The Role of Time in Cryptography: Play, Fast Forward, and Speed Up

by

Ahmed Zawia

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2024

© Ahmed Zawia 2024

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Reihaneh Safavi-Naini Professor, Dept. of Computer Science, University of Calgary
Supervisor(s):	M. Anwar Hasan Professor, Dept. of Electrical and Computer Engineering, University of Waterloo
Internal-External Member:	Douglas Stebila Associate Professor, Dept. of Combinatorics and Optimization, University of Waterloo
Internal Member:	Vijay Ganesh Professor, Dept. of Computer Science, Georgia Institute of Technology
Internal Member:	Sebastian Fischmeister Professor, Dept. of Electrical and Computer Engineering, University of Waterloo

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

The primary objectives of the thesis work are to conduct in-depth research in the field of delay-based cryptography and to improve the efficiency of evaluating ideal class group action, which serves as a building block in this area. Delay-based cryptographic primitives, such as time-lock puzzles and verifiable delay functions, play a crucial role in many cryptographic protocols by ensuring that certain computations cannot be completed faster than a predetermined time, even in parallel. Our study of delay-based cryptography aims to formalize and implement new delay-based primitives that use time-delay functions to improve the security and functionality of various applications.

We start by introducing a class of *trapdoor* verifiable delay functions (trapdoor-VDFs), an extended notion of verifiable delay functions, and providing a formal definition of their security properties. Trapdoor-VDFs feature a large ensemble of trapdoors, allowing a client to randomly select a private trapdoor to generate a unique challenge. Without the trapdoor, a server must perform a predetermined number of sequential steps to solve the challenge, while any client can efficiently verify the server's solution without knowing the trapdoor. We also propose a construction of trapdoor-VDFs based on large smooth-degree supersingular isogenies with bilinear pairings for efficient public verification. Our construction is useful for time-sensitive applications that require guaranteed timely release with public verification, such as sealed voting and front-running attack prevention.

Then, we introduce the notion of *ChronoCloak*, an integrated solution designed to prevent premature access to digital content while preserving recipients' privacy. Specifically, ChronoCloak enables a sender to transmit a collection of secret data through a puzzle that requires a predetermined amount of time to be solved. Once the puzzle is solved, the receiver can only extract the chosen subset of the sender's secret data without the sender knowing which subset was chosen. With public verifiability, the puzzle-solving process can be securely outsourced while ensuring that only the receiver can extract the chosen secret subset. Additionally, we introduce an ideal functionality for ChronoCloak and propose a generic protocol that implements ChronoCloak's functionality in the random oracle model.

Finally, we propose a new *batching strategy* for computing a set of *group* actions within the Commutative Supersingular Isogeny Diffie-Hellman (CSIDH) framework, which is a crucial component for developing post-quantum cryptographic schemes like zero-knowledge proofs and signature schemes. Our work presents two variants of the batching strategy, each of which is suited to different security requirements. The first variant is for computing a *public* set of CSIDH group actions and aims to reduce the number of finite field arithmetic operations required. Our results show a roughly up to 14% reduction in computational cost. Toward our second variant, we first propose an efficient new constant-time

CSIDH algorithm for evaluating a *private* group action based on state-of-the-art batching techniques from CTIDH. We then introduce a method for computing a private set of CSIDH group actions, which runs in constant time while reducing the overall number of finite field arithmetic operations needed. Combined with our constant-time algorithm, it shows a reduction in computational cost of roughly up to 8% compared to the current state-of-the-art constant-time CSIDH. Interestingly, the constant-time algorithm alone reduces computations by about 4%. Finally, we present a proof-of-concept implementation, showcasing the results of our work.

Acknowledgements

First and foremost, I am very grateful for the support of my supervisor Prof. Anwar Hasan. His guidance and feedback improved shaping my thesis as well as my growth as a researcher. It has been such a privilege to be under his supervision and learn from his significant expertise in the field.

I am also sincerely grateful to the members of my examining committee: Professors Reihaneh Safavi-Naini, Douglas Stebila, Vijay Ganesh and Sebastian Fischmeister. I appreciate their time and consideration in reviewing my work. As well, I would like to extend my heartfelt thanks to Professor Gordon Agnew, who served on my earlier examination panels. His insights and support were invaluable to me throughout my journey.

I wish to extend my appreciation to the anonymous reviewers of the publications that have resulted from this thesis so far. The reviewers' constructive critiques especially some from the 27th Information Security Conference, were very helpful in improving my research.

I want to express my deepest gratitude to my family for their unwavering support and encouragement throughout this journey. Their belief in me has been a constant source of motivation and strength.

Special thanks go to my wife, whose patience has been the cornerstone of this endeavor. She has been a constant source of strength and I could not have reached this milestone without her by my side.

Lastly, I am grateful to my colleagues, friends, University of Waterloo staff, and everyone who has supported me throughout my PhD journey, whether it be through academic guidance or personal encouragement. Your contributions have made this achievement possible.

Table of Contents

Examining Committee ii

Author's Declaration iii

Abstract iv

Acknowledgements vi

List of Figures xi

List of Algorithms xiii

List of Tables xiv

List of Abbreviations xv

List of Notations xvi

1 Introduction 1

1.1 Overview 1

1.2 Motivation 2

1.3 Contributions 3

1.4 Thesis Organization 4

2 Preliminaries **5**
2.1 Basics of Modern Cryptography . . . 5
2.1.1 Hash Functions and Random Oracles . . . 5
2.1.2 Secure Encryption Scheme . . . 6
2.1.3 Oblivious Transfer Protocol . . . 7
2.2 Security Models: Game-Based vs. UC Security . . . 8
2.2.1 Game-Based Security . . . 8
2.2.2 UC Security . . . 9
2.2.3 Choosing a Security Model and Modular Approach . . . 9
2.3 Mathematical Background . . . 9
2.3.1 Groups, Rings, Ideals, and Fields . . . 10
2.3.2 Ideal Class Group . . . 12
2.3.3 Elliptic Curves . . . 12
2.3.4 Bilinear Pairing . . . 18
2.3.5 Isogenous Curves . . . 19
2.3.6 Isogeny and Pairing . . . 22

3 Beyond the Clock: The Role of Time in Cryptography **23**
3.1 Introduction . . . 23
3.2 Main Building Blocks and Theoretical Foundations . . . 24
3.2.1 Constructions of Long Sequential Functions . . . 24
3.3 Notions in Delay-Based Cryptography . . . 28
3.3.1 Time-Lock Puzzles and Timed-Release Encryption . . . 28
3.3.2 Proof of Sequential Works . . . 30
3.3.3 Delay-Based Encryption . . . 31
3.3.4 Verifiable Delay Functions . . . 31
3.3.5 Discussion . . . 34
3.4 Proofs of Correct Computation . . . 35

3.4.1 Soundness in Unknown Order Group . . . 36
3.4.2 Soundness in Isogeny Based Construction . . . 46
3.5 Special Classes of Delay-Based Cryptography . . . 49
3.6 Chapter Summary . . . 49

4 A New Class of Trapdoor Verifiable Delay Functions 51
4.1 Introduction . . . 51
4.2 Proposed Trapdoor-VDFs . . . 53
4.3 Design Rationale . . . 57
4.3.1 Construction Elements . . . 58
4.3.2 The Graph in Use and the Single Step of Computation . . . 59
4.3.3 The Scheme Group Structure and Its Public Verification . . . 60
4.3.4 An Instance of the Trapdoor-VDF . . . 62
4.4 Security . . . 64
4.5 Chapter Summary . . . 66

5 ChronoCloak: An integrated Solution for Mitigating Premature Disclosure in Oblivious Digital Dissemination 68
5.1 Introduction . . . 69
5.1.1 Related Work . . . 70
5.2 Chapter Background . . . 71
5.2.1 UC Security Model . . . 72
5.2.2 Related Functionalities . . . 76
5.3 Underlying UC Framework and Operation Unit . . . 77
5.3.1 Modeling Operation Unit . . . 80
5.3.2 Non-Committing Technique . . . 81
5.4 ChronoCloak . . . 82
5.4.1 Definition of ChronoCloak . . . 82

5.4.2 Extended Definition of ChronoCloak . . . 85
5.4.3 The Ideal Functionality of ChronoCloak . . . 86
5.5 Generic ChronoCloak . . . 87
5.5.1 The Protocol . . . 88
5.5.2 Security . . . 89
5.6 ChronoCloak from Delayed Oblivious Transfer . . . 93
5.7 Chapter Summary . . . 99

6 Streamlining CSIDH: Cost-Effective Strategies for Group Actions Evaluation 100
6.1 Introduction . . . 101
6.2 Background and Preliminaries . . . 102
6.2.1 CSIDH . . . 103
6.2.2 Constant-Time Implementation of CSIDH . . . 104
6.3 Public Action Set Evaluation . . . 107
6.4 Private Action Set Evaluation . . . 111
6.5 Chapter Summary . . . 120

7 Concluding Remarks and Future Works 121
7.1 Concluding Remarks . . . 121
7.2 Future Work . . . 123

References 124

List of Figures

3.1 i^{th} iteration of Pietrzak's interactive succinct argument. 38
3.2 A tree representation of Pietrzak's proving protocol [128]. 38
3.3 A tree representation of Ephraim et al.'s proving protocol [69]. 41
3.4 Wesolowski's interactive succinct argument [156]. 42
3.5 i^{th} iteration of modified Pietrzak-like interactive succinct argument (mPoE). 44
3.6 De Feo et al. succinct argument for isogeny graphs over $\mathbb{F}_p$ [72]. 46
3.7 De Feo et al. succinct argument for isogeny graphs over $\mathbb{F}_{p^2}$ [72]. 46
3.8 Delay encryption [33]. 48
3.9 Shani's construction [137]. 48

4.1 The structure of the proposed trapdoor-VDF proof system. 62
4.2 The proposed trapdoor-VDF instance. 63

5.1 1-out-of-2 oblivious transfer ideal functionality (F_{OT}). 76
5.2 The random oracle ideal functionality (F_{RO}). 77
5.3 The functionality of $\mathsf{F}_{\mathsf{clk}}$-global ticker from [16]. 79
5.4 The ideal functionality of the operation unit, F_{OU}. 80
5.5 The ideal functionality of ChronoCloak, F_{CC}. 86
5.6 Proposed ChronoCloak protocol $\Gamma_{\mathsf{F}_{\mathsf{CC}}}$ implementing F_{CC}. 88
5.7 A simulation $\tilde{\mathcal{A}}'_S$ for a corrupted sender and an honest receiver executing $\Gamma_{\mathsf{F}_{\mathsf{CC}}}$. 90
5.8 A simulation $\tilde{\mathcal{A}}'_R$ for an honest sender and a corrupted receiver executing $\Gamma_{\mathsf{F}_{\mathsf{CC}}}$. 91

6.1 An example implementation for identifying public action systems. 109
6.2 Our results for evaluating a CSIDH-512 execution sets ranging from 2 to 58. 112
6.3 The distribution of Batch 11 in the CTIDH-512 key space. 116

List of Algorithms

1 CSIDH's original algorithm [41] for class group action 104
2 CTIDH's original algorithm for constant-time group action. 107
3 New constant-time group action. 113
4 Torsion point computation, GetTorsionPoint. 114
5 Initial round torsion point computation. 118
6 GetTorsionPoint2 (revised from Algorithm 4) 119

List of Tables

3.1 Summary of delay-based cryptographic primitives. 35

6.1 The cost for computing a CSIDH-512 public execution set with $c = 50$. . . 111

6.2 The computational cost for computing two action systems of size 2. 120

List of Abbreviations

CSIDH Commutative Supersingular Isogeny Diffie-Hellman 4, 27, 71, 100–108, 110–112, 117, 118, 120, 122

CTIDH Constant-Time CSIDH 105, 106, 110–112, 115, 117, 118

DARK Diophantine Argument of Knowledge 43

HTLP Homomorphic Time-Lock Puzzle 29, 30

PoSW Proof of Sequential Work 2, 30, 31, 34, 35, 51, 52

SIDH Supersingular Isogeny Diffie–Hellman 27, 71

TLE Time-Lock Encryption 29

TLP Time-Lock Puzzle 2, 28–30, 32, 34, 35, 48, 51, 52, 70

TRE Timed-Release Encryption 28

VDF Verifiable Delay Function 2, 3, 31–35, 40–42, 44–47, 51–53, 57, 66, 70

List of Notations

- $\lambda \in \mathbb{N}$: Represents the security level of a scheme, where λ is a natural number.
- $\mathsf{Pr}[\mathsf{Ev} : \mathsf{Ev}_1, \mathsf{Ev}_2, \ldots, \mathsf{Ev}_n]$: Denotes the probability of the event Ev occurring after the ordered events $\mathsf{Ev}_1, \mathsf{Ev}_2, \ldots, \mathsf{Ev}_n$.
- $A \parallel B$: Denotes the concatenation of the string representations of A and B.
- $[n]$: Denotes the set $\{1, \ldots, n\}$ for a positive integer n.
- $\mathcal{B}$: Denotes a finite set, with sets represented in calligraphic font.
- $|\mathcal{B}|$: Represents the size of the set $\mathcal{B}$.
- $e \leftarrow_{\$} \mathcal{B}$: Denotes the process of uniformly sampling a random element e from the set $\mathcal{B}$.
- $\boldsymbol{z} \leftarrow_{\$} \mathcal{B}^n$: Denotes the process of sampling the vector $(\boldsymbol{z}_i)_{i \in [n]}$ from a uniform distribution over $\mathcal{B}^n$, where $\boldsymbol{z}_i \leftarrow_{\$} \mathcal{B}$ for all $i \in [n]$.
- $e \leftarrow \mathcal{B}$: Denotes the deterministic choice of an element e from the set $\mathcal{B}$.
- $a \leftarrow_{\$} \mathsf{Alg}$: Represents the process of executing an algorithm Alg with fresh random coins sampled from a uniform distribution, where a is the output.
- $a \leftarrow \mathsf{Alg}$: Denotes the deterministic process of executing an algorithm Alg.
- $\mathsf{f}_1 \circ \mathsf{f}_2(x)$: Denotes the composition of functions such that $\mathsf{f}_1(\mathsf{f}_2(x))$ for a valid input x.
- $\mathsf{Primes}(n)$: Denotes the deterministic function that returns the set of all odd prime numbers less than or equal to n.
- $\left(\frac{*}{*}\right)$: Represents the Legendre symbol.

Chapter 1

Introduction

This thesis studies the field of cryptography where *time* plays a crucial role in security. Our focus is on expanding the applicability of this field by introducing new notions, building on existing ones, and improving the efficiency of one of the few building blocks.

1.1 Overview

With the rapid advancement of digital technologies, the need for modern secure methods has become increasingly important. Cryptographic protocols are the backbone of secure digital communication, ensuring its confidentiality, integrity, and authenticity. To secure data, traditional cryptography uses algorithms and keys, focusing on continuous and immediate security.

However, in certain applications, timing plays an essential role in maintaining security. Delay-based cryptography, or time-locked cryptography, addresses this need by incorporating time delays into cryptographic operations to impose temporal constraints. This approach requires specific cryptographic tasks to take a predetermined amount of time to complete, regardless of the computational resources in use, including parallel execution. Hence, up to some assumptions, even if secret keys or computational resources are available, access to the data is restricted until the required computation is completed (a time delay elapses). This temporal control is important in applications such as time-locked messages, time-based ownership, and randomness beacons, where it helps maintain fairness and resistance to manipulation by preventing premature computations. One practical example is its application in financial systems to mitigate front-running attacks. In such

attacks, malicious actors exploit early access to transaction details to manipulate market outcomes [50, 70]. We can prevent front-running by introducing a time delay that cannot be bypassed, ensuring that sensitive information, like transaction details, can only be accessed after a predetermined period. This guarantees fair access to sensitive information, preventing malicious actors from gaining an unfair advantage.

The first delay-based cryptographic primitive, Time-Lock Puzzle (TLP), was introduced by Rivest, Shamir, and Wagner in 1996 [131]. TLP allows a sender to encrypt a message so that it can only be decrypted after a predetermined period, without relying on third parties or synchronized clocks. TLP in [131] achieves this by employing sequential squaring in a group of unknown order, which inherently resists parallelization. This ensures a fixed decryption time, meaning that solving a TLP's puzzle will take a prescribed amount of time or sequential computational steps.

Since then, other delay-based notions have been developed, including Proof of Sequential Work (PoSW) and Verifiable Delay Function (VDF). PoSW, introduced by Mahmoody et al. [114], requires a prover to perform a sequence of computations to obtain an output that proves that they have invested a certain amount of time and effort. VDFs, proposed by Boneh et al. [25], guarantee not only a prescribed time delay in computing the function's output but also an efficient public verification of its output uniqueness. Among others, each of these primitives serves distinct purposes: TLPs focus on time-restricted message access, VDFs guarantee output uniqueness after a delay, and PoSWs prove sequential computational effort. More on this is provided in Chapter 3.

1.2 Motivation

In many real-world scenarios, such as sealed-bid auctions, secure voting, digital content distribution, and financial transactions, the timing of information release is critical to ensuring fairness and security. In practice, however, adopting delay-based cryptographic techniques present several challenges, including the need to outsource extensive computations in resource-limited environments, difficulties in achieving composability (i.e., securely integrating these primitives into larger cryptographic systems is not intuitive), efficiency, and complexities in integrating user privacy. Although TLPs, PoSWs, and VDFs each offer unique solutions, there are still opportunities to refine these notions and others, extending their applicability over a broader range of practical scenarios. In this thesis, we aim to refine the existing theoretical framework, introduce a new notion that is compact and safely composable into more complex systems, and enhance the efficiency of a building

block in delay-based cryptographic primitives, all with the aim of extending the role of time in secure digital systems.

1.3 Contributions

This thesis has three major contributions to the field of delay-based cryptography. As part of the first contribution, we formalize the notion of trapdoor-VDF's security definition and introduce an approach for constructing a VDF with a large ensemble of distinct shortcuts, known as trapdoors. The delay function involves the sequential evaluation of a large smooth-degree supersingular isogeny, as originally proposed by Feo et al. [72]. The trapdoor ensemble consists of smaller degree isogenies defined over finite field $\mathbb{F}_p$, where p is a prime. For public verification, a bilinear pairing equality serves as proof of the correct evaluation of a secret trapdoor, specifically a randomly chosen isogeny. We have shown that our proposal is secure under trapdoor-VDF security notions. Trapdoor-VDF is ideal for time-sensitive applications that require both public verification and timely release guarantees. It has numerous potential uses, including sealed voting and preventing front-running attacks [50]. Another advantage is allowing the reuse of public parameters in constructions with costly public setups.

As part of the second contribution, we present the notion of ChronoCloak—our integrated solution to premature disclosure of digital content in oblivious distribution. This pertains to a problem urged by the growing use of digital content dissemination, such as games, videos, and magazine articles, where it is imperative to maintain timely access while preserving recipients' privacy. In particular, ChronoCloak allows a sender to transmit a set of secrets through a puzzle, and then, once the puzzle is solved via a lengthy computation, the receiver recovers only a subset of the secrets that are oblivious to the sender. It also enables the receiver to securely outsource the computation (with public verifiability) to a public solver, yet only the intended receiver can retrieve the subset of secrets using the puzzle solution. Additionally, we propose a secure implementation of ChronoCloak within the universal composability framework of [36]. Unlike [15, 112, 157], the ChronoCloak framework takes a different approach that is more practical and efficient, as it is based on an integrated model and does not require a trusted time server.

Our third contribution addresses the practicality of a possible building block for implementing our aforementioned work, namely the evaluation of ideal class groups of imaginary quadratic fields via isogenies. Specifically, we propose a new batching strategy for computing multiple group actions, which is essential for constructing cryptographic schemes like ours, as well as others such as zero-knowledge proofs and signature schemes. We propose

two variants of our strategy. The first variant involves the evaluation of a public action set for which no security restrictions are required. This variant reduces the computational cost of group actions by roughly up to 14% when compared to non-batched action evaluation. The second variant addresses the evaluation of a secret group action set that requires a constant-time implementation to prevent information leakage. In addition, our work introduces a new constant-time implementation of Commutative Supersingular Isogeny Diffie-Hellman (CSIDH). Combined with the second variant, it reduces the computational costs of secret action sets by approximately 8% when compared to individual computation using the state-of-the-art constant-time algorithm. Interestingly, the constant-time implementation alone reduces the computation by about 4%.

1.4 Thesis Organization

The remainder of the thesis is structured as follows:

- Chapter 2 introduces some basics of modern cryptography and offers mathematical notions and background relevant to our research. This includes topics such as supersingular elliptic curves, bilinear pairing, and isogenies.
- Chapter 3 presents essential concepts of delay-based cryptography, covering key notions and special delay-based constructions, as well as the building blocks used in their constructions.
- Chapter 4 introduces our first contribution, a formal definition of the trapdoor verifiable delay function and an approach to constructing a secure instance.
- Chapter 5 introduces our second contribution, the notion of ChronoCloak, an integrated solution that addresses the problem of preventing premature disclosure of digital content in oblivious distribution.
- Chapter 6 presents our third contribution, a new batching strategy for enhancing (streamlining) CSIDH computations.
- Chapter 7 offers some final remarks summarizing our key findings and discusses potential directions for future research.

Chapter 2

Preliminaries

2.1 Basics of Modern Cryptography

In this section, we introduce a selected set of modern cryptographic tools that form the basis of our work.

2.1.1 Hash Functions and Random Oracles

In cryptography, a hash function $\mathsf{H} : \{0,1\}^* \to \mathcal{K}$ is a deterministic function that maps an input of arbitrary length to a fixed (size) predefined output space $\mathcal{K}$. For example, $\mathcal{K}$ can be the set of fixed length binary strings $\{0,1\}^k$ for $k \in \mathbb{Z}_{>0}$. A cryptographic hash function satisfies a set of properties, including preimage resistance (it is computationally infeasible to invert the function, i.e., H behaves as a one-way function), collision resistance (it is computationally infeasible to find two distinct inputs that map to the same output), and second preimage resistance (given an input, it is computationally infeasible to find another input with the same output).

In theoretical analyses, including our work, hash functions are sometimes modeled as random oracles, which is an idealized 'black-box' functionality that responds to any hashing query by returning an independent random output from the predefined output space $\mathcal{K}$. This abstraction simplifies the security proofs for cryptographic protocols; however, it has practical limitations because real-world hash functions are deterministic, so they may deviate from random oracle behavior.

2.1.2 Secure Encryption Scheme

Let $\Gamma_{\mathcal{E}}$:=(KGen, En, De) be an encryption scheme with a message space $\mathcal{M}$, key space $\mathcal{K}$, and ciphertext space $\mathcal{CT}$. We require that $\Gamma_{\mathcal{E}}$ satisfy the following with overwhelming probability:

$$\mathsf{De}_{sm}(\mathsf{En}_{sm}(m)) = m,\ \forall m \in \mathcal{M} \text{ and } \forall sm \in \mathcal{K}.$$

A secure encryption scheme $\Gamma_{\mathcal{E}}$ ensures confidentiality by preventing an adversary from obtaining secret information from ciphertext. In our work, we consider two types of encryption schemes as described below. Each of these schemes is intended to provide specific security guarantees and is used in specific cryptographic contexts.

IND-CPA Secure Encryption Scheme

We begin by presenting the semantic security of $\Gamma_{\mathcal{E}}$, which is formally equivalent to Indistinguishability under Chosen Plaintext Attack (IND-CPA). This property ensures that an adversary cannot distinguish the ciphertext of a chosen message from any ciphertext of a random message.

Definition 1 (IND-CPA secure $\Gamma_{\mathcal{E}}$)**.** *Let $\Gamma_{\mathcal{E}}$:=(KGen, En, De) be an encryption scheme with a message space $\mathcal{M}$, key space $\mathcal{K}$, and ciphertext space $\mathcal{CT}$. Let Env be a PPT adversary that participates in the following game against a challenger:*

- *The challenger generates a secret key $sm \leftarrow_{\$} \mathsf{KGen}(1^{\lambda})$, where $sm \in \mathcal{K}$.*
- *Env outputs the message pair $(m_0, m_1) \in \mathcal{M}^2$.*
- *The challenger selects a random bit $b \in \{0, 1\}$, computes the ciphertext $ct = \mathsf{En}_{sm}(m_b)$, and then sends ct to Env.*
- *Upon receiving ct, Env outputs a guess $b' \in \{0, 1\}$ for b.*

*To win the game, Env's output b' must equal b. The encryption scheme $\Gamma_{\mathcal{E}}$ is said to be **semantically secure encryption scheme** (or IND-CPA secure) if, for all PPT adversaries Env, the advantage*

$$\mathsf{Adv}_{\mathsf{Env}}(\lambda) = \left| \mathsf{Pr}(b' = b) - 1/2 \right|$$

is a negligible function of λ

Non-Committing Encryption Scheme (NCES)

We primarily employ a non-committing encryption scheme, as defined in Definition 2, in Chapter 5 to construct secure generic protocol of ChronoCloak in universal composability framework. Essentially, NCES is a useful tool in the construction of cryptographic protocols within UC framework, as it allows for secure simulation in a malicious adversarial environment. Section 5.3.2 provides an in-depth explanation of the technical details of our applications of NCES.

Definition 2 (Non-Committing Encryption Scheme)**.** *Let* $\Gamma_{\mathcal{E}} :=(\mathsf{KGen}, \mathsf{En}, \mathsf{De})$ *be an encryption scheme with a message space* $\mathcal{M}$*, key space* $\mathcal{K}$*, and ciphertext space* $\mathcal{CT}$*. We say that* $\Gamma_{\mathcal{E}}$ *is a* ***non-committing encryption scheme*** *if it satisfies the following property:*

- *Simulatability and Indistinguishability. For any probabilistic polynomial time (PPT) distinguisher* Env*, there exists a PPT extractor algorithm* Ex *such that*

$$|Pr[\mathsf{Env}(ct, sm) = 1] - Pr[\mathsf{Env}(ct', sm') = 1]|$$

  is a negligible function in λ*, where* $ct = \mathsf{En}_{sm}(m)$*,* $m \in \mathcal{M}$*,* $sm \leftarrow_{\$} \mathsf{KGen}(1^{\lambda})$*,* $ct' \leftarrow_{\$} \mathcal{CT}$*, and* $sm' \leftarrow \mathsf{Ex}_{m}(ct')$*. The output of the distinguisher* Env *is interpreted as 1 for a guess of the real encryption distribution and 0 for a guess of the simulated distribution.*

2.1.3 Oblivious Transfer Protocol

Oblivious Transfer (OT) is a two-party cryptographic protocol that serves as an essential building block for many secure computation protocols. In our work, we consider a 1-out-of-2 OT protocol that allows for the transmission of one of two secret messages obliviously. This protocol involves a trusted setup party, a sender, and a receiver, typically requiring n rounds of interaction. The sender holds a pair of secret messages (m_0, m_1), while the receiver has a choice bit i. Through n rounds of interaction, a secret information is transferred from the sender to the receiver, allowing the receiver to reconstruct the selected message m_i. A secure OT protocol ensures that the receiver's choice i stays *oblivious* from the sender, while the receiver gains no information about the other secret message m_{1-i}.

Definition 3. *An Oblivious Transfer protocol with* $n = 2$ *is a tuple of four algorithms* $\Gamma_{OT} = (\mathsf{OT}_{setup}, \mathsf{OT}_R, \mathsf{OT}_S, \mathsf{OT}_{Dec})$*, defined as follows:*

- $\mathsf{OT}_{setup}(1^\lambda)$: *A randomized algorithm that takes a security parameter* λ *and outputs a common reference string* crs.
- $\mathsf{OT}_R(\mathsf{crs}, i \in \{0,1\})$: *A randomized algorithm that takes the common reference string* crs *and a bit* $i \in \{0,1\}$*; it returns the receiver's commitment* ot_1 *and secret* sk_r.
- $\mathsf{OT}_S(\mathsf{crs}, \mathsf{o}_1, m_0, m_1)$: *A randomized algorithm that takes the common reference string* crs*, the receiver's commitment* o_1*, and a pair of input messages* (m_0, m_1)*; it returns the sender's commitment* o_2.
- $\mathsf{OT}_{Dec}(\mathsf{crs}, \mathsf{o}_2, sk_r)$: *A deterministic algorithm that takes the common reference string* crs*, the sender's commitment* o_2*, and the receiver's secret* sk_r*; it returns a message* m_i.

2.2 Security Models: Game-Based vs. UC Security

Different models and frameworks can be used to formalize and prove the security of cryptographic protocols, including the game-based model and the UC framework. While game-based definitions are often simpler and are commonly used in applied cryptography, UC security offers stronger guarantee, that is, ensuring its security guarantee even when the protocol is combined with other protocols.

2.2.1 Game-Based Security

The security of a cryptographic primitive, in the game-based model, is defined through a game between a challenger and an adversary. This model addresses a single security property at a time, which simplifies security analysis making it more intuitive; two examples are $\Gamma_{\mathcal{E}}$'s properties presented in Definition 1 and 2. In the game, the challenger implements the cryptographic primitive, as predefined. The adversary's objective is to break a security property of the primitive with limited access to some information, such as encryption/decryption oracle queries. In the game, we also specify a success condition for the adversary, such as distinguishing a new encrypted messages. The primitive is considered secure (i.e., it satisfies the security property) if the adversary's probability of success is negligible. In summary, this model provides explicit mathematical definitions of security properties and a structured framework to analyze them. However, this model is stand-alone and does not guarantee that the cryptographic primitive remains secure when combined with other primitives or protocols arbitrarily.

2.2.2 UC Security

Introduced by Canetti [36] in 2001, the UC framework is a strong security model that ensures the security of cryptographic primitives regardless of how they are composed with other primitives. This is because it considers not only the isolated execution of the primitive but also the environment it operates in. However, in general, this broader scope of security adds extra requirements, which makes the security analysis under the UC framework more challenging than in the game-based model. More formally, the UC framework is a simulation-based definition of security. Section 5.2.1 provides a detailed explanation of the UC framework.

2.2.3 Choosing a Security Model and Modular Approach

Generally, the intended use of the cryptographic primitives often determines whether to use game-based or UC security. On one hand, the game-based model is often sufficient for scenarios where a primitive is deployed as a standalone entity. On the other hand, UC security is more appropriate in scenarios where a primitive *constantly* interacts with other entities, such as within a larger system. However, compared to the game-based model, the security analysis under UC tends to be more complex.

In this thesis, we analyze the security of our primitives in the game-based model, as detailed in Chapter 4. In Chapter 5, we consider defining our work in both the UC framework and game-based notions. Then, we propose a generic UC-secure protocol that is constructed using the game-based definition of our primitive. This approach simplifies the analysis by allowing researchers to focus on designing a simplest game-based secure instance of our primitive. Once such an instance is constructed, it can be directly integrated into our generic protocol, resulting in a UC-secure implementation. Using this modular approach, one can establish UC security without constructing a UC-secure protocol from scratch for every instance.

2.3 Mathematical Background

This section presents a brief overview of some relevant mathematical notions and background required for the work presented in this thesis.

2.3.1 Groups, Rings, Ideals, and Fields

Groups. Let G be a group. We denote the subgroup of G generated by an element $g \in G$ as $\langle g \rangle = \{ag : a \in \mathbb{Z}\}$. The order of g is the set size of $\langle g \rangle$. Group G is a cyclic group if $G = \langle g \rangle$ for an element $g \in G$. A cyclic group of order N is isomorphic to an integer additive group modulo N, denoted by $\mathbb{Z}/N\mathbb{Z}$.

