A COMPREHENSIVE STUDY OF TIME LOCK PUZZLES AND TIMED SIGNATURES IN CRYPTOGRAPHY

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF APPLIED MATHEMATICS
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

CEYLİN DOĞAN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
CRYPTOGRAPHY

JULY 2023

Approval of the thesis:

A COMPREHENSIVE STUDY OF TIME LOCK PUZZLES AND TIMED SIGNATURES IN CRYPTOGRAPHY

submitted by **CEYLİN DOĞAN** in partial fulfillment of the requirements for the degree of **Master of Science in Cryptography Department, Middle East Technical University** by,

Prof. Dr. A. Sevtap Selçuk Kestel
Dean, Graduate School of **Applied Mathematics**

Assoc. Prof. Dr. Oğuz Yayla
Head of Department, **Cryptography**

Assoc. Prof. Dr. Oğuz Yayla
Supervisor, **Cryptography, METU**

Examining Committee Members:

Prof. Dr. Murat Cenk
Cryptography, METU

Assoc. Prof. Dr. Oğuz Yayla
Cryptography, METU

Assoc. Prof. Dr. Ahmet Sınak
Mathematics and Computer Science,
Necmettin Erbakan University

Assist. Prof. Dr. Ayşe Nurdan SARAN
Computer Engineering, Çankaya University

Assist. Prof. Dr. Talha Arıkan
Mathematics, Hacettepe University

Date:

ABSTRACT

A COMPREHENSIVE STUDY OF TIME LOCK PUZZLES AND TIMED SIGNATURES IN CRYPTOGRAPHY

Doğan, Ceylin
M.S., Department of Cryptography
Supervisor : Assoc. Prof. Dr. Oğuz Yayla

July 2023, 69 pages

Timed-release cryptography is an innovative approach to sending information that is designed to be received at a specific time in the future. The thesis focuses on the evolution of timed-release cryptography, which was initially proposed by May in 1993 [21] as a means of sending encrypted messages into the future. This concept led to the development of time-lock puzzles by Rivest, Shamir, and Wagner in 1996 [25], which enabled the generation of puzzles with hidden solutions that became visible after a specific time had elapsed. This thesis provides a comprehensive survey of the existing literature on timed cryptography, including time-lock puzzles, timed commitment schemes, and timed signature schemes, to provide a historical background of timed cryptography. In addition, this study analyzes the efficiency and security levels of various cryptographic techniques, identifies areas for future research and development, and highlights the potential applications of timed cryptography in real-life scenarios such as contract signing and payment channel protocols.

Keywords: timed-release cryptography, time-lock puzzles, timed commitments, timed signatures

ÖZ

KRİPTOGRAFİK ZAMAN KİLİTLİ BULMACALAR VE ZAMANLANMIŞ İMZALAR ÜZERİNE KAPSAMLI BİR ARAŞTIRMA

Doğan, Ceylin
Yüksek Lisans, Kriptografi Bölümü
Tez Yöneticisi : Doç. Dr. Oğuz Yayla

Temmuz 2023, 69 sayfa

Zamanlanmış kriptografi, bilgilerin gelecekte belirlenmiş bir zamanda güvenli bir şekilde erişilebilir olması için tasarlanmış yenilikçi bir yaklaşımdır. Bu tez ilk olarak 1993 yılında May [21] tarafından önerilen şifrelenmiş mesajları geleceğe gönderebilmemiz üzerine çalışan zamanlanmış kriptografinin evrimsel gelişimine odaklanır. Zamanlanmış kriptografi fikri, 1996 yılında Rivest, Shamir ve Wagner [25] tarafından kilitli bulmacaların geliştirilmesine öncülük etmiştir. Bu yöntem, belirlenmiş bir süre geçtikten sonra gizlenmiş olan çözümlerine ulaşabildiğimiz zaman kilitli bulmacaların temelini oluşturur. Bu tez, okuyucuya zamanlanmış şifreleme alanındaki mevcut literatür hakkında kapsamlı bir araştırma sunar. Aynı zamanda, zaman kilitli bulmacalarını, zamanlanmış taahhüt ve imza şemalarını inceleyerek, zamanlanmış kriptografi alanını tarihsel arka planı ile birlikte ortaya koyar. Tüm bunlara ek olarak, bu çalışma gelecekteki araştırma ve geliştirme alanlarını belirleyerek ve zamanlanmış kriptografinin sözleşme imzalama ve ödeme kanalı protokolleri gibi gerçek hayat senaryolarındaki potansiyel uygulamalarını vurgulayarak, çeşitli kriptografik tekniklerin verimlilik ve güvenlik seviyelerini analiz etmektedir.

Anahtar Kelimeler: zamanlanmış kriptografi, zaman kilitli bulmacalar, zamanlanmış imzalar

To my beloved father whose light I always carry in my heart

ACKNOWLEDGMENTS

I am grateful to my supervisor, Assoc. Prof. Dr. Oğuz Yayla, for his guidance during this thesis. Working with him has been a pleasure. I extend my appreciation to Prof. Dr. Murat Cenk for his continuous support throughout my master's program. His guidance has been immensely valuable to my academic journey. I would also like to thank all of my committee members for their contributions and encouragement.

Furthermore, I am deeply thankful to my beloved professor, Assoc. Prof. Dr. Samet Bağçe, whose presence has enlightened my life. I always carry his support and kindness in my heart.

I am also indebted to my family, particularly my dear mother Hatice Doğan, and my sisters Derin Doğan and Ecem Ömeroğlu, for their love and support. Their presence and encouragement have been a constant source of strength. Most especially, I am grateful to my sister Derin, she was "the one who remembers to turn on the light even in the darkest of times". Thank you so much sister for being my best friend all my life.

Special thanks to my lifetime friends Emre Karabıyık, Dilan Yıldırım, Efe Kurtoğlu, and Büşra İlayda Ses. I am deeply grateful for their endless support and the incredible efforts they have made on my behalf. Their love has had a profound impact on me. During my time in Ankara, I have been fortunate to encounter individuals who have made this city and my life beautiful. Thanks to Emre Güngör, Arca Özçoban, Burak Kaya and Piki for their valuable friendships in my journey. It is precious for me to make all of my friends proud. I would like to extend my thanks to my colleagues Beste Akdoğan, Esra Günsay, and H. Bartu Yünüak for their exceptional work and contributions. Beste has been the best buddy for all the three years. Her support and contributions are priceless.

For the last year, one person showed me special support. I know he did his best for me to make this thesis better. He lifted me up when I felt down, much like the day that we were 30 meters underwater, and I got breathless, he came up to me with his octopus and give me the confidence to stay calm and move on. Thank you, Mahmut Şahin for all our good memories.

Lastly, I want to thank my lovely cat, Miyu, who is the only one who stayed awake with me during the late hours of the nights.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

BBS	Blum-Blum-Shub
BLS	Boneh, Lynn and Shacham
DCR	Decisional Composite Residuosity
DDH	Decisional Diffie-Hellman
DLP	Discrete Logarithm Problem
DSA	Digital Signature Algorithm
ECDSA	Elliptic Curve Digital Signature Algorithm
IFP	Integer Factorization Problem
HTLP	Homomorphic Time-Lock Puzzle
LHTLP	Linearly Homomorphic Time-Lock Puzzle
MHTLP	Multiplicatively Homomorphic Time-lock Puzzle
PRAM	Parallel Random Access Machines
PPT	Probabilistic Polynomial Time
RSW	Rivest, Shamir and Wagner
TLP	Time-Lock Puzzle

CHAPTER 1

INTRODUCTION

The concept of timed-release cryptography was initially conceived as a means of "sending information to the future". The origin of this idea can be traced back to Timothy May's 1993 [21] proposition, in which he posed the brilliant question of why anyone would want to send encrypted messages to the future. May's proposal opened the door for researchers to explore the practical applications of timed-release cryptography and devise cryptographic techniques to achieve this goal. Consequently, significant efforts have been made to develop cryptographic methodologies that enable the transmission of encrypted messages into the future.

In 1996, Rivest, Shamir, and Wagner [25] introduced the concept of time-lock puzzles, which evolved from May's timed-release cryptography. Their protocol allowed the sender to generate a puzzle with a solution that remained hidden until a specific time elapsed. This notion was based on the assumption that exponentiation modulo an RSA integer is an inherently sequential computation. This work stood as a leading candidate for a long time until other constructions were proposed. Bitansky et al. proposed a new candidate for time-lock puzzles as an alternative to Rivest, Shamir, and Wagner's scheme in 2015 [3]. Their proposal utilizes randomized encodings and can be proven secure under the assumption that non-parallelizing languages exist. These languages require a depth of at least T circuits to decide, which is necessary for the existence of time-lock puzzles. The authors implemented their construction with various randomized encodings from existing literature, which yielded increasingly better efficiency based on stronger cryptographic assumptions. These assumptions ranged from one-way functions to indistinguishability obfuscation. The authors also constructed other types of puzzles, such as proofs of work, using randomized encodings and a suitable worst-case hardness assumption. Recently, Malavolta and Thyagarajan [20] proposed homomorphic time-lock puzzles, increasing the previous constructions' efficiency. They also presented new protocols for applications, such as e-voting and fair contract signing. These advances have opened up new avenues for research and practical implementations in timed-release cryptography.

Time-lock puzzles are considered the primary building blocks of timed-release cryptography, which have enabled further exploration of various cryptographic use cases, including timed commitments and timed signatures. The introduction of timed commitment schemes by Boneh and Naor [7] extended the standard notion of commitments, allowing the receiver to retrieve the committed value, even if the committer is coerced into revealing the information, albeit with significant effort. Garay and Jakobsson [15] subsequently explored the timed release of digital signatures, demonstrating their application for RSA, Schnorr, and DSA signatures based on Boneh and Naor's work on timed commitments. More recently, Thyagarajan et al. [30] presented a comprehensive study on verifiable timed signatures, utilizing homomorphic time-lock puzzles, demonstrating their application in real-life scenarios such as contract signing and payment channel protocols. Additionally, the study presented the verifiable linkable timed signatures [29] as a solution to the weaknesses of Monero, the largest privacy-preserving cryptocurrency, without the need for system-wide changes such as a hard fork in its blockchain.

The central focus of this thesis is to conduct a comprehensive survey of the existing literature on timed cryptography. Our objective is to examine, summarize and compare various cryptographic techniques, which are time-lock puzzles, timed commitment schemes, and timed signature schemes, in chronological order to provide a historical background of timed cryptography. Despite the various benefits of timed cryptography in terms of efficiency and security, there is still ongoing research and development in this field to further improve their performance and address potential vulnerabilities. By conducting a detailed analysis of the existing literature on timed cryptography and comparing their efficiency and security levels, we can identify areas for future research and development and contribute to advancing the field. The results of this research will be valuable for researchers, practitioners, and policymakers in the field of cryptography, providing insights into the strengths and weaknesses of timed cryptography and its potential applications in various scenarios.

CHAPTER 2

PRELIMINARY TO THE SUBJECT

This chapter aims to equip the reader with an adequate theoretical foundation concerning timed cryptography by presenting several fundamental primitives, notations, and definitions. To begin with, the notation utilized throughout this thesis will be introduced, followed by a concise summary of essential mathematical and cryptographic concepts. An in-depth elucidation of commitment schemes will also be provided, as they play a crucial role in timed signatures. Furthermore, the final section of the chapter will explicate digital signatures and define specific digital signatures that are particularly relevant to subsequent chapters.

2.1 Notations

Let $\mathbb{Z}_N$ denotes residue class ring of modulo N. $\mathbb{Z}_N^*$ is defined as $\mathbb{Z}_N^* = \{x \in \mathbb{Z}_N : \text{gcd}(x, N) = 1\}$. $\mathbb{J}_N$ is the group of elements of $\mathbb{Z}_N^*$ with Jacobi symbol $+1$. Euler totient function is denoted as $\varphi(.)$ Let p and q be large random primes and $N = p.q$. N is called a strong RSA integer if $p = 2p'+1$ and $q = 2q'+1$ where p' and q' are also primes. N is called a Blum integer if p and q are distinct primes each congruent to $3 \mod 4$. We denote that g is randomly chosen over any group G by using $g \leftarrow_\$ G$ or $g \in_R G$. The set $\{1, ..., n\}$ is denoted by $[n]$ in come chapter.

Also, specific definitions and notations will be given in the related chapters.

Definition 2.1.1. [4] $f(n) = \mathcal{O}(g(n))$ means that there exist $c, k \in \mathbb{Z}^+$ such that $0 \leq f(n) \leq c.g(n)$ for all $n \geq k$. The values of c and k must be fixed for f and must not depend on n. Here $\mathcal{O}$ is a Landau symbol and also called asymptotic upper bound. It is a theoretical measure of the execution of an algorithm.

2.2 Mathematical Backgrounds

In this section, we present several noteworthy cryptographic hard problems that will serve as the foundation for the security proofs of puzzle and signature constructions in the following chapters. Additionally, we will provide definitions and assumptions as mathematical tools for the constructions.

Definition 2.2.1. (Probabilistic Algorithm) A probabilistic algorithm is an algorithm with an additional command RANDOM that returns "0" or "1", each with probability $1/2$.

Definition 2.2.2. (Deterministic Algorithm) A deterministic algorithm is an algorithm that produces the same output for a given input, regardless of how many times it is run. In other words, the output of the algorithm is entirely determined by its input, and there is no randomness involved in the computation.

Definition 2.2.3. (Discrete Logarithm Problem) Let $\mathbb{G}$ be a multiplicative group, g be a generator of $\mathbb{G}$ and $h \in \mathbb{G}$. When g and h are given determining x that satisfies the equation

$$g^x = h$$

is a hard problem and called the discrete logarithm problem.

Definition 2.2.4. (Integer Factorization Problem) Given a composite number $N \in \mathbb{Z}_N$, finding for which $x, y \in \mathbb{Z}_N$ satisfy $N = x \cdot y$ is a hard problem and called the integer factorization problem.

Definition 2.2.5. (Decisional Diffie-Hellman Assumption) [10] Let $\mathbb{G}$ be a cyclic group and g is a generator of $\mathbb{G}$ of order q. For given (g_a, g_b) such that $a, b \leftarrow_\$ \mathbb{Z}_q$, g^{ab} is indistinguishable from a random element in $\mathbb{G}$.

Definition 2.2.6. (Quadratic Residuosity Assumption) In $\mathbb{Z}_N^*$, for an RSA modulus N, let $a \in QR_N$ which is a square in $\mathbb{Z}_N^*$, then it is indistinguishable from a random element in $\mathbb{J}_N$.

Definition 2.2.7. (Decisional Composite Residuosity) In $\mathbb{Z}_{N^2}^*$, for an RSA modulus n, a random N-th power in $\mathbb{Z}_{N^2}^*$ is indistinguishable from a random element in $\mathbb{Z}_{N^2}^*$.

Theorem 2.2.1. [10] When $N = pq$ is strong RSA modulus, the DDH assumption on $\mathbb{J}_N$ is implied by the DDH assumption in both the large prime-order subgroups of $\mathbb{Z}_p^*$ and $\mathbb{Z}_q^*$ and the quadratic residuosity assumption over $\mathbb{Z}_N^*$.

Theorem 2.2.2. (The generalized Blum-Blum-Shub Assumption) [15] Let N be a Blum integer. A BBS sequence is $x_0, x_1, ..., x_n$ with $x_0 = g^2 \mod N$ for a random $g \in \mathbb{Z}_N$, and $x_i = x_{i-1}^2 \mod N, 1 \leq i \leq n$. The sequence defined by taking the least significant bit of the elements above is polynomial-time unpredictable (unpredictable to the left and to the right), provided the quadratic residuosity assumption (QRA) holds. Let k be an integer such that $l < k < u$. Then given the vector

$$< g^2, g^4, g^{16}, ..., g^{2^{2i}}, ..., g^{2^{2^k}} > \mod N$$

the (l, u, δ, ϵ) generalized BBS assumption states that no PRAM algorithm whose running time is less than $\delta \cdot 2^k$ can distinguish the element $g^{2^{2^{k+1}}}$ from a random quadratic residue R^2 with probability larger than ϵ. The bound l precludes the parallelization of the computation, while the bound u precludes the feasibility of computing square roots through factoring.

Theorem 2.2.3. (Sequential Squaring Assumption) [20] Let N be a uniform strong RSA integer, g be a generator of $\mathbb{J}_N$, and $T(\cdot)$ be a polynomial. Then there exists some $0 < \varepsilon < 1$ such that for every polynomial-size adversary $A = \{A_\lambda\}_{\lambda \in \mathbb{N}}$ who's depth is bounded from above by $T^\varepsilon(\lambda)$, there exists a negligible function $\mu(\cdot)$ such that

$$\Pr\left[b \leftarrow A(N, g, T(\lambda), x, y) : \begin{array}{c} x \leftarrow_\$ \mathbb{J}_N; b \leftarrow_\$ \{0,1\} \\ \text{if } b = 0 \text{ then } y \leftarrow_\$ \mathbb{J}_N \\ \text{if } b = 1 \text{ then } y := x^{2^{T(\lambda)}} \end{array} \right] \leq \frac{1}{2} + \mu(\lambda).$$

Definition 2.2.8. (Non-Interactive Zero-Knowledge: NIZK) [30] [26]

Let $R : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$ be an NP-witness-relation with corresponding NP-language $L := \{x : \text{ there exists } w \text{ such that } R(x, w) = 1\}$. A NIZK for R have three algorithms:

1. $\mathrm{ZKsetup}(1^\lambda) \rightarrow crs$ where λ is the security parameter and crs is the common reference string.

2. $\mathrm{ZKprove}(crs, x, w) \rightarrow \pi$ where the prover shows the validity of the statement x with a witness w, and π is the proof.

3. $\mathrm{ZKverify}(crs, x, \pi)$ checks the proof π.

A NIZK system satisfies two properties: the verifier does not have more information than the validity of x, and convincement of an invalid x is hard for any prover.

