. 2015, p. 162, 2015.

[72] A. Langlois, D. Stehlé, and R. Steinfeld, "Gghlite: More efficient multilinear maps from ideal lattices," in *EUROCRYPT*, vol. 8441 of *Lecture Notes in Computer Science*, pp. 239–256, Springer, 2014.

[73] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza, "Snarks for C: verifying program executions succinctly and in zero knowledge," in *CRYPTO (2)*, vol. 8043 of *Lecture Notes in Computer Science*, pp. 90–108, Springer, 2013.

[74] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Succinct non-interactive zero knowledge for a von neumann architecture," in *USENIX Security Symposium*, pp. 781–796, USENIX Association, 2014.

[75] N. Bitansky, A. Chiesa, Y. Ishai, R. Ostrovsky, and O. Paneth, "Succinct non-interactive arguments via linear interactive proofs," in *TCC*, vol. 7785 of *Lecture Notes in Computer Science*, pp. 315–333, Springer, 2013.

[76] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: nearly practical verifiable computation," *Commun. ACM*, vol. 59, no. 2, pp. 103–112, 2016.

[77] I. Tuomi, "The lives and death of moore's law," *First Monday*, vol. 7, no. 11, 2002.

[78] R. R. Schaller, "Moore's law: Past, present, and future," *IEEE Spectr.*, vol. 34, p. 52–59, June 1997.

[79] M. Gams and T. Kolenik, "Relations between electronics, artificial intelligence and information society through information society rules," *Electronics*, vol. 10, no. 4, 2021.

[80] A. Abadi and A. Kiayias, "Multi-instance publicly verifiable time-lock puzzle and its applications," 2021.

[81] N. Ephraim, C. Freitag, I. Komargodski, and R. Pass, "Non-malleable time-lock puzzles and applications," *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 779, 2020.

[82] A. Abadi and A. Kiayias, "Multi-instance publicly verifiable time-lock puzzle and its applications," *Financial Cryptography and Data Security*, 2021.

[83] D. Boneh, Ö. Dagdelen, M. Fischlin, A. Lehmann, C. Schaffner, and M. Zhandry, "Random oracles in a quantum world," in *ASIACRYPT*, vol. 7073 of *Lecture Notes in Computer Science*, pp. 41–69, Springer, 2011.

[84] G. Brassard, P. Høyer, K. Kalach, M. Kaplan, S. Laplante, and L. Salvail, "Merkle puzzles in a quantum world," in *CRYPTO*, vol. 6841 of *Lecture Notes in Computer Science*, pp. 391–410, Springer, 2011.

[85] L. Liu and M. Xu, "Analysis of the randomness generation for pos-based blockchains with verifiable delay functions," in *BlockSys*, vol. 1267 of *Communications in Computer and Information Science*, pp. 661–674, Springer, 2020.

[86] C. Network, "Chia network vdf specifications." `https://github.com/Chia-Network/chiavdf`, 2021". [Online; accessed 02.08.2021].

[87] S. Singh and N. Singh, "Blockchain: Future of financial and cyber security," in *2016 2nd international conference on contemporary computing and informatics (IC3I)*, pp. 463–467, IEEE, 2016.

[88] J. Liu, F. Garcia, and M. Ryan, "Time-release protocol from bitcoin and witness encryption for sat," *IACR Cryptology ePrint Archive*, vol. 2015, p. 482, 2015.

[89] E. Zaghloul, T. Li, M. Mutka, and J. Ren, "Bitcoin and blockchain: Security and privacy," *CoRR*, vol. abs/1904.11435, 2019.

Erklärung

Erklärung gemäss Art. 30 RSL Phil.-nat. 18

Ich erkläre hiermit, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen benutzt habe. Alle Stellen, die wörtlich oder sinngemäss aus Quellen entnommen wurden, habe ich als solche gekennzeichnet. Mir ist bekannt, dass andernfalls der Senat gemäss Artikel 36 Absatz 1 Buchstabe r des Gesetzes vom 5. September 1996 über die Universität zum Entzug des auf Grund dieser Arbeit verliehenen Titels berechtigt ist.

Für die Zwecke der Begutachtung und der Überprüfung der Einhaltung der Selbständigkeitserklärung bzw. der Reglemente betreffend Plagiate erteile ich der Universität Bern das Recht, die dazu erforderlichen Personendaten zu bearbeiten und Nutzungshandlungen vorzunehmen, insbesondere die schriftliche Arbeit zu vervielfältigen und dauerhaft in einer Datenbank zu speichern sowie diese zur Überprüfung von Arbeiten Dritter zu verwenden oder hierzu zur Verfügung zu stellen.

Bern 18.08.2021

Ort/Datum

Unterschrift