Rings. A ring R is an additive abelian group with a multiplication operation that distributes over addition (i.e., $r_1(r_2+r_3) = r_1r_2 + r_1r_3$, and $(r_1+r_2)r_3 = r_1r_3 + r_2r_3$ where $\forall r_i \in \mathsf{R}$). A subring of R is a subset of R that also forms a ring.

Ideals. An R-*ideal* I is a subring of R that satisfies $ri \in \mathsf{I}$ for all $r \in \mathsf{R}$ and $i \in \mathsf{I}$; and I is called *left* R-*ideal* (the same concept applies to *right* R-*ideal* and *two-sided* R-*ideal*). The R-ideal I is non-trivial proper if $\mathsf{I} \neq \{0\}$ and $\mathsf{I} \subsetneq \mathsf{R}$. The R-ideal $\mathfrak{a}$ is a principal if $\mathfrak{a}$ is generated by an element ($a \in \mathsf{R}$), denoted by (a) (i.e., for the left principal R-ideal, we have $\mathfrak{a} = \mathsf{R}a = (a) = \{ra : r \in \mathsf{R}\}$). A proper R-ideal $\mathfrak{a}$ is maximal if there are no R-ideals between $\mathfrak{a}$ and R (i.e., there is no proper R-ideal $\mathfrak{b}$ such that $\mathfrak{a} \subset \mathfrak{b} \subset \mathsf{R}$). For an R-ideal $\mathfrak{a}_1$ and $\mathfrak{a}_2$, a proper R-ideal $\mathfrak{a}$ is a prime ideal if $\mathfrak{a}_1\mathfrak{a}_2 \subseteq \mathfrak{a}$ then $\mathfrak{a}_1 \subseteq \mathfrak{a}$, or $\mathfrak{a}_1 \subseteq \mathfrak{a}$. Note that if $\mathfrak{a}_i = (a_i)$, the statement $\mathfrak{a}_i \subseteq \mathfrak{a}$ is equivalent to $a|a_i$; thus, $\mathfrak{a}_1\mathfrak{a}_2 \subseteq \mathfrak{a}$ means that $a|a_1a_2$ which implies that $a|a_1$ or $a|a_2$ (see [143], Propositions 5.7). The previous statement implies that every maximal ideal is a prime ideal.

Fields. Let R be a ring with multiplicative identity. An element $r_1 \in \mathsf{R}$ is zero divisor if $r_1r_2 = 0_{\mathsf{R}}$ and $r_2 \neq 0_{\mathsf{R}}$. We say R is an integral domain if R is commutative and has no non-zero zero divisor. We say that an element $r \in \mathsf{R}$ is a unit if it has multiplicative inverse. The group of unites is defined by $\mathsf{R}^{\times} := \{r \in \mathsf{R} : \exists e \in \mathsf{R} \text{ s.t., } er = 1_{\mathsf{R}}\}$. We say that R is a division ring if $\mathsf{R} \setminus \{0_{\mathsf{R}}\}$ forms a group under multiplication (i.e., $\mathsf{R} \setminus \{0_{\mathsf{R}}\} = \mathsf{R}^{\times}$). A ring R is a *field*, if R is commutative division ring. The ring $\mathbb{Z}/N\mathbb{Z}$ is a field if N is a prime.

A *finite field* is a field with a finite number of elements. A finite field with characteristic p is denoted by $\mathbb{F}_q$, where q is a prime p power (i.e., $q = p^m$ for a positive integer m). We say that $\mathbb{F}_q$ is a prime field if $q = p$, denoted by $\mathbb{F}_p$. For $m \geq 1$, $\mathbb{F}_q$ is the $\mathbb{F}_p$ extension of degree m such that $\mathbb{F}_p \subseteq \mathbb{F}_q$. An order q finite field $\mathbb{F}_q$ is isomorphic to $\mathbb{F}_p[i]/g(i)$ where g is a monic irreducible polynomial in $\mathbb{F}_p[i]$ of degree m. The multiplicative group of $\mathbb{F}_q$ is defined by $\mathbb{F}_q^*$. We say that the field $\overline{\mathbb{F}}_q$ is the algebraic closure of $\mathbb{F}_q$ if every non-constant $g \in \overline{\mathbb{F}}_q[i]$ has a root in $\overline{\mathbb{F}}_q$ (i.e., algebraically closed). In the case of a finite field, $\overline{\mathbb{F}}_q$ is union of all extensions of $\mathbb{F}_q$ (i.e., union of all $\mathbb{F}_{q^k}$ for every positive integer $k \geq 1$); thus naturally, $\mathbb{F}_q$ is subset of the algebraic closure of $\mathbb{F}_p$ for $m > 1$.

Let u be an element of $\mathbb{F}_q$. We say u is a square if there is an element $v \in \mathbb{F}_q$ such that

$u = v^2$; otherwise, u is non-square.

Quadratic Field

Let $\mathbb{Q}$ be the field of rational numbers. Let u be a non-square integer. A quadratic field is an extension of degree two of $\mathbb{Q}$ (is vector space of dimension two over $\mathbb{Q}$), and it is defined as

$$\mathbb{Q}(\sqrt{u}) = \{r + s\sqrt{u} : r, s \in \mathbb{Q}, u \notin \mathbb{Q}\} \tag{2.1}$$

We say that $\mathbb{Q}(\sqrt{u})$ is an *imaginary* quadratic field if $u < 0$.

The *fundamental discriminant* (D_m) of $\mathbb{Q}(\sqrt{u})$ is u if $u \equiv 1 \pmod 4$ or $4u$ if $u \equiv 2$ or $3 \pmod 4$. Two quadratic fields $\mathbb{Q}(\sqrt{u_1})$ and $\mathbb{Q}(\sqrt{u_2})$ are equivalent if $u_1 = c^2 u_2$ for $c \in \mathbb{N}$.

Let g be a polynomial of degree two. Then, a complex root of g is called an *algebraic integer* (α). The algebraic integer $\alpha \in \mathbb{Q}(\sqrt{u})$ is of form

$$\alpha \in \mathbb{Z}[\theta] = \{r + s\theta : r, s \in \mathbb{Z}\} \tag{2.2}$$

where

- If $u \equiv 2$ or $3 \pmod 4$, then $\theta = \sqrt{u}$; and their set is denoted as $\mathbb{Z}[\sqrt{u}]$ ($\mathbb{Z}[\sqrt{u}]$ forms a ring of integers closed under addition, subtraction, and multiplication)
- If $u \equiv 1 \pmod 4$, then $\theta = \frac{1+\sqrt{u}}{2}$; and their set is denoted as $\mathbb{Z}[\frac{1+\sqrt{u}}{2}]$ (Similarly, $\mathbb{Z}[\frac{1+\sqrt{u}}{2}]$ forms a ring of integers)

See [98], Proposition 19.1 for the proof.

An Order in a Quadratic Field

In general, an order $\mathcal{O}$ is a subring in a ring of algebraic integers of a field $\mathcal{K}$. Thus, the ring of algebraic integers is also an order. The formal definition adopted from [122, 102] is as follows.

Definition 1. *Let $\mathcal{K} = \mathbb{Q}(\sqrt{u})$ be a quadratic field. An order $\mathcal{O}$ is a subring of $\mathcal{K}$ such that $\mathcal{O}$ is also a lattice in $\mathbb{Z}$ (free $\mathbb{Z}$-module of rank two) and $\mathcal{O} \otimes_{\mathbb{Z}} \mathbb{Q} = \mathcal{K}$.*

As an example, the ring $\mathcal{O} = \mathbb{Z}[\sqrt{u}]$ is an order in $\mathbb{Q}(\sqrt{u})$ (the same for $\mathbb{Z}[\frac{1+\sqrt{u}}{2}]$). A maximal order ($\mathcal{O}_m$) is an order that is not *properly* contained in another order [102]. An order $\mathcal{O} \subset \mathcal{O}_m$ is of form $\mathcal{O} = \mathbb{Z} + c\,\mathcal{O}_m$ such that $D = c^2 D_m$ and c is called *conductor* of $\mathcal{O}$ in $\mathcal{O}_m$, and $c \in \mathbb{Z}_{\geq 1}$ (see [53]).

Finally, we end our discussion with the following.

Proposition 2.3.1 (see [52], and [95]). *Let D_m be a fundamental discriminant of a quadratic field $\mathcal{K}$. Any order in quadratic field $\mathcal{K}$ has discriminant $D = c^2 D_m < 0$ and $c \geq 1$. Conversely, there is a unique order in $\mathbb{Q}(\sqrt{D_m})$ of discriminant D such that $D \equiv 0$ or $1 (\mathrm{mod}\ 4)$ and D is a non-square uniquely of form $D = c^2 D_m$.*

2.3.2 Ideal Class Group

Let $\mathcal{O}$ be an order of number field $\mathcal{K}$. A *fractional ideal* of $\mathcal{O}$ is a finitely generated non-zero $\mathcal{O}$-submodule $\mathfrak{a}$ of $\mathcal{K}$ such that $r\mathfrak{a} \subset \mathcal{O}$ and $r \in \mathcal{O}$ is a non-zero. A fractional $\mathcal{O}$-ideal $\mathfrak{a}$ is of form $c^{-1}\mathfrak{b}$ such that $\mathfrak{b}$ is $\mathcal{O}$-ideal and $\mathfrak{b} = c\mathfrak{a}$ for a non-zero $c \in \mathcal{O}$. The fractional $\mathfrak{a}$ (of form $c^{-1}\mathfrak{b}$) is principal if $\mathfrak{b}$ is. A fractional $\mathcal{O}$-ideal $\mathfrak{a}$ is invertible if there is a fractional $\mathcal{O}$-ideal $\mathfrak{b}$ such that $\mathfrak{a}\mathfrak{b} = \mathcal{O}$. Note that if $\mathfrak{a} = (a)$ and $\mathfrak{b} = (b)$ are principals, then $\mathfrak{a}\mathfrak{b} = (ab)$. It follows that $(a)(a^{-1}) = (1_{\mathcal{O}}) = \mathcal{O}$. An $\mathcal{O}$-ideal of norm coprime to $\mathcal{O}$'s conductor is invertible and has a unique factors. Thus, every $\mathcal{O}$-ideal is invertible if $\mathcal{O}$ is maximal. A fractional $\mathcal{O}$-ideal $\mathfrak{a}$ can be represented in form $\prod_{i=1}^{e} \mathfrak{p}_i^{e_i}$ where $\mathfrak{p}_i$ are distinct prime $\mathcal{O}$-ideals, and $e_i \in e$ are integers.

The set of all fractional $\mathcal{O}$-ideals forms an abelian group under multiplication (see [143], Theorem 5.5). The ideal class group is a quotient group that forms a finite abelian group [143] defined as:

$$\mathrm{CI}(\mathcal{O}) = I_{\mathcal{O}}/P_{\mathcal{O}}$$

where $I_{\mathcal{O}}$ is the group of **i**nvertible fractional $\mathcal{O}$-ideals, and $P_{\mathcal{O}}$ is the subgroup of **p**rincipal fractional $\mathcal{O}$-ideals. The order of $\mathrm{CI}(\mathcal{O})$ is the class number of $\mathcal{O}$. The ideal class of $\mathfrak{a}$ denoted by $[\mathfrak{a}]$ is the class of equivalent fractional ideals (i.e., the fractional ideals $\mathfrak{b}$ and $\mathfrak{a}$ are equivalent ($[\mathfrak{a}] = [\mathfrak{b}]$) such that $\mathfrak{b} = (c)\mathfrak{a}$ for some $c \in \mathcal{O}$). For simplicity, we call a fractional ideal simply by ideal.

2.3.3 Elliptic Curves

In general, this section is adopted mainly from [79, 87, 140].

Elliptic Curves Representation

General Weierstraß equation. Generally, elliptic curves are represented by the following affine equation:

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{2.3}$$

and its projective form:

$$E : y^2z + a_1xyz + a_3yz^2 = x^3 + a_2x^2z^2 + a_4x + a_6z^3 \tag{2.4}$$

where $a_1, \cdots, a_6$ are elements in $\mathbb{F}_q$; and in this context, E is denoted to be *defined over* $\mathbb{F}_q$ ($E/\mathbb{F}_q$).

A point (P) on E is the pair $(x, y) \in (\mathbb{F}_q, \mathbb{F}_q)$ such that it satisfies E's equation. The set of $\mathbb{F}_q$-rational points in $E/\mathbb{F}_q$ is denoted as $E(\mathbb{F}_q)$ and the set size as $\#E(\mathbb{F}_q)$. The set $E(\mathbb{F}_q)$ is defined as follows:

$$E(\mathbb{F}_q) = \{(x, y) \in (\mathbb{F}_q, \mathbb{F}_q) : x, y \text{ satisfies } E\text{'s equation}\} \cup \{\infty_E\}$$

where ∞_E is called the point at infinity. The point at infinity is an extra point that is added to the set $E(\mathbb{F}_q)$ so that it defines a group. This point (∞_E) does not lie on the affine equation (2.3) but on its projective form (2.4).

Let Δ_E be a discriminant of E. Δ_E is defined as follows:

$$\Delta_E = -\hat{a}_2^2\hat{a}_8 - 8\hat{a}_4^3 - 27\hat{a}_6^2 + 9\hat{a}_2\hat{a}_4\hat{a}_6$$

where

$$\begin{aligned}
\hat{a}_2 &= a_1^2 + 4a_2 \\
\hat{a}_4 &= 2a_4 + a_1a_3 \\
\hat{a}_6 &= a_3^2 + 4a_6 \\
\hat{a}_8 &= a_1^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_4^2
\end{aligned}$$

It is necessary that $\Delta_E \neq 0$; so that every point in E has a unique distinct tangent line.

Short Weierstraß equation. Let p be a prime ($p > 3$). Equation (2.3) of an elliptic curves E is isomorphic over $\mathbb{F}_q$ to

$$E_{a_4,a_6} : y^2 = x^3 + a_4x + a_6 \tag{2.5}$$

where a_4, $a_6 \in \mathbb{F}_q$ and $\Delta_{E_{a_4,a_6}} = -16(4a_4^3+27a_6^2) \neq 0$ (i.e., it is required that $4a_4^3+27a_6^2 \neq 0$). For a curve in Equation (2.5), the j-invariant of E_{a_4,a_6} is defined as:

$$j(E_{a_4,a_6}) = 1728\frac{4a_4^3}{4a_4^3 + 27a_6^2} \tag{2.6}$$

The curve $E_{0,a_6} : y^2 = x^3 + a_6$ has $j(E_{0,a_6}) = 0$, and the curve $E_{a_4,0} : y^2 = x^3 + a_4x$ has $j(E_{a_4,0}) = 1728$.

Group Structure of an Elliptic Curve

This section will only consider the supersingular elliptic curves, which will be defined later on. Let E be an elliptic curve defined over $\mathbb{F}_q$. Let π_q be the q-power Frobenius endomorphism on $E/\mathbb{F}_q$. The endomorphism π_q is defined as follows

$$\pi_q : (x, y) \to (x^q, y^q)$$

where $\pi_q(\infty_E) = \infty_E$.

Moreover, the Frobenius endomorphism is of a degree q that satisfies

$$\pi_q^2 - t\pi_q + q = 0$$

where t is the trace of π_q, and π_q^2 is analogue to $\pi_q \circ \pi_q$.

Hasse's theorem provides a bound for the group order $\#E(\mathbb{F}_q)$.

Theorem 1. *Let E be an elliptic curve over $\mathbb{F}_q$. We have*

$$|t| = |\#E(\mathbb{F}_q) - q - 1| \leq 2\sqrt{q}$$

In other words, the order of the group $E(\mathbb{F}_q)$ is in the interval:

$$q + 1 - 2\sqrt{q} \leq \#E(\mathbb{F}_q) \leq q + 1 + 2\sqrt{q} \tag{2.7}$$

Let P be a point in $E(\overline{\mathbb{F}}_q)$. The Frobenius endomorphism acts trivially on P (i.e., $\pi_q(P) = P$) if and only if P is in $E(\mathbb{F}_q)$. As a result, the set of all those points is the whole set of $E(\mathbb{F}_q)$; hence, we have

$$E(\mathbb{F}_q) = \mathsf{Ker}(\pi_q - [1])$$

where $\mathsf{Ker}(\pi_q - [1])$ is the kernel of the map $\pi_q - [1]$.

Moreover, since the map $\pi_q - [1]$ is separable, it follows that

$$\#E(\mathbb{F}_q) = \#\mathsf{Ker}(\pi_q - [1]) = \deg(\pi_q - [1])$$

The Waterhouse theorem [155] further elaborates on the order of E. Waterhouse's theorem is presented here from the perspective of the supersingular elliptic curve (as in [61]), where $p|t$.

Theorem 2. *Let $\mathbb{F}_q$ be a finite field, and t be the trace of q-power Frobenius endomorphism. There is a supersingular elliptic curve $E/\mathbb{F}_q$ with $\#E(\mathbb{F}_q) = q+1-t$ and $|t| \leq 2\sqrt{q}$ if and only if one of the following is true:*

1. *m is even and*
   - *(a) $t = \pm 2\sqrt{q}$; or*
   - *(b) $t = \pm\sqrt{q}$, and $p \not\equiv 1 (\mathrm{mod}\ 3)$; or*
   - *(c) $t = 0$, and $p \not\equiv 1 (\mathrm{mod}\ 4)$,*
2. *m is odd and $t = 0$,*

where $t \in \mathbb{Z}$, $q = p^m$, and $p > 3$ is a prime.

In the case of an ordinary curve, there is an ordinary curve $E/\mathbb{F}_q$ with $\#E(\mathbb{F}_q) = q+1-t$ and $p \not| t$ is true.

In 1988, Schoof derived the group structure of a supersingular elliptic curve $E/\mathbb{F}_q$ (see [136], Lemma 4.8). Let L be a set of prime numbers. There is a supersingular elliptic curve $E/\mathbb{F}_q$ with $\#E(\mathbb{F}_q)$ of a prime factor $\prod_{l_i \in L} l_i^{e_{l_i}}$. The possible group structures of $E(\mathbb{F}_q)$ are one of the following cases:

- If $t = \pm 2\sqrt{q}$, then we have $E(\mathbb{F}_q) \cong \mathbb{Z}/(\sqrt{q} \mp 1)\mathbb{Z} \times \mathbb{Z}/(\sqrt{q} \mp 1)\mathbb{Z}$; or
- If $t = \pm\sqrt{q}$ and $p \not\equiv 1 (\mathrm{mod}\ 3)$, then $E(\mathbb{F}_q)$ is a cyclic group; or
- If $t = 0$ and $p \not\equiv 1 (\mathrm{mod}\ 4)$, then $E(\mathbb{F}_q)$ is either a cyclic group or $E(\mathbb{F}_q) \cong \mathbb{Z}/\frac{q+1}{2}\mathbb{Z} \times \mathbb{Z}/2\mathbb{Z}$; otherwise $E(\mathbb{F}_q)$ is a cyclic group.

where $t \in \mathbb{Z}$, $q = p^m$, $p \notin L$, and $p > 3$.

Let P be a point in $E(\overline{\mathbb{F}}_q)$. The cyclic subgroup generated by P is given by $\langle P \rangle$. The subgroup of points of order N is called the *N-torsion points* which is defined as follows

$$E[N] = \mathsf{Ker}([N])$$
$$E[N] = \{P \in E(\overline{\mathbb{F}}_q) : [N]P = \infty_E\} \cup \{\infty_E\}$$

Let N be co-prime to p. The structure of $E[N]$ is congruent to $\cong \mathbb{Z}/N\mathbb{Z} \times \mathbb{Z}/N\mathbb{Z}$, which has an order $\#E[N] = \deg([N]) = N^2$. Let P, Q be linearly independent points in $E[N]$. Any point $R \in E[N]$ can be uniquely represented as free $(\mathbb{Z}/N\mathbb{Z})$-module with bases (P, Q) (i.e., $R = [n]P + [m]Q \in E[N]$ and $n, m \in \mathbb{Z}/N\mathbb{Z}$).

Finally, we present a definition of supersingular elliptic curve.

Definition 2. *An elliptic curve $E/\mathbb{F}_q$ is a supersingular, if one of the following conditions is true:*

- *There is no $P \in E(\overline{\mathbb{F}}_q)$ with order p (i.e., $E[p] \cong \{\infty\}$).*
- *$\#E(\mathbb{F}_q) = q + 1 - t$ and $p|t$ (i.e., $gcd(p, t) \neq 1$).*
- *The endomorphism ring of $E/\overline{\mathbb{F}}_q$ is an order in a quaternion algebra.*

Otherwise, $E/\mathbb{F}_q$ is said to be an ordinary curve.

Mapping Between Elliptic Curves

Let E_1 and E_2 be elliptic curves defined over $\mathbb{F}_q$. The morphism $\phi : E_1 \to E_2$ is a rational map that is also a group homomorphism such that $\phi(\infty_{E_1}) = \infty_{E_2}$. The kernel of ϕ is the set of all points on E_1 such that $\phi(P) = \infty_{E_2}$.

$$\mathsf{Ker}(\phi) = \{P \in E_1(\overline{\mathbb{F}}_q) \ : \phi(P) = \infty_{E_2}\} \tag{2.8}$$

An isomorphism of elliptic curves defined over a finite field $\mathcal{K}$ is a morphism (is also invertible and a group homomorphism) of degree 1 (see [79], Lemma 8.1.15) such that $\#E_1(\mathcal{K}) = \#E_2(\mathcal{K})$ and $\phi(\infty_{E_2}) = \infty_{E_2}$. Generally, an isomorphism from E_1 to E_2 is defined as

$$\phi(P) = \phi(x, y) = (u^2x + c_x,\ u^3y + cu^2x + c_y) \tag{2.9}$$

where u, c, c_x, and $c_y \in \mathbb{F}_q$, and hence, ϕ is defined over $\mathbb{F}_q$; otherwise $u, c, c_x, c_y \in \overline{\mathbb{F}}_q$ and ϕ is said to be defined over the algebraic closure of $\mathbb{F}_q$ (see [79], Theorem 9.3.4).

Theorem 3 ([79]). *If $j(E_1) = j(E_2)$, then there is as an isomorphism defined over the algebraic closure of $\mathbb{F}_q$.*

Definition 3. *E_1 and E_2 is called isomorphic if an isomorphism from E_1 to E_2 exists; An isomorphism class is the set of all isomorphic curves.*

From [61] Proposition 2.3, there are finitely many $\mathbb{F}_p$-isomorphism classes of supersingular elliptic curves over $\mathbb{F}_p$, where p is a prime ($p > 3$). For the j-invariant $j \neq 0, 1728$, there are exactly 2 isomorphism classes. For the special cases $j = 0$, and $j = 1728$, there are 6 and 4 isomorphism classes, respectively, when $p \equiv 1 \pmod 3$ and $p \equiv 1 \pmod 4$. Hence, in most cases, there are two $\mathbb{F}_p$-isomorphism classes of supersingular elliptic curves over $\mathbb{F}_p$ with a given j-invariant for $p > 3$.

Endomorphism ring. The set of all group homomorphisms (i.e., isogenies) from E_1 to itself is called the endomorphism ring of E_1 ($\mathbf{End}(E_1)$). The endomorphism ring defined over $\mathbb{F}_p$ is denoted as $\mathbf{End}_{\mathbb{F}_p}(E_1)$. $\mathbf{End}(E_1)$ forms a ring under addition and composition. For $\phi_1, \phi_2 \in \mathbf{End}(E_1)$ and point P on E_1, the addition satisfies $(\phi_1 + \phi_2)(P) = \phi_1(P) + \phi_2(P)$; the composition satisfies $(\phi_1 \circ \phi_2)(P) = \phi_1(\phi_2(P))$. One example of $\phi \in \mathbf{End}(E_1)$ is the multiplication-by-d mapping ($[d] : E_1 \to E_1$). The Frobenius endomorphism is another example.

For the other conditions in Theorem 2 (other than condition 1.(a)), the endomorphism algebra $\mathcal{R}$ is an imaginary quadratic field $\mathbb{Q}(\sqrt{u})$. And all orders in $\mathbb{Q}(\sqrt{u})$ is possible $\mathbf{End}_{\mathbb{F}_q}(E)$ such that the order's conductor (c) is a co-prime to p.

The case of endomorphism ring over $\mathbb{F}_p$ have been discussed in [61]. Let $E/\mathbb{F}_p$ be a supersingular elliptic curve with $\#E(\mathbb{F}_p) = p+1$ (i.e., $t = 0$, see Theorem 2 condition 2). It is known that Frobenius endomorphim π_p is in $\mathbf{End}_{\mathbb{F}_p}(E_1)$ that generates the order $\mathbb{Z}[\pi_p] \subseteq \mathcal{O}_m$ with discriminant $D = t^2 - 4p = -4p = c^2 D_m$. The endomorphism ring $\mathbf{End}_{\mathbb{F}_p}(E_1)$ is an order in the imaginary quadratic field $\mathbb{Q}(\pi_p)$. In fact, the order $\mathcal{O} = \mathbf{End}_{\mathbb{F}_p}(E_1)$ is in $\mathbb{Q}(\sqrt{-p})$ given that $\pi_p^2 + p = 0$ is true.

From Section 2.3.1, the fundamental discriminant of $\mathbb{Q}(\sqrt{-p})$ is either $D_m = -p$, or $-4p$ (for case $p \equiv 3 \pmod 4$, or $p \equiv 1 \pmod 4$[1], respectively). Thus, $D_m \equiv 1 \pmod 4$ (or $D_m \equiv 0 \pmod 4$, respectively). From above and Proposition 2.3.1, we can conclude that $\mathbf{End}_{\mathbb{F}_p}(E_1)$ is $\mathbb{Z}[\sqrt{-p}]$ for (i) $p \equiv 1 \pmod 4$ with $c = 1$ or (ii) $p \equiv 3 \pmod 4$ with $c = 2$. Otherwise, we have $\mathbf{End}_{\mathbb{F}_p}(E_1)$ is $\mathbb{Z}[\frac{1+\sqrt{-p}}{2}]$ for $p \equiv 3 \pmod 4$ with $c = 1$.

[1]Skipping the case where $p \equiv 2 \pmod 4$ since we assume that p is odd prime.

2.3.4 Bilinear Pairing

Let N be a prime such that $N|\#E(\mathbb{F}_q)$. Let k be the smallest integer such that $N|q^k-1$. The integer k is called the embedding degree of the N-torsion points. Let G_1 and G_2 be cyclic subgroups in $E[N] \cap E(\mathbb{F}_{q^k})$. Let G_T be the target multiplicative group of the N-th roots of unity in $\mathbb{F}^*_{q^k}$. In the context of elliptic curves, a bilinear pairing on groups is the map $\hat{e} : G_1 \times G_2 \to G_T$. A property of bilinear pairing that is of our interest is non-degeneracy. A non-degenerate pairing guarantees non-trivial pairings such that $\hat{e}(P,Q) \neq 1$ for all points $P \in G_1/\infty_E$ and $Q \in G_2/\infty_E$.

An example of a bilinear pairing map considered in cryptography is the Weil pairing, a non-degenerate pairing map defined as $\hat{e}_N : E[N] \times E[N] \to \mu_N$ (μ_N is the order N subgroup in $\mathbb{F}^*_{q^k}$). Furthermore, the Weil pairing has an identity property, $\hat{e}_N(P,P) = 1, \forall P \in E[N]$. This property may result in a trivial pairing such that $\hat{e}_N(P,Q) = 1$ when P and Q are linearly dependent. Therefore, to avoid this issue, it is required that P and Q are linearly independent. As a result, we raise the question of how to sample random points and yet preserve their independence efficiently.

Galbraith et al. [80] address this issue and outline pairing into types based on the choice of G_1 and G_2. Type 1 is of most relevance since we use the supersingular elliptic curve in this work. In that context, we set our first cyclic group to the *base-field group* ($G_1 = E[N] \cap E(\mathbb{F}_q)$). We then use a distortion map (σ [2]) to map from G_1 to $E[N] \cap \mathsf{Ker}(\pi_q - [q])$. Finally, we redefine our pairing as

$$e_N(P,Q) = \hat{e}_N(P, \sigma(Q))$$

where $P, Q \in G_1$. In this particular case, we perform a secure hashing of a random point by multiplying the factor $\#E(\mathbb{F}_q)/N$ to a randomly selected point. However, the failure probability of hashing into a point of order N is $1/N$ (i.e., $P = \infty$), which means that we have to repeat this step with probability $1/N$. Hence, this hashing method may be inefficient for a small value of N.

Twist of an elliptic curve. A technique to improve pairing in elliptic curves is a *twist* of curves. This technique, introduced in [14], helps to compress the representation of the point in G_2. Additionally, it improves the computation efficiency of pairing in some cases [79]. We will denote a twist of a curve E by E^t. The non-trivial twist map of a curve is an isomorphism ($\sigma^{-1} : E \to E^t$) over $\overline{\mathbb{F}}_q$ such that $\sigma^{-1}(\infty_E) = \infty_{E^t}$. Both curves E^t and E are isomorphic over $\overline{\mathbb{F}}_q$ but not over $\mathbb{F}_q$. Thus, they (E^t and E) fall into different $\mathbb{F}_q$ isomorphism classes. E^t_1 and E^t_2 are said to be equivalent such that there is an isomorphism from E^t_1 to E^t_2 defined over $\mathbb{F}_q$.

[2] The definition of the distortion map (σ) can be found in Section 4.3.

2.3.5 Isogenous Curves

An isogeny between curves ($\phi : E_1 \to E_2$) is a surjective morphism that has a finite kernel such that $\phi(\infty_{E_1}) = \infty_{E_2}$. We say ϕ is defined over $\mathbb{F}_q$, if the non-constant rational map represents ϕ has coefficients in $\mathbb{F}_q$. Let $\langle S \rangle$ be a subgroup of $E(\overline{\mathbb{F}}_q)$. We compute an isogeny ϕ with kernel $\langle S \rangle$ using Vélu's formulae [149]. Vélu's formulae takes $O(\#\langle S \rangle)$ arithmetic operations. In this work, we will focus on separable isogenies where the isogeny degree is its kernel size. An isogeny is separable if its degree is co-prime to p. An isogeny ϕ is a rational map

$$\phi : (x, y) \to (\frac{\mathsf{p}_x}{\mathsf{q}_x}, \frac{\mathsf{p}_y}{\mathsf{q}_y})$$

where $\mathsf{p}_x, \mathsf{q}_x, \mathsf{p}_y$, and q_y are polynomial in (x, y). The monic polynomials p_x and p_y are relatively prime polynomials to q_x and q_y, respectively. The degree of ϕ is $\mathsf{max}(\deg(\mathsf{p}_x), \deg(\mathsf{q}_x))$. The derivative of $\frac{\mathsf{p}_x}{\mathsf{q}_x}$ is a non-zero if ϕ is separable; otherwise, ϕ is inseparable [138]. A separable isogeny of degree $\deg \phi$ can be a composite of isogenies of prime degree factors $\deg \phi$ (i.e., $\phi = \phi_n \circ \cdots \phi_2 \circ \phi_1$ such that $\deg \phi = \deg \phi_1 \cdots \deg \phi_n$).