2.3 Digital Signatures

To create the timed signature schemes, we will present the frameworks of well-known traditional ones.

A digital signature scheme typically consists of three essential functions: key generation KeyGen, signing Sign, and verification Verify. The KeyGen algorithm produces a set of private-public key pairs (p_k, s_k), which are used by the Sign algorithm to generate the signature σ. The Verify algorithm then utilizes the corresponding p_k to verify the validity of the signature. In this scheme, the message is signed using the private key and authenticated using the corresponding public key.

2.3.1 RSA Signature Scheme

RSA Signature Scheme was first proposed by R. L. Rivest, A. Shamir, and L. M. Adlemanin in 1978 [24]. The signature scheme uses $(\mathrm{KeyGen}, \mathrm{Sign}, \mathrm{Verify})$ algorithms, and the key generation phase is identical to RSA encryption [24]. For security, N needs at least 1024 bits to provide a security level of 80 bits, and the signature needs to be at least 1024 bits long. For the signing and verification phase, exponentiation operation is used with the public and private keys, respectively. Verification is efficient as a small public key e can be chosen. Security depends on the IFP. We take p and q as large prime numbers and $m \in \{0, 1\}^*$ in Figure 2.1.

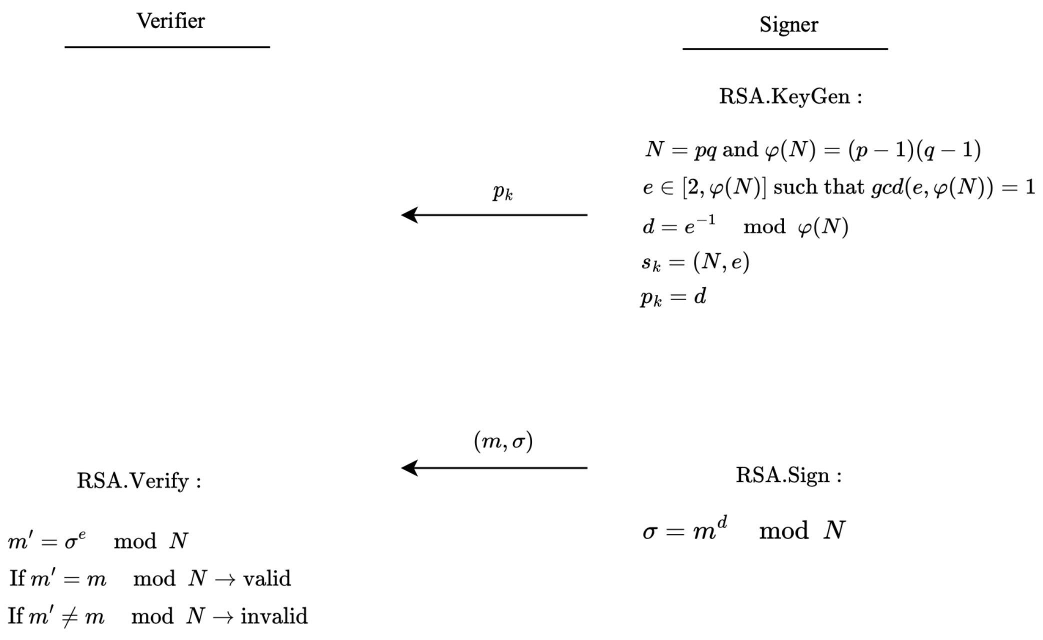

Verifier
Signer
RSA.KeyGen :
$N = pq$ and $\varphi(N) = (p-1)(q-1)$
$e \in [2, \varphi(N)]$ such that $gcd(e, \varphi(N)) = 1$
$d = e^{-1} \mod \varphi(N)$
$s_k = (N, e)$
$p_k = d$
p_k
(m, σ)
RSA.Verify :
$m' = \sigma^e \mod N$
If $m' = m \mod N \rightarrow$ valid
If $m' \neq m \mod N \rightarrow$ invalid
RSA.Sign :
$\sigma = m^d \mod N$

Figure 2.1: RSA Signature Scheme

2.3.2 Digital Signature Algorithm

DSA Signatures Scheme was proposed by the National Institute of Standards and Technology (NIST) in 1991. It is based on the Elgamal signature scheme [14]. For security of DSA that is presented in Figure 2.2, to solve the DLP in p, the powerful index calculus method can be applied. However, this method cannot be applied to the DLP of the subgroup q. Therefore q can be smaller than p. To provide a security level of 80 bits, p and q need to be at least 1024 and 160 bits, respectively. Signature verification is slower when compared with RSA. Also, in Figure 2.2, $m \in \{0, 1\}^*$.

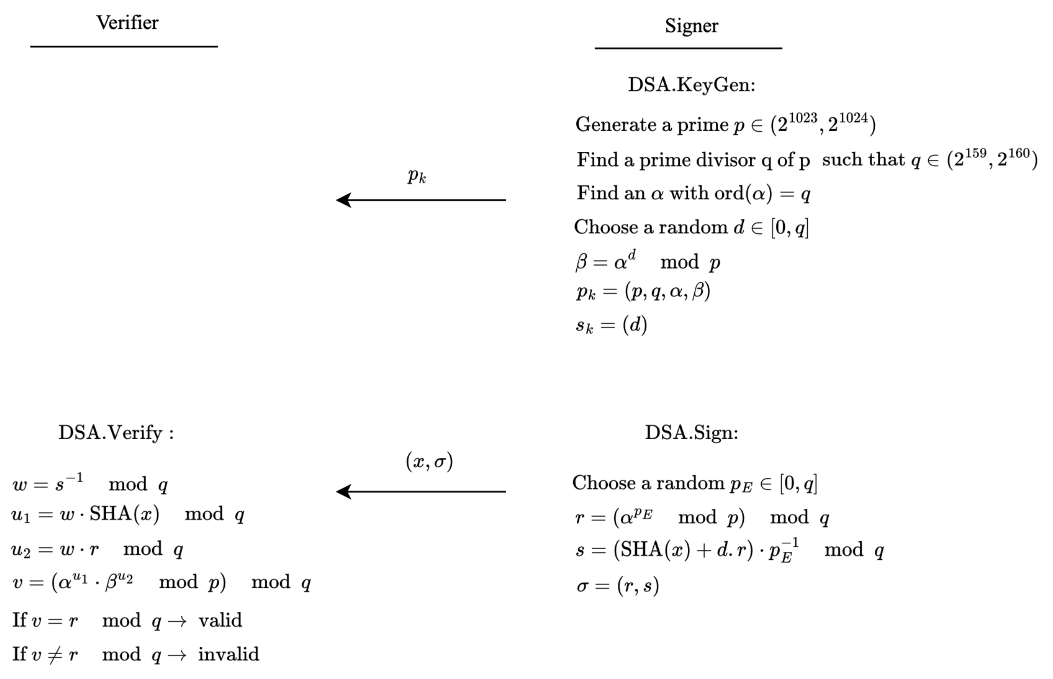

Verifier
Signer
DSA.KeyGen:
Generate a prime $p \in (2^{1023}, 2^{1024})$
Find a prime divisor q of p such that $q \in (2^{159}, 2^{160})$
Find an α with $\mathrm{ord}(\alpha) = q$
Choose a random $d \in [0, q]$
$\beta = \alpha^d \mod p$
$p_k = (p, q, \alpha, \beta)$
$s_k = (d)$
p_k
DSA.Verify :
$w = s^{-1} \mod q$
$u_1 = w \cdot \mathrm{SHA}(x) \mod q$
$u_2 = w \cdot r \mod q$
$v = (\alpha^{u_1} \cdot \beta^{u_2} \mod p) \mod q$
If $v = r \mod q \rightarrow$ valid
If $v \neq r \mod q \rightarrow$ invalid
(x, σ)
DSA.Sign:
Choose a random $p_E \in [0, q]$
$r = (\alpha^{p_E} \mod p) \mod q$
$s = (\mathrm{SHA}(x) + d.\, r) \cdot p_E^{-1} \mod q$
$\sigma = (r, s)$

Figure 2.2: DSA Scheme

2.3.3 Elliptic Curve Digital Signature Algorithm

Elliptic Curve Cryptography (ECC) has been demonstrated to possess several advantages over the widely-used RSA and other digital signature algorithms. ECC utilizes significantly shorter key lengths ranging from 160 to 256 bits, compared to 1024 to 3072 bits in RSA and DL schemes. Without strong attacks against elliptic curve cryptosystems, shorter bit lengths in ECC result in decreased processing time and shorter signatures. These benefits motivated the standardization of the Elliptic Curve Digital Signature Algorithm (ECDSA) by the American National Standards Institute (ANSI) in 1998. In the scheme of ECDSA Figure 2.3, $m \in \{0, 1\}^*$.

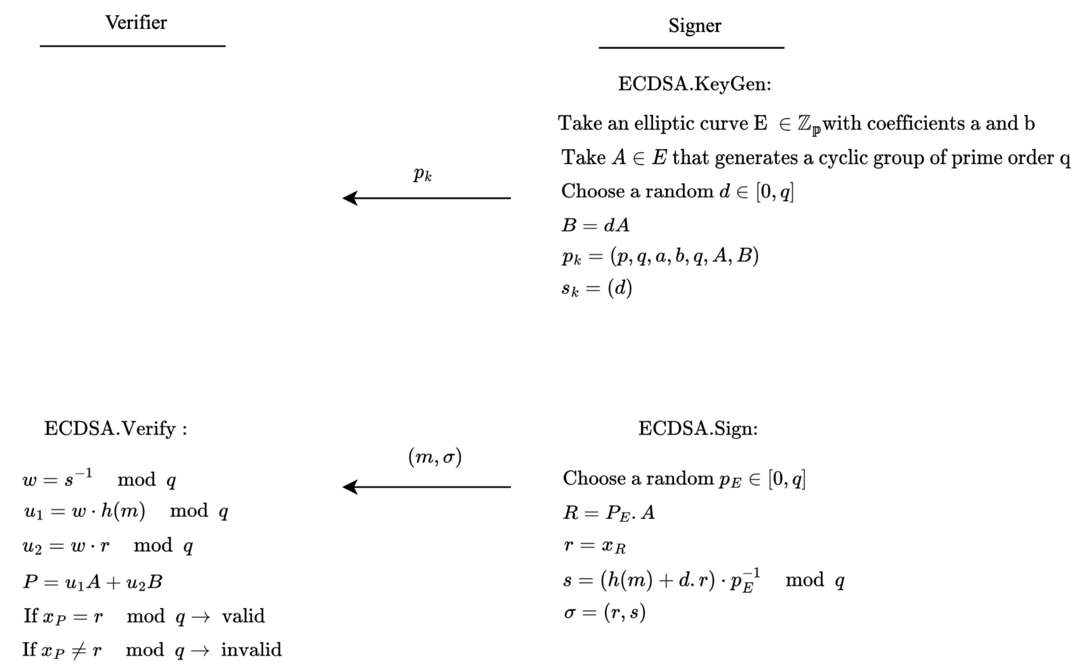

Verifier
Signer
ECDSA.KeyGen:
Take an elliptic curve E $\in \mathbb{Z}_p$ with coefficients a and b
Take $A \in E$ that generates a cyclic group of prime order q
Choose a random $d \in [0, q]$
$B = dA$
$p_k = (p, q, a, b, q, A, B)$
$s_k = (d)$
p_k
ECDSA.Verify :
$w = s^{-1} \mod q$
$u_1 = w \cdot h(m) \mod q$
$u_2 = w \cdot r \mod q$
$P = u_1 A + u_2 B$
If $x_P = r \mod q \rightarrow$ valid
If $x_P \neq r \mod q \rightarrow$ invalid
(m, σ)
ECDSA.Sign:
Choose a random $p_E \in [0, q]$
$R = P_E . A$
$r = x_R$
$s = (h(m) + d.r) \cdot p_E^{-1} \mod q$
$\sigma = (r, s)$

Figure 2.3: ECDSA Scheme

2.3.4 Schnorr Signature Scheme

Schnorr proposes Schnorr Signature Algorithm in 1991 [27]. The scheme's security in Figure 2.4 is based on the supposed intractability of DLP.

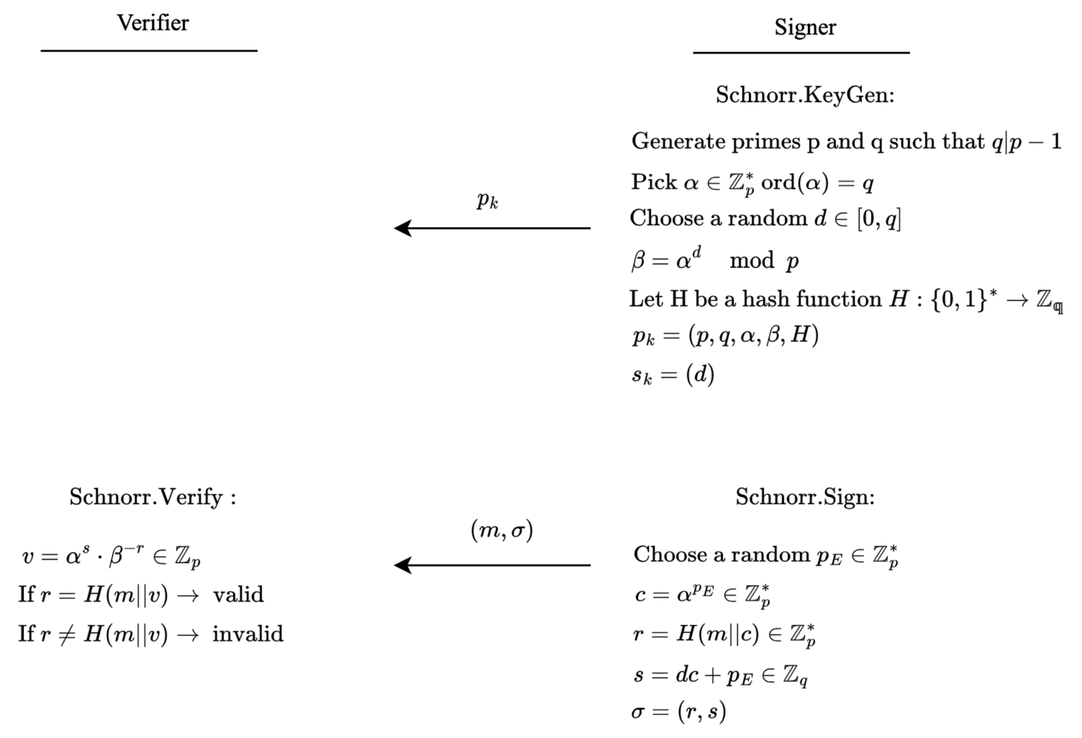

Verifier
Signer
Schnorr.KeyGen:
Generate primes p and q such that $q|p-1$
Pick $\alpha \in \mathbb{Z}_p^*$ $\text{ord}(\alpha) = q$
Choose a random $d \in [0, q]$
$\beta = \alpha^d \mod p$
Let H be a hash function $H : \{0,1\}^* \to \mathbb{Z}_q$
$p_k = (p, q, \alpha, \beta, H)$
$s_k = (d)$
p_k
Schnorr.Verify :
$v = \alpha^s \cdot \beta^{-r} \in \mathbb{Z}_p$
If $r = H(m||v) \to$ valid
If $r \neq H(m||v) \to$ invalid
(m, σ)
Schnorr.Sign:
Choose a random $p_E \in \mathbb{Z}_p^*$
$c = \alpha^{p_E} \in \mathbb{Z}_p^*$
$r = H(m||c) \in \mathbb{Z}_p^*$
$s = dc + p_E \in \mathbb{Z}_q$
$\sigma = (r, s)$

Figure 2.4: Schnorr Signature Scheme

2.3.5 BLS Signature Scheme

BLS signature scheme is a digital signature scheme proposed by Boneh, Lynn, and Shacham in 2001 [6]. In this scheme, bilinear mapping is used to verify. It can function in any group where the Decisional Diffie-Hellman Problem is easy, but the Computational Diffie-Hellman Problem is difficult. The scheme in Figure 2.5 consists of three algorithms $(\text{BLS. Gen}, \text{BLS. Sign}, \text{BLS. Verify})$, $m \in \{0,1\}^*$, and $H : \{0,1\}^* \to G_1$ that is a full-domain hash function and it is considered as a random oracle.

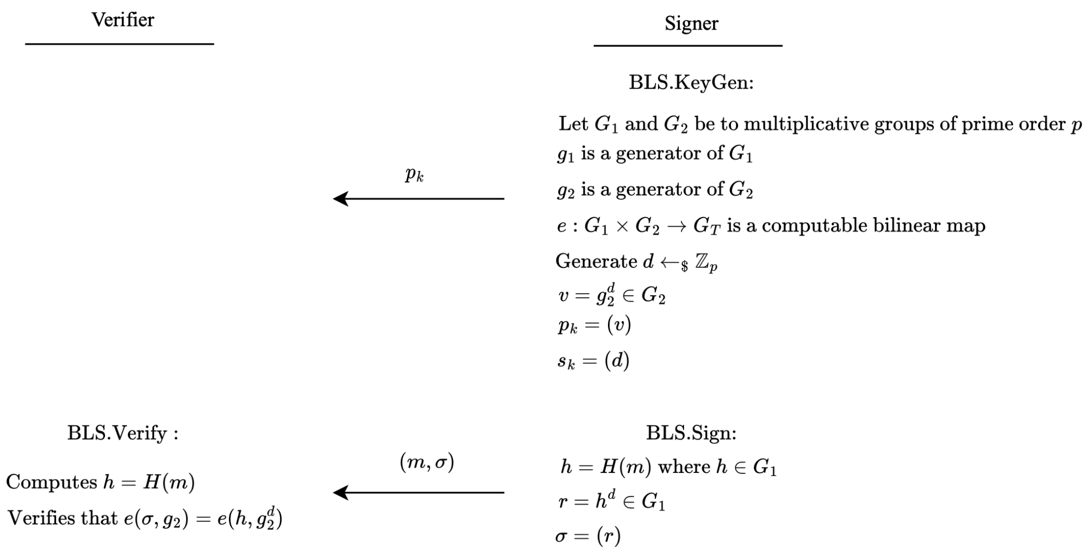

Verifier
Signer
BLS.KeyGen:
Let G_1 and G_2 be to multiplicative groups of prime order p
g_1 is a generator of G_1
p_k
g_2 is a generator of G_2
$e : G_1 \times G_2 \rightarrow G_T$ is a computable bilinear map
Generate $d \leftarrow_{\$} \mathbb{Z}_p$
$v = g_2^d \in G_2$
$p_k = (v)$
$s_k = (d)$
BLS.Verify :
Computes $h = H(m)$
Verifies that $e(\sigma, g_2) = e(h, g_2^d)$
(m, σ)
BLS.Sign:
$h = H(m)$ where $h \in G_1$
$r = h^d \in G_1$
$\sigma = (r)$

Figure 2.5: BLS Signature Scheme

CHAPTER 3

TIME-LOCK PUZZLES

This chapter is dedicated to examining the fundamental components of timed cryptography, focusing on time-lock puzzles. The discussion begins by introducing the concept of cryptographic puzzles, followed by an in-depth exploration of the notion of time in cryptography. Specifically, two notable constructions in the realm of time-lock puzzles will be thoroughly investigated: the repeated squaring-based time-lock puzzles and the homomorphic time-lock puzzles. By examining these constructions, a comprehensive understanding of the principles underlying timed cryptography and time-lock puzzles, in particular, will be achieved. Furthermore, this chapter will present two applications of homomorphic time-lock puzzles, namely e-voting and multiparty contract signing. These applications will provide insight into the computational differences between classical and homomorphic time-lock puzzles.

3.1 Merkle's Cryptographic Puzzles

In 1978, R. C. Merkle published a revolutionary paper [22] that discusses a paradigm of cryptography that tells us that for a cryptographic system, when two participants want to share information, the security of their key channel is inevitable. Nevertheless, Merkle shows that this is not a necessity. We can construct secure cryptographic systems using cryptographic puzzles (cryptograms that cannot be cryptanalyzed), although an adversary leaks into our key channels. This was the first time cryptographic puzzles have become an option for cryptographic schemes. The following algorithm is a summary of Merkle's cryptographic puzzle idea [22].

Merkle's Approach [22]:

1. Alice and Bob want to communicate, and Eve takes place in this scenario as an adversary. First, Alice and Bob decide on a strong encryption function F. Any strong and suitable encryption function can be used. They will use cryptographic puzzles for security. A cryptographic puzzle is nothing but the encryption of some information with F. Solving a puzzle means decryption of F. So we have F^{-1} as the inverse of F. They decide the total number of cryptographic puzzles N, arbitrary constant C where the random key is selected from a key space of size $C * N$. An exhaustive search through the key space can only solve a puzzle. So, each puzzle requires $\mathcal{O}(N)$ efforts to break. To control the difficulty of solving puzzles, they should make an optimization by restricting the key space of encryption. If C is too big,, solving puzzles will be harder and take much more time. Alice should not create a puzzle that can not be solved in a reasonable time.

2. Alice generates N puzzles and sends them to Bob over the key channel. Each puzzle has two pieces of information. The first one is a unique ID number for identifying the related puzzle. The second one is a puzzle key, i.e., one of the possible keys for subsequent encrypted communications. Alice chooses both of them randomly and assigns them to the puzzles.

3. Bob takes N puzzles, picks one of them randomly, and solves it. It takes $\mathcal{O}(N)$ time. After that, he gets two pieces of information, the ID of the puzzle and the puzzle key. He sends the ID to Alice over the key channel.

4. When Alice gets the ID information, she knows which puzzle has exactly that ID number and key.

5. After that, Alice and Bob use this puzzle key for further encrypted communications over the normal channel.

Security:

Since we assume that the key channel is not secure, Eve has the information of N puzzles and ID that Bob sent. In order to learn which puzzle is associated with this ID number and puzzle key, Eve should break puzzles at random until he finds the correct one, which takes, on average, $\mathcal{O}(N^2)$ time. Merkle's method has two main problems [25]: (i) brute-force key-search is parallelizable, and (ii) time estimation for expected running time is not certain and depends on the order in which the keys are examined.

3.2 Repeated Squaring Based Time-lock Puzzles

Rivest, Shamir, and Wagner proposed the first time-lock puzzles in 1996 [25]. They aim to encrypt a message so that it can not be decrypted by anyone until a pre-determined amount of time (real-time, not total CPU time) has passed, i.e., the goal is to send information into the future. Time-lock cryptographic puzzles are presented as computational mathematical problems that can not be solved without running a computer continuously for at least a certain amount of time. The problem was that the amount (large parallel computers) and the power of computers (hardware) could affect the CPU time required to solve a computational problem. Their solution is based on building time-lock puzzles with intrinsically sequential designs using repeated squaring and the random-access property of the Blum-Blum-Shub's x^2 mod n pseudo-random number generator [5].

Rivest, Shamir and Wagner's Algorithm [25]:

Suppose Alice wants to send the secret s to Bob. She encrypts s with a time-lock puzzle for a period of t seconds as follows:

1. Alice chooses two random large primes p and q and computes:

$$N = pq,$$

$$\varphi(N) = (p-1)(q-1),$$

$$T = tS$$

where S is the number of squarings modulo N per second.

2. She chooses a suitable cryptosystem and encryption algorithm Enc and then generates a random key k. She computes

$$C_s = Enc(k, s).$$

3. She picks a random x mod N where $1 < x < N$ and computes

$$e = 2^T \mod (\varphi(N)),$$

$$y = x^e \mod (N),$$

and encrypts k,

$$C_k = k + x^{2^T} = k + y \mod (N).$$

4. She outputs the time-lock puzzle (N, T, x, C_k, C_s), and erases the secret values p and q.

5. Bob takes the output and solves the puzzle by starting with x and performs T sequential squaring in order to compute

$$y = x^{2^T} \mod (N).$$

Security:

Factoring N and determining $\varphi(N)$ is a hard problem (IFP). So, there is no faster way of computing y than to start with x and perform T sequential squaring. In this approach, a possible problem is that a computer needs to work continuously on the puzzle until it is solved. Because the puzzle does not automatically become solvable at a given time, it suits simple puzzles better (e.g., with time-to-solution under a month).

Rivest, Shamir, and Wagner's methodical approach is fundamental to comprehending, conceptualizing, and developing time-lock puzzles and allows us to establish a precise and rigorous definition for time-lock puzzles.

Definition. A time-lock puzzle (TLP) is a tuple of two algorithms $(\mathrm{PGen}, \mathrm{PSolve})$ defined as follows:

- $Z \leftarrow \mathrm{PGen}(T, s)$ 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.
- $s \leftarrow \mathrm{PSolve}(Z)$ is a deterministic algorithm that takes as input a puzzle Z and outputs a solution s.

Completeness:

For every λ, T, $s \in \{0,1\}^\lambda$ and Z, $\mathrm{PGen}(T, s)$, $\mathrm{PSolve}(Z)$ outputs s.

Efficiency:

$Z \leftarrow \mathrm{PGen}(T, s)$ can be computed in time $poly(\lambda, logT)$ and $\mathrm{PSolve}(Z)$ can be computed in time $poly(\lambda, T)$.

Definition 3.2.1. [20] A TLP is secure with gap $\epsilon < 1$ if there exists a polynomial $\tilde{T}(\cdot)$ such that for all polynomials $T(.) \geq \tilde{T}(\cdot)$ and every polynomial-size adversary $A = \{A_\lambda\}_{\lambda \in \mathbb{N}}$ where the depth of A is bounded from above by $T^\epsilon(\lambda)$, there exists a negligible function μ such that for all $\lambda \in \mathbb{N}$, and every pair of solutions $s_0, s_1 \in \{0,1\}^\lambda$ it holds that:

$$\Pr\left[b \leftarrow A_\lambda(Z) : \begin{array}{c} b \leftarrow_{\$} \{0,1\} \\ Z \leftarrow \mathrm{TLP.PGen}(t(\lambda), s_b) \end{array} \right] \leq \frac{1}{2} + \mu(\lambda).$$

Time-lock puzzles are cryptographic constructs that possess distinctive features. The first characteristic is their ability to generate puzzles swiftly, which necessitates computational resources significantly lower than T. This property is especially crucial when concealing secrets over extended durations of time. The second feature is the robust security they offer against parallel algorithms. Regardless of their size, circuits possessing a depth lower than T cannot compromise the hidden secret s confidentiality.

Disadvantages:

- In the absence of a trusted third party, time-lock puzzles require the decrypter to execute a lengthy computational process to access the concealed secret. Though this is necessary to ensure security, it poses a potential disadvantage when time-lock puzzles are implemented on a larger scale, as the computational burden may become significant.

- No measures are taken to verify that the puzzle can be unlocked in the desired time.

This traditional form of time-lock puzzles has been extensively utilized in various cryptographic applications such as sealed-bid auctions [25], fair contract signing [7], zero-knowledge arguments [12], and non-malleable commitments [19]. It can be seen as the pioneer among all the other structures. Also, Bitansky et al. proposed a new candidate for time-lock puzzles as an alternative to Rivest, Shamir, and Wagner's scheme in 2015 [3] as we mentioned in Chapter 1. They designed randomized encoding based time-lock puzzles. The main contribution of their design was its universal feature, as its security can be established by relying on any family of non-parallelizing languages. It sets it apart from RSW's proposal, which depends on a specific computation's non-parallelizability (on average) concerning a specific distribution.

3.3 Homomorphic Time-lock Puzzles

The main disadvantage of repeated squaring-based and randomized encoding based time-lock puzzles is that one needs to solve (brute-force) many puzzles before computing some function over the embedded secrets. Both have a 'first compute, then solve' approach. Malavolta and Thyagarajan devised a solution to this critical shortcoming in 2019 [20]. They proposed a 'first compute then solve ' method by constructing homomorphic time-lock puzzles (HTLP).

To put it simply, an HTLP [20] is a time-lock puzzle that allows the evaluation of a circuit C over sets of puzzles $(Z_1, ..., Z_n)$ in a homomorphic manner without needing to know the secret messages $(s_1, ..., s_n)$ contained within those puzzles. The output of the evaluation, which is also a puzzle, contains the circuit output $C(s_1, ..., s_n)$, and the puzzle's timing difficulty does not depend on the size of the evaluated circuit C (Compactness). It is important to emphasize that maintaining the compactness of the evaluation algorithm is an essential requirement for HTLP. If this requirement is neglected, the simplistic approach of solving the puzzles $(Z_1, ..., Z_n)$ and evaluating C over the secrets would be sufficient. So, the work of [20] introduces the concept of HTLPs and characterizes their security guarantees formally.

Definition 3.3.1. [20] Say $C = \{C_\lambda\}_{\lambda \in \mathbb{N}}$ be a class of circuits and S is a finite domain. A homomorphic time-lock puzzle is a tuple of four algorithms:

- $pp \leftarrow \mathrm{HTLP.\,PSetup}(1^\lambda, T)$: a probabilistic algorithm that takes security parameter 1^λ and a hardness parameter T as inputs and outputs public parameters pp.
- $Z \leftarrow \mathrm{HTLP.\,PGen}(pp, s)$: a probabilistic algorithm that outputs a puzzle Z encapsulating a solution $s \in S$.
- $s \leftarrow \mathrm{HTLP.\,PSolve}(pp, Z)$: a deterministic algorithm that takes a puzzle Z and solves it to output a solution s.
- $Z' \leftarrow \mathrm{HTLP.\,PEval}(C, pp, Z_1, ..., Z_n)$: a probabilistic algorithm that takes a set of n puzzles, evaluates a circuit C on these puzzles to output a puzzle Z'.

Efficiency:

$pp \leftarrow \mathrm{HTLP.\,PSetup}(1^\lambda, T)$ can be computed in time $poly(\lambda, logT)$ and
$Z' \leftarrow \mathrm{HTLP.\,PEval}(C, pp, Z_1, ..., Z_n)$ can be computed in time $poly(\lambda, |C|)$.

Security:

As previously defined, the homomorphic time-lock puzzles' security shares the same theoretical concept as the classical time-lock puzzles. The security of both types of puzzles depends on ensuring that the solution remains hidden from any adversaries running in (parallel) time that are less than T. In this context, we will focus on a basic version of the homomorphic time-lock puzzles where the countdown starts when the public parameters become available. Formal requirements for the security and compactness of the homomorphic time-lock puzzles are as follows:

Definition 3.3.2. [20] An HTLP scheme is secure with gap $\epsilon < 1$ if there exists a polynomial $\tilde{T}(\cdot)$ such that for all polynomials $T(.) \geq \tilde{T}(\cdot)$ and every polynomial-size adversary $(A_1, A_2) = \{(A_1, A_2)_\lambda\}_{\lambda \in \mathbb{N}}$ where the depth of A_2 is bounded from above by $T^\epsilon(\lambda)$ there exists a negligible function $\mu(\cdot)$ such that for all $\lambda \in \mathbb{N}$ it holds that

$$\Pr\left[b \leftarrow A_2(pp, Z, \tau) : \begin{array}{l} (\tau, s_0, s_1) \leftarrow A_1(1^\lambda) \\ pp \leftarrow \text{HTLP.PSetup}(1^\lambda, \mathsf{T}(\lambda)) \\ b \leftarrow_\$ \{0,1\} \\ Z \leftarrow \text{HTLP.PGen}(pp, s_b) \end{array} \right] \leq \frac{1}{2} + \mu(\lambda),$$

and $(s_0, s_1) \in S^2$.

Definition 3.3.3. [20] Let $C = \{C_\lambda\}_{\lambda \in \mathbb{N}}$ be a class of circuits (along with their respective representations). An HTLP scheme is compact for the class C if for all $\lambda \in \mathbb{N}$, all polynomials T in λ, all circuits $C \in C_\lambda$ and respective inputs $(s_1, ..., s_n) \in S^n$, all pp in the support of $\text{HTLP.PSetup}(1^\lambda, T)$, and all Z_i in the support of $\text{HTLP.PGen}(pp, s_i)$ the following two conditions are satisfied:

- There exists a fixed polynomial $p(\cdot)$ such that $|Z| = p(\lambda, |C(s_1, ..., s_n)|)$, where $Z \leftarrow \text{HTLP.Eval}(C, pp, Z_1, ..., Z_n)$.
- There exists a fixed polynomial $\tilde{p}(\cdot)$ such that the runtime of $\text{HTLP.PEval}(C, pp, Z_1, ..., Z_n)$ is bounded by $\tilde{p}(\lambda, |C|)$.

Also, HTLP scheme is correct if the following conditions are satisfied:

- There exists a negligible function $\mu(.)$ such that

  $\Pr[\text{HTLP.PSolve}(pp, \text{HTLP.PEval}(C, pp, Z_1, ..., Z_n)) \neq C(s_1, ..., s_n)] \leq \mu(\lambda)$.
- There exists a fixed polynomial $p(\cdot)$ such that the runtime of $\text{HTLP.PSolve}(pp, Z)$ is bounded by $p(\lambda, T)$.

G. Malavolta and S. A. K. Thyagarajan [20] developed three distinct but related constructions for homomorphic time-lock puzzles and subsequently explained how they can be used in various practical settings. Through their research, they devised these constructions and explored their potential real-world applications, shedding light on the advantages and limitations of each method.

3.3.1 Linearly Homomorphic Time-lock Puzzles (LHTLP)

The group $\mathbb{Z}_{N^2}^*$ can be written as the product of the group generated by $(1+N)$ that has order N and the group of N-th residues $\{x^N : x \in \mathbb{Z}_N^*\}$ that has order $\varphi(N)$. When the tuple (N, x, x^{2^T}) where x is the random element of $\mathbb{Z}_N^*$ with Jacobi symbol $+1$ is fixed for the setup phase, a linearly homomorphic HTLP can be computed as

$$(N, T, x^r, (x^{N.2^T})^r \cdot (1+N)^s) = (N, T, y, y^{N.2^T} \cdot (1+N)^s)$$

where r is uniformly sampled from $\{1, ..., N^2\}$ whose distribution (modulo φ) is statistically close to sampling from $\{1, ..., \varphi\}$. It is correctly distributed, and the knowledge of $\varphi(N)$ is not needed to compute. The scheme is an HTLP for linear functions, assuming the inherent sequentiality of squaring modulo N and other standard intractability assumptions over hidden-order groups.

A linearly homomorphic time-lock puzzle over the ring $(\mathbb{Z}_N, +)$ is as follows [20]:

Setup:

- Takes two primes $p = 2p' + 1$ and $q = q' + 1$ such that p' and q' are primes and computes $N = pq$.
- Takes uniform $\tilde{g} \leftarrow_\$ \mathbb{Z_N}^*$ and set $g = -\tilde{g}^2 \mod N$.
- Computes $h = g^{2^T}$ by reducing $2^T \mod \varphi(N)/2$.
- $\text{LHTLP.PSetup}(1^\lambda, T) \rightarrow pp = (T, N, g, h)$.

Puzzle Generation:

- Chooses a uniform $r \leftarrow_\$ \{1, ..., N^2\}$ as the randomization factor.
- Computes $u = g^r \mod N$ and $v = h^{r.N} \cdot (1+N)^s \mod N^2$.
- $\text{LHTLP.PGen}(pp, s) \rightarrow Z = (u, v)$.

Puzzle Solving:

- Computes $w = u^{2^T} \mod N$ using repeated squaring and
$$s = \frac{(v/(w)^N \mod N^2) - 1}{N}$$
.

- $\mathrm{LHTLP.PEval}(pp, Z) \rightarrow s$.