From Tate's theorem [146], E_1 and E_2 are *isogenous* (over $\mathbb{F}_q$) if and only if there is an isogeny over $\mathbb{F}_q$ between them such that $\#E_1(\mathbb{F}_q) = \#E_2(\mathbb{F}_q)$. This implies that isogenies preserve the supersingularity (i.e., E_2 is supersingular if and only if E_1 is).

Furthermore, an isogeny ϕ has unique a *dual* isogeny $\hat{\phi}$ with the same degree ($\deg \phi = \deg \hat{\phi}$) such that $\hat{\phi} : E_2 \to E_1$, and $\phi \circ \hat{\phi} = [\deg \phi]$ on E_2, where $[m]$ is multiplication-by m mapping (see [138], Theorem 2.23).

An isogeny is called *horizontal* isogeny if $\mathbf{End}_{\mathbb{F}_p}(E_1) \cong \mathbf{End}_{\mathbb{F}_p}(E_2)$ [97]. Also, the non-backtracking walk is a sequence of isogenies that is not cyclic or followed by any dual isogeny(s).

Complex Multiplication

In this section, we briefly introduce group action on the set of all curves with complex multiplication. A curve whose endomorphism ring is larger than $\mathbb{Z}$ (i.e., contains $\mathcal{O}$) is said to have a complex multiplication by $\mathcal{O}$.

Let $\mathcal{K} = \mathbb{Q}(\sqrt{-p})$ be an imaginary quadratic field with a fundamental discriminant d_m. Let $\mathcal{O}$ be an order of $\mathcal{K}$. Let $\mathcal{ELL}(\mathcal{O})$ be a non-empty set of isomorphism classes of supersingular elliptic curves with $\mathbf{End}_{\mathbb{F}_p} \in \{\mathbb{Z}[\pi], \mathcal{O}_m\}$. Curves in $\mathcal{ELL}(\mathcal{O})$ are connected by horizontal isogeny of degree l with the same $\mathbf{End}_{\mathbb{F}_p}$. In short, the concept of complex multiplication deduces the following, adapted from [41].

Theorem 4. *The ideal class group* $\boldsymbol{cl}(\mathcal{O})$ *acts freely and transitively*[3] *on* $\mathcal{ELL}(\mathcal{O})$ *through the action:*

$$\boldsymbol{cl}(\mathcal{O}) \times \mathcal{ELL}(\mathcal{O}) \to \mathcal{ELL}(\mathcal{O})$$
$$\phi_{\mathfrak{l}} : [\mathfrak{l}] * E \to E/\mathfrak{l}$$

where $[\mathfrak{l}] \in \boldsymbol{cl}(\mathcal{O})$*, and* $E \in \mathcal{ELL}(\mathcal{O})$

By fixing $E \in \mathcal{ELL}(\mathcal{O})$, it can be observed that

$$\mathbf{cl}(\mathcal{O}) \to \mathcal{ELL}(\mathcal{O})$$
$$[\mathfrak{l}] \to E/\mathfrak{l}$$

which implies that $\#\mathcal{ELL}(\mathcal{O})$ is exactly $\#\mathbf{cl}(\mathcal{O}) = h(\mathcal{O})$, where h is the class number. Furthermore, the order of $\mathbf{cl}(\mathcal{O})$ is asymptotically equal to $\sqrt{p}$.

Since any ideal can be factored into prime ideals, we consider prime ideals in the discussion below. The $\mathfrak{l}$-torsion subgroup is defined as follows:

$$E[\mathfrak{l}] = \{P \in E(\overline{\mathbb{F}}_q) : \sigma(P) = \infty_E,\ \forall \sigma \in \mathfrak{l}\} \cup \{\infty_E\}$$

Let c be the conductor of $\mathcal{O}$ such that $d = c^2 d_m$. Let l be an Elkies prime that is coprime to c, generates the ideal $(l) \in \mathbf{cl}(\mathcal{O})$, and satisfies $\left(\frac{-p}{l}\right) = 1$. Thus, by the definition of Elkies primes, the characteristic equation of p-power Frobenius endomorphism splits modulo l:

$$\pi_p^2 + p = (\pi_p - r)(\pi_p - p/r) (\mathrm{mod}\ l)$$

where $r = 1$ and $p = -1 (\mathrm{mod}\ l)$ (i.e., the equation splits if $\left(\frac{-p}{l}\right) = 1$). Thus, we say that (l) splits into ideals of norm l such that $(l) = \mathfrak{l}\hat{\mathfrak{l}}$, where $\mathfrak{l} = (l, \pi_p - 1)$, and $\hat{\mathfrak{l}} = (l, \pi_p + 1)$. Further, the l-torsion group $E[l]$ has two eigenvalue 1 and -1 for each $\mathfrak{l}$ and $\hat{\mathfrak{l}}$ (i.e., 1 is eigenvalue of $E[\mathfrak{l}]$, and -1 is for $E[\hat{\mathfrak{l}}]$). The action of $\mathfrak{l}$ and $\hat{\mathfrak{l}}$ represent the isogenies $\phi_{\mathfrak{l}}$ and $\phi_{\hat{\mathfrak{l}}}$, respectively. Since $(l) = \mathfrak{l}\hat{\mathfrak{l}} = \hat{\mathfrak{l}}\mathfrak{l}$, each of $\mathfrak{l}$ and $\hat{\mathfrak{l}}$ are inverse of the other in $\mathbf{cl}(\mathcal{O})$ [41, 144]; thus, they represent an isogenies of opposite direction of each other.

To conclude, the ideal class $\mathfrak{l}$ whose norm $\mathbf{N}(\mathfrak{l})$ is coprime to p guarantees the existence of a separable isogeny ϕ of degree $\mathbf{N}(\mathfrak{l})$ defined over $\mathbb{F}_p$. Let $E/\mathbb{F}_p \in \mathcal{ELL}(\mathcal{O})$ be an elliptic curve. The isogeny ϕ is unique up to post-composition with an isomorphism; the curve $\phi(E)$ is also unique up to an isomorphism (see [130], and [79], Theorem 9.6.19). Further, the kernel of ϕ is unique and equal to $\mathsf{Ker}(\phi_{\mathfrak{l}}) = E[\mathfrak{l}]$. Thus, $E/\mathfrak{l}$ is $\mathbb{F}_p$-isomorphism to $\phi(E)$. The isogeny ϕ can be computed using Vélu's formulae [149].

[3] For an $E_1, E_2 \in \mathcal{ELL}(\mathcal{O})$, there is exactly one $[\mathfrak{l}] \in \mathbf{cl}(\mathcal{O})$ such that $\mathfrak{l} * E_1 = E_2$.

Isogeny Graph over a Finite Field $\mathbb{F}_p$

In this section, we will highlight the most relevant finding of [61] that is related to our research. All isogenies and curves considered in this section are defined over $\mathbb{F}_p$.

Let l be a prime such that $l \neq p$ and $p > 3$. A degree l isogeny graph over $\mathbb{F}_p$ has vertices as supersingular curves, and edges as degree l isogeny. Each vertex is represented by the curve's j-invariant and the curve's coefficients[4]. The graph has at most two levels[5] called *surface* and *floor*. In the case of $p \equiv 3 \pmod 4$, we say that E_1 is on the surface if $\mathbf{End}_{\mathbb{F}_p}(E_1) = \mathbb{Z}[(1 + \sqrt{-p})/2]$, and we say E_1 is on the floor if $\mathbf{End}_{\mathbb{F}_p}(E_1) = \mathbb{Z}[\sqrt{-p}]$. However, we end up with a one-level graph in the case of $p \equiv 1 \pmod 4$ since we always have $\mathbf{End}_{\mathbb{F}_p}(E_1) = \mathbb{Z}[\sqrt{-p}]$.

In the case of supersingular isogeny graph with two levels, we have two horizontal isogenies with degree l from each vertex such that $l > 2$ and $\left(\frac{-p}{l}\right) = 1$; also,

- if $p \equiv 7 \pmod 8$,
  - there are $h(-p)$ vertices on the surface (and similar on the floor).
  - there is a non-horizontal isogeny of degree two that connects each vertex on the surface to a vertex on the floor.
  - there are two horizontal isogenies of degree two from each vertex on the surface.
- if $p \equiv 3 \pmod 8$,
  - there are $h(-p)$ vertices on the surface and $3h(-p)$ on the floor.
  - there are three non-horizontal isogenies of degree two that connect every vertex on the surface to three vertices on the floor.
  - there are no horizontal isogenies of degree two.

where h is the class number for imaginary quadratic fields, and $\left(\frac{\cdot}{\cdot}\right)$ is the Legendre symbol.

In the case of $p \equiv 1 \pmod 4$, we have two horizontal degree l isogenies from every $h(-4p)$ vertices such that $l > 2$ and $\left(\frac{-p}{l}\right) = 1$; additionally, there is one outgoing horizontal isogeny of degree two from every vertex.

[4] This additional information is used to distinguish curves in different isomorphic classes.

[5] As a graph level, we mean the set of vertices with the same endomorphism ring.

2.3.6 Isogeny and Pairing

We now discuss the pairing relations over isogenous curves (e.g., E_1 and E_2). Let N be a prime such that $N \neq p$ and $N|\#E_1(\mathbb{F}_p)$. Let P and Q be points in $E_1[N]$ and $E_2[N]$, respectively. The following relationship holds (see [23], Theorem IX.10)

$$\hat{e}_N(P, \hat{\phi}(Q)) = \hat{e}_N(\phi(P), Q) \tag{2.10}$$

Furthermore, it is trivial to show that

$$\hat{e}_N(P, Q')^{\deg \phi} = \hat{e}_N(\phi(P), \phi(Q')) \tag{2.11}$$

where $Q = \phi(Q')$, and $\hat{\phi} \circ \phi = [\deg \phi]$ on E_1.

Chapter 3

Beyond the Clock: The Role of Time in Cryptography

This chapter provides an overview of delay-based cryptography, covering selected notions and special delay-based constructions, as well as the building blocks used to construct them.

3.1 Introduction

Delay-based cryptography is a unique area of cryptography that incorporates time to secure digital communication systems. It does so by introducing intentional computational delays into its cryptographic operations. This involves designing operations that need a *predetermined* amount of time to complete (without access to a secret structure or key). In general, delay-based cryptography aims to prevent predicting a protocol's outcome within a predetermined time period, thus ensuring timely access to sensitive information. In recent years, delay-based cryptography has gained significant attention, with multiple notions being proposed. Each of these integrates time delay into their design to expand cryptographic protocols' security and applicability.

Organization. We begin, in Section 3.2, with our definitions of time, time delay, and long sequential functions. Following this, we discuss a list of well-known methods from the literature for constructing secure sequential functions. Section 3.3 explores selected notions in delay-based cryptography, including time-lock puzzle, proof of sequential work, verifiable delay function, and delay encryption. Section 3.4 discusses selected techniques for verifying

computational correctness from the literature, which is categorized by their underlying assumptions. We then highlight some special classes of delay-based cryptography in Section 3.5. Finally, we conclude with our remarks in Section 3.6.

3.2 Main Building Blocks and Theoretical Foundations

Unit of time, and delay. Before we proceed, we elaborate on what we mean by time, or delay. Real-world time is generally established using physical references like atomic clocks, whereas in our context, we define time based on the abstract measure of the duration required to complete a predefined computational task. In this work, a unit of time is considered in a computational model, which relates to a specific base task that takes at least x amount of real-time to complete, even with parallel computation. We refer to a unit of time for a base task as a "step." A T-time refers to the time taken to complete a large task through T steps. In parallel computing, T-*sequential time*, or T-*delay* for short, is determined by the duration of the longest sequence of T dependent tasks (that must be executed sequentially), not the sum of all tasks; where the probability of predicting the output status with $T-1$ steps is negligible.

Long Sequential Functions. We use T to characterize the difficulty of a function, which represents the amount of sequential steps required to compute its output for any random input with a polynomially large number of parallel computations. A short function has a small value of T, while a function with a large value of T is considered a long function. Throughout this work, we refer to the long sequential function as $\mathsf{SeqEval}$. $\mathsf{SeqEval}$ is defined with parameters for its domain and range. A shorter version of $\mathsf{SeqEval}$, referred to as a shortcut or trapdoor, is denoted by tr.

3.2.1 Constructions of Long Sequential Functions

This section presents different constructions of long sequential functions (from the literature), which are classified by their sequentiality assumption.

Hash-Based Iterative Sequential Work

Let $\mathsf{H} : \{0,1\}^* \to \{0,1\}^{2\lambda}$ be a cryptographic hash function. In a hash-based sequential work, H serves as the primary task, iterated T times to achieve a delay of T, with the

output of one iteration being the input for the next, as shown below:

$$y = \mathsf{H}(\mathsf{H}(\cdots \mathsf{H}(x) \cdots)),$$

where x is a random string in $\{0,1\}^*$ and y is the delayed output. This method constructs a sequential work that maps from $\{0,1\}^*$ to $\{0,1\}^{2\lambda}$, where the input x is often referred to as a puzzle (or challenge) and the output y is the solution (or answer) to that puzzle. However, this method may not be suitable for building more complex applications for several reasons, including the inability to predict the output y before completing T-iterations. Another method, which allows us to predict the output y *beforehand*, is as follows.

1. Puzzle creation

    a. Choose a random string y and divide it into T equal parts, i.e.,

    $$y := y_0 \| y_1 \| \cdots \| y_T.$$

    b. Return the following as input to the iterative sequential work:

    $$x := x_0 \| x_1 \| \cdots \| x_T = y_0 \| (y_1 \oplus \mathsf{H}(y_0)) \| \cdots \| (y_T \oplus \mathsf{H}(y_{T-1})).$$

2. Puzzle solving

    a. To solve the puzzle x, compute $y_i \leftarrow x_i \oplus \mathsf{H}(y_{i-1})$ for all $i \in \{1, 2, \ldots, T\}$.

One drawback of this method is that the size of the puzzle depends on T, and creating the puzzle involves large amounts of parallel computation (and hence less sequential computation time). Furthermore, parts of the output y become visible with each iteration, but there may be a requirement not to reveal any part of the output until the last iteration. To address this, we define the output as the hash of all y_i (i.e., $\mathsf{H}(y_0 \| y_1 \| \cdots \| y_T)$).

Exponentiation in a Group of Unknown Order

Let G be a group of order o with a generator g'. Here, we assume that the order o is unknown. To construct a sequential function, we apply the group operation repeatedly to a random element $g \in G$. Let $\mathsf{Hash} : \{0,1\}^* \to G$ be an efficient hash function that maps a string to an element in G. Formally, the sequential function can be defined as follows:

$$g_0 = \mathsf{Hash}(x), \quad g_{i+1} = g_i^k \quad \text{for} \quad i = 0, 1, \ldots, T-1,$$

where k is a positive integer and g_T is the solution y to the puzzle x. With the knowledge of G's order, we can compute the solution y in fewer steps than T by exponentiating $\mathsf{Hash}(x)$ to the power of $k^T \bmod o$. As a result, the order of G must be unknown to *the puzzle solver*.

The common instances of these groups are the RSA groups [131] and the ideal class groups of imaginary quadratic fields [31]. Each of these candidates has its own advantages over the other. For example, RSA groups have been studied more extensively than ideal class groups of imaginary quadratic fields. In short, the RSA group is the multiplicative group of integers modulo a composite number N, represented as $\mathbb{Z}_N^*$. Usually, N is the product of two large primes, p and q, which gives the group an order of $\varphi(N) = (p-1)(q-1)$, and φ being Euler's totient function. A puzzle based on an RSA-based sequential function of difficulty T can therefore be solved in fewer than T steps by factoring N. Hence, a puzzle generator or a trusted third party is required to generate the group, specifically to generate the modulus N.

In contrast, ideal class groups of imaginary quadratic fields can be publicly generated while still maintaining an unknown order, unlike RSA groups. Here is [63]'s security model for groups of unknown order.

Definition 4 (Unknown-order Groups Security [63])**.** *Let λ be a security parameter. Let Gen be an algorithm that generates groups. We define Gen as achieving a security threshold of (λ, ρ) if, with a probability of $1 - \frac{1}{2^\rho}$ for its outputs, any algorithm A provided with a group G from Gen needs a minimum of 2^λ bit operations to accurately determine $|G|$ with a high probability.*

Jacobians of Hyperelliptic Curves. The interest in unknown order groups has increased in recent years. Dobson et al. [62] proposed a candidate for an unknown order group that requires no trusted setup. Their proposal uses the Jacobian of a genus three hyperelliptic curve over $\mathbb{F}_q$ with characteristic p. To compute group order, the group is subject to point-counting algorithms and discrete-log algorithms. The discrete logarithmic algorithm is estimated to have a complexity of $O(q \log^2 q)$ with required memory $O(q^{3/4} \log^{o(1)} q^{3/4})$ [101]. The attack, for instance, requires 4×10^{18} TB of storage with a total of 2^{134} field multiplications in a 120-bit field. However, Abelard's work [1] reduces point counting complexity to $O(\log^{14} q \log\log^{14} q)$, which may further be improved. Therefore, it is required that $\log q \approx 1100$-bit to achieve 128-bit of security (the reader is referred to [103] for further details). Interestingly, it is considerably less secure to form groups from *non-hyperelliptic* curves, which make up the majority of genus three curves. It is therefore crucial that the choice of a curve must be carefully considered (for more information, see [147]).

Group of Known Order

The security of group-based sequential functions depends on how challenging it is to shortening or speed up the function's sequential computation within the group, up to some assumption(s). Rotem et al. [134] show negative results on the feasibility of a secure sequential function construction based on a generic[1] cyclic group.

Isogeny Walk on Supersingular Elliptic Curves with Unknown Endomorphism Ring

In abstract, this type of construction is based on isogeny (ϕ) walk on a graph(s) defined over a finite field $\mathbb{F}_q$. The isogeny ϕ is separable with a large degree (e.g., $\deg \phi = l^T$ where l is a small prime). Let E be a *random*[2] elliptic curve in graph $\mathcal{G}$. To evaluate ϕ efficiently, we compute and evaluate a T-sequential composite of ϕ on $E \in \mathcal{G}$. Thus, we define a sequential function as follows:

$$E \xrightarrow{\phi_1} E_1 \xrightarrow{\phi_2} E_2 \xrightarrow{\phi_3} \cdots \xrightarrow{\phi_T} E_T \in \mathcal{G}$$

where ϕ_i is an isogeny of degree l_i such that $\phi \equiv \phi_1 \circ \cdots \circ \phi_T$, and there is no $\hat{\phi}_i \equiv \phi_j$ and $i \neq j$, where $\hat{\phi}_i$ is the dual isogeny of ϕ_j. Hence, breaking the sequentiality is based on finding a shortcut isogeny ϕ' of smoother and smaller degree. In general, finding a shortcut is heuristically equivalent to computing an endomorphism ring. Recovering the endomorphism ring of a randomly selected supersingular curve is a hard problem [127, 67, 57].

As far as we know, isogeny-based sequential functions (ϕ) can be divided into two main categories. The first category, similar to Supersingular Isogeny Diffie–Hellman (SIDH)-like setting [90], involves sequential functions defined over an extension field. In the second category, sequential functions are defined over a prime field, similar to the CSIDH setting [41].

The sequential function of the latter is constructed using the sequential walk over the $\mathbb{F}_p$ graph $\mathcal{G}$ that connects the set of supersingular elliptic curves (denoted by $\mathcal{E}_{\mathbb{F}_p}(\mathcal{O})$) over $\mathbb{F}_p$ with $\mathcal{O}$ being an order of an imaginary quadratic field. This function thus represents

[1] In this context, generic means that the construction does not utilize any representation properties of the group (for this abstract model, see [117]).

[2] The initial curve E_0 with a known endomorphism gives an adversary an advantage, reducing the intended difficulty of the isogeny-based sequential function.

the group action of the ideal class group $\mathbf{cl}(\mathcal{O})$ of an imaginary quadratic order $\mathcal{O}$[3] on the set $\mathcal{E}_{\mathbb{F}_p}(\mathcal{O})$. As a result, the security of the sequential function ϕ defined over the prime field $\mathbb{F}_p$ depends on the difficulty of computing the order of the class group $\mathbf{cl}(\mathcal{O})$.

Other Methods

Other proposals for constructing sequential functions are explored in the work of [22] and [108]. The first proposal in [22] introduces the concept of *non-parallelizing languages*, characterized by an evaluation difficulty T; a T-non-parallelizing language is one that cannot be decided by a circuit with a depth less than T. Essentially, this method is based on succinct randomized encodings, whose sequentiality depends on the existence of non-parallelizable languages.

The second proposal in [108] presents a notion of a *computational reference clock*, distinct from the definition of time provided in Section 3.2. This notion relates the time delay to some physical reference, representing the current state of an iterative, public computation. A concrete example of a computational reference clock serving as a time reference is the current state of the Bitcoin network [124]. To construct sequential functions, in short, they combine witness encryption [82] with the Bitcoin network as a reference clock.

To be complete, a theoretical proposal to construct a sequential function is presented in [92]. The core construction is based on an iterative sequential function derived from a minimal circuit that is encoded in a fully homomorphic encryption scheme.

3.3 Notions in Delay-Based Cryptography

3.3.1 Time-Lock Puzzles and Timed-Release Encryption

In 1993, May [118] proposed the notion of Timed-Release Encryption (TRE), introducing the concept of *sending a message into the future*. The goal is to ensure that the ciphertext of a secret message cannot be decrypted before a predetermined time period has elapsed. May suggested using a "trusted agent" to enforce this time delay, in which the agent reveals the decryption key of that ciphertext after the predetermined period.

A notable notion in delay-based cryptography is the Time-Lock Puzzle (TLP), introduced by Rivest, Shamir, and Wagner [131] in 1996, building on the work of May [118]. The

[3] with $\mathcal{O}$ being $\cong \mathbb{Z}[\sqrt{-p}]$ or $\cong \mathbb{Z}[\frac{1+\sqrt{-p}}{2}]$.

main idea involves replacing the trusted agent with a long sequential function to achieve an approximate time delay. Following their proposal, some TLP's constructions have been introduced, each based on different assumptions, including those discussed in [22, 108].

Generally, and for a security parameter λ, a time-lock puzzle scheme takes a message m and a difficulty T. The key component of TLP is the existence of a secret *short function* that has difficulty $t << T$. This function allows the sender to construct a puzzle and its solution, which is then used to encapsulate the secret message m. The sender outputs a challenge consisting of the puzzle x, the difficulty T, and the ciphertext. The puzzle solution serves as the deciphering key for the ciphertext, which is obtained by evaluating the T-sequential function on x. In summary, the goal is to keep the message m hidden until the T-sequential function evaluation has been completed, thereby achieving the predetermined time delay.

In Rivest et al.'s TLP, the sequential function is constructed by repeated squaring within the RSA group. For an RSA modulus N and a difficulty T, the T-sequential function involves exponentiating an element g by 2^T modulo N (i.e., $g^{2^T} \bmod N$). On the other hand, the short function is defined as the exponentiation of g to $2^T \bmod \varphi(N)$. Thus, in this context, each sender must generate their own RSA group, which must later be included in the output challenge. In more detail, the challenge is represented by the tuple $(N, T, g, kg^{2^T}, \mathsf{Encryption}_k(m))$, where $(k, g) \in (\mathbb{Z}_N^*)^2$.

Furthermore, the notion of Time-Lock Encryption (TLE) is a variant of time-lock puzzles, with a key difference. In contrast to TLE, the TLP notion requires efficient puzzle generation. In another words, the time t needed to generate the puzzle must be significantly shorter than T.

To our best knowledge, all existing methods of constructing time-lock puzzles employ some form of algebraic structure. The question arises of whether it is possible to design TLPs or similar time-lock mechanisms using a black-box approach. Research by Mahmoody et al. [113] suggests that within the random-oracle model, it is impossible to design TLPs using one-way permutations and collision-resistant hash functions. They essentially rule out the possibility of black-box construction of TLPs. We explore the (im)possibility and further details in Section 3.3.4.

Homomorphic Time-Lock Puzzles

More recently, the notion of the Homomorphic Time-Lock Puzzle (HTLP) has been proposed as a TLP with homomorphic and compactness properties. As introduced in [116], this proposal expands the practical application of time-lock puzzles.

Using this framework, multiple senders can generate several independent puzzles (x_1, ..., x_n) with a delay T, each of which is associated with a secret k_i for $i \in [n]$. HTLP allows the aggregation of these puzzles into a single puzzle (i.e., $\hat{x}$) that requires only a single solution. More specifically, without knowing the secrets $(k_1, \ldots, k_n)$, a circuit C can be homomorphically evaluated over these puzzles to obtain an aggregated puzzle $\hat{x}$, whose solution is $C(k_1, \ldots, k_n)$. For TLP to be HTLP, the circuit size must be independent of the time delay (i.e., T) required to solve the puzzle $\hat{x}$ (i.e., to obtain $C(k_1, \ldots, k_n)$). This property is known as compactness.

The most obvious advantage of homomorphic time-lock puzzle is that it allows one to solve a single puzzle to obtain an aggregated solution, whereas individual puzzle solutions remain unknown. Thus, HTLP can be used effectively in many applications, such as sealed-bid auctions and e-voting. Recently, HTLP has been used to construct a number of cryptographic protocols, including fair and sound secret sharing [96] and verifiable timed signatures [148].

Since Malavolta and Thyagarajan's proposal, there have been different improvements such as in [29, 109].

3.3.2 Proof of Sequential Works

In 2013, the notion of Proof of Sequential Work (PoSW) was first introduced by Mahmoody et al. [114]. PoSW works primarily as a *proof system* rather than a *function*, allowing a prover to convince a verifier that it received (or became aware of) input information χ at least T-sequential computational steps *ago, and not before*. For PoSW, it is required that the verification time of the proof be much less than T, specifically $\mathsf{Poly}(\log(T))$. In general, most PoSW constructions rely on (or build upon) some form of data structure in their proof construction, such as a depth-robust graph, a Merkle tree with extra edges, and a skip list-like structure.

Mahmoody et al. [114] construct PoSW within the random oracle model. In particular, their construction relied on the labeling of a depth-robust graph. Let $\mathsf{H}_\chi(\cdot)$ be a hash function modeled as a random oracle $\mathcal{H}(\chi, \cdot)$, and let G be a directed acyclic graph with T nodes. Given an initial input x, the prover computes all labels $l_1, \ldots, l_T$, where the initial node label is defined as $l_1 = \mathsf{H}_\chi(1 \,\|\, x)$, and for each subsequent label l_i, it is computed as $\mathsf{H}_\chi(i \,\|\, l_{p_1}, \ldots, l_{p_d})$, where $l_{p_1}, \ldots, l_{p_d}$ are the labels of l_i's parent nodes in the graph. The prover computes y to be the root of a binary tree built on all T labels. Then, the output y is sent to a verifier, who responds with a random subset $S \subset [T]$ as a challenge. For each $i \in S$, the prover sends back l'_i along with its path to the root y and its parents $l'_{p_1}, \ldots, l'_{p_d}$. The

verifier then checks the validity by recomputing l'_i and verifying if $l'_i = \mathsf{H}_\chi(i \,\|\, l'_{p_1}, \ldots, l'_{p_d})$. A key limitation of this construction and those that follow is that the proof does not verify the *correctness* of the output y; instead, it ensures that a total of T-sequential steps were completed to obtain this *valid* proof. To verify the output's correctness, one must also perform the T-sequential steps. This implies that the proof of sequential work is more of a *proof system* than a *function*.

Later, Cohen and Pietrzak [51] proposed an efficient and practical construction of PoSW that incorporates a modified Merkle tree structure as a replacement for depth-robust graphs. Other constructions for proofs of sequential work have been proposed, such as [2] and [66], each of which offers distinct properties. Building on the work of [104], the former uses the random permutation model instead of the random oracle (and is constructed with *skip-lists* data structure), allowing for computations in both forward and backward directions, which can be useful in applications such as the *proof of replica* [2]. In the latter, the notion of *incremental proofs of sequential work* is introduced, which allows a prover to incrementally carry on a previous prover's computation, without increasing the resources required for individual provers or the size of the proofs [66].

3.3.3 Delay-Based Encryption

In 2021, Burdges et al. [33] introduced the notion of Delay Encryption, a time-locked variant of identity-based encryption. Delay encryption involves delaying the extraction of a private key (or a session key) from a *random unpredictable* session identifier (*id*) using a puzzle derived from a long sequential function. Anyone with a session *id* can encrypt messages for that session. By solving the puzzle, the session key can be obtained, which can then be used to decrypt the messages identified with the session *id*.

In [33], the sequential function used is based on a dual isogeny of a large degree defined over $\mathbb{F}_{p^2}$, which required 12 TiB of storage to define the sequential function with a target delay of 1 hour. This estimate is based on $T = 10^7$ isogeny evaluations, with the assumption that one 2-isogeny evaluation takes 50 ns. As a result, adopting such a construction presents a significant challenge. Another disadvantage is that it requires a trusted setup with computational time dependent on the difficulty T.

3.3.4 Verifiable Delay Functions

In their pioneering work, Boneh et al. [25] introduced the notion of Verifiable Delay Function (VDF) as primitives designed to ensure a guaranteed and verifiable delay in de-

centralized systems. A VDF requires a predetermined amount of time delay to evaluate (so it is inherently sequential function), but its correctness can be efficiently and publicly verified using a separate, fast verification process. Their paper does not include a construction resembling a VDF, but rather explored the notion itself and potential approaches for its construction.

Unlike TLP, VDF does not require a short function to predict the puzzle solution beforehand. Instead, VDF requires an efficient method to verify the correctness of the puzzle solution. The main idea behind VDFs is to ensure that the solution remains unknown to all parties until it is computed, yet the correctness of any proposed solution can be efficiently and publicly verified. Typically, VDFs can be built by combining time-lock puzzles with proof of correct computation, forming what is formally known as a *trapdoor-VDF*. The main restriction is that the trapdoor (or the TLP's short function, known as a trapdoor) must remain unknown to all parties. This allows VDF to be a useful tool in various applications such as randomness beacon, proof-of-work system, and timestamping.

Building on the work of Boneh et al. [25], a variety of VDF constructions have been proposed that built upon the time-lock assumption by Rivest, Shamir, and Wagner [131], including [24], [69], [110], [128], [133], and [156]. Each of these introduces unique verification processes, new properties, and various improvements.

De Feo [72] proposed the first VDF based on an isogeny walk over a graph(s) defined over a finite field. In [72], the verification process is based on the bilinear pairing of the Boneh–Lynn–Shacham (BLS) signature scheme [28]. As a result of the work of [72], other isogeny-based constructions have been developed [44, 60, 72, 137]. In Section 3.4, we present a number of verification techniques from the aforementioned works.

In [44], Chavez-Saab et al. propose a post-quantum secure VDF built upon an isogeny-based sequential function combined with inefficient succinct non-interactive arguments (SNARGs) for verification. Decru, Maino, and Sanso [60] have proposed another quantum-secure construction, which implements an efficient verification approach based on the computation of isogenies between products of elliptic curves. However, a notable disadvantage of their VDF construction is that it requires $O(T)$ parallelism to be evaluated in parallel time T. Another VDF construction, proposed by Tan et al. [145], is based on ZKBoo, a zero-knowledge proof system for verifiable computation [85]. Their proposal, known as ZKBdf [145], introduces a quantum-secure VDF that does not require a trusted setup.

Other Verifiable Delay Functions

This section presents a scan of VDF proposals with some relaxed properties (a.k.a. weaker VDF). Of course, some might not consider a weaker VDF as a VDF.