Evaluation:

- Computes $\tilde{u} = \prod_{i=1}^n u_i \mod N$ and $\tilde{v} = \prod_{i=1}^n v_i \mod N^2$ for each $(u_i, v_i) \in \mathbb{J}_N \times \mathbb{Z}_{N^2}$ belongs to Z_i's.
- $\mathrm{LHTLP.PEval}(\oplus, pp, Z_1, ..., Z_n) \rightarrow (\tilde{u}, \tilde{v})$.

Correctness [20]:

$$\begin{aligned}
\tilde{s} &= \frac{(\tilde{v}/(\tilde{w})^N \mod N^2) - 1}{N} \\
&= \frac{(\prod_{i=1}^n v_i/(\prod_{i=1}^n u_i^{2^T} \mod N)^N \mod N^2) - 1}{N} \\
&= \frac{(\prod_{i=1}^n h^{r_i.N} \cdot (1+N)^{s_i}/(\prod_{i=1}^n h^{r_i} \mod N)^N \mod N^2 - 1}{N} \\
&= \frac{\prod_{i=1}^n h^{r_i.N}(1+N)^{s_i}/\prod_{i=1}^n h^{r_i.N} \mod N^2) - 1}{N} \\
&= \frac{((1+N)^{\sum_{i=1}^n s_i} \mod N^2) - 1}{N} \\
&= \frac{1 + N \cdot \sum_{i=1}^n s_i - 1}{N} = \sum_{i=1}^n s_i.
\end{aligned} \tag{3.1}$$

Security depends on the sequential squaring, DDH over $\mathbb{J}_N$ and DCR over $\mathbb{Z}^*_{N^2}$ assumptions. The scheme is perfectly correct and satisfies the notion of randomness homomorphism.

3.3.2 Multiplicatively Homomorphic Time-lock Puzzles

Given that the tuple (N, x, x^T) is fixed in a setup phase, a puzzle to encapsulate a secret $s \in \mathbb{J}_N$ such that $\mathbb{J}_N$ is the subgroup of $\mathbb{Z}^*_N$ whose elements have Jacobi symbol $+1$ is generated as
$$(N, T, x^r, (x^{2^T})^r \cdot s)$$

for some uniformly chosen r.

A multiplicatively homomorphic time-lock puzzle over the ring $(\mathbb{J}_N, \cdot)$ is as follows [20]:

Setup:

- $\mathrm{MHTLP.\,PSetup}(1^{\lambda, T}) \rightarrow pp = (T, N, g, h)$ is the same with the $\mathrm{LHTLP.\,PSetup}$ function.

Puzzle Generation:

- Chooses a uniform $r \leftarrow_{\$} \{1, ..., N^2\}$ as the randomization factor.
- Generates $u = g^r \mod N$ and $v = h^r \cdot s \mod N$.
- $\mathrm{MHTLP.\,PGen}(pp, s) \rightarrow Z = (u, v)$.

Puzzle Solving:

- Computes $w = u^{2^T} \mod N$ using repeated squaring and $s = v/w$.
- $\mathrm{MHTLP.\,PEval}(pp, Z) \rightarrow s$.

Evaluation:

- Computes $\tilde{u} = \prod_{i=1}^{n} u_i \mod N$ and $\tilde{v} = \prod_{i=1}^{n} v_i \mod N$ for each $(u_i, v_i) \in \mathbb{J}_N^2$ belongs to Z_i's.
- $\mathrm{MHTLP.\,PEval}(\otimes, pp, Z_1, ..., Z_n) \rightarrow (\tilde{u}, \tilde{v})$.

Correctness [20]:

$$\begin{aligned}\tilde{s} &= \frac{\tilde{v}}{\tilde{w}} = \frac{\tilde{v}}{\tilde{u}^{2^T}} = \frac{\prod_{i=1}^{n} v_i}{\prod_{i=1}^{n} u_i^{2^T}} \\ &= \frac{\prod_{i=1}^{n} h^{r_i} \cdot s_i}{\prod_{i=1}^{n} g^{r_i . 2^T}} = \frac{\prod_{i=1}^{n} h^{r_i} \cdot s_i}{\prod_{i=1}^{n} h^{r_i}} \\ &= \prod_{i=1}^{n} s_i.\end{aligned} \tag{3.2}$$

Security depends on the sequential squaring and DDH assumptions over $\mathbb{J}_N$.

3.3.3 Fully Homomorphic Time-lock Puzzles

Using linear and multiplicative homomorphic time-lock puzzles is limited to evaluating certain restricted function classes over private data. However, whether an HTLP can be constructed for any polynomially-computable function remains unanswered, and existing techniques have yet to be instrumental in this regard [20]. Constructing homomorphic encryption from RSA groups has proven to be difficult, and as a result, the focus has shifted to constructions based on indistinguishability obfuscation [17]. Although a scheme has been developed using this approach, it is considered a possible result, and the challenging task of constructing HTLPs for any function without obfuscation remains an open problem. The fully-homomorphic encryption (FHE) scheme [20] blueprint can be used to construct a fully homomorphic time-lock puzzle. Before constructing a fully homomorphic time-lock puzzle, first, we need to define some primitives. Let,

1. $(\mathrm{KGen}, \mathrm{Enc}, \mathrm{Dec}, \mathrm{tKeyGen})$ be a trapdoor encryption scheme.
2. $(\mathrm{Key}, \mathrm{Puncture}, \mathrm{PRF})$ be a puncturable PRF.
3. $(\mathrm{PGen}, \mathrm{PSolve})$ be a homomorphic time-lock puzzle.
4. piO and iO be obfuscators for probabilistic and deterministic circuits, respectively.
5. $\mathrm{Prog}^{(s_k,p_k)}(\alpha, \beta)$ and $\mathrm{MProg}^{(s_{k0},k,k')}(i)$ are circuits such that:

   (a) $\mathrm{Prog}^{(s_k,p_k)}(\alpha, \beta)$:

   ```
   ________________________________
   $\alpha = (z_\alpha, c_\alpha)$
   $\beta = (z_\beta, c_\beta)$
   $s_\alpha \leftarrow \mathrm{Dec}(s_k, c_\alpha), s_\beta \leftarrow \mathrm{Dec}(s_k, c_\beta)$
   $s = s_\alpha$ NAND $s_\beta$
   $z \leftarrow \mathrm{PGen}(T, s)$
   $c \leftarrow \mathrm{Enc}(p_k, s)$
   return$(z, c)$
   ```

(b) $\mathrm{MProg}^{(s_{k0},k,k')}(i)$:

```
$r_{i-1} \leftarrow \mathrm{PRF}(k, i-1)$
$r_i \leftarrow \mathrm{PRF}(k, i), r'_i \leftarrow \mathrm{PRF}(k', i)$
$(p_{ki-1}, s_{ki-1}) \leftarrow \mathrm{KeyGen}(1^\lambda; r_{i-1})$
$(p_{ki}, s_{ki}) \leftarrow \mathrm{KeyGen}(1^\lambda, r_i)$
$P_i \leftarrow \mathrm{Prog}^{(s_{ki-1}, p_{ki})}$
$\Lambda_i \leftarrow piO(1^p, P_i; r'_i)$
return$(\lambda_i)$
```

6. L be a super-polynomial function such that $L(\lambda) = 2^{w(log(\lambda))}$.

A fully homomorphic time-lock puzzle [20] is as follows:

Setup:

- Takes key pairs $(p_{k0}, s_{k0}) \leftarrow \mathrm{KeyGen}(1^\lambda)$.
- Takes two PRF keys $k, k' \leftarrow \mathrm{Key}(1^\lambda)$.
- Obfuscates using iO the circuit $\mathrm{MProg}^{s_{k0},k,k'}$, i.e., take $\mathrm{MEvk} \leftarrow iO(1^p, \mathrm{MProg}^{(s_{k0},k,k')})$ where the security parameter $p = p(\lambda)$ for obfuscation is an upper-bound on the size of $\mathrm{MProg}^{(s_{k0},k,k')}$.
- $\mathrm{FHTLP.\,PSetup}(1^\lambda, T) \rightarrow pp = (T, p_{k0}, \mathrm{MEvk})$.

Puzzle Generation:

- Generates $c \leftarrow \mathrm{Enc}(p_{k0}, s)$ and $z \leftarrow \mathrm{PGen}(T, s)$ where c is the ciphertext and z is the puzzle.
- $\mathrm{FHTLP.\,PGen}(pp, s) \rightarrow Z = (z, c)$.

Puzzle Solving:

- Computes $s \leftarrow \mathrm{PSolve}(z)$
- $\mathrm{FHTLP.\,PSolve}(pp, Z) \rightarrow s$.

Puzzle Evaluation:

- Evaluates C layer by layer. For iteration $i \in \{0, ..., l\}$, generate the evaluation key for the layers as $\Lambda_i \leftarrow \mathrm{MEvk}(i)$.
- For each NAND gate g in this layer i, let $\alpha(g), \beta(g)$ be the puzzles of the values of its input wires.
- Evaluate g homomorphically by computing $\gamma(g) \leftarrow \Lambda_i(\alpha(g), \beta(g))$ as the puzzle of the value of g's output wire.
- $\mathrm{FHTLP.\,PEval}(C, pp, Z_1, ..., Z_n) \rightarrow$ the puzzle generated in the last iteration l.

Correctness: It is straightforward to establish the correctness of the overall system by relying on the correctness of the underlying components.

Security depends on the security of μ-hiding trapdoor encryption scheme, indistinguishable obfuscators (piO, iO) for the class of S and circuits respectively, and the puncturable PRF.

3.4 Applications of Homomorphic Time-Lock Puzzles

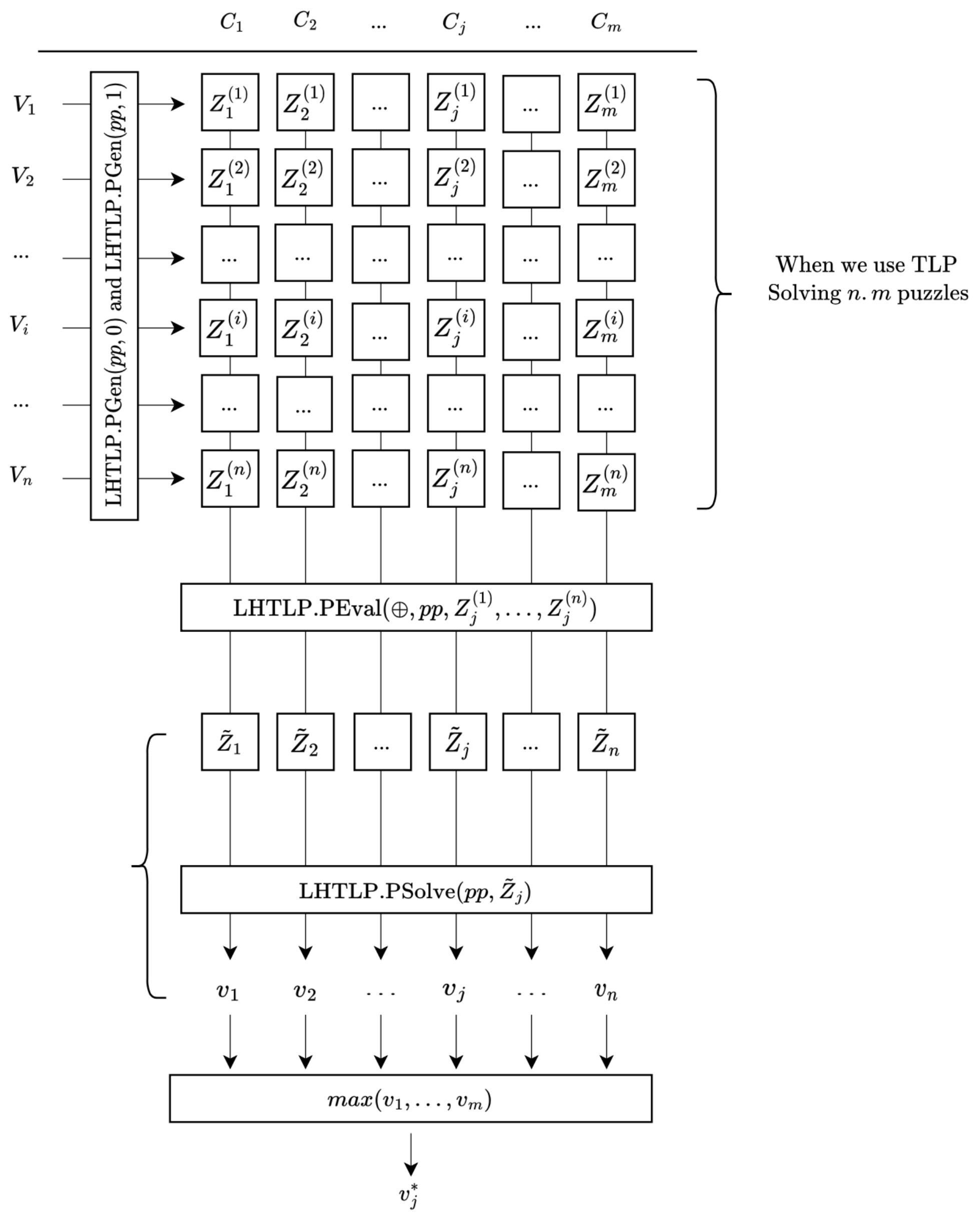
Malavolta and Thyagarajan [20] have comprehensively analyzed the application scenarios for homomorphic puzzles, highlighting their significance in various domains. In particular, they have explored the use of linearly homomorphic puzzles for e-voting, sealed bid auctions over blockchains and multi-party coin flipping, and multiplicatively homomorphic time-lock puzzles for multi-party contract signing. Constructing models for these scenarios has shed light on the difference between classical time-lock puzzles (RSW) and homomorphic ones. These models serve as crucial tools for understanding the practical implications of homomorphic puzzles in real-world applications.

3.4.1 Linearly Homomorphic Time-Lock Puzzles on E-Voting

The authors present a protocol [20] for selecting a single candidate from m options with n voters that involve two phases: voting and counting. The voting phase involves each voter generating a vector of m linearly-homomorphic puzzles, where only the j-th puzzle, corresponding to the voter's preferred candidate, encodes 1 while the remaining puzzles encapsulate 0. The vectors are publicly posted on a blockchain, and the final result is determined by summing up all vectors and opening the resulting m puzzles during the counting phase. This protocol does not require a trusted authority for tallying, and the computational effort remains constant regardless of the number of voters. The authors also suggest that similar techniques can be used for sealed bid auctions, although the resulting protocol currently requires fully-homomorphic time-lock puzzles due to the circuit's complexity [20]. The proposed approach improves upon previous solutions that may require opening a large number of puzzles, making it more efficient.

Algorithm 1 Electronic Voting Scheme [20]

```
1: Election Setup:
2: Generate $pp \leftarrow \mathrm{LHTLP.PSetup}(1^\lambda, T)$ and publish them so that they are accessible to all the voters.
3: Voting Phase: To deciding to vote the $j$-th candidate $C_j$ such that $j \in \{1, ..., m\}$
4: for each voter $V_i$ do
5:     Generate $Z_{j'} \leftarrow \mathrm{LHTLP.PGen}(pp, 0)$ for all $j' \in \{1, ..., m\}/j$.
6:     Generate $Z_j \leftarrow \mathrm{LHTLP.PGen}(pp, 1)$.
7:     Compute $vote_i = (Z_1, ..., Z_m)$ and outputs $vote_i$.
8: end for
9: Counting Phase:
10: Collect $(vote_1, ..., vote_n)$ from all voters and parse each vote as $vote_i = (Z_1^{(i)}, ..., Z_m^{(i)})$
11: for each candidate $j \in \{1, ..., m\}$ do
12:     Compute the puzzle $\tilde{Z}_j \leftarrow \mathrm{LHTLP.PEval}(\oplus, pp, Z_j^{(1)}, ..., Z_j^{(n)})$.
13:     Count the votes received by $j$-candidate by $v_j \leftarrow \mathrm{LHTLP.PSolve}(pp, \tilde{Z}_j)$.
14: end for
15: Compute $v_j = \max(v_1, ..., v_m)$.
16: Output $j$-th candidate as the winner of the election.
```



C_1 C_2 ... C_j ... C_m
V_1
V_2
...
V_i
...
V_n
LHTLP.PGen$(pp, 0)$ and LHTLP.PGen$(pp, 1)$
$Z_1^{(1)}$ $Z_2^{(1)}$ $Z_j^{(1)}$ $Z_m^{(1)}$
$Z_1^{(2)}$ $Z_2^{(2)}$ $Z_j^{(2)}$ $Z_m^{(2)}$
$Z_1^{(i)}$ $Z_2^{(i)}$ $Z_j^{(i)}$ $Z_m^{(i)}$
$Z_1^{(n)}$ $Z_2^{(n)}$ $Z_j^{(n)}$ $Z_m^{(n)}$
When we use TLP Solving $n.m$ puzzles
LHTLP.PEval$(\oplus, pp, Z_j^{(1)}, \ldots, Z_j^{(n)})$
$\tilde{Z}_1$ $\tilde{Z}_2$ ... $\tilde{Z}_j$ ... $\tilde{Z}_n$
LHTLP.PSolve$(pp, \tilde{Z}_j)$
v_1 v_2 ... v_j ... v_n
$max(v_1, \ldots, v_m)$
v_j^*

Figure 3.1: LHTLP Electronic Voting Scheme

The provided Figure 3.1 illustrates the proposed model for electronic voting, which utilizes homomorphic time-lock puzzles to determine the winner of an election. Employing a "compute-then-solve" method shows that the computation of n puzzles is required for winner selection. In contrast, the classical method of time-lock puzzles necessitates the solution of all Z_i^j puzzles before computation and selection of the winner can occur, resulting in $n.m$ puzzle-solving computations. Based on this analysis, it can be concluded that using homomorphic time-lock puzzles in real-life election scenarios is significantly more efficient than the classical approach.

3.4.2 Multiplicatively Homomorphic Time-Lock Puzzles on Multi-Party Contract Signing

Boneh and Naor addressed the problem of n mutually distrusting parties exchanging signatures on a document with their protocol for fair exchange of signatures [7], based on time-lock puzzles. In each round, each party generates a time-lock puzzle of their signature and broadcasts it until all signatures are published. However, suppose any party fails to broadcast its puzzle in any round. In that case, all other parties must solve all the puzzles from the previous round to learn the signatures necessary for the contract's validity. To solve this issue, the authors propose using their multiplicatively homomorphic time-lock puzzles and RSA-aggregate signatures, which allow homomorphic aggregation of signatures in QR_N , where N is fixed. This tool enables replacing the time-lock puzzle of Boneh and Naor with the multiplicatively homomorphic construction so that in the case of any offline party, each other party can homomorphically aggregate the signatures from the previous round and solve a single time-lock puzzle.

The proposed solution for the problem of mutually distrusting parties jointly signing a contract involves using multiplicatively homomorphic time-lock puzzles and RSA-aggregate signatures. An aggregate signature scheme publicly combines signatures over different messages and under different keys so that the digest remains verifiable. In the proposed contract-signing protocol, each party generates a signature on contract M and time-locks it with a timing hardness T. If all parties successfully broadcast their time-lock puzzles, the protocol proceeds to the next iteration. Otherwise, each party collects the puzzles from the previous iteration and generates the final puzzle, which is solved to reveal the aggregated signature σ_{aqq} on M. The protocol proceeds until $T_l := 2$ is reached. The public parameters of the Hohenberger-Waters signature scheme [18] and MHTLP are generated in the setup phase. Each user generates a key pair and enters the loop for generating signatures and time-lock puzzles.

Algorithm 2 Multi-Party Contract Signing [20]

```
1: **Setup Phase:**
2: Generate $pp_1 \leftarrow \mathrm{Setup}(1^\lambda, 1^T)$ for the aggregate signature scheme and $pp_2 \leftarrow \mathrm{MHTLP.PSetup}(1^\lambda, T_1, ..., T_l)$ and broadcast it to all parties.
3: **Key Generation Phase:**
4: **for** each party $P_i$ **do**
5:     Execute $(p_{k_i}, s_{k_i}) \leftarrow \mathrm{KeyGen}(pp_1)$ to generate $(pk_i, sk_i)$.
6: **end for**
7: **Signing Phase:**
8: **for** $k = 1$ to $l$ **do**
9:     **for** each party $P_i$ **do**
10:        Generate $\sigma_i^{(k)} \leftarrow \mathrm{Sign}(pp_1, s_{ki}, M)$.
11:        Time-lock the signature $Z_i^{(k)} \leftarrow \mathrm{MHTLP.PGen}(pp_2, \sigma_i^{(k)}, T_k)$.
12:        Broadcast the puzzle $Z_i^{(k)}$ to all parties.
13:    **end for**
14:    **Aggregation Phase:**
15:    **if** all parties have broadcast their puzzles or $k = l$ **then**
16:        Collect the puzzles $(Z_1^{(k-1)}, ..., Z_n^{(k-1)})$ from the $(k-1)$-th iteration
17:        Generate the final puzzle
18:        $Z^{(k-1)} \leftarrow \mathrm{MHTLP.PEval}(\otimes, pp_2, Z_1^{(k-1)}, ..., Z_n^{(k-1)})$
19:        Solve the puzzle to obtain $\sigma_{agg} \leftarrow \mathrm{MHTLP.Solve}(pp_2, Z^{(k-1)})$ on $M$.
20:        Output $(M, \sigma_{agg})$.
21:    **end if**
22: **end for**
```

CHAPTER 4

TIMED SIGNATURES

In the ensuing chapter, our objective is to comprehensively delineate timed commitment schemes and timed signatures in chronological order. The underlying constructions rely on time-lock puzzles expounded in the preceding chapter. Primarily, we shall explain Boneh and Naor's methodologies [7], premised on the repeated squaring time-lock puzzles. Following this, we shall elucidate Garay and Jakobsson's Timed Signatures [15], which are grounded on the principle of Boneh and Naor's timed signatures. Lastly, we shall present the verifiable timed signatures [30] predicated on homomorphic time-lock puzzles. By adopting this systematic approach, we aim to proffer a thorough and rigorous exposition of these essential concepts.

Initially, we present the precise definition of commitment schemes, as the subsequent sections are predicated upon this fundamental concept.

Definition 4.0.1. A commitment scheme is a cryptographic primitive that allows one to commit to a chosen value while keeping it hidden from others, with the ability to reveal the committed value later. It consists of two phases:

1. Commitment Phase: A value is chosen and committed to (at the end of which the sender is bound to the value).

2. Reveal Phase: The sender reveals the value to the receiver.

4.1 Boneh and Naor's Timed Signature Scheme

In 2000, D. Boneh and M. Naor introduced the notion of timed commitments [7] in which a potential forced opening phase permits the receiver to recover the committed value without the help of the committer. Then, they construct a timed signature scheme analogously to their timed commitment scheme, which resists parallel attacks by relying on modular exponentiation.

Contribution:

- In 1995, Damgard [11] proposed a well-designed pure two-party contract signing protocol (without a trusted third party). However, Damgard's protocol releases the signature bit-by-bit and is not resistant to parallel exhaustive search attacks. Boneh and Naor designed a two-party contract signing protocol without this deficiency.

- The authors used a similar function with RSW's time-lock puzzles. But, as we mentioned before, in RSW's settings, we do not have measures for the desired time of verification. Boneh and Naor worked for this deficiency.

Definition 4.1.1. A timed commitment scheme of Boneh and Naor [7] is as follows:

1. Commit phase: To commit to a string $S \in \{0,1\}^{\lambda}$, the committer and the verifier execute a protocol whose outcome is a commitment string C, which is given to the verifier.

2. Open phase: Later, the committer may reveal the string S to the verifier. They execute a protocol so that at the end of the protocol, the verifier has proof that S is the committed value.

3. Forced open phase: If the committer refuses to execute the open phase and does not reveal S, then the forced-open algorithm takes the commitment string C as input and outputs S and a proof that S is the committed value. The algorithm's running time is at most T.

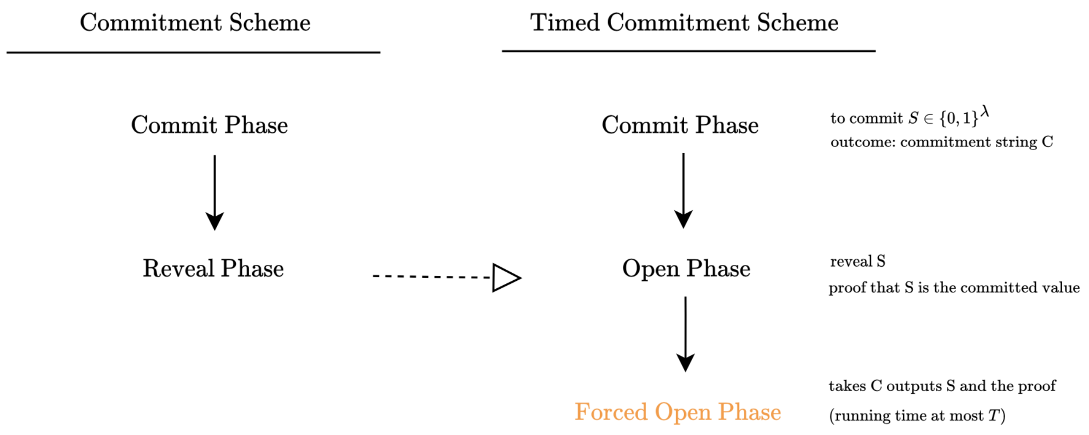

Commitment Scheme
Timed Commitment Scheme
Commit Phase
Reveal Phase
Commit Phase
Open Phase
Forced Open Phase
to commit $S \in \{0,1\}^\lambda$
outcome: commitment string C
reveal S
proof that S is the committed value
takes C outputs S and the proof
(running time at most T)

Figure 4.1: Schema for Timed Commitment Scheme

It should satisfy three security properties:

Binding: During the open phase, the committer cannot convince the verifier that C is a commitment to a different S.

Soundness: [7] At the end of the commit phase, the verifier is convinced that, given C, the forced open algorithm will produce the committed string S in time T.

Privacy: [7] Every PRAM algorithm whose running time is at most t such that $t < T$) on polynomially many processors will succeed in distinguishing s from a random string, given the transcript of the commit protocol as input, with an advantage at most ϵ. This requirement ensures that even an adversary equipped with a highly parallel machine must spend at least time t to open the commitment (with high probability) forcibly.

A timed signatures scheme enables the committer to give the verifier a signature s on a message M in two steps. The committer commits to the signature s, and the verifier accepts the commitment. After a time, the committer completely reveals the signature S to the verifier. If the committer does not reveal the signature, the verifier can spend time T to forcibly retrieve the signature from the commitment. As before, the committer is assured that after time t $(t < T)$, the verifier will not be able to retrieve the signature with probability more than ϵ. For a timed commitment scheme;

- During the commit phase, the committer must convince the verifier that the forced open algorithm will successfully retrieve the committed value. It must be done without actually running the forced open algorithm. Boneh and Naor use an efficient zero-knowledge protocol in the commit phase for the following protocol.
- An adversary with thousands of machines cannot forcibly open the commitment much faster than a legitimate party with only one machine.

4.1.1 Boneh and Naor's Timed Commitment Scheme: [7]

Setup:

- Let $T = 2^k \in \mathbb{Z}$ and security parameter $n \in \mathbb{Z}^+$.
- The committer takes n-bit random primes $p_1 = p_2 = 3 \mod (4)$ as private keys.
- He computes $N = p_1.p_2$ and $\varphi(N) = (p-1).(p_2-1)$ then publishes $< N >$.

Commit Phase:

The committer wants to commit to a message M of length l. The committer performs the following:

- Picks $h \in_R \mathbb{Z}_N$.
- Computes $g = h^{(\prod_{i=1}^{r} q_i^n)} \mod N$ where $q_i < B$ for some bound B and all of them are prime.
- Computes $a = 2^{2^k} \mod \varphi(N)$ using repeated squaring and set $u = g^a \mod N$.
- Hides the bits of M by XORing them with the LSB's of succesive square roots of $u \mod N$. Sets $S_i = M_i \oplus lsb(g^{2^{(2^k-i)}} \mod N)$ and let $S = S_1 ... S_l \in \{0,1\}^l$.
  Outputs the commitment string $C =< h, g, u, S >$ and sends to the verifier.

When the verifier takes h and g, he verifies that g is constructed properly from h. The verifier is assured that the order of g in $\mathbb{Z}_N^*$ is not divisible by any prime less than B. The committer must convince the verifier that $u = g^{2^{2^k}} \mod N$.

<u>*Convincing Step (Zero-Knowledge)*</u>:

- The committer constructs $W =< g^2, g^4, g^{16}, g^{256}, ..., g^{2^{2^i}}, ..., g^{2^{2^k}} > \mod N$ and sends to the verifier.
- For each i, the committer proves in zero-knowledge to the verifier that the triple (g, b_{i-1}, b_i) is a triple of the form (g, g^x, g^{x^2}) for some x.
- By verifying that the last element in W is equal to u the verifier is assured that indeed $u = g^{2^{2^k}} \mod N$.
- Each of these k proofs take four rounds and they can all be done in parallel. These proofs are based on a classic zero-knowledge proof that a tuple $< g, A, B, C >$ is a Diffie-Hellman tuple.

<u>*Proving integrity of W*</u>: Let q be the order of g in $\mathbb{Z}_N^*$ and R be a security parameter.

- The verifier picks $c_1, ..., c_k \in_R \{0, ..., R\}$ and sends c_i's using any regular commitment scheme.
- The committer picks $d_1, ..., d_k \in_R \mathbb{Z}_q$. He computes for all $i \in \{1, ..., k\}$, $z_i = g^{d_i}$ and $w_i = b_{i-1}^{d_i}$. He sends the pairs $< z_i, w_i >$ to the verifier.
- The verifier reveals all c_i's.
- The committer computes $y_i = c_i.2^{2^{i-1}} + d_i \mod q$ for all i values and sends to the verifier.
- The verifier check for all i, $g^{y_i}.b_{i-1}^{-c_i} = z_i$ and $b_{i-1}^{y_i}.b_i^{-c_i} = w_i$.

Open Phase: The committer sends $v' = h^{2^{(2^k - l)}} \mod N$ to the verifier and the verifier performs the following:

- Computes $v = (v')^{\prod_{i=1}^{r} q_i^n} \mod N$ and ensures that v has odd order.

- Verifies $v^{2^l} = u \mod N$ (v is in the subgroup generated by g since it has order and the 2^l'th root o u).

- Constructs the l-bit Blum-Blum-Shub pseudo-random sequence $R \in \{0,1\}^l$ starting at v.

- Sets for $i = \{1, ..., l\}$, R_i to be the least significant bit of $v^{2^{l-i}}$.

- $M = R \oplus S$.

M is the only possible outcome of the open phase protocol and the protocol satisfies the binding property. It is proven by Boneh and Naor in [7].

Forced Open Phase:

- The verifier computes $v = g^{2^{(2^k - l)}} \mod N$ using $(2^k - l)$ squarings in modulo N.

The above timed commitment scheme takes $T = 2^k$ modular exponentiations to retrieve the committed string forcibly. So, the time notion for the timed commitment scheme is the T value. The commit phase takes $\mathcal{O}(k)$ modular exponentiations. The forced open algorithm takes a few hours or days depending on $k \in [30, 50]$.

Security of the timed commitment scheme depends on the generalized BBS assumption. The protocol satisfies soundness, zero-knowledge and binding properties.

4.1.2 Boneh and Naor's Timed Signature Scheme:

Let (σ, V, G) be a signature scheme where σ takes a message and a private key and generates a signature, V takes a signature and a public key and verifies the signature and G is a public and private key pair generator. The timed signature scheme (T, t, ϵ) is as follows:

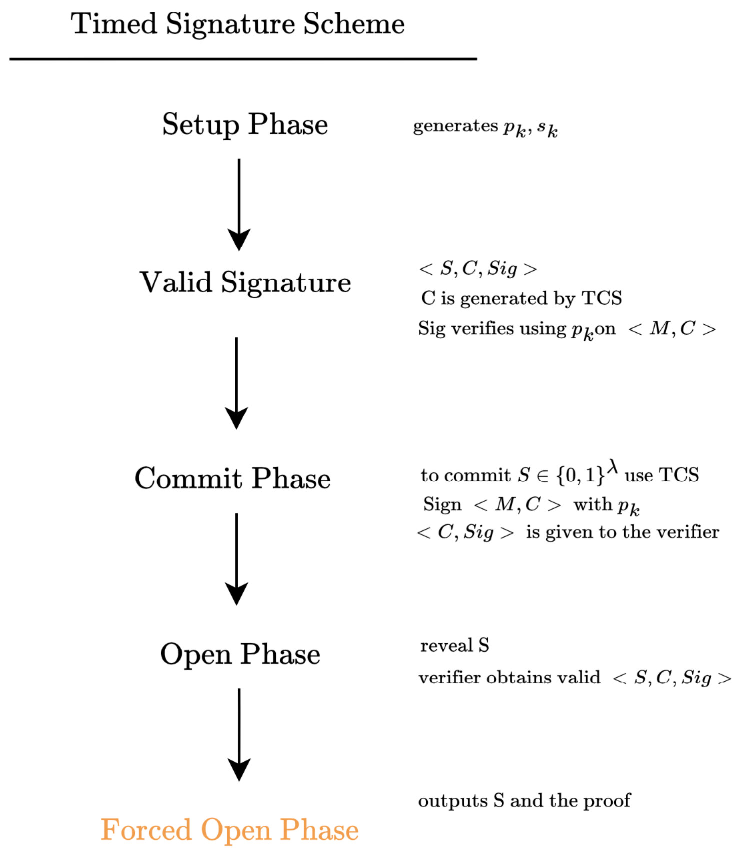

Timed Signature Scheme
Setup Phase
generates p_k, s_k
Valid Signature
$< S, C, Sig >$
C is generated by TCS
Sig verifies using p_k on $< M, C >$
Commit Phase
to commit $S \in \{0,1\}^\lambda$ use TCS
Sign $< M, C >$ with p_k
$< C, Sig >$ is given to the verifier
Open Phase
reveal S
verifier obtains valid $< S, C, Sig >$
Forced Open Phase
outputs S and the proof

Figure 4.2: Schema for Timed Signature Scheme

An abuse of the protocol may occur since once the commit phase is done, the verifier can convince a third party that the signer is about to give him a signature on M. Indeed, the value Sig in the commitment string given to the verifier could have only come from the signer. Boneh and Naor also designed a contract signing protocol by constructing a timed signature so that after the commit phase, the verifier cannot convince a third party that he has been talking to the signer.

4.2 Garay and Jakobsson's Timed Signatures

J. A. Garay and M. Jakobsson discuss a method for releasing digital signatures at a specific time and show how to implement this for RSA, Schnorr, and DSA signatures in 2002 [15]. Once released, these types of signatures cannot be distinguished from those not released at a specific time, making the process transparent to anyone observing the end result. The notion and construction of Boneh and Naor's timed commitments is the starting point for Garay and Jakobsson. As RSW's time-lock puzzles and Boneh and Naor's structures, iterated squaring is used as a solution for parallelization of the problem-solving.

Contribution:

- Reuse of the proposed timed structure and achieves lower session costs than Boneh and Naor's by introducing the concept of a reusable time-line.

- The robust timed release of standard signatures where the committed value from the time-line as a "blinding" factor for the signature.

Now, first, we present the notion of Garay and M. Jakobsson's time-line, then the proposed time-line commitment scheme, time-line signature scheme, and the versions of the robust timed release of standard signatures.