The work of Lenstra and Wesolowski [104]. Let f be a function such that the inverse computation of f (i.e., f^{-1}) is cheaper than f. Fundamentally, the [104] construction is based on f evaluation T-times in one computation direction and verification in the opposite direction.

$$x_0 \to f(x_0) \to f(x_1) \to f(x_2) \to \cdots \to f(x_T) = y$$

where $x_{i+1} = f(x_i)$, and $x_i \in \mathcal{X}$, $y \in \mathcal{Y}$. To verify, we check $x_0 =^? f^{-1} \circ f^{-1} \circ \cdots f^{-1}(y)$ One observation is that the verification time is linear in the computation advantage of f^{-1} over f (and not exponential).

Lenstra and Wesolowski [104] choose $f(x)$ to be $\mathbf{Per}(\sqrt{x})$, where $\mathbf{Per} : \mathbb{F}_p \to \mathbb{F}_p$ is an efficient random permutation and $\sqrt{x}$ is the square root in a prime field $\mathbb{F}_p$. Computing $\sqrt{x}$ requires $\log(p)$ squaring, but computing its inverse requires only one square operation in $\mathbb{F}_p$.

Furthermore, this construction is based on random permutations and algebraic assumptions. In our discussion section, we examine the advantages of having a [104]-like construction with f that is solely based on random permutation/inefficient one-way functions with known secret information.

The work of [25]. In their paper, Boneh et al. proposed some example constructions of VDFs. One construction is based on Incrementally Verifiable Computation with the succinct argument SNARG. In general, the cost of computing the proof is higher than the actual evaluation (around 100,000 times); this shows the need to achieve efficient verification. An impractical amount of parallel processing is required to close the gap between evaluation time and prove computation.

The second construction is based on injective[4] rational maps. This VDF sequential work is a GCD computation to invert a permutation polynomial of degree T (i.e., given a polynomial F and an output x, compute y such that $F(y) = x$). The evaluation requires $T - O(T)$ steps on parallel T^2 processors; otherwise, $O(T^2)$ steps (see [25], Assumption 2). The verification is polynomial evaluation in $O(\log(T))$ operation (i.e., given F, x, and y, check $F(y) =^? x$).

[4]The reason for the injective map is to ensure unique output.

A Note on Random Oracle Constructions

In previous sections, we noted that all of the presented proposals are based on some algebraic structures. Then a question that naturally arises: Is it possible to construct VDF from an unstructured (black-box) assumption such as hash functions?

There have been several attempts to construct VDF, such as VDF from random permutations [104, 2] and hash functions [114, 51, 66]. Although the proposed constructions are sequential, they are inefficient at verifying the uniqueness of the output, meaning the evaluation must be redone to ensure that a proposed answer is the correct one for a given challenge. Consequently, they are referred to as *proof of sequential work*. In their paper [115], Mahmoody et al. have tackled the open problem of constructing VDF *solely*[5] from a random oracle. One of their results rules out the existence of such construction that fully satisfies the uniqueness property.

Another work by [65, 115] studies the *tight*ness of VDF constructions. A VDF is tight if the evaluation time of $\mathsf{SeqEval}$ is not much more than or less than T. Unlike all current VDF constructions, their main result is negative on the existence of a tight VDF construction based solely on a random oracle model (see [65, 115] for more information).

3.3.5 Discussion

All of the aforementioned notions have distinct properties as summarized in Table 3.1; however, they also intersect in some aspects. For instance, both VDF and PoSW prevent the prediction of a challenge's solution before the long sequential work is completed, thus requiring an efficient verification mechanism. While PoSW verifies that sequential work has been done, VDF also verifies the correctness of the challenge solution. Hence, one can say that VDF is a PoSW with a unique proof system.

Unlike VDFs and PoSWs, TLPs require a short function to predict the puzzle solution beforehand but do not necessarily require a verification mechanism. Hence, TLPs are not publicly verifiable (i.e., considered privately verifiable) because the verification process requires knowledge of the secret short function, that is, the secret trapdoor.

Typically, VDFs can be built by combining time-lock puzzles with proof of correct computation, forming what is formally known as a trapdoor VDF. The main restriction is that the trapdoor (or the TLP's short function) must remain unknown to all parties.

[5] Solely in the sense that there are no additional structured assumptions.

Most delay-based primitives struggle with scalability because each challenge requires its own sequential work to compute the solution. However, delay encryption and homomorphic time-lock puzzles are notable exceptions to this issue. Interestingly, to the best of our knowledge, there is only one construction of delay encryption, which has inefficient setup and a large storage overhead.

	Key Properties	Verification	Additional Notes
PoSW	Ensures their output, in response to a random challenge, stays unpredictable until T sequential work is completed.	Efficient and public proof of time spent on completing the work.	Constructed using black-box assumptions; does not guarantee output uniqueness.
VDF		Efficient and public ensuring uniqueness of the output.	Functions as a PoSW with a public proof ensuring output uniqueness.
TLP	Enables encapsulation to the future via a short function (secret trapdoor) that predicts the puzzle solution in advance.	Verification requires knowledge of the secret trapdoor, making it privately verifiable.	Can be combined with proof of correct computation to form trapdoor-VDFs or VDFs under a trusted setup model.
Delay Encryption	Enables encapsulation to the future and addresses scalability issues by allowing multiple encapsulations to the future using a single challenge.	Does not require public verification; however, it can incorporate proof of correct computation, similar to TLP.	Each challenge requires a random and unpredictable session identifier; current implementations are inefficient and require large storage.
Homomorphic TLP	Enables encapsulation to the future and addresses scalability issues by aggregating multiple challenges into a single one.		As a result of an aggregated puzzle solution, more practical applications can be realized, such as fair secret sharing, multi-party coin flipping, contract signing, and electronic voting.

Table 3.1: Summary of delay-based cryptographic primitives.

3.4 Proofs of Correct Computation

Proofs of correct computation are essential tools for developing different applications and constructing primitives, including the one discussed in the previous sections. For example, while a sequential function can be developed to guarantee a predetermined delay, a more complete solution might require an efficient proof system to verify the amount of time spent and/or the correctness of the output. In this section, we will explore several

proof techniques used in delay-based primitives, classifying them based on their underlying structures.

3.4.1 Soundness in Unknown Order Group

Through our discussions, we have established that in delay-based cryptographic primitives, the groups used must be such that it is infeasible, within a negligible function of λ, to shorten the computation time of a long function. In this section, we will show that these groups must also maintain properties such as the *low order assumption* or the *adaptive root assumption*, as discussed in [128, 69] and [156], respectively.

Adopted from [26], both assumptions are defined in the following.

Definition 5 (Low Order Assumption)**.** *Let* Setup *be a randomized algorithm that, on input* 1^λ*, outputs a description of a randomly generated group* G*. The low order assumption is an assumption based on the infeasibility of finding a low order element* $x \in G$ *(other than* $\pm 1_G$*) with order* $\mathrm{ord}(x)$ *greater than* 1 *and less than* 2^λ*. Formally, for any probabilistic polynomial-time adversary* A*, the probability*

$$\Pr\left[\begin{array}{l|l} x^{\mathrm{ord}(x)} = 1_G, & G \leftarrow_\$ \mathsf{Setup}(1^\lambda), \\ x \neq \pm 1_G, & (x, \mathrm{ord}(x)) \leftarrow \mathsf{A}(G) \\ 1 < \mathrm{ord}(x) < 2^\lambda & \end{array}\right]$$

is a negligible function of λ*.*

Definition 6 (Adaptive Root Assumption)**.** *Let* G *be a description of randomly generated group from a randomized algorithm* $\mathsf{Setup}(1^\lambda)$*. The adaptive root assumption holds for* $G \leftarrow_\$ \mathsf{Setup}(1^\lambda)$ *if no efficient adversary* $\mathcal{A} := (\mathsf{A}_1, \mathsf{A}_2)$ *has more than a negligible chance of winning the following game: i)* A_1 *outputs an element* $w \in G$*; ii) a random prime is chosen (*$l \leftarrow_\$ \mathsf{Primes}(2^\lambda)$*); iii)* A_2 *outputs* $u = w^{1/l} \in G$*. Formally, for any probabilistic polynomial-time adversary* $\mathcal{A} = (\mathsf{A}_1, \mathsf{A}_2)$*, the probability*

$$\mathsf{Pr}\left[w \neq 1_G,\, u = w^{1/l} \in G : \begin{array}{l} G \leftarrow_\$ \mathsf{Setup}(1^\lambda) \\ (w, \mathsf{state}) \leftarrow \mathsf{A}_1(G) \\ l \leftarrow_\$ \mathsf{Primes}(2^\lambda) \\ u \leftarrow \mathsf{A}_2(w, \mathsf{state}, l) \end{array}\right]$$

is a negligible function of λ*.*

The adaptive root assumption is closely related to the strong RSA assumption. Essentially, the strong RSA assumption states that for a given RSA modulus N and an element w from the multiplicative group of integers modulo N, denoted as $G := \mathbb{Z}_N^*$, it is infeasible to find both a new exponent l and a new base u that satisfies $u^l \equiv w \pmod{N}$. On the other hand, the adaptive root assumption asks to find the element $u \in G$, given w and l, such that $u \equiv w^{1/l} \pmod{N}$.

Pietrzak [128] suggests using an RSA modulus to be a composite of two safe primes for an RSA group (e.g., the prime p of form $p = 2p' + 1$ where p' is prime). Accordingly, the low order assumption should always hold in the RSA groups of quadratic residue. Further, to generate an RSA group, a *trusted party* must run the Setup algorithm to find N and forget N's factorization.

To eliminate the need for a trusted setup, Wesolowski [156] proposes using the class group of an imaginary quadratic number field[6]. Based on the fastest algorithm, the complexity of finding the order of the class group of an imaginary quadratic field of discriminant $-p$ is $L_{|p|}(1/2) = \exp(O(\log(|p|)^{1/2} \log\log(|p|)^{1/2}))$ (see [86, 151], and [156] Section 1.1)

The following subsections discuss a variety of interactive proving arguments proposed in [128], [69], [156], and [24].

Pietrzak's Construction [128]

An honest Evaluator must convince a Verifier that his evaluation output y_0 is indeed $x_0^{2^{T_0}} \in G$ such that $y_0, x_0 \in G$ and $T_0 = T$. The proof statement is represented as $(x_0, T_0, y_0) \in \mathcal{L}_G$ where $\mathcal{L}_G = \{(x_i, t, y_i) : y_i = x_i^{2^t}, x_i \in G\}$. The protocol is recursive, as it proves a sequence of sub-statements of the original one. With each iteration, we reduce the proof from proving T_i sequential steps to $T_{i+1} = T_i/2$ at the cost of having two proving statements instead of just one (i.e., we add $(x_{i+1}, T_{i+1}, y_{i+1})$ to the original statement (x_i, T_i, y_i)). The proof is tailed in that proving a sub-statement $(x_{i+1}, T_{i+1}, y_{i+1})$ implies proving the original statement (x_i, T_i, y_i).

A single iteration of the proving protocol proceeds as in Figure 3.1, where we assume that T is even, and the i^{th} statement is $(x_i, T_i, y_i) \in \mathcal{L}_G$.

A total of $\log_2(T)$ iterations are necessary to fully prove the statement (x_0, T, y_0), at a cost of $\log_2(T)$ new statements. In the last step (with the statement of $(x_{\log_2 T}, 1, y_{\log_2 T})$), the verifier checks that $y_{\log_2 T} = x_{\log_2 T}^2 \in G$ holds in order to accept the proof.

[6] it is $\mathbb{Q}(\sqrt{-p})$ of discriminant $-p$, where if p is prime and $p \equiv 3 \bmod 4$, it ensures an odd order [156].

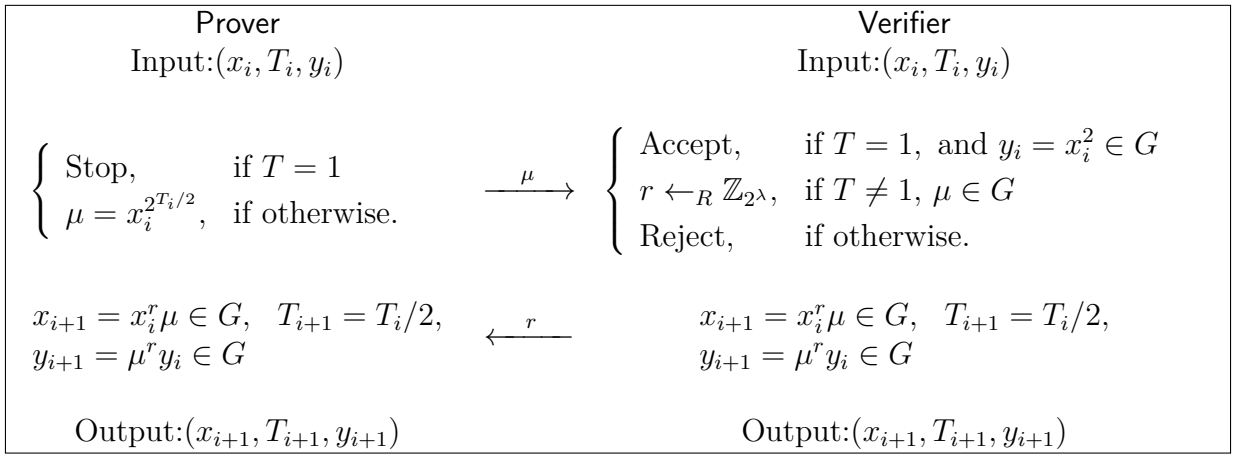


Figure 3.1: i^{th} iteration of Pietrzak's interactive succinct argument.

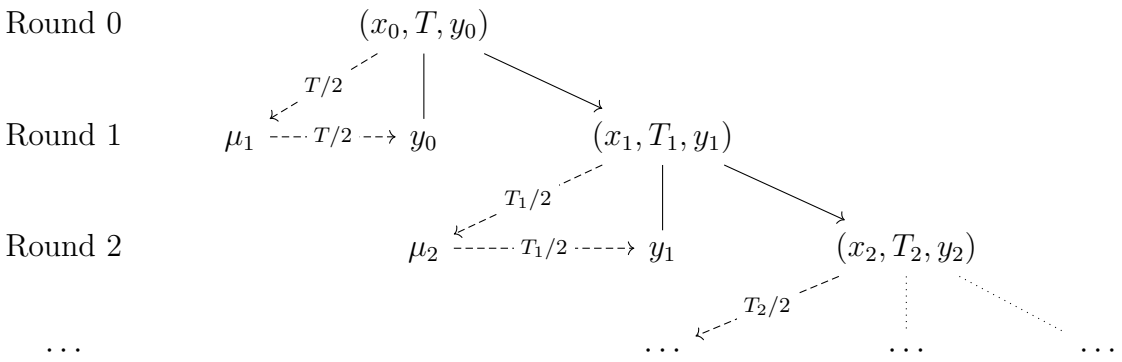


Figure 3.2: A tree representation of [128] proving protocol where it shows that verification is only done recursively on $y_i = (x_{i-1}^{r_i} \mu_i)^{2^{T_i}} \in G$ to prove the final evaluation result, where $\mu_0 = r_0 = 1$.

The beauty of [128]'s succinct argument is that it only requires proving one part of the statement $y_i = (x_{i-1}^{r_i} \mu_i)^{2^{T/2^i}}$ to validate $y_0 = x_0^{2^T}$. That is because $y_i = (x_{i-1}^{r_i} \mu_i)^{2^{T/2^i}}$ will be valid if and only if both μ_i and y_{i-1} are computed honestly. Thus, we do not need to recursively prove either of the exponentiations $\mu_i = x_{i-1}^{2^{T/2^i}}$, or $y_{i-1} = \mu_i^{2^{T/2^i}}$ for i^{th} computation step, see Figure 3.2.

Non-Interactive Argument. Fiat-Shamir heuristic can be applied to obtain a non-interactive proof argument. The prover and verifier generate a random $r_i \leftarrow_\$ \mathbb{Z}_{2^\lambda}$, $\forall i \in$

$[\log T]$ by hashing the last input statement (x_i, T_i, y_i) for each iteration.

Efficiency of the Prover. During each iteration, the prover must compute μ_i, which can fortunately be done using an intermediate calculation of $y_0 = x_0^{2^T}$. For example, μ_3 can be computed as:

$$\mu_3 = x_2^{2^{T/8}} = (x_0^{2^{T/8}})^{r_1 r_2} \, (x_0^{2^{3T/8}})^{r_1} \, (x_0^{2^{5T/8}})^{r2} \, x_0^{2^{7T/8}}$$

We can therefore store $x_0^{2^{T/8}}$, $x_0^{2^{3T/8}}$, $x_0^{2^{5T/8}}$, and $x_0^{2^{7T/8}}$ and use them later to compute μ_3. Pietrzak [128] stores around $s \approx \log_2 \sqrt{T}$ to optimize computation and storage requirements. Instead of all T elements, the evaluator stores only 2^s group elements $\{x_0^{2^{\frac{iT}{2^s}}}\}$, $\forall i \in \{0, \cdots, 2^s - 1\}$, where $s \in \mathbb{Z}_{\log_2 T}$.

To construct the proof statements, we have to compute μ_i for all $i \in [2^s - 1]$ using all 2^s-stored elements. This costs 2^s group exponentiations of a size approximately $\mathsf{Poly}(\lambda)$ (the average exponent size is actually $\lambda(s-1)/2$). Further, it takes a total of $T/2^s$ group multiplications [26] to compute the remaining elements $\{\mu_{s+1}, \cdots, \mu_{\log_2 T}\}$. Hence, the full proof requires around $2^s + T/2^s$ group multiplications. The proof requires around $2\sqrt{T}$ multiplications for $s = \log_2 \sqrt{T}$.

Efficiency of the Verifier. In each of the $\log_2 T$ rounds, we must perform two group exponentiations of size λ to compute $x_i = x_{i-1}^{r_i} \mu_i$ and $y_i = \mu_i^{r_i} y_{i-1}$.

Proof Soundness. Let us consider an attack of [26] that finds a low-order element $g \in G$. The attacker $\mathcal{A}$'s target is to send a malicious answer $(y_0' \neq x_0^T)$ other than the correct answer y (i.e., $\mathcal{A}$ sends the incorrect statement $(x_0, T, y_0') \notin \mathcal{L}_G$ that passes the verification). An attack proceeds as follows:

- The prover finds $g \leftarrow \mathcal{A}$ of small order $\mathsf{ord}(g)$.
- The prover sends $\mu = g x_0^{2^{T/2}} \in G$, and $y_0' = g y_0 \in G$.
- As described in the protocol, the verifier computes $x_1 = x_0^r \mu \in G$, $T_1 = T/2$, and $y_1 = \mu^r y_i g \in G$
- The prover and the verifier have the new malicious statement (G, x_1, T_{i+1}, y_1) that *might* pass the verification.

For the following reasons:

$$\begin{aligned} x_1^{2^{T_1}} &=^? y_1 \\ (x_0^r \mu)^{2^{T/2}} &=^? \mu^r y_0 g \\ (x_0^{r+2^{T/2}})^{2^{T/2}} g^{2^{T/2}} &=^? \mu^r y_0 g \\ (x_0^{r2^{T/2}+2^T}) g^{2^{T/2}} &=^? (x_0^{r2^{T_i/2}} y_0) g^{r+1} \end{aligned}$$

the new statement passes the verification with probability $1/\mathsf{ord}(g)$, given that $r+1 \equiv 2^{T/2} \pmod{\mathsf{ord}(g)}$ for a random r.

Hence, the soundness of Pietrzak's proof argument (as pointed out by Boneh et al. [26]) is crucially dependent on the low-order assumption. Pietrzak proposes using the RSA group of quadratic residues to ensure proof soundness

$$QR_N = \{z^2 \bmod N : z \in \mathbb{Z}_N^*\}$$

which has no low order subgroup given that $N = (2p'+1)(2q'+1)$ is a safe prime composite, and QR_N's order is $p'q'$. This implies that the size of $\min(p', q')$ should be at least $\mathsf{Poly}(\lambda)$. As a result, Pietrzak's argument is unconditionally sound. To simplify the proving protocol, Pietrzak also suggests using the group of signed quadratic residues $G = QR_N^+ = \{|z| : z \in QR_N\}$, where $\mathbb{Z}_N^*$ is represented as $\{-\frac{N-1}{2}, \cdots, \frac{N-1}{2}\}$.

Ephraim's Construction [69]

A recent paper by Ephraim et al. [69] proposes a variant of Pietrzak's work [128], namely *Continuous* VDF (cVDF). The variant is based on exponentiation in a group of unknown orders. The cVDF enables verification of intermediate evaluation step $T' < T$. A different entity can then *trust* and *continue* the evaluation from intermediate step T'.

Iteratively, Pietrzak [128] only creates a proving statement for $(y_i = (x_{i-1}^{r_i} \mu_i)^{2^{T_i}})$ to prove the final evaluation result (y_0). In each round, the prover sends a half-computation (μ_i) to generate a new statement (x_i, T_i, y_i), as shown in Figure 3.2. As of yet, we have not completed computing y_0 or the necessary proof elements. Therefore, we cannot validate an intermediate computation step. The solution is to prove the intermediate computation from x_{i-1} to μ_i and from μ_i to y_{i-1} in each round. While computing y_0, we recursively compute and prove all intermediate sub-instances μ_i and y_{i-1}. As an illustration, we provide a toy example with $T = 4$, see Figure 3.3.

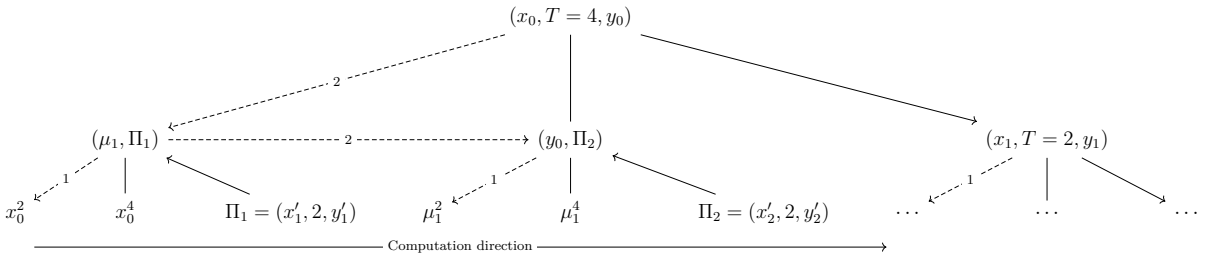


Figure 3.3: A tree representation of [69]'s protocol, with $k = 2$, that shows an intermediate proof computation while computing y_0.

To facilitate optimization, Ephraim et al. [69] divide the proving statement into $(k+1)$ segments that all prove the parent node, where $T = k^d$ (in Pietrzak's work, we have $k = 2$)

Non-Interactive Variant. The Fiat-Shamir heuristic can be used to transform the proof to a non-interactive one.

Protocol Efficiency. The evaluator must hold on to all proving statements for the verification step, which means they must store all intermediate proofs (at most $k \log_k T$). However, an evaluator can disregard the statements (and their segments) of an intermediate step once it has been proven. The size of a proof at any node depends on the height of the tree [69]. Unlike [128], the efficiency of [69] heavily depends on the parameter chosen (such as k and T). In conclusion, we emphasize that there is still much room for optimization to prove efficiency and proof size.

Protocol Security. Parameter selection is not straightforward; it is instead a tradeoff between security and efficiency. It is because an attacker (or even an honest evaluator) can skip all the intermediate proofs and compute only for y_0. An intermediate computation, for example, requires $(k + 1)^d$, while the malicious computations require k^d.

Wesolowski's Construction [156]

Wesolowski [156] presents a VDF based on a group of unknown order (e.g., an RSA group or the class group of an imaginary quadratic field). The construction is based on solving the time-lock puzzle [131] and computing a publicly verifiable proof of correctness.

Let G be a finite abelian group of unknown order, $\mathcal{H} : \mathcal{X} \to G$ be an efficient random oracle, and $\mathcal{X}$ be sampling space for the input x. The VDF construction of [156] is illustrated in Figure 3.4.

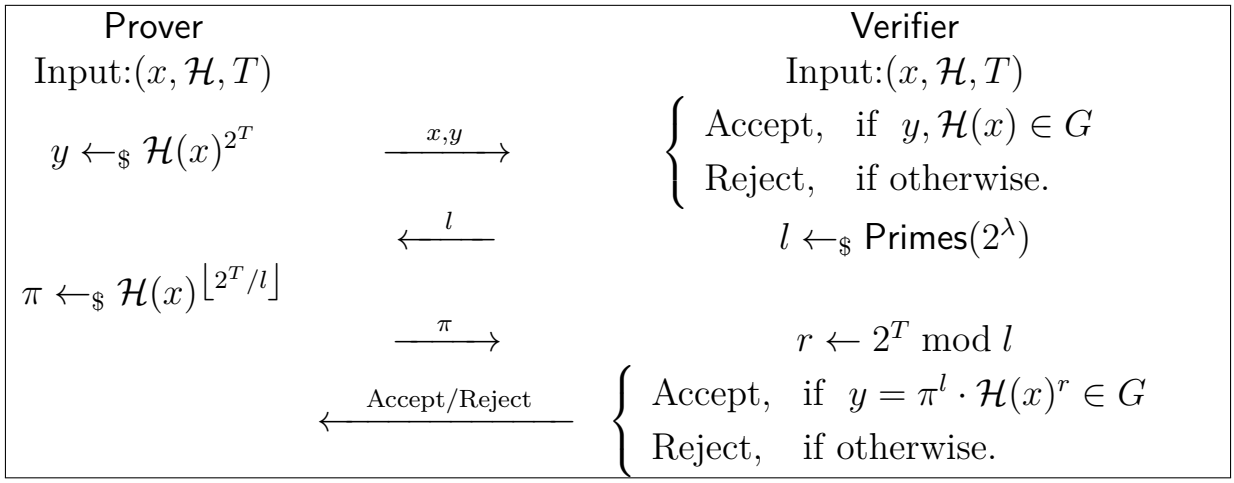


Figure 3.4: Wesolowski's interactive succinct argument [156].

Figure 3.4 clearly shows that the proof size in G is a *single* element (i.e., $\pi \in G$). An honest evaluator can compute y, and the proof works as follows:

$$y = \pi^l \, \mathcal{H}(x)^r = \mathcal{H}(x)^{ql+r} = \mathcal{H}(x)^{2^T} \in G$$

Non-Interactive Argument. We transform the protocol to be non-interactive using the Fiat-Shamir heuristic [74]. Initially, both prover and verifier agree on a random oracle $\mathcal{H}_p : \{0,1\}^* \rightarrow \mathsf{Primes}(2^{\hat{\lambda}})$. In response to the attack described in [26], we set $\hat{\lambda} = 2\lambda$. We remark that it is essential to include the answer as an input to $\mathcal{H}_p$. VDF sequentiality can be compromised if this is not done.

Efficiency of the Prover. With a long-division algorithm, the evaluator computes the proof ($\pi = \mathcal{H}(x)^{\left\lfloor \frac{2^T}{l} \right\rfloor}$) using $2T$ group operations. Boneh et al. [26] suggest using a windowing method that reduces the computation to T. This method can be further improved by using s-based processors (and s space) to T/s. Hence, the time to compute the answer (y) and proof is $(1 + \frac{1}{s})T$ [26].

Efficiency of the Verifier. The verifier has to compute two exponentiation in the group G and one exponentiation in $\mathbb{Z}_l$.

Proof Soundness. Due to the homomorphism property, it is essential to notice that we do not perform the computation directly on the input x. Instead, they use a random oracle

$\mathcal{H}$ (i.e., $y = \mathcal{H}(x)^{2^T}$). It is because an attacker can predict the answer y', given a previous answer y in less than T steps (i.e., given an evaluation $y = x^{2^T} \in G$, an adversary can compute $y' = y^\gamma \in G$ where $x' = x^\gamma \in G$ for any positive integer γ).

The soundness of the succinct argument relies on the assumption that it is hard to find u, such that $u \neq 0, \pm 1_G$, and u is the l-th roots of an element in G for a prime $l \leftarrow_\$ \mathsf{Primes}(2^{\hat{\lambda}})$ sampled uniformly at random. To be more specific, the adversary aims to send an incorrect answer (y') based on forged valid proof (π') such that

$$y' = \pi'^l \mathcal{H}(x)^r = w\mathcal{H}(x)^{2^T} = wy \in G$$

where $w = u^l \in G$, and $y = \mathcal{H}(x)^{2^T}$. The adversary forges the proof to be $\pi' = u\,\mathcal{H}(x)^q$. The verifier will accept π' as valid proof because of the following:

$$\pi'^l \mathcal{H}(x)^r = (u\,\mathcal{H}(x)^q)^l\,\mathcal{H}(x)^r = w\,\mathcal{H}(x)^{2^T} \in G$$

Therefore, the hardness of finding the l-th root assumption in the selected group G of unknown order is crucial for the soundness of the succinct argument.

Block et al. Construction [24]

Block et al. [24] proposed a variant of the Diophantine Argument of Knowledge (DARK)[7] polynomial commitment scheme [32]. Initially, DARK is a proving system that proves the knowledge of an integer $a \in \mathbb{Z}$ to a system of polynomials f such that $f(a) = 0$; DARK uses groups of unknown order to commit to f.

DARK. In short, DARK is conceptually quite similar to [128]. Iteratively, we prove the knowledge of a polynomial (f) by committing to f_i and then proving f_i^L and f_i^R's knowledge, such that $f_i = f_i^L || f_i^R$ [8], and $f_{i+1} = \alpha f_i^L + f_i^R$. In each iteration, DARK uses [156]-like proof of exponentiation (PoE) to validate $f_i = f_i^L || f_i^R$.

In a nutshell, Block et al. [24]'s variant of DARK replaces [156] PoE with a modified Pietrzak's PoE (mPoE). mPoE is interactive with $\log(T)$ rounds, as is Pietrzak's PoE. However, in each mPoE round, the prover commits to proving λ statements instead of just one [128]'s statement (i.e., the i^{th} iteration statement to be proven is the set $\{(x_{i,j}, T_i, y_{i,j})\}_{j\in[\lambda]} \in \mathcal{L}_G^\lambda$ where $\mathcal{L}_G = \{(x, t, y) : x \in G, y = x^{q^t} \in G\}$). Hence, the protocol consists of proving all λ statements.

[7] Diophantine refers to all polynomials over integers, where all coefficients and variables are integers.

[8] This notion is to represent an encoding of f_i as $f_i^L + q^N f_i^R$ where q is a large integer.

The modified PoE is of interest in our context since it can be used to instantiate a VDF. Let $\star$ be the subset product of elements (e.g., $\{u_j\}_{j\in[\lambda]} \star \{g\}_{j\in[\lambda]} = \prod_{j\in[\lambda]} g_j^{u_j}$, where we use $\{u_j\}_{j\in[\lambda]}$ as the sampler $\{u_j\}_{j\in[\lambda]} \in \{0,1\}^\lambda$). Let $\odot$ be the pair-wise product (e.g., $\{g_j\}_{j\in[\lambda]} \odot \{h\}_{j\in[\lambda]} = \{g_j h_j\}_{j\in[\lambda]}$). The mPoE protocol is illustrated in Figure 3.5.