Definition 4.2.1. [15] A time-line is a vector of elements, where each element is obtained by iterated squaring of the previous element. Its endpoints represent the time-line, and its commitment corresponds to a value on the line. This value can be computed by iterated squaring of one of the values representing the time-line. Derived time-lines are obtained by "shifting" of a master time-line. Derived time-lines are backward compatible.

Definition 4.2.2. [15] Let N be a Blum integer and g an element of large odd order in $\mathbb{Z}_N*$. A time-line is a partial description of a BBS sequence, given by the subsequence $\{g^{2^{2^i}}\}_{i=0}^{k} \mod N$. We call the the value $g^{2^{2^k-1}}$ the time-line's hidden value. A value lies on a time-line if obtained by iterated squaring of the time-line's start value.

Definition 4.2.3. [15] A (T, t, ϵ) time-line commitment is a scheme in which the committer gives the receiver a timed commitment to a hidden value. At a later time, she can reveal this value and prove that it's the correct one. However, if the committer fails to reveal it, the receiver can spend time T and retrieve it. It should also satisfy binding, soundness, and privacy properties.

4.2.1 Garay and Jakobsson's Time-Line Commitment Scheme:

Setup:

- Let $T = 2^k \in \mathbb{Z}$ and the security parameter $n \in \mathbb{Z}^+$.
- The committer takes $n - bit$ random primes $p_1 = p_2 = 3 \mod 4$ as private keys.
- He computes $N = p_1.p_2$ and publishes $< N >$.
- He picks the master time-line's generator g as in first two steps of Boneh and Naor's Timed Commitment Scheme's commit phase.

The committer performs the following steps to generate the master-line:

- Computes $M_i = g^{2^{2^i}} \mod N$, $0 \leq i \leq k$, sets $m_{1st} = g^{2^{2^k-1}} \mod N$ and $m_{2nd} = g^{2^{2^k-2}} \mod N$. If the party knows $\varphi(N)$ this computation can be performed efficiently by repeated squaring $a_i = 2^{2^i} \mod \varphi(N)$ and then $M_i = g^{a_i} \mod N$.
- Shows that each $< g, M_i, M_{i+1} >$, $0 \leq i \leq k-1$ is of the form $< g, g^x, g^{x^2} >$ for some x (proof of $M_i = g^{2^{2^i}} \mod N$, for $0 \leq i \leq k$).
- Outputs $\{M_i\}_{i=0}^k$, m_{1st} and m_{2nd}, along with the above non-interactive proof. Before using the master time-line, any party checks that ${m_{2nd}}^2 = m_{1st} \mod N$, ${m_{1st}}^2 = M_k \mod N$.

Each of these k proofs take four rounds and they can all be done in parallel. These proofs are based on a classic zero-knowledge proof that a tuple is a Diffie Hellman tuple as in the case of Boneh and Naor's Timed Commitment Scheme.

Commit phase:

The committer performs the following steps:

- Picks at random $\alpha \in \mathbb{Z}_{\varphi'(N)}$ where $\varphi'(N) = \varphi(N)$ is used if known otherwise, $\varphi'(N) = \lfloor N - 2\sqrt{N} \rfloor$.

- Computes the new time-line: $f = g^{\alpha}$, $R_{k-1} = (M_{k-1})^{\alpha}$, $r_{aux} = (m_{2nd})^{\alpha}$, $r = (m_{1st})^{\alpha}$, and $R_k = (M_k)^{\alpha} \mod N$. Outputs h, R_{k-1} and R_k and keeps r and r_{aux} secret. r is the new time-line's hidden value.
- Proves that the new time-line is correctly derived from the master time-line

  $$log_g f = log_{M_{k-1}} R_{k-1} = log_{M_k} R_k = (\alpha).$$

  He uses equality of discrete logs for the proof [8], [9], [2].

Open phase:

- The committer sends α and $h' = h^{2^{(2^k-1)}}$ to the verifier.
- The verifier computes $r = (h'^{\alpha})^{\prod_{i=1}^{p} q_i^n} \mod N$ and checks that $f = g^{\alpha} \mod N$ and $r^2 = R_k \mod N$.

Forced open phase:

- The verifier computes r from R_{k-1} by repeated squaring in modulo N using $2^k - 2$ operations.

Security of the time-line commitment scheme depends on DLP and the generalized BBS assumption.

4.2.2 Garay and Jakobsson's Time-Line Signature Scheme:

Let $\mathrm{Sig}(m)$ denote the signature on message m, generated with the signer's public key. The timed signature (T, t, ϵ) is as follows:

Commit Phase:

The signer performs (T, t, ϵ) time-line commitment scheme and blinds the signature with r. She gives the blinded signature and the time-line commitment $\mathrm{TL}(r)$ with the proofs of well-formedness of the time-line, uniqueness of the blind factor, and correctness of the blinding operation to the verifier.

Open Phase:

The signer performs open phase algorithm to reveal r. The verifier unblinds the signature with r and gets $\mathrm{Sig}(m)$.

Forced Open Phase:

The verifier performs forced open phase algorithm of time-line commitment scheme, retrieves r and unblinds the signature.

Let N be a publicly known modulus, $N' = N^2$ be a second auxiliary modulus and $g' = N + 1$. The order of the subgroup generated by g' equals to N. We will use g' and N' to perform auxiliary commitments in the following three signature schemes. For the following timed signature schemes, we assume that the signer and the verifier have already done a time-line commitment $\mathrm{TL}(r)$ before the commitment phase.

4.2.3 Timed RSA Signature Scheme:

Setup:

Let n be an RSA modulus, m be the message to be signed, (e, n) be the signer's public key, and d his secret key where $m^{ed} = m \mod n$ for all values $m \in \mathbb{Z}_n$. The signer's signature on a hashed message m is $s = m^d \mod n$.

Commit Phase:

The signer performs the following:

- Computes $s = m^d \mod n$.
- Blinds the signature by $\tilde{s} = s^{1/r} \mod n$ where r is the time-line's hidden value, and sends the pair $(m, \tilde{s})$ to the verifier.
- Computes

$$b_{aux} = g'^{r_{aux}} \mod N',$$

$$b = g'^{r} \mod N',$$

$$B = g'^{R_k} \mod N'.$$

and proves that

$$log_{g'} b_{aux} = log_{b_{aux}} b (= r_{aux})$$

using the properties of equality of discrete logs

$$log_{g'} b = log_b B (= R).$$

- The verifier computes $X = \tilde{s}^e$.
- The signer proves that $log_X m = log_{g'} b (= r)$.

Open and Forced Open Phases:

The verifier obtains r from the signer or from the time-line forced open algorithm and unblinds the signature by computing $s = \tilde{s}^r \mod n$.

The committer has to compute $s^{1/r} \mod n$ in the blinding step. It raises the signature s to the value $1/r \mod \varphi(n)$. If the committer is also the signer, he knows $\varphi(n)$. But, if he is not, he can not compute $\varphi(n)$. This scheme does not allow one party to commit to a signature generated by another party.

4.2.4 Timed Schnorr Signature Scheme:

Setup:

Let $\bar{g}$ be a generator of a group of size $\bar{q}$ and let all computation be modulo $\bar{p}$ where $\bar{q} \mid \bar{p} - 1$. Let $\bar{x} \in \mathbb{Z}_{\bar{q}}$ be the secret key and $\bar{y} = \bar{g}^{\bar{x}}$ be the public key. The signer selects $\bar{k} \in_R \mathbb{Z}_{\bar{q}}$ and computes $\bar{\rho} = \bar{g}^{\bar{k}} \mod \bar{p}$ and $\bar{s} = \bar{k} + \bar{x} h(m, \bar{\rho}) \mod \bar{q}$ where m is the message to be signed.

Commit Phase:

The signer knows a signature $(\bar{\rho}, \bar{s})$ on a message m known by the committer and the receiver. Unlike Timed RSA Signature Scheme, the committer and the signer do not need to be the same parties.

- Let r be the blinding factor. Let $\tilde{s} = \bar{s}/r \mod \bar{q}$ be the blinded signature, and $\tilde{g} = \bar{g}^r \mod \bar{p}$ be the blinded generator. The committer outputs $(\tilde{g}, \bar{\rho}, \tilde{s})$,

- The committer computes

$$b_{aux} = g'^{r_{aux}} \mod N',$$

$$b = g'^{r} \mod N',$$

$$B = g'^{R_k} \mod N'.$$

  and proves that

$$log_{g'} b_{aux} = log_{b_{aux}} b (= r_{aux})$$

  using the properties of equality of discrete logs

$$log_{g'} b = log_b B (= R).$$

- The verifier checks that $\tilde{g}^{\tilde{s}} = \overline{\rho y}^{h(m,\bar{\rho})} \mod \bar{p}$.
- The committer proves that $log_{\bar{g}} \tilde{g} = log_{g'} b (= r)$.

Open and Forced Open Phases:

The verifier obtains r from the signer or from the time-line forced open algorithm and unblinds the signature by computing $\bar{s} = \tilde{s}^r \mod \bar{q}$. He outputs $(m, \bar{\rho}, \bar{s})$.

4.2.5 Timed DSA Signature Scheme:

Setup:

Let m be the hashed message to be signed. The signer selects $\bar{k} \in_R \mathbb{Z}_{\bar{q}}$ and computes $\bar{\rho} = \bar{g}^{\bar{k}} \mod \bar{p}$, $\bar{\lambda} = \bar{\rho} \mod \bar{q}$ and $\bar{s} = \bar{k}^{-1}(m + \bar{x}\bar{\lambda}) \mod \bar{q}$. Like the Timed Schnorr Signature Scheme, the committer and the signer do not need to be the same parties. Assume that he knows $\bar{\rho}$.

Commit Phase:

The committer knows the signature $(\bar{\rho}, \bar{s})$ on message m. The committer and the receiver perform the following:

- Let r be the blinding factor. Let $\tilde{\rho} = \bar{\rho}^r \mod \bar{p}$ and $\tilde{s} = \bar{s}/r \mod \bar{q}$. The committer outputs $(\tilde{\rho}, \bar{\rho}, \bar{\lambda}, \tilde{s})$,

- The committer computes

$$b_{aux} = g'^{r_{aux}} \mod N',$$

$$b = g'^{r} \mod N',$$

$$B = g'^{R_k} \mod N'.$$

  and proves that

$$log_{g'} b_{aux} = log_{b_{aux}} b (= r_{aux})$$

  using the properties of equality of discrete logs

$$log_{g'} b = log_b B (= R).$$

- *T*he verifier checks that $\tilde{\rho}^{\tilde{s}} = \bar{g}^m \bar{y}^{\bar{\lambda}} \mod \bar{p}$.
- The committer proves that $log_{\bar{\rho}} \tilde{\rho} = log_{g'} b (= r)$.

Open and Forced Open Phases:

The verifier obtains r from the signer or from the time-line forced open algorithm and unblinds the signature by computing $\bar{s} = \tilde{s} r \mod \bar{q}$. He outputs $(m, \bar{\lambda}, \bar{s})$.

Security of the Timed RSA Signature Scheme, Timed Schnorr Signature Scheme, and Timed DSA Signature Scheme based on the hardness of DLP and the generalized BBS assumption.

4.3 Verifiable Timed Signature Scheme

The first Verifiable Timed Signature (VTS) and Verifiable Timed Commitment scheme (VTC) was presented by S. Thyagarajan et al. in 2020 [30]. A VTS scheme allows a signature on a known message to be locked for a specific amount of time T, such that after a sequential computation has been performed for time T, anyone can extract the signature from the time lock. Verifiability ensures that anyone can publicly verify that a time lock contains a valid signature on the message without solving it and that the signature can be obtained by solving it for time T. That means VTS can be considered a timed version of verifiably encrypted signatures without requiring a trusted third party to retrieve the signature.

The design is an efficient cut-and-choose protocol based on homomorphic time-lock puzzles to prove the validity of a signature within a time-lock puzzle that presents an efficient range proof protocol that significantly improves upon previous proposals in terms of proof size, which is also of independent interest. Homomorphic time-lock puzzles reduce the number of puzzles to only one puzzle. Also, they satisfy the notion of randomness homomorphism, which was expounded upon in great detail in Chapter 3.3. Also, threshold secret sharing [28] is used in the design, and the construction is secure against parallel computations.

Contribution:

- Chapter 4.2 of a source discusses the work of Garay and Jakobsson (and later Garay and Pomerance [16]), who proposed a way to create special-purpose zero-knowledge proofs that prove to a verifier that a time-lock puzzle contains a valid signature. However, their method requires both the prover and the verifier to locally store a "time-line" list of group elements whose length equals the number of timed checkpoints, which has not yet been implemented. The new scheme implements a difference in using an RSA modulus N for setup, which can be shared among all users in the system or sampled by the signer depending on the specific application.

- Banasik, Dziembowski, and Malinowski [1] have presented a method called "cut and choose" to verify that a time-lock puzzle contains a valid signing key. In this technique, the prover sends puzzles with signing keys for a public keys, and the verifier asks to open $a - b$ of them. If the opened puzzles are valid, the verifier solves the remaining ones. The verifier can then post a transaction using a multisig script where b-1 of the keys belongs to the verifier. But, the VTS scheme requires solving a single puzzle and posting a transaction with a single signature for a corresponding public key.

We will now introduce the concept of verifiable timed signatures and different variations of robust timed releases for standard signatures.

Definition 4.3.1. [30] A verifiable timed signature (VTS) for a signature scheme $\prod = (\text{KeyGen}, \text{Sign}, \text{Verify})$ is a tuple of four algorithms $(\text{Commit}, \text{Vrfy}, \text{Open}, \text{ForceOp})$ such that:

1. $(C, \pi) \leftarrow \text{Commit}(\sigma, T)$: Takes as input a signature σ (generated using $\Pi \cdot \text{Sign}(s_k, m)$) and hiding time T and outputs a commitment C and a proof π.

2. $0/1 \leftarrow \text{Vrfy}(p_k, m, C, \pi)$: Takes as input a public key p_k, a message m, a commitment C of hardness T and a proof π and outputs 1 if σ embedded in c is a valid signature on the message m with respect to the public key p_k that is $\Pi \cdot \text{Verify}(p_k, m, \sigma) = 1$. Otherwise, it outputs 0.

3. $(\sigma, r) \leftarrow \text{Open}(C)$: Takes as input a commitment C and outputs the committed signature σ and the randomness r used in generating C.

4. $\sigma \leftarrow \text{ForceOp}(C)$: Takes as input the commitment C and outputs a signature σ.

Soundness: [30]

A VTS scheme for a signature scheme $\Pi = (\text{KeyGen}, \text{Sign}, \text{Verify})$ is sound if there is a negligible function $negl$ such that for all probabilistic polynomial time adversaries A and all $\lambda \in \mathbb{N}$, we have:

$$\Pr\left[b_1 = 1 \wedge b_2 = 0 : \begin{array}{l} (p_k, m, C, \pi, T) \leftarrow A^{1^\lambda} \\ (\sigma, r) \leftarrow \text{ForceOp}(C) \\ b_1 := \text{Vrfy}(p_k, m, C, \pi) \\ b_2 := \Pi := \text{Verify}(p_k, m, \sigma) \end{array} \right] \leq negl(\lambda).$$

Definition 4.3.2. [30] A VTS is simulation-sound if it is sound even when the prover has access to simulated proofs for (possibly false) statements of his choice; i.e., the prover must not be able to compute a valid proof for a fresh false statement of his choice.

Privacy: [30]

A VTS scheme for a signature scheme $\Pi = (\text{KeyGen}, \text{Sign}, \text{Verify})$ is private if there exists a PPT simulator S, a negligible function $negl$, and a polynomial $\tilde{T}$ such that for all polynomials $T > \tilde{T}$, all PRAM algorithms A whose running time is at most $t < T$, all messages $m \in \{0,1\}^*$, and all $\lambda \in \mathbb{N}$ it holds that

$$\left| \begin{array}{l} \Pr\left[A(pk, m, C, \pi) = 1 : \begin{array}{c} (p_k, s_k) \leftarrow \Pi \cdot \text{KeyGen}\left(1^\lambda\right) \\ \sigma \leftarrow \Pi.\, \text{Sign}(s_k, m) \\ (C, \pi) \leftarrow \text{Commit}(\sigma, \mathbf{T}) \end{array} \right] \\ - \Pr\left[A(p_k, m, C, \pi) = 1 : \begin{array}{c} (p_k, s_k) \leftarrow \Pi.\, \text{KeyGen}\left(1^\lambda\right) \\ (C, \pi, m) \leftarrow S(pk, \mathbf{T}) \end{array} \right] \end{array} \right| \leq \text{negl}(\lambda).$$

Definition 4.3.3. [30] Lets say $(\text{LHTLP.\,PSetup}, \text{LHTLP.\,PGen}, \text{LHTLP.\,PSolve}, \text{LHTLP.\,PEval})$ is an **LHTLP**. The public coin interactive zero-knowledge proofs for the language L are described as follows:

$$\begin{array}{l} L := \{\text{stmt} = (p_k, m, Z, T) : \exists wit = (\sigma, r) \text{ s.t} \\ (\text{Verify}(p_k, m, \sigma) = 1) \wedge (Z \leftarrow \text{LHTLP.\,PGen}(\mathrm{T}, \sigma; \mathrm{r}))\}. \end{array}$$

The Commit algorithm embeds the signatures inside the time-lock puzzles and uses the zero-knowledge proof system for L to prove the validity of the time-locked signature. Say $(\text{ZKsetup}, \text{ZKprove}, \text{ZKverify})$ be a a zero-knowledge proof system for the language L_{range} defined as follows:

$$\begin{array}{l} L_{range} := \{\text{stmt} = (Z, a, b, T) : \exists wit = (\sigma, r) \text{ s.t} \\ (\sigma \in [a, b]) \wedge (Z \leftarrow \text{LHTLP.\,PGen}(\mathrm{T}, \sigma; \mathrm{r}))\}. \end{array}$$

For all the following verifiable timed signature schemes, we have some common assumptions. Let n be the security parameter and take t as $n/2 + 1$. Also, $|\sigma| = \lambda$ be the maximum number of bits of the signature σ. The hash function H' is defined as $H' : \{0,1\}^* \rightarrow I \subset [n]$ where $|I| = t - 1$. We assume that ForceOp algorithm solves in parallel $\tilde{n} = (n - t + 1)$ puzzles.

4.3.1 VT-BLS Signatures

Let $(\mathbb{G}_0, \mathbb{G}_1, \mathbb{G}_t)$ be a bilinear group of prime order q where q is a λ-bit prime. Let e be an efficiently computable biliniear pairing $e : \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_T$, where g_0 and g_1 are generators of $\mathbb{G}_0$ and $\mathbb{G}_1$ respectively. Let H be a hash function $H : \{0,1\}^* \to \mathbb{G}_1$. BLS construction and VT-BLS protocol [30] are as follows:

- $(p_k, s_k) \leftarrow \text{KeyGen}(1^\lambda)$: Chooses $\alpha \leftarrow \mathbb{Z}_q$, set $h \leftarrow g_0^\alpha \in \mathbb{G}_0$ and outputs $p_k := h$ and $sk := \alpha$.
- $\sigma \leftarrow \text{Sign}(s_k, m)$: Outputs $\sigma := H(m)^{s_k} \in \mathbb{G}_1$.
- $0/1 \leftarrow \text{Verify}(p_k, m, \sigma)$: If $e(g_0, \sigma) = e(p_k, H(m))$, then outputs 1 and otherwise output 0.

Setup:

1. Takes 1^λ as the input.
2. Runs $\text{ZKsetup}(1^\lambda)$ to generate crs_{range}.
3. Generates the public parameters $pp \leftarrow \text{LHTLP. PSetup}(1^\lambda, T)$.
4. Outputs $crs := (crs_{range}, pp)$.

Commit and Prove:

1. Takes (crs, wit) as the input.
2. Parses $wit := \sigma$, $crs := (crs_{range}, pp)$, p_k as the BLS public key, and m as the message to be signed.
3. For all $i \in [t-1]$ samples a uniform $\alpha_i \leftarrow \mathbb{Z}_q$ and set $\sigma_i := H(m)^{\alpha_i} \cdot h_i := g_0^{\alpha_i}$.
4. For all $i \in \{t, ..., n\}$ computes
$$\sigma_i = \left(\frac{\sigma}{\prod_{j \in [t-1]} \sigma_j^{l_j(0)}} \right)^{l_i(0)^{-1}}, h_i = \left(\frac{p_k}{\prod_{j \in [t-1]} h_j^{l_j(0)}} \right)^{l_i(0)^{-1}}$$
where $l_i(\cdot)$ is the $i - th$ Lagrange polynomial basis.

5. For $i \in [n]$, generates puzzles with corresponding range proofs as

$$r_i \leftarrow \{0,1\}^\lambda, Z_i \leftarrow \mathrm{LHTLP.PGen}(pp, \sigma_i; r_i)$$

$$\pi_{range,i} \leftarrow \mathrm{ZKprove}(crs_{range}, (Z_i, 0, 2^\lambda, T), (\sigma_i, r_i)).$$

6. Computes $I \leftarrow H'(p_k, (h_1, Z_1, \pi_{range,1}), ..., (h_n, Z_n, \pi_{range,n}))$.

7. Outputs $C := (Z_1, ..., Z_n, T)$ and $\pi := (\{h_i, \pi_{range,i}\}_{i\in[n]}, I, \{\sigma_i, r_i\}_{i\in I})$.

Verification:

1. Takes (crs, p_k, m, C, π) as the input.

2. Parses $C := (Z_1, ..., Z_n, T), \pi := (\{h_i, \pi_{range,i}\}_{i\in[n]}, I, \{\sigma_i, r_i\}_{i\in I})$ and $crs := (crs_{range}, pp)$.

3. If any of the following conditions is satisfied, outputs 0; else, outputs 1:

   (a) There exists some $j \notin I$ such that $\prod_{i\in I} {h_i}^{l_i(0)} \cdot {h_j}^{l_j(0)} \neq p_k$.

   (b) There exists some $i \in [n]$ such that $\mathrm{ZKverify}(crs_{range}, (Z_i, 0, 2^\lambda, T), \pi_{range,i}) \neq 1$.

   (c) There exists some $i \in I$ such that $Z_i \neq \mathrm{LHTLP.PGen}(pp, \sigma_i; r_i)$ or $e(g_0, \sigma_i) \neq e(h_i, H(m))$.

   (d) $I \neq H'(p_k, (h_1, Z_1, \pi_{range,1}), ..., (h_n, Z_n, \pi_{range,n}))$.

Open Phase:

Outputs $(\sigma, \{r_i\}_{i\in[n]})$.

Force Open Phase:

1. Takes as input $C := (Z_1, ..., Z_n, T)$.

2. Runs $\sigma_i \leftarrow \mathrm{LHTLP.PSolve}(pp, Z_i)$ for $i \in [n]$ to obtain all signatures. Since $t-1$ puzzles are already opened by the committer, this only means that $\mathrm{ForceOp}$ has to solve only $(n - t + 1)$ puzzles.

3. Outputs $\sigma := \prod_{j\in[t]} (\sigma_j)^{l_j(0)}$ where wlog., the first t signatures are valid shares.

4.3.2 VT-Schnorr Signatures

The Schnorr signature scheme is defined over a cyclic group $\mathbb{G}$ of prime order q with generator g and uses a hash function $H : \{0,1\}^* \to \mathbb{Z}_q$. Schnorr construction and VT-Schnorr protocol [30] are as follows:

- $(p_k, s_k) \leftarrow \mathrm{KeyGen}(1^\lambda)$: Chooses $x \leftarrow \mathbb{Z}_q$, set $s_k := x$ and $p_k := g^x$.
- $\sigma \leftarrow \mathrm{Sign}(s_k, m; r)$: Samples a randomness $r \leftarrow \mathbb{Z}_q$ to computes $R := g^r, c := H(g^x, R, m), s := r + cx$ and outputs $\sigma := (R, s)$.
- $0/1 \leftarrow \mathrm{Verify}(p_k, m, \sigma)$: Parses $\sigma := (R, s)$ and then computes $c := H(p_k, R, m)$ and if $g^s = R \cdot {p_k}^c$ outputs 1, otherwise outputs 0.

Setup:

1. Takes 1^λ as input.
2. Runs $\mathrm{ZKsetup}(1^\lambda)$ to generate crs_{range}.
3. Generates the public parameters $pp \leftarrow \mathrm{LHTLP.\,PSetup}(1^\lambda, T)$.
4. Outputs $crs := (crs_{range}, pp)$.

Commit and Prove:

1. Takes (crs, wit) as input.
2. Parses $wit := \sigma = (R, s)$, $crs := (crs_{range}, pp)$, p_k as the Schnorr public key, and m as the message to be signed.
3. For all $i \in [t-1]$ samples a uniform pair $(x_i, k_i) \leftarrow \mathbb{Z}_q$ and set s$h_i := g_i^{x_i}$, $R_i := g^{k_i}$ and $s_i := k_i + cx_i$ where $c = H(p_k, R, m)$.
4. For all $i \in \{t, ..., n\}$ computes

$$s_i = \left(s - \sum_{j \in [t-1]} s_j \cdot l_j(0) \right) \cdot l_i(0)^{-1}, h_i = \left(\frac{p_k}{\prod_{j \in [t-1]} h_j^{l_j(0)}} \right)^{l_i(0)^{-1}},$$

$$R_i = \left(\frac{R}{\prod_{j \in [t-1]} R_j^{l_j(0)}} \right)^{l_i(0)^{-1}},$$

where $l_i(\cdot)$ is the $i-th$ Lagrange polynomial basis.

5. For $i \in [n]$, generates puzzles with corresponding range proofs as ($|\sigma| = \lambda$) is the max number of bits of λ) :

$$r_i \leftarrow \{0,1\}^\lambda, Z_i \leftarrow \text{LHTLP. PGen}(pp, s_i; r_i)$$

$$\pi_{range,i} \leftarrow \text{ZKprove}(crs_{range}, (Z_i, 0, 2^\lambda, T), (s_i, r_i))$$

.

6. Computes $I \leftarrow H'(p_k, R, (h_1, R_1, Z_1, \pi_{range,1}), ..., (h_n, R_n, Z_n, \pi_{range,n}))$.

7. Outputs $C := (R, Z_1, ..., Z_n, T)$ and $\pi := (\{h_i, R_i, \pi_{range,i}\}_{i \in [n]}, I, \{s_i, r_i\}_{i \in I})$.

Verification:

1. Takes (crs, p_k, m, C, π) as input.

2. Parses $C := (R, Z_1, ..., Z_n, T), \pi := (\{h_i, R_i, \pi_{range,i}\}_{i \in [n]}, I, \{s_i, r_i\}_{i \in I})$ and $crs := (crs_{range}, pp)$.

3. If any of the following conditions is satisfied, outputs 0; else, returns 1:

   (a) There exists some $j \notin I$ such that $\prod_{i \in I} {h_i}^{l_i(0)} \cdot {h_j}^{l_j(0)} \neq pk$ or $\prod_{i \in I} {R_i}^{l_i(0)} \cdot {R_j}^{l_j(0)} \neq R$.

   (b) There exists some $i \in [n]$ such that Z Kverify$(crs_{range}, (Z_i, 0, 2^\lambda, T), \pi_{range,i})$ $\neq 1$.

   (c) There exists some $i \in I$ such that $Z_i \neq \text{LHTLP. PGen}(pp, s_i; r_i)$ or $g^{s_i} \neq R_i \cdot h_i^c$.

   (d) $I \neq H'(pk, R, (h_1, R_1, Z_1, \pi_{range,1}), ..., (h_n, R_n, Z_n, \pi_{range,n}))$.

Open Phase:

Outputs $((R, s), \{r_i\}_{i \in [n]})$.

Force Open Phase:

1. Takes $C := (R, Z_1, ..., Z_n, T)$ as input.
2. Runs $s_i \leftarrow \text{LHTLP.}\,\text{PSolve}(pp, Z_i)$ for $i \in [n]$ to obtain all signatures. Since $t-1$ puzzles are already opened by the committer, this only means that ForceOp has to solve only $(n - t + 1)$ puzzles.
3. Outputs $(R, s := \sum_{j\in[t]}(s_j) \cdot l_j(0))$ where wlog., the first t signatures are valid shares.

4.3.3 VT-ECDSA Signatures

ECDSA signature has a non-linear verification, unlike in Schnorr. Consequently, unlike VT-BLS and VT-Schnorr, the public key is not secretly shared in VT-ECDSA.

The ECDSA signatures scheme is defined over an elliptic curve group $\mathbb{G}$ of prime order q with base point(generator) g and uses a hash function $H : \{0,1\}^* \rightarrow \mathbb{Z}_q$. ECDSA construction on VT-ECDSA protocol [30] is as follows:

- $(p_k, s_k) \leftarrow \text{KeyGen}(1^\lambda)$: Chooses $x \leftarrow \mathbb{Z}_q$, set $s_k := x$ and $p_k := g^x$.
- $\sigma \leftarrow \text{Sign}(s_k, m; r)$: Samples an integer $k \leftarrow \mathbb{Z}_{||}$ and computes $c \leftarrow H(m)$. Let $(r_x, r_y) := R = g^k$, then set $r := r_x \mod q$ and $s := (c + rx)/k \mod q$. Output $\sigma := (r, s)$.
- $0/1 \leftarrow \text{Verify}(p_k, m, \sigma)$: Parses $\sigma := (r, s)$ and computes $c := H(m)$ and returns 1 if and only if $(x, y) = (g^c, h^r)^{s^{-1}}$ and $x = r \mod q$. Otherwise, output 0.

Setup:

1. Takes 1^λ as input.
2. Runs $\text{ZKsetup}(1^\lambda)$ to generate crs_{range}.
3. Generates the public parameters $pp \leftarrow \text{LHTLP.}\,\text{PSetup}(1^\lambda, T)$.
4. Outputs $crs := (crs_{range}, pp)$.

Commit and Prove:

1. Takes (crs, wit) as input.
2. Parses $wit := \sigma = (r, s)$, $crs := (crs_{range}, pp)$, p_k as the ECDSA public key, and m as the message to be signed.
3. Defines $R := (x, y) = (g^c, h^r)^{s^{-1}}$ and $B = g^c \cdot h^r$, where $c = H(m)$.
4. For all $i \in [t-1]$ samples a uniform pair $s_i \leftarrow \mathbb{Z}_q$ and set $R_i := B^{s_i}$.
5. For all $i \in \{t, ..., n\}$ computes
$$s_i = \left(s^{-1} - \sum_{j\in[t-1]} s_j \cdot l_j(0)\right) \cdot l_i(0)^{-1}, R_i = \left(\frac{R}{\prod_{j\in[t-1]} R_j^{l_j(0)}}\right)^{l_i(0)^{-1}},$$
where $l_i(\cdot)$ is the $i-th$ Lagrange polynomial basis.
6. For $i \in [n]$, generates puzzles with corresponding range proofs as ($|\sigma| = \lambda$) is the max number of bits of λ) :
$$r_i \leftarrow \{0,1\}^\lambda, Z_i \leftarrow \text{LHTLP. PGen}(pp, s_i; r_i)$$
$$\pi_{range,i} \leftarrow \text{ZKprove}(crs_{range}, (Z_i, 0, 2^\lambda, T), (s_i, r_i).)$$
7. Computes $I \leftarrow H'(p_k, r, R, (R_1, Z_1, \pi_{range,1}), ..., (R_n, Z_n, \pi_{range,n}))$.
8. Outputs $C := (r, R, Z_1, ..., Z_n, T)$ and $\pi := (\{R_i, \pi_{range,i}\}_{i\in[n]}, I, \{s_i, r_i\}_{i\in I})$.

Verification:

1. Takes (crs, p_k, m, C, π) as input.
2. Parses $C := (r, R, Z_1, ..., Z_n, T), \pi := (\{R_i, \pi_{range,i}\}_{i\in[n]}, I, \{s_i, r_i\}_{i\in I})$ and $crs := (crs_{range}, pp)$.
3. If any of the following conditions is satisfied, outputs 0; else, returns 1:
   (a) It holds that $x \neq r \mod q$ where $(x, y) := R$.
   (b) There exists some $j \notin I$ such that $\prod_{i\in I} {R_i}^{l_i(0)} \cdot {R_j}^{l_j(0)} \neq R$.

(c) There exists some $i \in [n]$ such that $\mathrm{ZKverify}(crs_{range}, (Z_i, 0, 2^\lambda, T), \pi_{range,i})$ $\neq 1$.

(d) There exists some $i \in I$ such that $Z_i \neq \mathrm{LHTLP.PGen}(pp, s_i; r_i)$ or $R_i \neq (g^c \cdot h^r)^{s_i}$.

(e) $I \neq H'(pk, r, R, (R_1, Z_1, \pi_{range,1}), ..., (R_n, Z_n, \pi_{range,n}))$.

Open Phase:

Outputs $((r, s), \{r_i\}_{i \in [n]})$.

Force Open Phase:

1. Takes $C := (r, R, Z_1, ..., Z_n, T)$ as input.
2. Runs $s_i \leftarrow \mathrm{LHTLP.PSolve}(pp, Z_i)$ for $i \in [n]$ to obtain all signatures. Since $t-1$ puzzles are already opened by the committer, this only means that $\mathrm{ForceOp}$ has to solve only $(n - t + 1)$ puzzles.
3. Outputs $(r, s := \sum_{j \in [t]}(s_j) \cdot l_j(0))$ where wlog., the first t signatures are valid shares.

CHAPTER 5

COMPARISON AND ANALYSIS

5.1 Time-Lock Puzzles

- Chapter 3 introduced three different structures of time-lock puzzles based on mathematical concepts. The initial structure discussed was the repeated squaring-based time lock puzzles developed by RSW. We also mentioned about its possible application scenarios. The other presented structures are constructed in the light of RSW's time-lock puzzles, making it the pioneer among all the other structures.
- About homomorphic time-lock puzzles, we introduced two notable use cases in detail: e-voting scenarios and multi-party contract signing [20]. Furthermore, LHTP is also relevant for use in sealed bid auctions on blockchains and multi-party coin-flipping scenarios. In addition, homomorphic time-lock puzzles structure verifiable timed signatures in 4.3.
- In a demonstration utilizing an e-voting scenario, we presented a visualization, as depicted in Figure 3.1, which suggests that homomorphic time-lock puzzles offer **better efficiency** compared to their classical counterparts. While the polynomial time complexity of the setup and generation phases are equivalent, at $poly(\lambda, log(T))$, the solving phase for classical puzzles incurs a cost of $poly(\lambda, T)$. Conversely, the homomorphic variant requires only $poly(\lambda, |C|)$ for the same task.
- We also worked on the computational complexity of RSW's time-lock puzzles, LHTLP and MTHLP, which can be seen in the 5.2 in detailed.