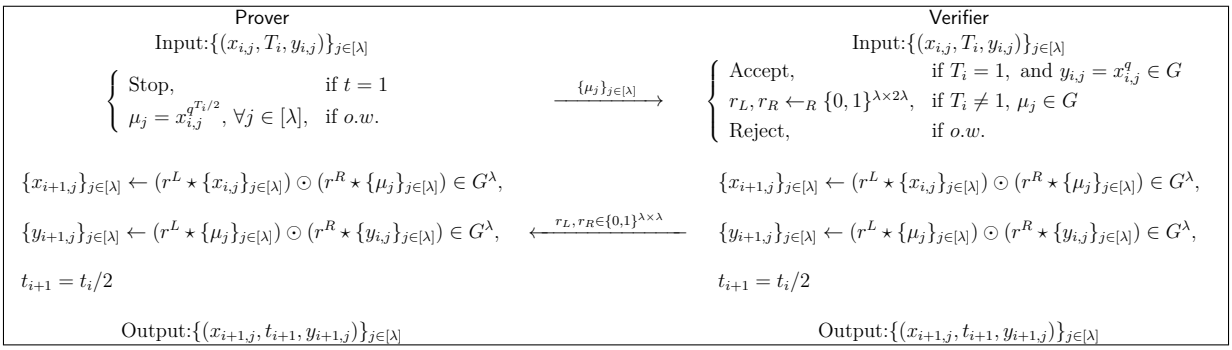


Figure 3.5: i^{th} iteration of modified Pietrzak-like interactive succinct argument (mPoE).

Non-Interactive Argument. The scheme is interactive with $O(\log T)$ proving rounds and communication complexity $\mathsf{Poly}(\lambda, \log T)$. However, applying the Fiat-Shamir heuristic, the protocol can be non-interactive.

Efficiency of the Prover. In contrast to the i^{th}-round one statement of [128], the prover has to generate λ proving statements $\{(x_{i,j}, T_i, y_{i,j})\}_{i\in[\lambda]} \in \mathcal{L}_G^\lambda$ where $\mathcal{L}_G = \{(x, t, y) : x \in G, y = x^{q^t} \in G\}$. To reduce the prover's VDF evaluation, we can argue that the starting statement ($\{(x_{0,j}, T, y_{0,j})\}_{i\in[\lambda]}$) can be λ a copy of [128]'s statement $(x, t, y) \in \mathcal{L}_G$ (i.e., we can set $x_{0,1} = \cdots = x_{0,\lambda}$). Compared to [128], this , however, adds an overhead of λ statements to the communication complexity per round (i.e., the complexity is $\mathsf{Poly}(\lambda, \log T)$). Further, the prover runs in time $T\mathsf{Poly}(\lambda, \log T)$ with a space of $\mathsf{Poly}(\lambda, \log T)$ group element.

Efficiency of the Verifier. In each $\log T$ round, the verifier has to sample and compute a subset product of $\{(x_{i,j}, T_i, y_{i,j})\}_{i\in[\lambda]}$ for λ-times. This adds an approximate factor of λ overhead in comparison to the original [128]. Hence, the running time of the verifier is $\mathsf{Poly}(\lambda, \log T)$.

Proof of Security. In a nutshell, the difference of mPoE from [128]' PoE is selecting random linear combinations of group elements rather than one group element. This fact is used to prove the soundness of mPoE. To do so, each element of $(i+1)^{th}$ statement is

equivalents to

$$x_{i+1,j} = \prod_{z_i \in S_x} z_i \prod_{m_i \in S_\mu} m_i$$

$$y_{i+1,j} = \prod_{w_i \in S_y} w_i \prod_{m_i \in S_\mu} m_i$$

such that $(x_{i+1,j}, T_i, y_{i+1,j}) \in \mathcal{L}_G$, where S_x, S_y, and S_μ are random subsets of $\{x_{i,j}\}_{j\in[\lambda]}$, $\{y_{i,j}\}_{j\in[\lambda]}$, and $\{\mu_j\}_{j\in[\lambda]}$, respectively.

Actually, the statement $(x_{i+1,j}, T_i, y_{i+1,j}) \notin \mathcal{L}_G$ is a valid statement with probability at most $1/2$:

$$\mathsf{Pr}[x_{i+1,j}^{T_i} = y_{i+1,j}] = \mathsf{Pr}[x_{i+1,j}^{T_i} y_{i+1,j}^{-1} = 1_G] \leq 1/2$$

such that at least one of $x_{i,j} \in S_x$ (or $\mu_j \in S_\mu$) is malicious, and at least one of $x_{i,j} \in S_x$ is not 1_G (if all $x_{i,j} \in S_x$ (and $\mu_j \in S_\mu$) are legitimate, it is obvious that the probability is 1). The above holds by the random subset product principle (see [24], Fact 8.1).

Hence, the new λ statements are constructed by the product of a random subset from the previous λ statements. As in Figure 3.5, both prover and verifier compute

$$\{x_{i+1,j}\}_{j\in[\lambda]} \leftarrow (r^L \star \{x_{i,j}\}_{j\in[\lambda]}) \odot (r^R \star \{\mu_j\}_{j\in[\lambda]})$$
$$\{x_{i+1,j}\}_{j\in[\lambda]} \leftarrow (\prod_{k\in[\lambda]} x_{i,k}^{r^L[k,j]}) \odot (\prod_{k\in[\lambda]} \mu_k^{r^R[k,j]})$$

and

$$\{y_{i+1,j}\}_{j\in[\lambda]} \leftarrow (r^L \star \{\mu_j\}_{j\in[\lambda]}) \odot (r^R \star \{y_{i,j}\}_{j\in[\lambda]})$$
$$\{y_{i+1,j}\}_{j\in[\lambda]} \leftarrow (\prod_{k\in[\lambda]} \mu_k^{r^L[k,j]}) \odot (\prod_{k\in[\lambda]} y_{i,k}^{r^R[k,j]})$$

where both r^L and r^R are the product subset sampler, which is also a matrix $\in \{0,1\}^{\lambda\times\lambda}$. As a result, the statement $\{(x_{i+1,j}, t_{i+1}, y_{i+1,j})\}_{j\in[\lambda]} \notin \mathcal{L}_G^\lambda$ passes the verification with a probability of at most $(1/2)^\lambda$.

As a direct result of the above, it appears that the security implication of low order elements is *reduced* to the principle of a random subset product from a sequence of group elements. Thus, mPoE can construct VDF in generic groups of unknown order G (i.e., group G might not need to preserve the low order assumption). Hence, the mPoE is statistically sound in the statistical parameter λ.

3.4.2 Soundness in Isogeny Based Construction

De Feo et al. Construction [72]

In their paper, De Feo et al. present a proof-of-concept implementation of two VDF based on isogeny graphs of supersingular elliptic curves (one of which is defined over $\mathbb{F}_p$ and the other over $\mathbb{F}_{p^2}$) and bilinear pairings. Fundamentally, the VDF succinct argument is inspired by the BLS signature scheme [28]. Let N be a large prime. Let P and Q be points on $E[N] \cap E(\mathbb{F}_{p^2}/\mathbb{F}_p)$, and $E_T[N] \cap E_T(\mathbb{F}_p)$, respectively. Given that $\hat{\phi}$ is the dual isogeny, the proving protocol goes as in Figure 3.6 and 3.7.

Prover		Verifier
Input:$(E, E_T, \hat{\phi}, P, \phi(P), Q)$		Input:$(E, E_T, \hat{\phi}, P, \phi(P), Q)$
Compute: $\hat{\phi}(Q)$	$\xrightarrow{\hat{\phi}(Q)}$	Accept, if $\hat{\phi}(Q) \in E[N] \cap E(\mathbb{F}_p)$, and $e(P, \hat{\phi}(Q)) = e(\phi(P), Q)$ Reject, if otherwise.

Figure 3.6: De Feo et al. succinct argument for isogeny graphs of supersingular elliptic curves over $\mathbb{F}_p$ [72].

Prover		Verifier
Input:$(E, E_T, \hat{\phi}, P, \phi(P), Q)$		Input:$(E, E_T, \hat{\phi}, P, \phi(P), Q)$
Compute: $Q' = \mathsf{Tr} \circ \hat{\phi}(Q)$	$\xrightarrow{Q'}$	Accept, if $Q' \in E[N] \cap E(\mathbb{F}_p)$, and $e(P, Q') = e(\phi(P), Q)^2$ Reject, if otherwise.

Figure 3.7: De Feo et al. succinct argument for isogeny graphs of supersingular elliptic curves over $\mathbb{F}_{p^2}$ [72].

The evaluation of $\phi(P)$ is a fixed part of the public parameters. Therefore, it is necessary for a trusted setup to pre-compute the long isogeny ϕ. Consequently, neither construction can be *incremental*, i.e., it cannot support multiple difficulties other than T.

Also, for the VDF over $\mathbb{F}_{p^2}$, the trace map ($\mathsf{Tr} : E/\mathbb{F}_{p^2} \to E/\mathbb{F}_p$) maps all N subgroups of $E[N] \cap E(\mathbb{F}_{p^2}/\mathbb{F}_p)$ to the base group. Thus this construction is not decodable. Finally,

for the VDF over $\mathbb{F}_p$, the only possible definition of ζ-Dec is $\hat{\phi}$, which makes it inefficient (i.e., $\zeta = T$; we do not consider it decodable as well).

Efficiency of the Trusted Setup and Prover. Specifically, [72]'s VDF relies on an isogeny of a very large degree that can be evaluated in sequential time, T. In such an approach, the storage requirement is linear in T (we need $O(T)$ points and curves to describe $\hat{\phi}$). Their Setup algorithm is as expensive as their evaluation process, which runs in $\mathsf{Poly}(T, \lambda)$ instead of $\mathsf{Poly}(\lambda)$.

Efficiency of the Verifier. To check the evaluation correctness, we require two bilinear pairings that run in time polynomial in size of N and p. Their selection is the Tate pairing because they have a large rho-value ($\rho = \log(p)/\log(N)$), and the Tate pairing has a shorter Miller loop.

Scheme's Security. Similar to VDF based on RSA groups, both of their proposals require a trusted setup to compute an isogeny $\phi : E \to E_T$ of degree l^T. To break the succinct argument, an attacker must find a random point such that $e(P, ?) = e(\phi(P), Q)$ holds (or $e(P, ?) = e(\phi(P), Q)^2$ holds). This problem is known as the pairing inversion problem that is strongly related to the size of p and N. The best attack is using Number Field Sieve with complexity $L_p(1/3)$. Since Q is a random challenge, the attacker must re-apply the attack for every new challenge Q.

On the other hand, the attacker can try to break the sequentiality of VDF by finding a shortcut isogeny ($\hat{\phi}' : E_T \to E$) that takes less than T-isogeny steps of a small degree. For graph over $\mathbb{F}_p$, such isogeny can be computed by finding the generators of the ideal class group of $\mathbf{End}_{\mathbb{F}_p}(E)$. The best attack runs in $L_p(1/2)$ due to [91]. For graph over $\mathbb{F}_{p^2}$, the classical attack is the birthday paradox with complexity $O(\sqrt{p})$. The shortcut attack is required once to break the VDF sequentiality.

The attack's last step is to find the endomorphism $\omega \in \mathbf{End}(E)$ such that $\hat{\phi} \equiv \omega \circ \hat{\phi}'$ holds. This step requires the knowledge of $\mathbf{End}(E)$ (e.g., in $\mathbb{F}_p$ graph, the endomorphism ω is multiplication-by-d map such that $\hat{\phi} = [d] \circ \hat{\phi}'$).

Delay encryption [33]. A direct application to [72], VDF over $\mathbb{F}_{p^2}$ is time-release primitive. Let $H_1 : \{0,1\}^\lambda \to E_T[N]$ be a hash function from identifier (id) to point of order N on $E_T/\mathbb{F}_{p^2}$. Let $H_2 : \mathbb{F}_{p^2} \to \{0,1\}^\lambda$ be a key derivation function. The Burdges and De Feo [33]'s delay encryption is described in Figure 3.8.

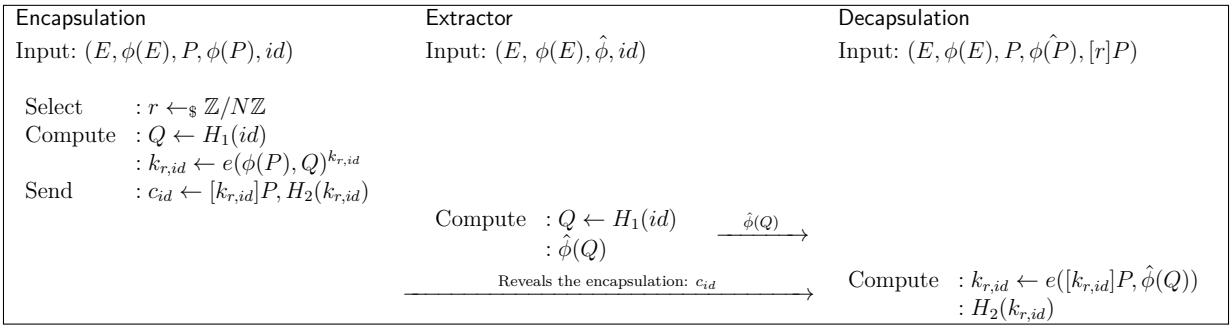


Figure 3.8: Delay encryption [33].

For the delay encryption to be secure, the computation of $k_{r,id}$ should at least take T sequential isogeny evaluation. The key $k_{r,id}$ can be computed in two ways: one by evaluating $\hat{\phi}(Q)$ or the other by $\phi([k_{r,id}]P)$. This questions the computation gap between ϕ and $\hat{\phi}$. Another note is that the Extractor work is identical. Hence, all *id*s should be applied only once for a fixed *id*. Additionally, they demonstrate how to find a random-looking curve E using an iterative secret isogeny walk by a number of participants. At least one participant must be honest for their approach to be secure.

Shani's Sketch Construction [137]

A note by Shani [137] describes a TLP based on an unbalanced isogeny-based key exchange. A Challenger generates a secret and computes a random challenge (as an elliptic curve E). Then, a Verifier validates SeqEval's evaluation using the time-lock puzzle's secret (revealed by the Challenger). Hence, an *interactive* Challenger is now required. Let ϕ be a public isogeny of a large degree (in T). A high level description is in Figure 3.9.

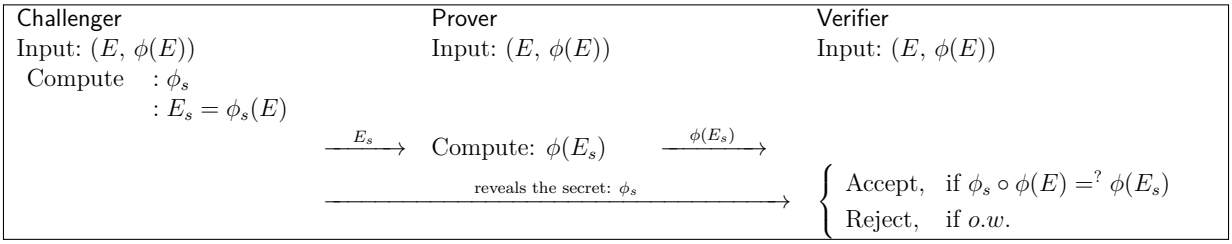


Figure 3.9: Shani's construction over an isogeny graphs of supersingular elliptic curves [137].

In a nutshell, Shani's proposal requires a trusted setup to secure the public parameters' selection and walk a T-long isogeny. Thus, it would not be suitable for delay construct with very large difficulty. Also, the construction of both [137] and [72] share similar characteristics (i.e., in a graph over $\mathbb{F}_p$) and therefore are not quantum secure.

3.5 Special Classes of Delay-Based Cryptography

Several specialized notions of delay-based cryptography have been introduced, focusing on more narrowly defined primitives tailored to specific applications or scenarios. Most of these primitives extend existing traditional cryptographic notions by incorporating time delay into their security characteristics.

Delay-based Signatures. One example is Time-Locked Signatures (TLS), which allows a signature to be time-locked to a known message for a predetermined duration T. In other words, with TLS, senders' signatures related to past messages *expire* after a time delay T, making them forgeable by anyone without the secret. A closely related notion is Verifiable Timed Signatures (VTS), which is TLS with public verifiability. With VTS, anyone can verify that a time-locked signature is valid without solving the puzzle through the T-sequential computation. Numerous TLS and VTS constructions have been proposed, such as [4, 6, 141] and [46, 148, 162].

Delay-based Secret Sharing. Timed-Release Secret Sharing (TR-SS) is a variation of secret-sharing schemes where the reconstruction of the secret is constrained not only by the need for a threshold number of shareholders, but also by a specified time delay set by the dealer. The concept and construction of TR-SS have been explored in various works, such as [94, 152, 153]. In addition, Verifiable TR-SS (VTSS) strengthens the notion of TR-SS by adding a verification mechanism. This ensures that shareholders can verify that the dealer's challenges, which encapsulate their shares, are well-formed with valid shares. It is also designed to prevent malicious shareholders from submitting invalid shares (which might be in the form of a challenge) during the secret reconstruction process.

3.6 Chapter Summary

In this chapter, we covered several key concepts in delay-based cryptography, including the idea of time delay and sequential functions. We discussed the main constructions of long

sequential functions that are essential to achieve computational delays. Furthermore, we explored different notions of delay-based cryptography that have been proposed in the literature, such as time-lock puzzles and timed-release encryption, proof of sequential works, and so on. In addition, we presented several techniques for proofs of correct computation, which are crucial for developing a wide range of applications and constructing different primitives. Finally, we explored certain special classes of delay-based cryptography, including delay-based signatures and secret sharing notions.

Chapter 4

A New Class of Trapdoor Verifiable Delay Functions

This chapter presents a new class of trapdoor-VDFs featuring a large ensemble of trapdoors for each instantiation of the function. This way, a client can randomly choose a private trapdoor from the ensemble, thereby using it to encapsulate a secret to the future as a unique puzzle. To solve the puzzle, the server, which does not know the trapdoor, requires a prescribed number of sequential steps to evaluate the function. Any client can efficiently verify the correctness of the server's evaluation with zero knowledge of the trapdoor being used. We present an approach for constructing the proposed class of trapdoor-VDFs based on bilinear pairings and a long walk on supersingular isogeny graphs. Finally, we examine the security of our construction under trapdoor-VDF security notions. An article based on this chapter appeared in [159].

4.1 Introduction

This work examines a remedy for the vulnerability that arises from knowing or predicting a protocol's outcome. The vulnerability stems from malicious participants influencing the outcome or gaining an advantage by knowing the outcome beforehand. One way to solve this problem is to impose a prescribed number of sequential steps to obtain the desired outcome. This solution has been introduced in the previous works such as time-lock puzzle (TLP) [131], proofs of sequential work (PoSW) [2, 51, 104, 114], and verifiable delay functions (VDFs) [25]. These primitives are all time-sensitive in that they only release the

outcome after a prescribed delay (T). The outcome of VDF and PoSW is publicly verifiable, while that of TLP requires a secret. The public verification of the outcome's *uniqueness* is more efficient in VDF than in PoSW since the latter requires all T steps. The uniqueness property ensures that there are no multiple valid proofs for different outcome.

This work focuses more on the properties, specifically uniqueness and public verifiability, that make VDF stand out from others. Furthermore, we are interested in a VDF-like primitive that allows any participant (other than the trusted setup) to predict the outcome in advance by knowing a secret trapdoor. In the absence of the secret trapdoor, participants obtain the desired outcome through a prescribed number of sequential evaluations. It is, however, possible to verify the result efficiently and publicly. We refer to primitives with such characteristics by trapdoor-VDF. Considering its importance, trapdoor-VDF is suitable for time-sensitive applications that require both public verification and timely release assurances. There are many possible applications, such as sealed-voting, and front-running attack prevention [50].

In addition, the term "trapdoor-VDF" has been used previously in [156], though their approach differs from that presented in this chapter. In [156], each participant constructs independent instances of trapdoor-VDF. Every instance has a secret trapdoor that shortens the long evaluation process. Therefore, an instance generator can answer any challenge faster as it has the secret trapdoor. Our work presents a new class of trapdoor-VDFs, where each instantiation includes a description of a finite trapdoor ensemble. Hence, any participant can generate a unique challenge, together with its answer, using a hidden trapdoor sampled at random from the ensemble.

Contributions

In this chapter, we first formalize the security notions of trapdoor-VDF. We then introduce a novel approach to trapdoor-VDF; the proposed trapdoor-VDF is a VDF with a large ensemble of distinct efficient shortcuts called trapdoors. The delay function involves sequentially evaluating a large smooth-degree supersingular isogeny, which was first proposed by Feo et al. [72]. The trapdoor ensemble is a set of isogenies of a smaller degree defined over $\mathbb{F}_p$, where p is a prime. In public verification, a bilinear pairing equality serves as proof of the correct evaluation of a secret trapdoor, i.e., a randomly sampled isogeny. Finally, we show that our proposal is secure under trapdoor-VDF security notions.

Organization. In Section 4.2, we start by introducing the notion of trapdoor verifiable delay function. Following this, Section 4.3 discusses an approach to construct a proof-of-concept trapdoor-VDF. Then, we present, in Section 4.3.4, a concrete instance built on

isogeny-based sequential function. Section 4.4 shows that our proposed instance is a secure and well-defined trapdoor-VDF. Finally, we wrap up with concluding remarks in Section 4.5.

4.2 Proposed Trapdoor-VDFs

By $\lambda \in \mathbb{N}$, we indicate the security level of a scheme. A function difficulty is denoted by T, which quantifies the amount of sequential work/steps necessary to produce/compute its output against any random input and with a polynomially large number of parallel processes. A function with a small T is identified as a short function, whereas one with a large T is called a long sequential function. Generally, we will denote the long sequential function by $\mathsf{SeqEval}$, with T being super-polynomial in λ.

An Informal Exposition. The proposed trapdoor-VDF is a VDF with a large ensemble of distinct shortcuts denoted by $\mathcal{F}$, called trapdoors. We say two trapdoors are equivalent if they produce the same output for the same inputs. The set of equivalent trapdoors is called a class. The class difficulty is the shortest trapdoor difficulty. Hence, the ensemble $\mathcal{F}$ is a collection of trapdoor classes.

Trapdoor-VDF setup agreement. Participants of a setup protocol agree on a long function $\mathsf{SeqEval}$ with a difficulty T, a large ensemble of trapdoors $\mathcal{F}$, and possibly additional parameters for a security level λ.

Generation of a challenge. In trapdoor-VDF, a Challenger can select a secret trapdoor tr_{sk} indexed (identified) in $\mathcal{F}$ by a random secret (sk). Using the trapdoor, the challenger can efficiently generate a challenge ($\mathbf{c}$) and its unique answer ($\mathbf{a}$). With the same parameters, distinct trapdoors produce different challenges (and accordingly different answers).

Obtaining the answer. In the absence of sk, a Solver can only evaluate $\mathsf{SeqEval}$ function in time no less than T to output the answer to $\mathbf{c}$ with a proof π. Akin to VDF, the Solver gains no advantage from parallel computation. The Challenger can, however, get the answer $\mathbf{a}$ via the trapdoor tr_{sk} (i.e., in time less than T).

The public verification. The answer $\mathbf{a}$ is publically verifiable that is also a zero knowledge of the secret sk. In a timeframe less than T, a *perfect* trapdoor-VDF is one in which the Challenger, who owns $\mathbf{c}$ and knows $\mathbf{a}$, cannot pass the public verification protocol. This is because the required proof cannot be fully computed before the specific time T.

A Formal Definition. We present our formal definition of trapdoor-VDF, which naturally overlaps with [25] and [156]'s VDF definition.

Definition 7. *Let $\mathcal{C}$, $\mathcal{S}$, and $\mathcal{Y}$ be the challenge, secret, and answer spaces, respectively. Our trapdoor-VDF is a tuple of algorithms (*Setup*, *Challenger*, *Solver*, *Verify*) defined below*

- *Setup: a randomized algorithm (runs in time $\mathsf{Poly}(\lambda)$) that takes a security parameter λ and a difficulty T and outputs public parameter pk.*
- *Challenger: a randomized algorithm (runs in time $\mathsf{Poly}(\log T, \lambda)$) that takes pk and selects a secret trapdoor tr_{sk} from the trapdoor ensemble $\mathcal{F}$ using a random secret sk (i.e., $\mathsf{tr}_{sk} \leftarrow \mathcal{F}$ given $sk \leftarrow_{\$} \mathcal{S}$); then, it generates and outputs a challenge $\boldsymbol{c} \in \mathcal{C}$ associated with an answer $\boldsymbol{a} \in \mathcal{Y}$.*
- *Solver: a deterministic algorithm that takes pk and a challenge $\boldsymbol{c} \in \mathcal{C}$ and outputs the answer $\boldsymbol{a} \in \mathcal{Y}$ and a "possibly empty" proof π. This algorithm must at least run in time T with $\mathsf{Poly}(\lambda)$ parallel processors.*
- *Verify: a deterministic algorithm (runs in total time polynomial in $\log T$ and λ) takes a challenge $\boldsymbol{c}$, an answer $\boldsymbol{a}$, a proof π, and pk; the algorithm outputs* ACCEPT *if $\boldsymbol{a}$ is indeed the corresponding answer to $\boldsymbol{c}$ under a given π, otherwise* REJECT.

Trapdoor-VDF Properties. The following assumes that all statements are true for any λ, T and $pk \leftarrow_{\$} \mathsf{Setup}(1^{\lambda}, T)$. A trapdoor-VDF construction is well-defined if it is correct, unique, and efficient.

- *Correctness.* A trapdoor-VDF is correct only if the Verifier accepts, with probability one, an honest Solver's answer $\mathbf{a}$ for any honest challenge $\mathbf{c} \leftarrow_{\$} \mathsf{Challenger}(pk)$.
- *Uniqueness.* A trapdoor-VDF is unique only if there is exactly one valid answer $\mathbf{a}$ accepted by Verify for every challenge $\mathbf{c}$ generated by $\mathsf{Challenger}(pk)$ using the secret key sk.
- *Efficiency.* A trapdoor-VDF is efficient if the Verify algorithm runs in a time $\mathsf{Poly}(\log T, \lambda)$ that is significantly faster than the Solver algorithm, which has a total running time polynomial in T and λ. Further, trapdoor-VDF must retain efficiency for any public parameters generated by Setup, runs in $\mathsf{Poly}(\log T, \lambda)$, and any challenge generated by $\mathsf{Challenger}$, which runs in time $\mathsf{Poly}(\log T, \lambda)$.

Let $\mathcal{A}$ be a PPT adversary who has no knowledge of the secret sk. Let $\mathcal{A}_1$ be an algorithm that outputs pre-computation (pc) on pk. Let $\mathcal{A}_2$ be an online efficient evaluating

algorithm that runs in parallel time with $\mathsf{Poly}(\lambda)$ processors and returns an answer $\mathbf{a}'$. Let $\mathcal{A}_3$ be an online forging algorithm that runs in time $\mathsf{Poly}(T, \lambda)$ and returns a malicious answer and proof $(\mathbf{a}' \neq \mathbf{a}, \pi')$. To be secure, a well-defined trapdoor-VDF should satisfy two key properties: sequentiality and soundness.

— *Sequentiality.* A trapdoor-VDF is sequential only if there is no adversary $\mathcal{A} := (\mathcal{A}_1, \mathcal{A}_2)$, where $\mathcal{A}_2$, an online algorithm, runs in less than T-sequential time and succeeds with a probability:

$$\mathsf{Pr}\left[\begin{array}{lcl} & & pk \leftarrow_{\$} \mathsf{Setup}(1^{\lambda}, T), \\ & & \mathsf{pc} \leftarrow \mathcal{A}_1(pk), \\ \mathbf{a}' = \mathbf{a} & : & \mathbf{c} \leftarrow_{\$} \mathsf{Challenger}(pk), \\ & & (\mathbf{a}', -) \leftarrow \mathcal{A}_2(pk, \mathbf{c}, \mathsf{pc}), \\ & & (\mathbf{a}, \pi) \leftarrow \mathsf{Solver}(pk, \mathbf{c}). \end{array}\right]$$

that is greater than a negligible function of λ.

— *Soundness.* A trapdoor-VDF is sound only if, for every challenge $\mathbf{c}$ generated by $\mathsf{Challenger}(pk)$, the Verifier rejects any proof π' for any answer $\mathbf{a}'$ that is not an output from $\mathsf{Solver}(pk, \mathbf{c})$. The probability of success for the adversary $\mathcal{A} := (\mathcal{A}_1, \mathcal{A}_3)$ to output a proof π' for an answer $(\mathbf{a}', -) \neq \mathsf{Solver}(pk, \mathbf{c})$ is

$$\mathsf{Pr}\left[\begin{array}{l} \text{ACCEPT} \leftarrow \mathsf{Verify}(pk, \mathbf{c}, \mathbf{a}', \pi') \\ \text{and } \mathbf{a}' \neq \mathbf{a} \end{array} : \begin{array}{l} pk \leftarrow_{\$} \mathsf{Setup}(1^{\lambda}, T), \\ \mathsf{pc} \leftarrow \mathcal{A}_1(pk), \\ \mathbf{c} \leftarrow_{\$} \mathsf{Challenger}(pk), \\ (\mathbf{a}, \pi) \leftarrow \mathsf{Solver}(pk, \mathbf{c}), \\ (\mathbf{a}', \pi') \leftarrow \mathcal{A}_3(pk, \mathbf{c}, \mathsf{pc}). \end{array}\right]$$

that is a negligible function of λ.

Additionally, a well-defined trapdoor-VDF may comprise further properties, and one that is most relevant to our work is given below.

— *Perfectness.* The knowledge of tr_{sk} and the pair $(\mathbf{c}, \mathbf{a})$ solely does not provide an advantage in passing the public verification protocol. Let $\hat{\mathcal{A}}$ be an algorithm implementing $\mathsf{Challenger}$, which outputs the pair $(\mathbf{c}, \mathbf{a})$ and an efficient algorithm $\hat{\mathcal{A}}_2$. $\hat{\mathcal{A}}_2$ runs in parallel with $\mathsf{Poly}(\lambda)$ processors and returns a proof (π'). The probability of success for $\hat{\mathcal{A}}_2$ to output an acceptable proof π', in less than T-sequential time, for any pair $(\mathbf{c}, \mathbf{a})$ is

$$\mathsf{Pr}\left[\begin{array}{l} \\ \text{ACCEPT} \leftarrow \mathsf{Verify}(pk, \mathbf{c}, \mathbf{a}, \pi') \\ \text{and ACCEPT} \leftarrow \mathsf{Verify}(pk, \mathbf{c}, \mathbf{a}, \pi) \\ \\ \end{array} : \begin{array}{l} pk \leftarrow_{\$} \mathsf{Setup}(1^{\lambda}, T), \\ (\mathbf{c}, (\mathsf{tr}_{sk}, \mathbf{a}), \hat{\mathcal{A}}_2) \leftarrow \hat{\mathcal{A}}(\mathsf{Challenger}, pk), \\ (-, \pi') \leftarrow \hat{\mathcal{A}}_2(pk, \mathbf{c}, \mathsf{tr}_{sk}, \mathbf{a}), \\ (\mathbf{a}, \pi) \leftarrow \mathsf{Solver}(pk, \mathbf{c}). \end{array}\right]$$

that is a negligible function of λ.

Further on Trapdoor-VDF Properties. A secure well-defined trapdoor-VDF accounts also for the trapdoor properties and assumption(s). This is due to the fact that a secure well-defined Challenger requires a secure well-defined trapdoor. Formally, a trapdoor $\mathsf{tr}_{sk} \in \mathcal{F}$, associated with a domain $\mathcal{D}(\mathsf{tr}_{sk})$ and range $\mathcal{R}(\mathsf{tr}_{sk})$, is defined as follows

- tr_{sk}: a short function in $sk \leftarrow_{\$} \mathcal{S}$, with evaluation time $\mathsf{Poly}(\log T, \lambda)$, that takes an input pk; the function evaluation returns a challenge and answer pair $(\mathbf{c}, \mathbf{a}) \in \mathcal{C} \times \mathcal{Y}$.

For any random secret $sk \leftarrow_{\$} \mathcal{S}$ and $\mathsf{tr}_{sk} \leftarrow \mathcal{F}$, the trapdoors in trapdoor-VDF feature several properties.

— *Challenger correctness.* Let $(\mathbf{c}, \mathbf{a})$ be a challenge and its answer pair generated by tr_{sk}. The correctness property requires that the answer $(\mathbf{a}', -) \leftarrow \mathsf{Solver}(pk, \mathbf{c})$ be equal to $\mathbf{a}$ with probability one (i.e., we must have $\mathbf{a}' = \mathbf{a}$ with probability one). There is, however, an extension to the previous statement. For instance, we can allow $\mathbf{a}' \neq \mathbf{a}$ only if there is a one-way public function (f) such that $\mathbf{a} \leftarrow \mathsf{f}(\mathbf{a}')$ and $\text{ACCEPT} \leftarrow \mathsf{Verify}(pk, \mathbf{c}, \mathsf{f}(\mathbf{a}'), \pi)$ is *true* for all $\mathbf{c} \in \mathcal{C}$ and all valid proofs π. Having such an extension allows us to construct a *perfect* trapdoor-VDF in which the Challenger, who owns $\mathbf{c}$ and knows $\mathbf{a}$, cannot pass the public verification protocol before time T.

— *Challenger efficiency.* All trapdoors in a trapdoor-VDF must also be efficient. For any λ, T and $pk \leftarrow_{\$} \mathsf{Setup}(1^{\lambda}, T)$, the efficiency implies that (as in [17]):

  – There is an algorithm that runs in time $\mathsf{Poly}(\log T, \lambda)$ and implements the process of sampling tr_{sk} from $\mathcal{F}$ for all $sk \in \mathcal{S}$.

  – There is an algorithm that runs in time $\mathsf{Poly}(\log T, \lambda)$ and implements the process of sampling an element from $\mathcal{D}(\mathsf{tr}_{sk})$.

- There is an algorithm that evaluates tr_{sk} in time $\mathsf{Poly}(\log T, \lambda)$ for any element of $\mathcal{D}(\mathsf{tr}_{sk})$.

— *Challenger security.* To be secure, the following problems must be hard for any λ, T and $pk \leftarrow_{\$} \mathsf{Setup}(1^{\lambda}, T)$.

- Given any challenge $\mathbf{c} \in \mathcal{C}$ generated under the secret $\mathsf{tr}_{sk} \in \mathcal{F}$, find the challenge's answer $\mathbf{a} \in \mathcal{Y}$ in less than T-sequential time.
- Given any challenge and answer pair $(\mathbf{c}, \mathbf{a}) \in \mathcal{C} \times \mathcal{Y}$ generated by $\mathsf{tr}_{sk} \in \mathcal{F}$, find $sk \in \mathcal{S}$.
- Let $(\mathbf{c}', \mathbf{a}') \in \mathcal{C} \times \mathcal{Y}$ be any challenge and answer pair generated under $\mathsf{tr}_{sk'} \in \mathcal{F}$. Given a challenge $\mathbf{c} \in \mathcal{C}$ under $\mathsf{tr}_{sk} \in \mathcal{F}$, find its answer $\mathbf{a} \in \mathcal{Y}$ in less than T-sequential time.
- Given any polynomial number of challenge-answer pairs $\{(\mathbf{c}_i, \mathbf{a}_i) \in \mathcal{C} \times \mathcal{Y}\}_{i \in [m]}$ generated under $\mathsf{tr}_{sk} \in \mathcal{F}$, find the answer $\mathbf{a} \in \mathcal{Y}$ to a new challenge $\mathbf{c} \in \mathcal{C}$ (also under tr_{sk}) in less than T-sequential time.

Similar to VDF, the difficulty T is restricted to subexponential in λ. It is therefore cheaper to perform the T-sequential evaluation than to compromise the trapdoor-VDF security.

4.3 Design Rationale

This section discusses our approach to construct a proof-of-concept instance of trapdoor-VDF. To construct trapdoor-VDF, we begin by defining a long sequential public function $\mathsf{SeqEval} : \mathcal{X} \rightarrow \mathcal{X}$ with a public challenge and answer (i.e., $x, y \in \mathcal{X}$ such that $y \leftarrow \mathsf{SeqEval}(x)$). The secret is a random string ($sk \leftarrow_{\$} \mathcal{S}$) that indexes a secret short map ($\mathsf{tr}_{sk} : \mathcal{X} \rightarrow \mathcal{X}$) in the ensemble $\mathcal{F}$. Using a random secret map, one may craft a trapdoor. A new challenge (x') can be obtained by masking x with tr_{sk} (i.e., $x' \leftarrow \mathsf{tr}_{sk}(x)$). The secret map tr_{sk}, which can also determine $y' \leftarrow \mathsf{tr}_{sk}(y)$, becomes the trapdoor. This statement is true assuming that the action $\mathsf{tr}_{sk} \circ \mathsf{SeqEval}$ is equivalent to $\mathsf{SeqEval} \circ \mathsf{tr}_{sk}$. In the absence of tr_{sk}, one can obtain y' by evaluating $y' \leftarrow \mathsf{SeqEval}(x')$, which involves a large number of sequential steps.

Lastly, the verification procedure involves validating tr_{sk}'s correct computation statement with zero knowledge of tr_{sk}, where the Solver's answer (y'') serves as the statement

witness. The validation arguments should be efficient, validating only the unique answer (i.e., a Verifier accepts only if $y'' = y'$).

In the following, we present a sketch construction of the proposed trapdoor-VDF. Essentially, the construction consists of evaluating a series of non backtracking *horizontal* isogenies with a large degree (representing $\mathsf{SeqEval}$), whereas the secret trapdoor is a shorter *horizontal* isogeny walk in graphs over a finite field $\mathbb{F}_p$ (representing tr_{sk}). Our public verification protocol uses bilinear pairings similar to BLS signature scheme [28], which is also used in [33] and [72].

4.3.1 Construction Elements

First, we define the public parameters that we will use to construct our scheme. Following that, we will briefly describe the supersingular isogeny graph, upon which both the long evaluation function and short trapdoor operate. Lastly, we will discuss the scheme group structure and the scheme's public verification method.

Selection of the Scheme Parameters. The parameters for our scheme are generated with the help of the following algorithms.

- $(p, \mathcal{L}, N, \mathcal{S}, \boldsymbol{t}, E_0) \leftarrow \mathsf{GGen}(1^\lambda, T)$ is a public parameter generation algorithm that takes a security parameter λ and difficulty T as an input and outputs:
  - a large odd prime p such that $p = 7 \bmod 8$,
  - a set $\mathcal{L}$ of n small distinct primes defined as follow

$$\mathcal{L} := \{2\} \cup \{l \in \mathsf{Primes}(6\log(p)^2) : (\frac{-p}{l}) = 1\},$$

  - a large prime N such that $N \notin \mathcal{L}$ and $N \mid p+1$,
  - the set $\mathcal{S} := \{-e, \ldots, e\}$ for a positive integer e where $|\mathcal{S}| = 2e+1$,
  - the vector $\boldsymbol{t}$ of n elements such that $\sum_{m \in \boldsymbol{t}} \mathsf{abs}(m) = T < 2^{o(\lambda)}$.
  - a supersingular elliptic curve $E_0/\mathbb{F}_p$ on the surface of $\mathcal{G}(\mathbb{F}_p, 2)$, where $|E_0(\mathbb{F}_p)| = p+1$ and possibly with j-invariant $\mathsf{j}(E_0) \in \{0, 1728\}$.
- $E \leftarrow_{\$} \mathsf{Alg}(1^\lambda, E', \mathcal{L})$ is a randomized algorithm that takes an initial curve E', the set $\mathcal{L}$, and security parameter λ. The algorithm Alg outputs a random supersingular elliptic curve $E/\mathbb{F}_p$ on the graph surface. Basically, this algorithm involves taking

a random horizontal isogeny walk (see [72]) of a length of at least $\mathsf{Poly}(\log p)$. It is required that the probability of finding isogeny (path) between the output curve E and an initial curve E' be a negligible function in λ.

The prime p can be expressed as $p = 2^{e_1} l_2^{e_2} \cdots l_n^{e_n} N^{e_{n+1}} - 1$, where $l_k \in \mathcal{L}$, $e_1 \geq 3$, and $e_k \in \mathbb{Z}^+$ for all k with the sequence l_k strictly increasing (i.e., $l_k < l_{k+1}$) to satisfy the condition discussed earlier. Furthermore, for $i \in [n]$, we define the vector $\boldsymbol{t}$ with difficulty T by setting $\boldsymbol{t}[i] = T$ and $\boldsymbol{t}[j] = 0$ for all other indices $j \in [n] \setminus \{i\}$, where i is the smallest index such that $e_i < \sum_{\substack{j=1 \\ j \neq i}}^{n+1} e_j$.

4.3.2 The Graph in Use and the Single Step of Computation

The graph over $\mathbb{F}_p$ described in [61, Theorem 2.7] is the abstraction behind our scheme. In our scheme, the graph consists of two levels, namely a surface (i.e., $\mathcal{E}_{\mathbb{F}_p}(\mathbb{Z}[\frac{1+\sqrt{-p}}{2}])$) and a floor (i.e., $\mathcal{E}_{\mathbb{F}_p}(\mathbb{Z}[\sqrt{-p}])$). In this graph, surface and floor have one-to-one connection by 2-isogenies, and there are no odd-degree isogenies connecting them. On the graph surface, there are two horizontal isogenies of degree $\ell_i \in \mathcal{L}$ from each vertex, whereas the floor is connected by isogenies of degree $\ell_i \in \mathcal{L}/\{2\}$.

In our construction, a *sequential walk* on the graph is represented by the group action of the ideal class group $\mathbf{cl}(\mathcal{O})$ of an imaginary quadratic order $\mathcal{O} \cong \mathbb{Z}[\frac{1+\sqrt{-p}}{2}]$ on the set $\mathcal{E}_{\mathbb{F}_p}(\mathcal{O})$. Hence, both $\mathsf{SeqEval}$ and tr_{sk} act on the set $\mathcal{E}_{\mathbb{F}_p}(\mathcal{O})$. The choice of $\mathcal{L}$'s elements enables us to represent the elements of $\mathbf{cl}(\mathcal{O})$ as a product of ideals of small norm $\mathsf{N}(\mathfrak{l})$ such that $\mathsf{N}(\mathfrak{l}) \in \mathcal{L}$. Thus, we represent a *single step* in the sequential walk by the group action of an ideal of a norm in $\mathcal{L}$ that acts on $\mathcal{E}_{\mathbb{F}_p}(\mathcal{O})$.

The elements of $\mathcal{L}$ are chosen to be Elkies primes. Hence, the ideal $\ell_i\mathcal{O}$ splits into $\mathfrak{l}_i = (\ell_i, \pi - 1)$ and $\hat{\mathfrak{l}}_i = (\ell_i, \pi + 1)$ for every $\ell_i \in \mathcal{L}$ (i.e., $\ell_i\mathcal{O} = \mathfrak{l}_i\hat{\mathfrak{l}}_i, \forall \ell_i \in \mathcal{L}$). This defines the direction of a *single step*. From every vertex in the graph surface, there are two actions of an ideal with norm ℓ_i that can be applied, either $\mathfrak{l}_i$ or $\hat{\mathfrak{l}}_i$. Further, each direction can be computed via an isogeny using Vélu's formulas [149]. In other words, a single step in one direction, denoted by ϕ_{ℓ_i}, is an isogeny of a kernel of order ℓ_i intersecting with $\mathsf{ker}(\pi - [1])$. As for the step in the opposite direction, denoted by $\phi_{\ell_i}^{-1}$, represents an isogeny of a kernel of order ℓ_i intersecting with $\mathsf{ker}(\pi + [1])$. Finally, for an integer m_i, we denote ℓ_i-sequential walk by $\phi_{\ell_i}^{m_i}$, which represents m_i sequences of ℓ_i-isogeny evaluation in the same direction.

The Long Evaluation Function. As with [72] and [137], the long evaluation is represented by an isogeny of large degree, exponential in T,

$$\mathsf{SeqEval} := \phi_{\mathcal{L}}^{\boldsymbol{t}} : E \to E_A$$

which is a composite of all $\phi_{\ell_i}^{m_i}$ for all $\ell_i \in \mathcal{L}$ and $m_i \in \boldsymbol{t}$ (i.e., $\phi_{\mathcal{L}}^{\boldsymbol{t}} := \phi_{\ell_n}^{m_n} \circ \cdots \circ \phi_{\ell_1}^{m_1}$, where $m_i \in \boldsymbol{t}$ and $|\boldsymbol{t}| = n$). The degree of $\phi_{\mathcal{L}}^{\boldsymbol{t}}$ is $\prod_{i=1}^{n} l_i^{\mathsf{abs}(m_i)}$ and it has a difficulty of $\sum_{i=1}^{n} \mathsf{abs}(m_i) = T$. The presumption is that to compute $\mathsf{SeqEval}$ efficiently, all T composites of $\phi_{\mathcal{L}}^{\boldsymbol{t}}$ must be evaluated sequentially.

The Secret Trapdoor and Secret Trapdoor Set. As with [41, 137], we want to be able to efficiently sample a random trapdoor from their ensemble $\mathcal{F}$. We will therefore randomly sample $sk := \boldsymbol{s} \leftarrow_{\$} \mathcal{S}^n$ to represent the trapdoor as

$$\mathsf{tr}_{sk} := \phi_{\mathcal{L}}^{\boldsymbol{s}} : E \to E_B.$$

The set $\mathcal{S}$ is chosen so that the trapdoor $\phi_{\mathcal{L}}^{\boldsymbol{s}}$ is much shorter than $\phi_{\mathcal{L}}^{\boldsymbol{t}}$. However, the size of $\mathcal{S}$ must be large enough to ensure that there exist $|\mathcal{S}|^n \geq 2^{2\lambda}$ possible secrets.

4.3.3 The Scheme Group Structure and Its Public Verification

Let G_i^E be a subgroup of $E[N]$ where $E \in \mathcal{E}_{\mathbb{F}_p}(\mathcal{O})$ and $i \in [N]$. The trace-zero and the base-field subgroup of $E[N]$ are represented by subscripts 1 and 2, respectively[1]. Under the assumption that (i) N is coprime to $|\mathsf{ker}(\phi_{\mathcal{L}}^{\boldsymbol{t}})|$ and $N \neq p$, and (ii) E and E_A are isogenous (so that $|E[N]| = |E_A[N]|$), the surjective morphism $\phi_{\mathcal{L}}^{\boldsymbol{t}} : E[N] \to E_A[N]$ must also be injective induced by Lagrange's theorem; hence it is a bijective group homomorphism on the N-torsion subgroup (similar argument applies for $\phi_{\mathcal{L}}^{\boldsymbol{s}}$). Let σ^{-1} be a *quadratic twist* defined as follows

$$\sigma^{-1} : E \to E^{(d)} \in \mathcal{E}_{\mathbb{F}_p}(\mathcal{O})$$
$$\sigma^{-1} : G_{i\in\{1,2\}}^{E} \to G_{2i \bmod 3}^{E^{(d)}}$$

where $E^{(d)}$ is a twist of the curve E. The inverse quadratic twist (i.e., $\sigma : E^{(d)} \to E$) is efficiently computable and it is defined as follows

$$\sigma : (x, y) \to (x/w^2, y/w^3)$$

where $w \in \mathbb{F}_{p^2}$, $w \notin \mathbb{F}_p$, and it has $w^2 \in \mathbb{F}_p$. In our configuration, it should be noted that $\sigma \circ \phi_{\ell_i}^{m_i}(E)$ and $\phi_{\ell_i}^{-m_i} \circ \sigma(E)$ are equivalent.

[1]The subgroup G_1^E is defined as $G_1^E := E[N] \cap \mathsf{ker}(\pi + [1])$, whereas, $G_2^E := E[N] \cap \mathsf{ker}(\pi - [1])$.

Pairing Based Public Verification in a Nutshell. Let $\hat{\mathsf{e}}_N^E : E[N] \times E[N] \to \mu_N$ be a non-degenerate Weil pairing map on E, where $\mu_N \subset \mathbb{F}_{p^2}^*$. A non-trivial bilinear Weil pairing can be defined as $\hat{\mathsf{e}}_N^E : G_1^E \times G_2^E \to \mu_N$. This definition is equivalent to $\hat{\mathsf{e}}_N^E : \sigma(G_2^{E^{(d)}}) \times G_2^E \to \mu_N$, which is efficiently computable with inputs of short representation. Let E, $E_A = \phi_{\mathcal{L}}^{\boldsymbol{t}}(E)$, $E_B = \phi_{\mathcal{L}}^{\boldsymbol{s}}(E)$, $E_{AB} = \phi_{\mathcal{L}}^{\boldsymbol{s}}(E_A)$, and $E_{BA} = \phi_{\mathcal{L}}^{\boldsymbol{t}}(E_B)$ be a set of isogenous supersingular elliptic curves. The public verification proceeds as follows:

- A Challenger, Verifier, and Solver all share as an input the description of the isogeny $\phi_{\mathcal{L}}^{\boldsymbol{t}} : E \to E_A$ of difficulty T, and the point pair $(x^{G_1^E}, y^{G_1^{E_A}}) \in G_1^E \times G_1^{E_A}$ such that $y^{G_1^{E_A}} = \phi_{\mathcal{L}}^{\boldsymbol{t}}(x^{G_1^E})$. All inputs are chosen for a security parameter λ.
- A Challenger selects $\boldsymbol{s} \leftarrow_{\$} \mathcal{S}^n$ that defines the secret short isogeny $\phi_{\mathcal{L}}^{\boldsymbol{s}} : E \to E_B$; then, it
  - randomly samples integers k_A, k_B, and k_{AB} such that $k_A k_B^{-1} k_{AB}^{-1} = \deg \phi_{\mathcal{L}}^{\boldsymbol{s}} \pmod N$,
  - computes and broadcasts the challenge $([k_A]Q^{G_2^{E_A}}, \hat{x}^{G_1^{E_B}}, Q^{G_2^{E_{AB}}}) \in G_2^{E_A} \times G_1^{E_B} \times G_2^{E_{AB}}$, where $\hat{x}^{G_1^{E_B}} = [k_B]\phi_{\mathcal{L}}^{\boldsymbol{s}}(x^{G_1^E})$, $Q^{G_2^{E_A}} \leftarrow_{\$} G_2^{E_A}$, and $Q^{G_2^{E_{AB}}} = [k_{AB}]\phi_{\mathcal{L}}^{\boldsymbol{s}}(Q^{G_2^{E_A}})$.
- A Solver computes the answer $\hat{y}^{G_1^{E_{BA}}} \leftarrow \phi_{\mathcal{L}}^{\boldsymbol{t}}(\hat{x}^{G_1^{E_B}})$, which requires T sequential isogeny evaluations, and then broadcasts $\hat{y}^{G_1^{E_{BA}}}$.
- A Verifier outputs ACCEPT if $\hat{y}^{G_1^{E_{BA}}}$ satisfies the following equality

$$\hat{\mathsf{e}}_N^{E_A}(y^{G_1^{E_A}}, [k_A]Q^{G_2^{E_A}}) = \hat{\mathsf{e}}_N^{E_{AB}}(\hat{y}^{G_1^{E_{BA}}}, Q^{G_2^{E_{AB}}}), \tag{4.1}$$

  otherwise returns REJECT.

From Equation (2.11), the above-mentioned verification is complete, as shown below.

$$\begin{aligned} \hat{\mathsf{e}}_N^{E_A}(y^{G_1^{E_A}}, [k_A]Q^{G_2^{E_A}}) &= \hat{\mathsf{e}}_N^{E_{AB}}(\phi_{\mathcal{L}}^{\boldsymbol{t}} \circ [k_B]\phi_{\mathcal{L}}^{\boldsymbol{s}}(x^{G_1^E}), [k_{AB}]\phi_{\mathcal{L}}^{\boldsymbol{s}}(Q^{G_2^{E_A}})), \\ \hat{\mathsf{e}}_N^{E_A}(y^{G_1^{E_A}}, [\deg \phi_{\mathcal{L}}^{\boldsymbol{s}}]Q^{G_2^{E_A}}) &= \hat{\mathsf{e}}_N^{E_{AB}}(\phi_{\mathcal{L}}^{\boldsymbol{s}}(y^{G_1^{E_A}}), \phi_{\mathcal{L}}^{\boldsymbol{s}}(Q^{G_2^{E_A}})). \end{aligned}$$

Moreover, it is obvious that the correctness of the scheme depends on the terminal elliptic curves (i.e., E_{AB} and E_{BA}) being identical. To be consistent with previous work [72], Figure 4.1 shows the proof system's structure.

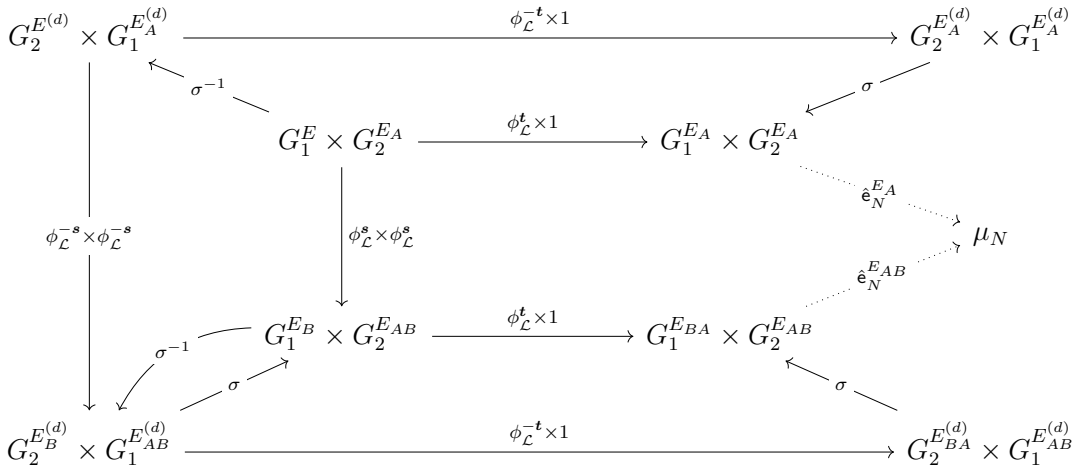


Figure 4.1: The diagram illustrates the structure of the proposed trapdoor-VDF's proof system. The map $\phi_{\mathcal{L}}^{-t}$ is the $\mathcal{L}$-sequential walk $\phi_{\mathcal{L}}^{t}$ in the opposite direction (and similar for $\phi_{\mathcal{L}}^{s}$).

4.3.4 An Instance of the Trapdoor-VDF

The following is a formal description of a trapdoor-VDF instance constructed from the action of class groups on supersingular elliptic curves with bilinear pairings being used for public verification. They are all defined by the parameters chosen in the previous section.

Setup$(1^\lambda, T)$**:**

$(\mathcal{L}, N, p, \mathcal{S}, \boldsymbol{t}, E_0) \leftarrow \mathsf{GGen}(1^\lambda, T)$
$E \leftarrow_\$ \mathsf{Alg}(1^\lambda, E_0, \mathcal{L})$
$x^{G_1^E} \leftarrow_\$ G_1^E$, s.t $\langle x^{G_1^E} \rangle = G_1^E$
$E_A \leftarrow \phi_{\mathcal{L}}^{\boldsymbol{t}}(E)$
$y^{G_1^{E_A}} \leftarrow \phi_{\mathcal{L}}^{\boldsymbol{t}}(x^{G_1^E})$
$pk \leftarrow (\mathcal{L}, N, p, \mathcal{S}, \boldsymbol{t}, E, E_A, x^{G_1^E}, y^{G_1^{E_A}})$
return (pk)

Solver$(pk, \mathbf{c})$**:**

if $\boldsymbol{c}.Q^{G_2^{E_A}} \notin G_2^{E_A}$ **then**
  return REJECT
if $\boldsymbol{c}.\hat{x}^{G_1^{E_B}} \notin G_1^{E_B}$ **then**
  return REJECT
if $\boldsymbol{c}.Q^{G_2^{E_{AB}}} \notin G_2^{E_{AB}}$ **then**
  return REJECT
$E_{BA} \leftarrow \phi_{\mathcal{L}}^{\boldsymbol{t}}(\mathbf{c}.E_B)$
$\hat{y}^{G_1^{E_{BA}}} \leftarrow \phi_{\mathcal{L}}^{\boldsymbol{t}}(\mathbf{c}.\hat{x}^{G_1^{E_B}})$
$\mathbf{a} \leftarrow (E_{BA}, \hat{y}^{G_1^{E_{BA}}})$
return $(\mathbf{a})$

Challenger(pk)**:**

$\boldsymbol{s} \leftarrow_\$ pk.\mathcal{S}^n$
$k_B^{-1}, k_{AB}^{-1} \leftarrow_\$ (\mathbb{Z}_N^*)^2$
$k_A = k_B k_{AB} \deg \phi_{\mathcal{L}}^{\boldsymbol{s}} \bmod N$
$Q^{G_2^{E_A}} \leftarrow_\$ G_2^{E_A}$
$E_{AB} \leftarrow \phi_{\mathcal{L}}^{\boldsymbol{s}}(pk.E_A)$
$Q^{G_2^{E_{AB}}} \leftarrow [k_{AB}]\phi_{\mathcal{L}}^{\boldsymbol{s}}(Q^{G_2^{E_A}})$
$E_B \leftarrow \phi_{\mathcal{L}}^{\boldsymbol{s}}(pk.E)$
$\hat{x}^{G_1^{E_B}} \leftarrow [k_B]\phi_{\mathcal{L}}^{\boldsymbol{s}}(pk.x^{G_1^E})$
$Q^{G_2^{E_A}} \leftarrow [k_A]Q^{G_2^{E_A}}$
$\mathbf{c}_{\mathsf{SeqEval}} \leftarrow (E_B, \hat{x}^{G_1^{E_B}})$
$\mathbf{c}_{\text{verify}} \leftarrow (E_{AB}, Q^{G_2^{E_A}}, Q^{G_2^{E_{AB}}})$
$\mathbf{c} \leftarrow (\mathbf{c}_{\mathsf{SeqEval}}, \mathbf{c}_{\text{verify}})$
return $(\mathbf{c})$

Verify$(pk, \mathbf{c}, \mathbf{a})$**:**

if $\boldsymbol{a}.\hat{y}^{G_1^{E_{BA}}} \notin G_1^{E_{BA}}$ **then**
  return REJECT
if $\boldsymbol{c}.Q^{G_2^{E_{AB}}} \notin G_2^{E_{AB}}$ **then**
  return REJECT
if $\boldsymbol{c}.Q^{G_2^{E_A}} \notin G_2^{E_A}$ **then**
  return REJECT
$l_{\text{Side}} \leftarrow \hat{\mathsf{e}}_N^{E_A}(pk.y^{G_1^{E_A}}, \mathbf{c}.Q^{G_2^{E_A}})$
$r_{\text{Side}} \leftarrow \hat{\mathsf{e}}_N^{E_{BA}}(\mathbf{a}.\hat{y}^{G_1^{E_{BA}}}, \mathbf{c}.Q^{G_2^{E_{AB}}})$
if $l_{Side} = r_{Side}$ & $l_{Side} \neq 1_{\mu_N}$ **then**
  return ACCEPT
return REJECT

Figure 4.2: The proposed trapdoor-VDF instance.

Discussion. It is imperative that a trusted process runs Setup due to an attack that exploits an elliptic curve with known endomorphism rings (see [72]). To generate the public parameters, the Setup must select E and then compute $\phi_{\mathcal{L}}^{\boldsymbol{t}}$, which takes $O(T)$ steps. A Challenger must sample a one-time secret $\boldsymbol{s}$ to compute $\hat{x}^{G_1^{E_B}}$ and $Q^{G_2^{E_{AB}}}$, with the evaluation time being a polynomial function in λ. Essentially the challenge consists of

three elliptic curve points. To obtain the answer, the Solver must compute $\phi_{\mathcal{L}}^{\boldsymbol{t}}$, which is a separable isogeny of a degree exponential in T. Typically, the best approach for computing $\phi_{\mathcal{L}}^{\boldsymbol{t}}$ is to sequentially compute each of its T compositions, of degrees $\ell_i \in \mathcal{L}$, using Vélu's formulas. A single point on the target curve E_{BA} serves as both the answer and the proof. The verification involves two bilinear pairings. It also involves determining whether all points are members of the appropriate group, a relatively trivial operation.

4.4 Security

We will show that the trapdoor-VDF proposed in Section 4.3.4 is secure and well-defined with an honest challenge and the aforementioned setup that takes $O(T)$ steps.

Theorem 5. *The trapdoor-VDF instance in Section 4.3.4 is a secure well-defined trapdoor-VDF with a $O(T)$ steps long Setup and under the assumption of an honest Challenger.*

Correctness and Uniqueness. The following assumes that all statements are true for any $\lambda, T, pk \leftarrow_{\$} \mathsf{Setup}(1^{\lambda}, T)$. For all *honest* $\mathbf{c} \leftarrow_{\$} \mathsf{Challenger}(pk)$, the correctness argument implies that any honest evaluator should be able to obtain the answer $\mathbf{a} \leftarrow \mathsf{Solver}(pk, \mathbf{c})$ which is Verify's acceptable answer.

However, the correctness argument is not complete yet. We also require that the Challenger and Solver must land on the same terminal elliptic curve and not only curves on the same isomorphism class. In other words, the evaluation output $(\phi_{\mathcal{L}}^{\boldsymbol{t}}(\mathbf{c}.\hat{x}^{G_1^{E_B}}))$ and the challenge $(\mathbf{c}.Q^{G_2^{E_{AB}}})$ have to be on the same curve (i.e., $E_{BA} = E_{AB}$). This is required so that the Verifier can evaluate Equation (4.1). The verification in Equation (4.1) is likewise unique, as we employ a similar version of the [28]'s (and [72]) verifier.