Table 5.1: Table of Time-lock Puzzles

Schemes	Security	Proposed Applications
RSW [25]	IFP Sequential Squaring	Sealed-bid auctions Fair contract signing Zero-knowledge arguments Non-malleable commitments
LHTLP [20]	Sequential Squaring DDH over $\mathbb{J}_N$ DCR over $\mathbb{Z}^*_{N^2}$	E-Voting Multiparty Coin Flipping
MHTLP [20]	Sequential Squaring DDH over $\mathbb{J}_N$	Multiparty Contract Signing
FHTLP [20]	Sub-exponentially secure iO Sub-exponentially secure OWF	E-Voting Multiparty coin flipping Sealed-bid auctions

Table 5.2: Comparison of time-lock puzzles

Schemes	PSetup	PGen	PSolve	PEval
RSW [25]	3 $Mult_{\mathbb{Z}}$	1 Enc 1 $Exp_{\mathbb{Z}_N}$ 1 $Exp_{\mathbb{Z}_{\varphi(N)}}$ 1 $Add_{\mathbb{Z}_N}$	1 $Exp_{\mathbb{Z}_N}$	
LHTLP [20]	2 $Mult_{\mathbb{Z}}$ 1 $Mult_{\mathbb{Z}_N}$ 1 $Exp_{\mathbb{Z}_{\varphi(N)/2}}$ 1 $Exp_{\mathbb{Z}}$ 1 $UniSamp_{\mathbb{Z}^*_N}$	1 $Exp_{\mathbb{Z}_N}$ 3 $Exp_{\mathbb{Z}_{N^2}}$ 2 $Mult_{\mathbb{Z}_{N^2}}$ 1 $UniSamp_{\mathbb{Z}_{N^2}}$	1 $Exp_{\mathbb{Z}_N}$ 1 $Exp_{\mathbb{Z}_{N^2}}$ 1 $Div_{\mathbb{Z}_N}$ 1 $Div_{\mathbb{Z}_{N^2}}$ 1 $Add_{\mathbb{Z}_N}$	$(n-1)$ $Mult_{\mathbb{Z}_N}$ $(n-1)$ $Mult_{\mathbb{Z}_{N^2}}$
MHTLP [20]	2 $Mult_{\mathbb{Z}}$ 1 $Mult_{\mathbb{Z}_N}$ 1 $Exp_{\mathbb{Z}_{\varphi(N)/2}}$ 1 $Exp_{\mathbb{Z}}$ 1 $UniSamp_{\mathbb{Z}^*_N}$	2 $Exp_{\mathbb{Z}_N}$ 1 $Mult_{\mathbb{Z}_N}$ 1 $UniSamp_{\mathbb{Z}_{N^2}}$	1 $Div_{\mathbb{Z}}$ 1 $Exp_{\mathbb{Z}_N}$	$2(n-1)$ $Mult_{\mathbb{Z}_N}$

Exp_A: Exponentiation in group A, where $A \in \{\mathbb{Z}, \mathbb{Z}_N, \mathbb{Z}_{N^2}, \mathbb{Z}_{\varphi(N)/2}, \mathbb{Z}_{\varphi(N)}\}$.
Div_A: Divison in group A, where $A \in \{\mathbb{Z}, \mathbb{Z}_{N^2}\}$.
$Mult_A$: Multiplication in group A, where $A \in \{\mathbb{Z}, \mathbb{Z}_N, \mathbb{Z}_{N^2}, \mathbb{Z}_{\varphi(N)/2}\}$.
Add_A: Addition in group A, where $A \in \{\mathbb{Z}_N\}$.
Enc: A suitable encryption algorithm such as RC5 or RSA.
$UniSamp_A$: Uniform sampling from the group A, where $A \in \{\mathbb{Z}_{N^2}, \mathbb{Z}^*_N\}$

5.2 Timed Commitments and Timed Signature Schemes

- In Chapter 4, we began by discussing timed signatures and timed commitment schemes developed by Boneh and Naor. Their timed-commitment scheme has significant use in contract signing, and the authors presented a strongly fair protocol for this purpose in their publication [7]. They present a two-party protocol that allows exchanging RSA, Rabin, or Fiat-Shamir signatures. Also, other applications, including honesty-preserving auctions and collective coin-flipping, are discussed.
- Garay and Jakobsson's structure 4.2 was an extension of Boneh and Naor's, and their main contribution and the application was the use of our derived time-lines for a robust release of standard signatures such as RSA, Schnorr, and DSA [15].

  In Boneh and Naor's schemes, expensive proofs are used to verify the correctness of time-lines for each commitment. It requires repeating many times a protocol with k proofs of equality of discrete logs ($T = 2^k$). Nevertheless, in Garay and Jakobsson's scheme, they perform a simpler version of proof of correctness in the setup phase, which does not repeat. Once the setup is performed and the master time-line established, they only perform two such equality of discrete log proofs for each time-line reuse. Despite the open and forced open phases having similar complexity in both schemes, when we consider the setup phase, Garay and Jakobsson's scheme provides **better efficiency**.
- Chapter 4.3 introduced the initial verifiable timed signatures on homomorphic time-lock puzzles. Thyagarajan et al. exhibit various applications for VTS [30]. They demonstrate how VTS can serve as the fundamental cryptographic component for creating payment channel networks [13] [23] that provide better on-chain unlinkability for users participating in transactions, enabling multi-party signing of transactions for cryptocurrencies without any on-chain concept of time, and establishing a cryptocurrency-based equitable multi-party computation protocol. They construct VT-BLS, VT-Schnorr, and VT-ECDSA by exploiting the group structure of the classical ones and applying threshold secret-sharing with

a cut-and-choose type of argument. Homomorphic time lock puzzles provide efficiency gains, as we explained while arguing the efficiency differences between TLP's and HTLP's. The authors implemented VT-BLS, VT-ECDSA, and VT-Schnorr, then presented and compared the cost of commit-prove and verification steps and defended the idea that these structures can be used practically in real-life applications. Also, they noted that the implementations could be significantly improved using concurrency and other efficient data structures. Future work may be targeted on this subject.

- In 2022, Thyagarajan et al. [29] presented a verifiable timed linkable ring signature version of VTS for Monero. It is an important work to show how we can use timed signatures in one of the most privacy-providing blockchains.
- We worked detailed on computational complexities of Thyagarajan et al. [30]'s verifiable timed signatures and provided the table 5.3. It can be observed that the costs of setup phases are equivalent for each verifiable timed signature scheme, and the forced open phase takes same amount of time for VT-Schnorr and VT-ECDSA signatures. But for VT-BLS, we have additional t exponentiations. Despite this, the signature construction phase of VT-BLS offers the least computational cost.

Table 5.3: Computational complexities of verifiable timed signatures

Schemes	Signature Construction	Setup	Commit and Prove Phase	Verification Phase	ForcedOpen Phase
VT-BLS [25]	1 $Exp_{\mathbb{G}_0}$ 1 $Exp_{\mathbb{G}_1}$ 1 $Hash$ 1 $Pairing$	1 ZKSetup 1 LSetup 1 Div	$2(n+t-1)$ Exp $2(t-2)$ $Mult$ 2 Div $(n-t+1)$ Inv 1 $Hash$ n LGen n ZKProve $(t-1)$ $Sampling$ $(n-t+1)$ $Eval$	$(t-1)$ Exp $(3/2)(t-1)(t-2)$ $Mult$ 1 $Hash$ $(t-1)$ LGen n ZKVerify $(t-1)$ $Pairing$	t Exp $(t-1)$ $Mult$ $(n-t+1)$ LSolve t $Eval$
VT-Schnorr [20]	4 Exp 2 $Mult$ 2 $Hash$ 1 Add	1 ZKSetup 1 LSetup 1 Div	$2(n+t-2)$ Exp $(3t+n-5)$ $Mult$ 2 Div $(n-t+1)$ Inv 1 $Hash$ n LGen n ZKProve $(t-1)$ $Sampling$ $(n-t+1)$ $Eval$ $(2t-3)$ Add	$4(t-1)$ Exp $(t-1)(3t-5)$ $Mult$ 1 $Hash$ $(t-1)$ LGen n ZKVerify	t $Mult$ $(n-t+1)$ LSolve t $Eval$ $(t-1)$ Add
VT-ECDSA [20]	5 Exp 1 $Mult$ 1 $Mult_{\mathbb{Z}_q}$ 1 Inv 1 $Div_{\mathbb{Z}_q}$ 2 $Hash$ 1 $Add_{\mathbb{Z}_q}$	1 ZKSetup 1 LSetup 1 Div	$(n+t-1)$ Exp $(n+t-2)$ $Mult$ 1 Div $(n-t+2)$ Inv 1 $Hash$ n LGen n ZKProve $(t-1)$ $Sampling$ $(n-t+1)$ $Eval$ $(t-1)$ Add	$2t$ Exp $(t-1)(3t-4)/2$ $Mult$ 1 $Hash$ $(t-1)$ LGen n ZKVerify	t $Mult$ $(n-t+1)$ LSolve t Eval $(t-1)$ Add

Exp_A: Exponentiation in group A where $A \in \{\mathbb{G}, \mathbb{G}_0, \mathbb{G}_1\}$.
$Mult_A$: Multiplication in group A where $A \in \{\mathbb{G}, \mathbb{Z}_q\}$.
Div_A: Division in group A where $A \in \{\mathbb{G}, \mathbb{Z}_q\}$.
Inv: Inversion in group $\mathbb{G}$.
Add: Addition operation in group A where $A \in \{\mathbb{G}, \mathbb{Z}_q\}$.
$Pairing$: Bilinear pairing operation.
$Hash$: Hash functions in the schemes.
$Sampling$: Uniform sampling operation in group $\mathbb{Z}_q$.
ZKSetup, ZKVerify, ZKProve: The zero-knowledge algorithms that we used in 4.3.
LSetup, LGen, LSolve: Linearly homomorphic time-lock puzzle algorithms in 3.3.1 and 5.2.
$Eval$: Lagrange polynomial basis evaluation computations.

CHAPTER 6

CONCLUSION

So far, our investigation has focused on the prominent cryptographic structures and protocols of timed cryptography with the motivation of sending information to the future. This journey started with a brilliant question posed by cyberpunk and subsequently led to the development of sophisticated and practical cryptographic frameworks.

In the scope of this thesis, firstly, a brief overview of the mathematical background of various commonly used structures is given to the readers. Then the focus shifts to exploring timed cryptography, starting with the concept of time-lock puzzles. The chapter discusses three constructions based on the same idea, and we highlight their similarities and differences. The next chapter uses the same methodology to examine three different timed commitment and timed signature schemes. It is important to note that the literature presented in this thesis does not contain entirely distinct structures but rather builds upon each other chronologically. In the chapter dedicated to comparison and analysis, we concentrate on the possible application scenarios of the protocols under investigation. Additionally, we discussed the key aspects of the efficiency differences among the given constructions. The utilization of time-lock puzzles and timed signatures have a significant potential to be integrated into widely used systems such as e-voting, contract signing protocols, and blockchain areas.

As future work,

- There is potential to develop new methods for timed commitments and signatures using different primitives of cryptography. The security of existing structures relies on classic hard problems, but we know these assumptions are vulnerable to post-quantum computers. The use of lattice basis reduction algorithms, which can be executed sequentially [7], may be incorporated into timed cryptography.

- The complexity of the time-lock puzzles and timed signature schemes can be analyzed by implementing them, and new cryptographic primitives can be integrated into the structures in order to offer more efficient schemes. For example, in Thyagarajan et al.'s work [29], the curves used for VT-BLS needed to be optimized, leading to faster computations. An optimization work for BLS can lead to observing the VT signature's potential.
- Timed cryptography has a relatively old history, but its practical implementations have only been recently developed. It suggests the potential for creating novel applications through further research in this field.

REFERENCES

[1] W. Banasik, S. Dziembowski, and D. Malinowski, Efficient zero-knowledge contingent payments in cryptocurrencies without scripts, volume 9879, pp. 261–280, 09 2016, ISBN 978-3-319-45740-6.

[2] F. Bao, An efficient verifiable encryption scheme for encryption of discrete logarithms, in J.-J. Quisquater and B. Schneier, editors, *Smart Card Research and Applications*, pp. 213–220, Springer Berlin Heidelberg, Berlin, Heidelberg, 2000, ISBN 978-3-540-44534-0.

[3] N. Bitansky, S. Goldwasser, A. Jain, O. Paneth, V. Vaikuntanathan, and B. Waters, Time-lock puzzles from randomized encodings, in *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, ITCS '16, p. 345–356, Association for Computing Machinery, New York, NY, USA, 2016, ISBN 9781450340571.

[4] P. E. Black, Big-o notation, in Dictionary of Algorithms and Data Structures [online], Paul E. Black, ed. 6 September 2019. (accessed TODAY) Available from: https://www.nist.gov/dads/HTML/bigOnotation.html, 1978.

[5] L. Blum, M. Blum, and M. Shub, A simple unpredictable pseudo random number generator, SIAM J. Comput., 15(2), p. 364–383, may 1986, ISSN 0097-5397.

[6] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, Aggregate and verifiably encrypted signatures from bilinear maps, in *Proceedings of the 22nd International Conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'03, p. 416–432, Springer-Verlag, Berlin, Heidelberg, 2003, ISBN 3540140395.

[7] D. Boneh and M. Naor, Timed commitments, in M. Bellare, editor, *Advances in Cryptology — CRYPTO 2000*, pp. 236–254, Springer Berlin Heidelberg, Berlin, Heidelberg, 2000, ISBN 978-3-540-44598-2.

[8] D. Chaum, J.-H. Evertse, and J. van de Graaf, An improved protocol for demonstrating possession of discrete logarithms and some generalizations, in D. Chaum and W. L. Price, editors, *Advances in Cryptology — EUROCRYPT' 87*, pp. 127–141, Springer Berlin Heidelberg, Berlin, Heidelberg, 1988, ISBN 978-3-540-39118-0.

[9] D. Chaum and T. P. Pedersen, Wallet databases with observers, in *Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings*, volume 740 of *Lecture Notes in Computer Science*, pp. 89–105, Springer, 1992.

[10] G. Couteau, T. Peters, and D. Pointcheval, Encryption switching protocols, in *Proceedings, Part I, of the 36th Annual International Cryptology Conference on Advances in Cryptology — CRYPTO 2016 - Volume 9814*, p. 308–338, Springer-Verlag, Berlin, Heidelberg, 2016, ISBN 9783662530177.

[11] I. Damgård, Practical and provably secure release of a secret and exchange of signatures, J. Cryptology, 8, pp. 201–222, 1995.

[12] C. Dwork and M. Naor, Zaps and their applications, in *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pp. 283–293, Nov 2000, ISSN 0272-5428.

[13] C. Egger, P. Moreno-Sanchez, and M. Maffei, Atomic multi-channel updates with constant collateral in bitcoin-compatible payment-channel networks, in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, p. 801–815, Association for Computing Machinery, New York, NY, USA, 2019, ISBN 9781450367479.

[14] T. ElGamal, A public key cryptosystem and a signature scheme based on discrete logarithms, in G. R. Blakley and D. Chaum, editors, *Advances in Cryptology*, pp. 10–18, Springer Berlin Heidelberg, Berlin, Heidelberg, 1985, ISBN 978-3-540-39568-3.

[15] J. A. Garay and M. Jakobsson, Timed release of standard digital signatures, in M. Blaze, editor, *Financial Cryptography*, pp. 168–182, Springer Berlin Heidelberg, Berlin, Heidelberg, 2003, ISBN 978-3-540-36504-4.

[16] J. A. Garay and C. Pomerance, Timed fair exchange of standard signatures, in R. N. Wright, editor, *Financial Cryptography*, pp. 190–207, Springer Berlin Heidelberg, Berlin, Heidelberg, 2003, ISBN 978-3-540-45126-6.

[17] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters, Candidate indistinguishability obfuscation and functional encryption for all circuits, in *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pp. 40–49, Oct 2013, ISSN 0272-5428.

[18] S. Hohenberger and B. Waters, Synchronized aggregate signatures from the rsa assumption, Jesper Buus Nielsen and Vincent Rijmen, editors, EUROCRYPT 2018, Part II, volume 10821 of LNCS. Springer, Heidelberg, p. 197–229, May 2018.

[19] H. Lin, R. Pass, and P. Soni, Two-round and non-interactive concurrent non-malleable commitments from time-lock puzzles, in *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 576–587, Oct 2017, ISSN 0272-5428.

[20] G. Malavolta and S. Thyagarajan, *Homomorphic Time-Lock Puzzles and Applications*, pp. 620–649, 08 2019, ISBN 978-3-030-26947-0.

[21] T. C. May, Timed-release crypto, http://www.hks.net/cpunks/cpunks-0/1460.html., February 1993.

[22] R. C. Merkle, Secure communications over insecure channels, Commun. ACM, 21(4), p. 294–299, apr 1978, ISSN 0001-0782.

[23] J. Poon and T. Dryja, The bitcoin lightning network: Scalable off-chain instant payments, 2016.

[24] R. L. Rivest, A. Shamir, and L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, Commun. ACM, 21(2), p. 120–126, feb 1978, ISSN 0001-0782.

[25] R. L. Rivest, A. Shamir, and D. A. Wagner, Time-lock puzzles and timed-release crypto, Technical report, USA, 1996.

[26] A. D. Santis, S. Micali, and G. Persiano, Non-interactive zero-knowledge proof systems, in *A Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology*, CRYPTO '87, p. 52–72, Springer-Verlag, Berlin, Heidelberg, 1987, ISBN 3540187960.

[27] C. P. Schnorr, Efficient signature generation by smart cards, J. Cryptol., 4(3), p. 161–174, jan 1991, ISSN 0933-2790.

[28] A. Shamir, How to share a secret, Commun. ACM, 22(11), p. 612–613, nov 1979, ISSN 0001-0782.

[29] S. A. Thyagarajan, G. Malavolta, F. Schmid, and D. Schröder, Verifiable timed linkable ring signatures fornbsp;scalable payments fornbsp;monero, in *Computer Security – ESORICS 2022: 27th European Symposium on Research in Computer Security, Copenhagen, Denmark, September 26–30, 2022, Proceedings, Part II*, p. 467–486, Springer-Verlag, Berlin, Heidelberg, 2022, ISBN 978-3-031-17145-1.

[30] S. A. K. Thyagarajan, A. Bhat, G. Malavolta, N. Döttling, A. Kate, and D. Schröder, Verifiable timed signatures made practical, in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, p. 1733–1750, Association for Computing Machinery, New York, NY, USA, 2020, ISBN 9781450370899.