Lemma 1. *Given any λ, T, and $pk \leftarrow_{\$} \mathsf{Setup}(1^{\lambda}, T)$, a Solver's honest output $\boldsymbol{a}.\hat{y}^{G_1^{E_{BA}}}$ is indeed a point on the Challenger's terminal curve (E_{AB}) such that $\boldsymbol{a}.\hat{y}^{G_1^{E_{BA}}} = \hat{y}^{G_1^{E_{AB}}}$ and $\textsc{accept} \leftarrow \mathsf{Verify}(pk, \boldsymbol{c}, \boldsymbol{a})$, for all honest $\boldsymbol{c} \leftarrow_{\$} \mathsf{Challenger}(pk)$; further, there is no answer $\boldsymbol{a}'$ that is a valid answer (i.e, $\textsc{accept} \leftarrow \mathsf{Verify}(pk, \boldsymbol{c}, \boldsymbol{a}')$) unless $\boldsymbol{a}' = \boldsymbol{a} = \phi_{\mathcal{L}}^{\boldsymbol{t}}(\boldsymbol{c}.\hat{x}^{G_1^{E_B}})$.*

Proof. The proof to the first part of the lemma follows from the result of Leonardi [105, Theorem 3.1]. Leonardi's result implies the equivalency between the action of $\phi_{\mathcal{L}}^{\boldsymbol{s}} \circ \phi_{\mathcal{L}}^{\boldsymbol{t}}$ and $\phi_{\mathcal{L}}^{\boldsymbol{t}} \circ \phi_{\mathcal{L}}^{\boldsymbol{s}}$ when all computed with Vélu's formulas. As for the uniqueness, we have set our parameters so that (i) as mentioned in Section 4.3.3, the surjective morphism

$\phi_{\mathcal{L}}^{\boldsymbol{t}} : E_B[N] \to E_{BA}[N]$ is a group isomorphism on the N-torsion subgroup and (ii) as discussed in [72], Equation (4.1)'s right-hand side is a group isomorphism from $G_1^{E_{BA}}$ to μ_N for a given $\mathbf{c}.Q^{G_2^{E_{AB}}}$. Hence, $\mathsf{Verify}((pk, \mathbf{c}), \mathbf{a}')$ outputs ACCEPT if and only if $\mathbf{a}' = \mathbf{a} = \phi_{\mathcal{L}}^{\boldsymbol{t}}(\mathbf{c}.\hat{x}^{G_1^{E_B}}) \in G_1^{E_{BA}}$. □

Soundness. Since there is only one valid answer, the verification is unique, as is a property of [28]'s verifier. Thus, and similar to [72], our trapdoor-VDF is perfectly sound.

Sequentiality. The trapdoor-VDF instance presented in Section 4.3.4 satisfies sequentiality if no PPT adversary $\mathcal{A} := (\mathcal{A}_1, \mathcal{A}_2)$ has a non-negligible probability of winning the following shortcut game:

- A trusted process computes and publishes $pk \leftarrow_{\$} \mathsf{Setup}(1^{\lambda}, T)$.
- $\mathcal{A}$ performs a pre-computation in time $\mathsf{Poly}(T, \lambda)$ and outputs $\mathsf{pc} \leftarrow \mathcal{A}_1(pk)$.
- A trusted process then computes and publishes $\mathbf{c} \leftarrow_{\$} \mathsf{Challenge}(pk)$.
- $\mathcal{A}$ outputs $\hat{y}^{G_1^{E_{BA}}} \leftarrow \mathcal{A}_2(pk, \mathbf{c}, \mathsf{pc})$ in parallel time less than T, where $\hat{y}^{G_1^{E_{BA}}} \in G_1^{E_{BA}}$.

To win the game, $\mathcal{A}$'s output, $\hat{y}^{G_1^{E_{BA}}}$, must be the correct evaluation of $\phi_{pk.\mathcal{L}}^{pk.\boldsymbol{t}}$ on $\mathbf{c}.\hat{x}^{G_1^{E_B}}$ (i.e., it must be equal to $\mathbf{a} \leftarrow \mathsf{Solver}(pk, \mathbf{c})$).

An adversary $\mathcal{A}$ can win the game by finding (i) a shorter isogeny than $\phi_{\mathcal{L}}^{\boldsymbol{t}}$, (ii) a faster method for computing $\phi_{\mathcal{L}}^{\boldsymbol{t}}$, (iii) the inverse of pairing Equation (4.1), or (iv) the secret isogeny $\phi_{\mathcal{L}}^{\boldsymbol{s}}$ (or a short equivalent isogeny). The three former points have already been discussed in [72, Section 6]. With regards to the last point, the sequentiality is also determined by recovering the secret isogeny $\mathsf{tr}_{sk} := \phi_{\mathcal{L}}^{\boldsymbol{s}}$ or obtaining a short equivalent isogeny. In other words, this is equivalent to recovering the ideal class $[\mathfrak{b}] \in \mathbf{cl}(\mathcal{O})$ that is computed via the secret isogeny $\phi_{\mathcal{L}}^{\boldsymbol{s}}$.

Definition 8 (Key recovery [41])**.** *Given two supersingular elliptic curves E and E_B defined over $\mathbb{F}_p$ with the same $\mathbb{F}_p$-rational endomorphism ring $\mathcal{O}$, find an ideal $\mathfrak{b}$ such that $[\mathfrak{b}]E = E_B$ and $\mathfrak{b}$ is the product of ideals of small norm in $\mathcal{L}$.*

There are several possible attacks for recovering the secret, some of which are discussed in [41].

Brute-Force Attack on the Secret Key. A basic attack searches over all potential keys. For a secret $\boldsymbol{s}$, its corresponding secret ideal class $[\mathfrak{b}]$ is represented as $[\mathfrak{l}_1^{m_1}\mathfrak{l}_2^{m_2}\cdots\mathfrak{l}_n^{m_n}] \in \mathbf{cl}(\mathcal{O})$

such that the norm $\mathsf{N}(\mathfrak{l}_i) = \ell_i \in \mathcal{L}$ and $m_i \in \boldsymbol{s}$. Thus, it can be argued that $[\mathfrak{b}]$ has several representations (i.e., there are an equivalent $\hat{\boldsymbol{s}} \leftarrow \mathcal{S}^n$ that yields $[\mathfrak{b}] \in \mathbf{cl}(\mathcal{O})$), which is vulnerable to exhaustive search. Castryck et al. [41] show that the expected number of equivalent representations is $|\mathcal{S}|^n/\mathsf{ord}(G)$, assuming that $\mathbf{cl}(\mathcal{O})$ is almost cyclic with a very large cyclic component of order $\mathsf{ord}(G)$ close to $|\mathbf{cl}(\mathcal{O})|$. Therefore, it suffices to choose $n \log |\mathcal{S}| \approx \log(\sqrt{p})$, where $|\mathbf{cl}(\mathcal{O})| \approx \sqrt{p}$.

Pohlig–Hellman-Style Attack. To our knowledge, the Pohlig–Hellman style attacks cannot be effectively applied to the construction in Section 4.3.4 to recover the secret degree ($\deg \phi_{\mathcal{L}}^{\boldsymbol{s}}$) or the answer ($\hat{y}^{G_1^{E_{BA}}}$).

Castryck-Decru-Style Attack [40]. As in [75], we mask the torsion points in pairing-based verification. Therefore, the point pairs $(y^{G_1^{E_A}}, [k_A]Q^{G_2^{E_A}})$ on E_A and $(\hat{y}^{G_1^{E_{AB}}}, Q^{G_2^{E_{AB}}})$ on E_{AB} are not images of the secret isogeny $\phi_{\mathcal{L}}^{\boldsymbol{s}}$. In addition, the secret isogeny $\phi_{\mathcal{L}}^{\boldsymbol{s}}$ has a hidden degree determined by $\boldsymbol{s}$, which is similar to the countermeasure suggested in [123, 76]. Hence, to the best of our knowledge, our proposal is naturally resistant to a Castryck-Decru-style attack because we conceal torsion points' preimage and the secret isogeny degree.

Meet-in-the-Middle Attack (MITM). To find the path $(\phi_{\mathcal{L}}^{\boldsymbol{s}} : E_A \to E_{AB})$ in the isogeny graph $\bigcup_{\ell_i \in \mathcal{L}} \mathcal{G}(\mathbb{F}_p, \ell_i)$, MITH starts from E_A and E_{AB} to construct search trees. The attacker looks for the collision in the tree. The halfway point between E_A and E_{AB} is the isogeny evaluation of the halves of the set $\mathcal{L}$, $\mathcal{L}_{\mathbf{left}} := \{\ell_1, \ell_2, \ldots, \ell_{n/2}\}$, and $\mathcal{L}_{\mathbf{right}} := \{\ell_{\frac{n}{2}+1}, \ldots, \ell_n\}$ (for simplicity assume that n is even). Let $\mathcal{J}$ be the set size of all elliptic curves defined over $\mathbb{F}_p$ that are isogenous to E_A, constrained to $\mathcal{S}$, in the halfway to E_{AB}. We observe that the set size is $|\mathcal{J}| = |\mathcal{S}|^{n/2} - 1$; thus the attack average-case isogeny computation is about $\approx 2^{n \log(|\mathcal{S}|)/2}$. Setting $n \log(|\mathcal{S}|)$ to be $\approx \log(p)/2$, the attack's average-case complexity is $2^{\log(p)/4}$.

4.5 Chapter Summary

In this chapter, we have presented a new class of trapdoor-VDFs that have a large ensemble of trapdoors for each VDF instantiation. In the proposed scheme, a secret trapdoor, chosen randomly from the ensemble by Challenger, is used to generate a challenge. In a few steps, Challenger can obtain the answer to a challenge by using the secret trapdoor. Without knowing the secret trapdoor, a Solver on the other hand must perform the long sequential computations. We have also presented a trapdoor-VDF instance from the action

of class groups on supersingular elliptic curves with bilinear pairings being used for public verification.

Chapter 5

ChronoCloak: An integrated Solution for Mitigating Premature Disclosure in Oblivious Digital Dissemination

With the increasing use of online digital content delivery, such as games, videos, and magazine articles, there are scenarios where it is crucial to prevent premature revelation of the distributed content and, at the same time, allow a recipient to choose any specific item of the content bundle in a privacy-preserving manner. This chapter presents an integrated solution, namely ChronoCloak, which aims to address both the premature exposure of the sender data and the privacy preservation of the receiver interaction. ChronoCloak allows a sender to transmit a set of secrets through a puzzle, which can be solved via a lengthy computation. Upon solving the puzzle, the receiver recovers only a subset of the secrets that is oblivious to the sender. It also allows the receiver to securely outsource the computation with public verifiability, yet only the intended receiver can retrieve a subset of secrets using the puzzle solution. We also present an ideal functionality for ChronoCloak and provide a generic protocol implementing this functionality in the random oracle model, utilizing a ChronoCloak instance constructed using a trapdoor-VDF and an oblivious transfer protocol. A slightly shorter version of the work presented in this chapter appeared in [160].

5.1 Introduction

In certain cases of digital content distribution, it is crucial for the distributor to be able to send a selection of time-locked items (e.g., games, videos, or magazine articles) to a recipient who can then choose and unlock one of those items after a predetermined time in a privacy preserving manner so that the distributor is unaware of the recipient's choice. For example, the publisher of a weekly magazine may send digital copies of encrypted articles of an upcoming issue to its reader base in advance. Then, a *pay-per-article* subscriber, on or after the publication date, can decrypt one of the articles of their choice that is oblivious to the publisher. The core issues, namely time-locking of contents and their oblivious transfer, in the aforementioned applications can potentially be addressed by carefully deploying relevant cryptographic primitives and other necessary functional blocks. For example, one existing solution is a time-release oblivious transfer employing a verifiable ID-based encrypted blind signature [157], where a trusted third party (time server) is used to ensure the release of the message at a predetermined time in the future. Time tokens can be broadcast periodically only if the server is online. In the work of [112], privacy is guaranteed only for a limited period of time. As in [157], [112] uses a time server to release a private key that enables the sender to learn the receiver's choice after a specified amount of time has passed. A second solution can be constructed using [15]'s framework, which is an inefficient method of constructing similar functionality using multiple primitives. However, no concrete construction has been proposed. As a trusted setup can be difficult to perform securely, using multiple primitives as a solution may require multiple calls to a trusted setup. This significantly complicates the system setup process, increasing potential security risks. We then ask: If both issues occur simultaneously in a system, how can we address them in a single solution?

Contributions. In this work, we consider an integrated approach to address time-delayed decryption and oblivious transfer and propose a solution called *ChronoCloak*. Through ChronoCloak, the sender and receiver interact via secret trapdoors and construct a joint puzzle. After solving the puzzle via a lengthy computation (i.e., a sequential computation of length $T \in \mathbb{N}$), the receiver can obliviously retrieve a subset of the sender's secrets. More importantly, our ChronoCloak allows the recipient to outsource the lengthy computation to an external solver without worrying that the latter can decrypt the sender's secret. This is because the puzzle is jointly generated using the secrets of both the sender and the intended recipient; thus, recovering the transmitted secret requires more than just the solution to the puzzle - it also requires the recipient's secret. In this chapter, we introduce an ideal functionality for ChronoCloak and propose a generic protocol that takes any ChronoCloak construction proven secure in a game-based setting and *lifts* it into a

UC-secure implementation of this functionality, resulting in stronger security guarantees. Specifically, we present a ChronoCloak instance constructed using an oblivious transfer protocol and a trapdoor-VDF, which, when integrated into our protocol, achieves a UC-secure implementation.

5.1.1 Related Work

As discussed previously, in the literature there are proposals for constructing Oblivious Transfer (OT) protocols with time-release delays, among which [55, 112, 157] are the most pertinent. The problem of sending messages into the future with conditional oblivious transfer was considered by Crescenzo et al. [55]. In the case of delayed release, the authors of [157] incorporate a trusted third party as a time server to ensure the scheduled release of the message. However, in the work of [112], the receiver's choice privacy is maintained within a limited time frame. Similar to [157], the work in [112] uses a time server to release a private key that allows the sender to learn the receiver's choice after a specified time. As the time server generally requires the release of some secrets at every time interval, the reliability and security requirement for the protocol's trusted-party extend beyond the one-time setup of initializing the protocol parameters.

Delay-based primitives. Aside from time servers, several cryptographic primitives have been proposed to guarantee time delay, such as time-lock puzzles (TLPs) [131], proof of sequential work [2, 51, 104, 114], and verifiable delay functions (VDF) [44, 69, 110, 128, 156] (and its relative trapdoor VDF [156, 159]). The aforementioned primitives however posit an adversarial model that is non-simulation based. Hence, they do not provide any composability guarantees (that is, securely integrating them into more complex protocols is not intuitive). This shortfall can be addressed by the universal composability framework [36]. Baum et al. [16] propose UC-secure TLP construction based on the repeated-squaring assumption in the random oracle model. Later on, they extend their work to a UC-secure TLP with public verifiablility and VDF in [15]. There are also other works that discuss time-lock puzzles, with public verifiability, that uphold composability, such as [5, 68, 77, 93].

Oblivious transfer. The concept of oblivious transfer was initially introduced by [129]. Since then, it has been a crucial building block in many secure multi-party computations [47, 89, 83]. Over time, different constructions of OT protocols have been developed based on different assumptions. One notable example is the Chou-Orlandi OT scheme [48], which is a three-round OT based on Diffie-Hellman (DH) key exchange. Although the Chou-Orlandi scheme did not achieve UC security [13, 84, 106], their core idea inspired the

design of secure and more efficient OT protocols such as [8, 38, 100, 58]. Also, there are a number of isogeny-based oblivious transfer constructions, including [3, 9, 12, 99, 58, 150]. In [3], the authors propose a framework for developing cryptographic primitives, which is amenable to group actions such as CSIDH [41] and CSI-FiSh [21]. Using their framework, the authors describe the construction of a variety of cryptographic primitives, including a (inefficient) statistically sender-private OT protocol. A notable work by Lai et al. [100] presented an efficient OT construction based on a variant of computational CSIDH, the reciprocal CSIDH assumption. They rely on the existence of an efficient quadratic twist map in a CSIDH setting (which has no analog in a DLP setting) to obtain an optimized 2-round version of a tweaked 3-round variant of the Chou-Orlandi OT scheme. In a trusted setup model, they use the quadratic twist and "proof of decryption" (PoD) mechanisms [48] (that was used in [13] and pointed out in [48]) in order to achieve a fully-simulable OT construction. However, their PoD assumption had a security flaw that was rectified in the revised version [99], which required two extra rounds, resulting in a four-round OT construction. An optimized OT extension to the work of Lai et al. [100] has been proposed in [9], reducing the overall number of isogeny computations. However, it does lead to a weaker notion of OT security, i.e., OT with selective failure attack. Moreover, they propose four and two round-optimal OTs in the plain and common reference string models, respectively. On the other hand, we note that some of the isogeny-based schemes such as [12, 58, 150] have been affected by recent attacks [40] on SIDH, but there is no known way to extend such attacks to the general isogeny problem. There are several other assumptions that can be used to construct oblivious transfer, such as lattice-based [121] and code-based [56, 64, 30].

Organization. We begin Section 5.2 by introducing the essential concepts of the security model and the related definitions of the building blocks of our work. Then, we discuss the underlying UC Framework and our proposed *Operation Unit* in Section 5.3. In Section 5.4, we introduce the notion of ChronoCloak and its ideal functionality. Section 5.5 presents a generic protocol based on the ChronoCloak definition. Section 5.5.2 then shows that our generic protocol is UC secure, implementing the intended functionality of ChronoCloak. In Section 5.6, we present an instance from delayed oblivious transfer. Finally, Section 5.7 concludes with our closing remarks.

5.2 Chapter Background

Computational indistinguishability. Following previous works, we will adopt some notions from [36, 100]. We represent a probability ensemble X by $\{X(z,\lambda)\}_{z,\lambda}$ to define

an infinite sequence of random variables indexed by $z \in \{0,1\}^*$ and λ. Further, we say that $X = \{X(z,\lambda)\}_{z\in\{0,1\}^*,\lambda}$ and $Y = \{Y(z,\lambda)\}_{z\in\{0,1\}^*,\lambda}$ are computationally indistinguishable (denoted by $X \stackrel{c}{=} Y$) if for every PPT distinguisher D there exists a negligible function in λ (denoted by $\mathsf{neg}(\lambda)$) on all $z \in \{0,1\}^*$ and large $\lambda \in \mathbb{N}$, it holds that

$$|\mathsf{Pr}[D(X(z,\lambda)) = 1] - \mathsf{Pr}[D(Y(z,\lambda)) = 1]| \leq \mathsf{neg}(\lambda).$$

5.2.1 UC Security Model

In this work, we use the universal composability framework to evaluate the security of our proposed protocol, using the notations from [36, 88, 107]. Introduced by Canetti in 2001, the UC framework is a strong security model that ensures the security of the protocol regardless of how it is *composed* with other protocols. This is achieved by considering not only the isolated execution of our protocol, but also the environment it operates in. Here is a summary of the key definitions within the UC security model.

Ideal Functionality

In the UC framework, the ideal functionality F acts as a trusted third party. It allows parties to securely compute output(s) based on their secret input(s) while ensuring that no unintended information is disclosed to either party beyond what is specified by F. Hence, it is an abstract definition that precisely outlines what the protocol is supposed to achieve without giving specifics on its implementation.

For example, consider two-party protocol Γ in the UC framework where its functionality F is defined as $\mathsf{F} := (\mathsf{F}_1, \mathsf{F}_2) : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$. In this setup, the first party, providing input x_1, aims to receive the output $\mathsf{F}_1(x_1, x_2)$, while the second party, providing input x_2, seeks $\mathsf{F}_2(x_1, x_2)$. This setup ensures that each party obtains only their designated output based on both of their inputs, so that no unintended information about the inputs of the other is disclosed beyond what is strictly specified by F.

The ultimate goal of using the concept of ideal functionality F in the UC framework is to ensure that any protocol Γ realizing its intended functionality F can be safely replaced with F itself in any larger more complex system (or protocol) without compromising security.

Static Corruption

Parties in the UC model are classified either as honest or corrupted. Honest parties follow the protocol specification, whereas corrupted parties (controlled by an adversary) deviate

from the protocol specification. In this work, we consider the UC model with static corruption, which means that honest parties remain honest throughout the execution of the protocol and corrupted parties remain corrupted.

Environment Distinguisher

The Environment Distinguisher Env serves as an external entity that models all external influences on the protocol. This includes generating inputs of the protocol's parties (including the corrupted ones and the simulator $\tilde{\mathcal{A}}$'s auxiliary input) and observing the outputs that these parties return. Also, Env interacts with the adversary A, who may corrupt some of the parties.

The core task of Env is to distinguish between the *ideal world* (where parties interact via the intended ideal functionality F) and the *real world* (where parties execute the actual protocol Γ). Informally, it differentiates between these two worlds based on its observations of the outputs and its knowledge of the inputs it has provided. In the context of our work, Env also implicitly accounts for the total execution time, or delay, when making its decision. In addition, Env engages in active interaction (not passive) in an attempt to identify whether it operates in the ideal world or the real world.

In the UC framework, the security argument asserts that if no Env can distinguish between the ideal and real worlds with a non-negligible advantage, then the protocol is considered to securely realize the intended ideal functionality of Γ.

Adversarial Models

In the UC security model, two types of adversaries are typically considered based on their capabilities: *semi-honest* and *malicious* adversaries.

Semi-Honest Model: A semi-honest adversary is a curious actor who adheres to protocol specifications and does not alter its initial inputs, as per some strategy, but attempts to gain critical information from its observations.

In this context, the view of the jth party, denoted by view_j^{Γ}, during the execution of a protocol Γ includes everything observed by the party, such as its initial input x_j, random tape and received messages. The output of the jth party, denoted by $\mathsf{output}_j^{\Gamma}$, is obtained from its $\mathsf{view}_j^{\Gamma}(x_1, x_2)$. In a protocol that implements a deterministic functionality $\mathsf{F} := (\mathsf{F}_1, \mathsf{F}_2)$, the correctness of the protocol is simply defined as $\{\mathsf{output}_1^{\Gamma}(\lambda, x_1, x_2), \mathsf{output}_2^{\Gamma}(\lambda, x_1, x_2)\}_{x_1,x_2,\lambda} = \{\mathsf{F}_1(x_1, x_2), \mathsf{F}_2(x_1, x_2)\}$ for the input pair (x_1, x_2) and a security

parameter $\lambda \in \mathbb{N}$. This ensures that the output produced by the protocol with given inputs deterministically reflects the expected output of F.

Our goal is to show that the view of any party during the execution of a protocol can be simulated using only its input and output, without needing any direct information from the actual execution of the protocol.

Definition 4. *Against semi-honest adversaries, a protocol Γ securely implements a deterministic functionality $F := (F_1, F_2)$, if there exists a simulator $\tilde{\mathcal{A}}_j$ such that*

$$\{\mathit{output}_j^{\Gamma}(\lambda, x_1, x_2)\}_{x_1,x_2,\lambda} = \{F_j(x_1, x_2)\} \text{ and}$$
$$\{\tilde{\mathcal{A}}_j(\lambda, x_j, F_j(x_1, x_2))\}_{x_1,x_2,\lambda} \stackrel{c}{=} \{\mathit{view}_j^{\Gamma}(\lambda, x_1, x_2)\}_{x_1,x_2,\lambda}. \ \forall j \in \{1, 2\},$$

Malicious Adversarial Model: A malicious adversary, denoted as A, has no restrictions on its choice of adversarial strategy. Depending on its approach, it may deviate from protocol specifications, alter its inputs, or adoptively change its strategy. This may involve sending incorrect messages, aborting the execution, or even choosing not to send any messages. Hence, the malicious adversarial model allows us to capture a wide range of real-world attacks.

In this work, we discuss the security of our proposed protocol in the presence of a malicious adversary. In this model, we cannot analyze security solely based on the existence of a simulator that indistinguishably simulates the view of the jth party (corrupted by A_j) as outlined in Definition 4. This is because of the capability of A_j to deviate arbitrarily from Γ and alter its inputs (e.g., x_j). As a result, in the security analysis, our goal is to establish that what an adversary A_j can do in the real execution of a protocol is no more than what it can do in an ideal scenario.

Modes of Executions

Here, we outline protocol execution in different scenarios taking into account the complexities introduced by a malicious adversary A, including the actual execution of the protocol (*real* execution), the execution with an intended behavior (*ideal* execution), and a *hybrid* version that combines elements of both.

Real execution. This scenario involves parties interacting directly to execute a protocol Γ. We denote the distribution ensemble of all parties' outputs from the real execution of Γ, where one party is corrupted by A, as $\mathsf{Real}_{\Gamma,\mathsf{A},\mathsf{Env}}$.

Ideal execution. As the name suggests, the ideal execution represents a scenario where parties interact through the ideal functionality F of a protocol Γ, which acts as a trusted

incorruptible third party. In other words, this execution should follow the expected behavior of Γ through F.

Briefly, after parties obtain their inputs, the honest party sends its private input to F, and a simulator (i.e., an ideal world adversary, denoted by $\tilde{\mathcal{A}}$) sends the corrupted parties' inputs to F. At an early stage, $\tilde{\mathcal{A}}$ can also send an *abort* message to F, which then forwards it to all parties to terminate the execution. If no abort message is sent, F computes the outputs for all parties based on the given inputs and then sends the outputs of the corrupted parties to $\tilde{\mathcal{A}}$. $\tilde{\mathcal{A}}$ then signals F to either send the corresponding outputs to the honest parties or abort the execution. We denote the distribution ensemble of all parties' outputs from the ideal execution through F, where one party is corrupted by $\tilde{\mathcal{A}}$, as $\mathsf{Ideal}_{\mathsf{F},\tilde{\mathcal{A}},\mathsf{Env}}$.

Security definition with ideal and real execution. Put simply, our goal is to establish that a malicious adversary A in the real execution of the protocol does not has more advantage than $\tilde{\mathcal{A}}$ has in the ideal execution. In other words, for a protocol to be secure, we need to show that there exists a simulator $\tilde{\mathcal{A}}$ in the ideal execution that can simulate the execution of the real protocol in the presence of A.

Definition 5. *Against malicious adversaries, a protocol Γ securely implements a deterministic functionality F if*

$$\exists \text{ a simulator } \tilde{\mathcal{A}} \text{ s.t. } \mathsf{Ideal}_{\mathsf{F},\tilde{\mathcal{A}},\mathsf{Env}} \stackrel{c}{=} \mathsf{Real}_{\Gamma,\mathsf{A},\mathsf{Env}},$$

for any probabilistic polynomial-time adversary A and distingusher Env.

It is obvious that, for $\tilde{\mathcal{A}}$ to correctly simulate the real execution, it must obtain the input of A (which may be altered) and then use it to participate in the ideal execution. In more detail, using a blackbox approach, the simulator $\tilde{\mathcal{A}}$ interacts with A through real protocol interactions to extract A's inputs. The extracted inputs are then passed to F to obtain A's corresponding output. As a result, $\tilde{\mathcal{A}}$ can correctly simulate the real execution based on inputs from both honest parties (in the ideal world) and A. The simulator's transcript is therefore indistinguishable from the real execution and consistent with the ideal execution (up to some assumptions).

In many contexts, including our work, the $\tilde{\mathcal{A}}$ extraction process can be challenging and may require some kind of trusted setup to simplify it. This complexity is pointed out by Canetti et al. [37]. Hence, we present a scenario for a hybrid version of the execution.

Hybrid execution. In this scenario, we assume the execution of a complex protocol that involves calls to a subroutine(s) like a hash function or other primitives/protocols such

as oblivious transfer protocol. We also assume that some of these subroutines are UC-secure, enabling us to replace them with their ideal functionalities (e.g., $\hat{\mathsf{F}}_1, \cdots, \hat{\mathsf{F}}_n$) when constructing the protocol. This replacement simplifies the security proof of our protocol. Therefore, we say that the $(\hat{\mathsf{F}}_1, \cdots, \hat{\mathsf{F}}_n)$-hybrid execution of the protocol is the one where parties interact not only with each other, as in real execution, but also through a trusted entity, $\hat{\mathsf{F}}_1, \cdots, \hat{\mathsf{F}}_n$. Finally, we use the notation $\mathsf{Hybrid}^{\hat{\mathsf{F}}_1,\cdots,\hat{\mathsf{F}}_n}_{\Gamma,\mathsf{A},\mathsf{Env}}$ to represent all parties' output ensemble from the real execution in which parties (one of whom is corrupted by A) have access to functionalities $\hat{\mathsf{F}}_1, \cdots, \hat{\mathsf{F}}_n$.

Security definition with ideal and hybrid execution. Following our discussion, we redefine our security in Definition 5 as follows:

Definition 6. *Against malicious adversaries, a construction Γ securely implements a deterministic functionality F in the $(\hat{F}_1, \cdots, \hat{F}_n)$-hybrid model if*

$$\exists \text{ a simulator } \tilde{\mathcal{A}} \text{ s.t. } \mathit{Ideal}_{F,\tilde{\mathcal{A}},\mathit{Env}} \stackrel{c}{=} \mathit{Hybrid}^{\hat{F}_1,\cdots,\hat{F}_n}_{\Gamma,\mathcal{A},\mathit{Env}},$$

for any probabilistic polynomial-time adversary $\mathcal{A}$ and distingusher Env.

5.2.2 Related Functionalities

In this subsection, we define some key ideal functionalities essential to our work, including those for the oblivious transfer protocol and the random oracle.

Ideal Functionality of the Oblivious Transfer Protocol, F_{OT}

In Figure 5.1, we present the ideal functionality of 1-out-of-2 oblivious transfer protocol, namely F_{OT}.

The ideal functionality F_{OT}.

Commit: Upon receiving (Commit, *sid*, P_S, $i \in \{0,1\}$) from P_R, check if a (P_R, P_S, *sid*, $i \in \{0,1\}$) was previously stored. If yes, ignore; otherwise, store (P_R, P_S, *sid*, $i \in \{0,1\}$) and output (Commit, *sid*, P_R) to P_S.

Challenge: Upon receiving (Challenge, *sid*, P_R, m_0, m_1) from P_S, check if a (P_R, P_S, *sid*, i) was previously stored. If not, ignore; otherwise, send (Challenge, *sid*, P_S, m_i)

continued on the next page

continued from the previous page

to P_R and ($\mathsf{Challenge}$, sid, P_R) to P_S.

Figure 5.1: 1-out-of-2 oblivious transfer ideal functionality (F_{OT}).

Ideal Functionality of a Random Oracle, F_{RO}

In our study, we analyze our protocol within the random oracle model. In a context relevant to our work, it has been shown that achieving a universally composable secure time-lock puzzle is not possible outside of this model [5, 111]. In our case, the reason will be discussed in Subsection 5.3.2.

In Figure 5.2, we present the ideal functionality of the random oracle F_{RO} that defines a uniformly random function : $\{0,1\}^* \rightarrow \mathcal{H}$. This functionality is initiated by the simulator $\tilde{\mathcal{A}}$ with an empty record, $\mathbf{H}$. When a query is received from a party, P, it checks if there is a record containing an entry ($-$, inString) in $\mathbf{H}$. If there is, it retrieves $str \in \mathcal{H}$ from $\mathbf{H}$, i.e., retrieve (str, inString) $\leftarrow \mathbf{H}$. If there is not, it samples an element $str \leftarrow_{\$} \mathcal{H}$, and then adds the tuple (str, inString) to $\mathbf{H}$. Finally, it returns $str \in \mathcal{H}$ to P.

The ideal functionality F_{RO}.

Initiate: Upon receiving ($\mathsf{Initiate}$, sid) from $\tilde{\mathcal{A}}$, setup an empty record $\mathbf{H}$, representing the state of $\mathsf{F}_{\mathsf{RO}} : \{0,1\}^* \rightarrow \mathcal{H}$; and no further messages of this type is accepted.

Query: Upon receiving (Query, sid, inString $\in \{0,1\}^*$) from participant P, return (Query, sid, $str \in \mathcal{H}$) as follows: (i) If (-, inString) $\notin \mathbf{H}$ (i.e., previously, there is no record containing an entry inString in $\mathbf{H}$), sample $str \leftarrow_{\$} \mathcal{H}$ and then add the tuple (str, inString) to $\mathbf{H}$. (ii) If there is a record containing inString in $\mathbf{H}$, retrieve str from $\mathbf{H}$ (i.e., retrieve (str, inString) $\leftarrow \mathbf{H}$).

Figure 5.2: The random oracle ideal functionality (F_{RO}).

5.3 Underlying UC Framework and Operation Unit

In this section, we present an overview of the underlying UC framework to be used in the remainder of this chapter. We first present an informal definition related to a generic

sequential function. Following this, we highlight the concept of capturing the abstract progression of computations through a series of ticks from a global ticker $\mathsf{F}_{\mathsf{clk}}$. To capture computation costs (expressed in ticks from $\mathsf{F}_{\mathsf{clk}}$), we then introduce an *operation unit* (OU) and its ideal functionality (F_{OU}), which each party accesses to execute a protocol's algorithms.

Long Sequential Functions. We use T to characterize the difficulty of a function, which represents the amount of sequential work required to compute its output for any random input with a polynomially large number of parallel computations. A short function has a small T, while a function with a large T is considered a long function. Throughout this work, we refer to the long sequential function as $\mathsf{SeqEval}$. $\mathsf{SeqEval}$ is defined with parameters for its domain and range. A shorter version of $\mathsf{SeqEval}$, referred to as a shortcut or trapdoor, is denoted by tr.

Abstraction of Steps in UC Framework

In this chapter, we use the abstract concept of "ticks", introduced in [16], to capture sequential computation, a replacement for the step-based abstraction used in previous chapters. The objective is to express different timing assumptions within the UC framework through the tick abstraction. In other words, tick serves as a unifying abstraction to coordinate between different events (e.g., computational steps and message delivery) without explicitly synchronizing parties or exposing concrete time. So, this abstraction models a universal discrete units of passing time. Using this abstraction, parties and protocols do not ask for current time itself but rather observe events triggered by time passing. Thus, timing-related actions and activities become implicit within functionalities or protocols, as they have the flexibility to interpret ticks according to their own assumption. For instance, computational steps evaluating $\mathsf{SeqEval}$ can be represented as x number of ticks while at the same time represents y number of ticks to deliver a message in a channel.

Ideal functionality of Global Ticker

As depicted in Figure 5.3, our framework relies on the ideal functionality of a global ticker ($\mathsf{F}_{\mathsf{clk}}$) from [16], providing ticks on a one-at-a-time basis. $\mathsf{F}_{\mathsf{clk}}$ is designed to capture the units of passing time and starts with an empty record of enrolled functionalities and parties. Our work implicitly assumes that all relevant functionalities and parties of a protocol are enrolled with $\mathsf{F}_{\mathsf{clk}}$.

The ideal functionality $\mathsf{F}_{\mathsf{clk}}$.

Enrollment: Upon receiving Enroll from a party P (resp. for a functionality F), add P to $\mathbf{P}$ (resp. add F to $\mathbf{F}$); then send back Enrolled.

Ticked request: Upon receiving IsTicked from $\mathsf{F} \in \mathbf{F}$ and $\mathsf{F} \notin L_{\mathsf{F}}$, add F to L_{F}, and send ticked to F; otherwise, send NotTicked to F.

Ready: Upon receiving Ready from $\mathsf{P} \in \mathbf{P}$, add P to L_P, and send Success to P.

Tick: Upon receiving tick from Env, if $\mathbf{P} = L_p$, set both $L_p = \varnothing$ and $L_{\mathsf{F}} = \varnothing$, and send ticked to the adversary $\tilde{\mathcal{A}}$. Otherwise, send NotTicked to Env.

Figure 5.3: The functionality of $\mathsf{F}_{\mathsf{clk}}$-global ticker from [16].

$\mathsf{F}_{\mathsf{clk}}$'s ticks are initiated by the environment Env's query Tick, but $\mathsf{F}_{\mathsf{clk}}$ will only tick if all parties have sent (in an arbitrary order) a Ready query. Moreover, to emulate a delay based on computation, parties do not (directly) receive notification of ticks (excluding simulator $\tilde{\mathcal{A}}$). They rather discretely query the relevant functionalities, which in turn respond appropriately to each query based on its internal state. Upon query such as Elicitate, functionalities first check (via IsTicked) with $\mathsf{F}_{\mathsf{clk}}$ to determine if a tick has occurred. In such a case, the functionality's Tick interface is triggered internally. After that, the functionality returns the appropriate message (on request) to the relevant party(ies). Once all relevant functionalities have responded, the party submits a Ready query to $\mathsf{F}_{\mathsf{clk}}$, requesting an advance to the next tick. For clarity, we assume that the query messages to/from $\mathsf{F}_{\mathsf{clk}}$ are handled implicitly in our protocol due to their repetitiveness.

Generally, the environment is capable of conducting computations instantaneously relative to other parties, e.g., it can perform computation before activating honest parties. To effectively simulate time delay and align with a more realistic model, we propose a global operation unit OU, mandating that all parties utilize the OU for any form of *arbitrary computation*. The OU operates under tick-based functionality governed by $\mathsf{F}_{\mathsf{clk}}$. This ensures that no party, including the environment, can advance computationally ahead of others, constraining the environment's advantage.

5.3.1 Modeling Operation Unit

Here we introduce an operation unit OU with a finite instruction set (INST). OU is formally defined as follows: $\mathsf{OU} : \mathsf{INST} \times \mathsf{STR} \to \mathsf{STR}$, where STR is the input/output space. Each party P has private access to OU, through which P executes algorithm steps. To execute an algorithm, P translates the steps into a sequence of instructions (i.e., a sequence of instructions executing a step S is referred to as $\mathbf{Inst}_{\mathrm{S}} = \{\mathsf{Inst}_j\}_{j\in[k]}$ s.t. $k \in \mathbb{N}$ and all $\mathsf{Inst} \in \mathsf{INST}$) and parses the input in STR's format. P has the flexibility to construct $\mathbf{Inst}_{\mathrm{S}}$ arbitrarily (of polynomial size), reflecting the strategy used to achieve the desired output.

OU's execution model: Let τ_F be a *global* function that takes $\mathbf{Inst}_{\mathrm{S}}$ as input and outputs an integer $\tau \in \mathbb{N}$, representing the number of ticks required for $\mathbf{Inst}_{\mathrm{S}}$'s execution. To emulate computation costs (in ticks), we introduce the ideal functionality F_{OU}, which captures the process of executing OU described earlier. To mimic OU's execution cost, F_{OU} computes the instruction sequence $\mathbf{Inst}_{\mathrm{S}}$ instantly on a given input in but withholds the output until $\tau = \tau_F(\mathbf{Inst}_{\mathrm{S}}, in)$ ticks have elapsed. As depicted in Figure 5.4, to initiate a session with F_{OU}, P sends ($\mathsf{Initiate}$, sid) to F_{OU}, which creates two empty lists, namely $\mathbf{L}_{in}$ and $\mathbf{L}_{out}$. To perform an execution, P submits ($\mathsf{Evaluate}$, sid, $\mathbf{Inst}_S$, in) to F_{OU}. Promptly, F_{OU} evaluates the instruction sequence, storing the query and the execution output in $\mathbf{L}_{in}$, then it informs P with τ. At every tick, for every query in $\mathbf{L}_{in}$, if $\tau = 0$, move the query from $\mathbf{L}_{in}$ to $\mathbf{L}_{out}$, otherwise it decreases τ by one. After τ ticks have passed, P sends $\mathsf{Elicitate}$ query to F_{OU}, which returns all queries of P, containing the execution outputs. To simplify the notations, for evaluating a function $\mathsf{SeqEval}$, P simply sends ($\mathsf{Evaluate}$, sid, $\mathsf{SeqEval}$, $\hat{in} := \mathbf{c} \parallel \mathrm{RandSeed}$) to F_{OU}, instead of submitting a set of explicit instructions as ($\mathsf{Evaluate}$, sid, $\mathbf{Inst}_{\mathsf{SeqEval}} \leftarrow \mathsf{P}(\mathsf{SeqEval})$, $in \leftarrow \mathsf{P}(\hat{in})$) to F_{OU}. In light of the above discussion, we can define T to be $\tau_F(\mathsf{SeqEval}, \mathbf{c})$.

The ideal functionality F_{OU}.

Initiate: Upon receiving ($\mathsf{Initiate}$, sid) from P_j, store sid and initiate two empty lists, namely $\mathbf{L}^j_{in}$ and $\mathbf{L}^j_{out}$. Output ($\mathsf{Success}$, sid) to P_j.

Evaluate: Upon receiving ($\mathsf{Evaluate}$, sid, $\mathbf{Inst}_S$, $in \in \mathsf{STR}$) from P_j, compute $out \leftarrow \mathsf{OU}(\mathbf{Inst}_S, in)$ and $\tau = \tau_F(\mathbf{Inst}_S, in)$. Add ($\mathsf{P}_j$, $\mathbf{Inst}_S$, sid, τ, out) to $\mathbf{L}^j_{in}$ and send ($\mathsf{Success}$, sid, $\mathbf{Inst}_S$, τ) to P_j.

Tick: In every tick, and for every query in $\mathbf{L}^j_{in}$, if $\tau > 0$, then decrease τ by one and

continued on the next page

continued from the previous page

update the query. If $\tau = 0$, then move it from $\mathbf{L}_{in}^{j}$ and add it to $\mathbf{L}_{out}^{j}$.

Elicitate: Upon receiving (Elicitate, *sid*) from P_j, pop all entries in $\mathbf{L}_{out}^{j}$ with (P_j, $\cdots$) and send (Elicitate, *sid*, **resp**) to P_j, where **resp** denotes the popped entries.

Figure 5.4: The ideal functionality of the operation unit, F_{OU}.

5.3.2 Non-Committing Technique

Achieving universal composable security for delay-based protocols, as in our work, is challenging. As in the UC framework, the simulator cannot extract more secret inputs from honest parties than is permitted by the ideal functionality of the protocol. By design, this ensures that the protocol transcript does not unintentionally leak any of the parties' secrets. As a result, the simulator must simulate the real execution of a protocol without some of the honest parties' inputs, as these inputs may not be available before the delay has passed. This makes the simulation of the protocol distinguishable from the ideal scenario.

Example. Consider a time-release protocol constructed using an encryption scheme $\Gamma_{\mathcal{E}} := (\mathsf{En}, \mathsf{De})$, and two *equivalent* functions: $\mathsf{tr} : \mathcal{C} \to \mathcal{K}$ in $\mathcal{F}$ with difficulty t, and $\mathsf{SeqEval} : \mathcal{C} \to \mathcal{K}$ with difficulty T in which $t << T$. This protocol is designed to send a secret message to a receiver, which can only be retrieved after a predetermined time T. To send a message $m \in \{0,1\}^{\lambda}$, the sender computes $ct = \Gamma_{\mathcal{E}}.\mathsf{En}_{sk}(m)$, where $sk = \mathsf{tr}(x) \in \mathcal{K}$ and x is a random element in $\mathcal{C}$. The sender then sends the pair (ct, x) to the receiver. To extract m, the receiver needs to compute $sk = \mathsf{SeqEval}(x)$, and then computes $m = \Gamma_{\mathcal{E}}.\mathsf{De}_{sk}(ct)$. In this setup, the simulator cannot obtain the secret input message (m) from the honest sender in the ideal world scenario without following the protocol's intended functionality, which involves waiting for the specified time delay T. Before this time elapses, the simulator, acting as the honest sender in the real world, is required to send the pair (ct', x), in which the ciphertext (ct') is constructed without knowing the honest sender's input (m). Later, after time T, the ciphertext can be decrypted using the secret key $sk = \mathsf{SeqEval}(x)$ to obtain $m' = \Gamma_{\mathcal{E}}.\mathsf{De}_{sk}(ct')$. As a result, there is an overwhelming probability that the ciphertext in the real world scenario will not decrypt to the correct message (m). It follows that the real and ideal execution of the protocol can be distinguished simply by comparing the message obtained in the real scenario (m') with the expected message from the ideal scenario (m).

To address this issue, we redesign the protocol to run within the random oracle model. This also involves modifying the protocol to use a non-committing encryption scheme and

forcing the sender to make one or more queries to the random oracle. This approach is known in the literature and is used in different works, such as [5, 13, 100].

Example continued. Let $\mathsf{Hash} : \{0,1\}^* \to \mathcal{K}$ be a cryptographic hash function and $\Gamma_{\mathcal{E}}$ be a non-committing encryption scheme defined in Subsection 2.1.2. In the new design, we model Hash by a random oracle F_{RO}. Another key difference here is that the sender computes $ct = \Gamma_{\mathcal{E}}.\mathsf{En}_h(m)$ instead of $ct = \Gamma_{\mathcal{E}}.\mathsf{En}_{sk}(m)$, where $h := \mathsf{Hash}(sk)$, $sk = \mathsf{tr}(x)$, and $x \leftarrow_{\$} \mathcal{C}$. In the hybrid scenario, the simulator, acting as the honest sender, sends a dummy ciphertext (ct') at an early stage, before time T elapses, where ct' is sampled at random from the ciphertext space $\mathcal{CT}$. After time T, the simulator, acting as a receiver in the ideal scenario, obtains the honest sender's message (m) from the ideal functionality of the protocol. The dummy ciphertext can then be decrypted to the obtained message (m) using the appropriate decryption key, which is derived from the extractor algorithm ($\Gamma_{\mathcal{E}}.\mathsf{Ex}$) and supplied through F_{RO}. In particular, after obtaining m, the simulator observes any query to F_{RO} of the form (Query, sid, $sk = \mathsf{SeqEval}(x)$), and responds with (Query, sid, sk, $h := \Gamma_{\mathcal{E}}.\mathsf{Ex}_m(ct')$). As a result, the decryption of the dummy ciphertext (ct') will return the expected message (m), and thus the hybrid scenario of the protocol will be indistinguishable from the ideal scenario.

5.4 ChronoCloak

In this section, we first present a formal definition of ChronoCloak and its desired properties. This definition serves as a basis for arguing the security of a ChronoCloak construction within the game-based framework. Toward UC security, we introduce an extended definition that also requires simulatability (Subsection 5.4.2) and then define the ideal functionality of ChronoCloak (Subsection 5.4.3). As we will show in Section 5.5, the extended definition will enable us to uplift a ChronoCloak construction from a game-based model to one that realizes ChronoCloak's ideal functionality, achieving UC security.

5.4.1 Definition of ChronoCloak

Definition 7 (ChronoCloak)**.** *ChronoCloak is a tuple of algorithms (*Setup*,* Commit*,* $\mathsf{Challenge}$*,* Solve*,* Verify*,* Open*) defined as follows, where* $\mathcal{M}$*,* $\mathcal{S}$*,* $\mathcal{C}_R$*,* $\mathcal{C}_S$*,* $\mathcal{Y}$ *and* Π *refer to the message, secret, commitment, challenge, answer and proof spaces, respectively.*

— Setup*: a randomized algorithm that takes a security parameter* λ *and a difficulty* T*. It runs in time* $\mathsf{Poly}(\lambda)$ *and outputs a public parameter* pp*.*

— *Commit:* *a randomized algorithm that takes pp and $i \in \{0,1\}$; it selects a random secret $sk_r \leftarrow_\$ \mathcal{S}$, and generates and outputs, in time $\mathsf{Poly}(\lambda)$, a commitment $\boldsymbol{c}' \in \mathcal{C}_R$ of i, and sk_r.*

— *Challenge:* *a randomized algorithm with input pp, $(m_0, m_1) \in \mathcal{M}^2$, and $\boldsymbol{c}' \in \mathcal{C}_R$. It selects $sk_c \leftarrow_\$ \mathcal{S}$, and computes and returns, in time $\mathsf{Poly}(\lambda)$, a challenge $\boldsymbol{c} \in \mathcal{C}_S$ of $\boldsymbol{c}'$ associated with an answer $\boldsymbol{a}$.*

— *Solve:* *a deterministic algorithm that runs in time T with $\mathsf{Poly}(\lambda)$ parallel processors. It takes as an input pp and a challenge $\boldsymbol{c} \in \mathcal{C}_S$; it returns an answer $\boldsymbol{a} \in \mathcal{Y}$ and a proof $\pi \in \Pi$.*

— *Verify:* *a deterministic algorithm that receives pp and $\boldsymbol{c} \in \mathcal{C}_S$ with its proposed answer $\boldsymbol{a} \in \mathcal{Y}$ and proof $\pi \in \Pi$. In time polynomial in $\log T$ and λ, it returns ACCEPT if $\boldsymbol{a}$ is the correct unique answer to $\boldsymbol{c}$ with respect to π, otherwise it returns REJECT.*

— *Open:* *a deterministic algorithm that runs in time $\mathsf{Poly}(\log T, \lambda)$. It takes as an input pp, sk_r, and a challenge $\boldsymbol{c} \in \mathcal{C}_S$ with its answer $\boldsymbol{a} \in \mathcal{Y}$ and proof $\pi \in \Pi$; with the knowledge of $sk_r \in \mathcal{S}$, it returns $m_i \in \{m_0, m_1\}$ if $\boldsymbol{a}$ is the unique answer to $\boldsymbol{c}$ regarding π, otherwise it returns $\perp$.*

ChronoCloak properties. Let $\mathcal{A}$ be adversary represented by a set of polynomially bounded algorithms (or/and strategies, which are both referred to by A), e.g., $\mathcal{A} := (\mathsf{A}_1, \mathsf{A}_2)$. Let A_1 be an offline algorithm, running on time $Poly(T, \lambda)$ that outputs pc a pre-computation of *pp*. Furthermore, it is the internal coin tosses of ChronoCloak's algorithms that yield all the probabilities. The outputs generated by an algorithm and enclosed within brackets [-] are secret, e.g., sk_r is the secret output of $(\mathbf{c}', [sk_r]) \leftarrow_\$ \mathsf{Commit}(pp, i)$. The following assumes that all statements are true for any λ, T, and $pp \leftarrow_\$ \mathsf{Setup}(1^\lambda, T)$. A secure ChronoCloak consists of a set of properties.

— *Correctness.* For every $\{m_0, m_1\} \leftarrow_\$ \mathcal{M}^2$, and $i \in \{0,1\}$ and for every honest challenge $\mathbf{c} \leftarrow_\$ \mathsf{Challenge}(pp, \mathbf{c}')$ of an honest $(\mathbf{c}', [sk_r]) \leftarrow_\$ \mathsf{Commit}(pp, i)$, $\mathsf{Open}(pp, sk_r, \mathbf{c}, \mathbf{a}, \pi)$ always returns the intended m_i from an honest answer $(\mathbf{a}, \pi) \leftarrow \mathsf{Solve}(pp, \mathbf{c})$ that Verify accepts with probability one.

— *Uniqueness.* ChronoCloak is unique only if, for every challenge $\mathbf{c} \in \mathcal{C}_\mathrm{S}$ of $\mathbf{c}'$, there exists exactly one valid answer (i.e., $(\mathbf{a}, \pi) \leftarrow \mathsf{Solve}(pp, \mathbf{c})$), that Verify accepts, and this is the only answer through which Open returns $m \in \{m_0, m_1\}$, satisfying:

$$\mathsf{Pr}\left[\begin{array}{c} \mathbf{a}' \neq \mathbf{a} \\ \text{and} \\ \left(\begin{array}{c} \mathsf{Verify}(pp, \mathbf{c}, \mathbf{a}', \pi') = \text{ACCEPT} \\ \text{or} \\ \mathsf{Open}(pp, sk'_r, \mathbf{c}, \mathbf{a}', \pi') \neq \perp \end{array}\right) \end{array} : \begin{array}{l} pp \leftarrow_{\$} \mathsf{Setup}(1^\lambda, T),\ \mathsf{pc} \leftarrow \mathsf{A}_1(pp), \\ (\mathbf{c}', [sk_r]) \leftarrow \mathsf{A}_2(\mathsf{Commit}, pp, [i]), \\ \mathbf{c} \leftarrow_{\$} \mathsf{Challenge}(pp, \mathbf{c}', [m_0, m_1] \in \mathcal{M}^2), \\ (\mathbf{a}, -) \leftarrow \mathsf{Solve}(pp, \mathbf{c}), \\ (\mathbf{a}', \pi', sk'_r) \leftarrow \mathsf{A}_3(pp, \mathsf{pc}, [sk_r], \mathbf{c}). \end{array}\right]$$

is a negligible function of λ, for any adversary $\mathcal{A} := (\mathsf{A}_1, \mathsf{A}_2, \mathsf{A}_3)$. Although the answer $\mathbf{a}$ must be unique, the proof π does not.

— *Sequentiality.* ChronoCloak is sequential only if no adversary $\mathcal{A}$ can obtain the correct answer before an honest solver. Let $\mathsf{A}_{\text{short}}$ be an online algorithm that proposes $\mathbf{a}'$ in a shorter time than executing $\mathsf{Solve}(pp, \mathbf{c})$ (i.e., in less time than T). For any adversary $\mathcal{A} := (\mathsf{A}_1, \mathsf{A}_2, \mathsf{A}_{\text{short}})$, we have

$$\mathsf{Pr}\left[\mathbf{a}' = \mathbf{a} \ : \begin{array}{l} pp \leftarrow_{\$} \mathsf{Setup}(1^\lambda, T), \mathsf{pc} \leftarrow \mathsf{A}_1(pp), \\ (\mathbf{c}', [sk_r]) \leftarrow \mathsf{A}_2(\mathsf{Commit}, pp, [i]), \\ \mathbf{c} \leftarrow_{\$} \mathsf{Challenge}(pp, \mathbf{c}', [m_0, m_1] \in \mathcal{M}^2), \\ \mathbf{a}' \leftarrow \mathsf{A}_{\text{short}}(pp, \mathsf{pc}, \mathbf{c}), \\ (\mathbf{a}, \pi) \leftarrow \mathsf{Solve}(pp, \mathbf{c}). \end{array}\right]$$

is a negligible function of λ. This must also hold for both $\mathcal{A}_2 \in \mathcal{A}$ and $\mathcal{A}_2 \notin \mathcal{A}$.

— *Hiding.* ChronoCloak is hiding only if, for every challenge $\mathbf{c} \in \mathcal{C}_{\text{S}}$ of $\mathbf{c}'$, and answer $\mathbf{a} \leftarrow \mathsf{Solve}(pp, \mathbf{c})$, it holds that:

$$\left|\mathsf{Pr}\left[b' = b \ : \begin{array}{l} pp \leftarrow_{\$} \mathsf{Setup}(1^\lambda, T),\ (\mathsf{pc}, [m_0, m_1]) \leftarrow \mathsf{A}_1(pp), \\ (\mathbf{c}', [sk_r]) \leftarrow \mathsf{Commit}(pp, [i]),\ [b] \leftarrow_{\$} \{0, 1\}, \\ \mathbf{c} \leftarrow_{\$} \mathsf{Challenge}(pp, \mathbf{c}', \left[\begin{array}{ll} m_0, m_1 & \text{for } b = 0 \\ [m'_0, m'_1] \leftarrow_{\$} \mathcal{M}^2 & \text{for } b = 1 \end{array}\right]), \\ (\mathbf{a}, \pi) \leftarrow \mathsf{Solve}(pp, \mathbf{c}),\ b' \leftarrow \mathsf{A}_4(pp, \mathsf{pc}, \mathbf{c}', \mathbf{c}, \mathbf{a}). \end{array}\right] - \frac{1}{2}\right|$$

is a negligible function of λ, for any adversary $\mathcal{A} := (\mathsf{A}_1, \mathsf{A}_4)$.

— *Secrecy.* ChronoCloak is secretive only if there is no adversary, corrupting the receiver, inferring any extra information than inferred by an honest receiver. Hence, for any adversary $\mathcal{A} := (\mathsf{A}_1, \mathsf{A}_2, \mathsf{A}_3, \mathsf{A}_5, \mathsf{A}_6)$, we have

$$\mathsf{Pr}\left[\begin{array}{l|l} \hat{m}_{1-i} = m_{1-i}, \\ \text{and } \hat{m}_i = m_i \end{array} : \begin{array}{l} pp \leftarrow_{\$} \mathsf{Setup}(1^\lambda, T),\ \mathsf{pc} \leftarrow \mathsf{A}_1(pp), \\ (\mathbf{c}', [sk_r]) \leftarrow \mathsf{A}_2(\mathsf{Commit}, pp, [i]), \\ \mathbf{c} \leftarrow_{\$} \mathsf{Challenge}(pp, \mathbf{c}', [m_0, m_1] \in \mathcal{M}^2), \\ (\mathbf{a}', -, sk'_r) \leftarrow \mathsf{A}_3(pp, \mathsf{pc}, [sk_r], \mathbf{c}),\ \hat{m}_i \leftarrow \mathsf{A}_5(pp, \mathsf{pc}, [sk'_r], \mathbf{c}, \mathbf{a}') \\ \hat{m}_{1-i} \leftarrow \mathsf{A}_6(pp, \mathsf{pc}, [sk'_r], \mathbf{c}, \mathbf{a}', \hat{m}_i). \end{array}\right]$$

is a negligible function of λ.

— *Privacy.* ChronoCloak is private only if the $\mathbf{c}'_i$ of i is indistinguishable from $\mathbf{c}'_{i'}$ of i'. For any adversary $\mathcal{A}$, we have

$$\left|\mathsf{Pr}\left[\mathcal{A}(pp, \mathbf{c}') = i : \begin{array}{l} pp \leftarrow_{\$} \mathsf{Setup}(1^\lambda, T), \\ [i] \leftarrow_{\$} \{0,1\}, \\ \mathbf{c}' \leftarrow_{\$} \mathsf{Commit}(pp, [i]) \end{array}\right] - 1/2\right|$$

is a negligible function of λ.

5.4.2 Extended Definition of ChronoCloak

We now present an extended definition of ChronoCloak.

Definition 8 (Simulatable ChronoCloak)**.** *A Simulatable ChronoCloak is a secure ChronoCloak (Setup, Commit, Challenge, Solve, Verify, Open) as defined in Definition 7, where the Challenge algorithm is identical to the original definition except that it returns the secret answer $\boldsymbol{a}$.*

Challenge: a randomized algorithm with input pp, $(m_0, m_1) \in \mathcal{M}^2$, and $\boldsymbol{c}' \in \mathcal{C}_R$. It selects $sk_c \leftarrow_{\$} \mathcal{S}$, and computes and returns, in time Poly(λ), a challenge $\boldsymbol{c} \in \mathcal{C}_S$ of $\boldsymbol{c}'$ and its secret answer $[\boldsymbol{a}]$.

A Simulatable ChronoCloak satisfies correctness, hiding, sequentiality, uniqueness, secrecy, privacy, and the following additional properties:

Simulatability of Commit. *There exists a PPT $\mathcal{E}_{\mathsf{Commit}}$ such that, for any commitment $\boldsymbol{c}'$ generated under sk_r, it outputs a bit $i' \in \{0,1\}$ satisfying $\hat{m} = m_{i'}$, where $(\boldsymbol{c}, [\boldsymbol{a}]) \leftarrow_{\$}$ Challenge$(pp, \boldsymbol{c}', [m_0, m_1] \in \mathcal{M}^2)$, and $\hat{m} \leftarrow$ Open$(pp, [sk_r], \boldsymbol{c}, \mathsf{Solve}(pp, \boldsymbol{c}))$.*

Simulatability of Challenge. *There exists a PPT $\mathcal{E}_{\mathsf{Challenge}}$ such that, for any valid challenge $\boldsymbol{c} \in \mathcal{C}_S$ generated with $(m_0, m_1) \in \mathcal{M}^2$ and $\boldsymbol{c}' \in \mathcal{C}_R$, and under sk_c, it outputs a pair $(m'_0, m'_1) \in \mathcal{M}^2$ satisfying $(m'_0, m'_1) = (m_0, m_1)$.*

5.4.3 The Ideal Functionality of ChronoCloak

Now, we present the ideal functionality $\mathsf{F_{CC}}$ that captures ChronoCloak features, shown in Figure 5.5, which involves a series of interactions among participants. By submitting an Enrollment message to $\mathsf{F_{CC}}$, participants are added to the Participant List ($\mathcal{P}$). A receiver P_R initiates interactions by submitting Commit to $\mathsf{F_{CC}}$ with a choice i. The functionality only forwards the receiver's request along with its identity to the sender, illustrating the concealment of the receiver's choice. Upon receiving notice of P_R's query from $\mathsf{F_{CC}}$, the sender P_S responds by sending (m_0, m_1) via Challenge to $\mathsf{F_{CC}}$. $\mathsf{F_{CC}}$ broadcasts a challenge (χ_0) to all parties in $\mathcal{P}$. The functionality only shows the initial chronicle progress to the receiver, which is independent of any information regarding the messages, illustrating the prevention of any leak of secret messages (except their length). $\mathsf{P}_j \in \mathcal{P}$ prompts sequential Advance queries to $\mathsf{F_{CC}}$, obtaining $\{\chi_k\}_{k\in[T]}$ after all T-ticks, where to obtain χ_k at a tick, P_j must have χ_{k-1} from the previous tick, illustrating the sequentiality of progress. Note that, following each tick, all parties receive their respective responses from $\mathsf{F_{CC}}$ by calling Elicitate. Once T ticks have elapsed, P_j can obtain a proof π from $\mathsf{F_{CC}}$ by sending Prove with (χ_0, χ_T). If P_j's input is not the expected, it receives a REJECT from $\mathsf{F_{CC}}$. This ensures that only a unique answer to the challenge will yield a valid proof, which does not need to be unique itself. Upon obtaining/receiving (χ_T, π') from $\mathsf{F_{CC}}$, P_R retrieves m_i by invoking Open on $\mathsf{F_{CC}}$, completing the ChronoCloak process.

The ideal functionality $\mathsf{F_{CC}}$.

Enrollment: Upon receiving (Enroll, sid) from a party P_j, add P_j to $\mathcal{P}$; then send back (Enrolled, sid).

Commit: Upon receiving (Commit, sid, P_S, i) from P_R, add the tuple (P_R, P_S, sid, $m_0 = \varnothing$, $m_1 = \varnothing$, i, $T = \varnothing$, $\chi_0 = \varnothing$, $\chi_T = \varnothing$) to **InState**, where χ_j is the chronicle progress; and no further messages of this type is accepted. Output (Commit, sid, P_R) to P_S.

Challenge: Upon receiving (Challenge, sid, P_R, T, m_0, m_1) from P_S, if Commit is received, sample $(\mathbf{c}_j)_{j\in[T+1]} \leftarrow_{\$} \mathcal{C}_{\mathrm{S}}^{T}$ s.t. $\nexists \mathbf{c}_j = \mathbf{c}_k \ \forall j \neq k$. Assign $\mathbf{ChronoSteps}[\mathbf{c}_j] = \mathbf{c}_{j+1} \ \forall j \in [T+1]$ and updates (P_S, P_R, sid, m_0, m_1, T, $\chi_0 = \mathbf{c}_1$, $\chi_T = \mathbf{c}_{T+1}$, $\pi_{\mathbf{list}} = [\varnothing]$) in **InState**. Lastly, broadcast (Challenge, sid, χ_0) to $\mathcal{P}$ and no further messages of this type is accepted.

continued on the next page

continued from the previous page

Advance: Upon receiving (Step, sid, χ) from P_j, perform the following: (i) If $\mathbf{ChronoSteps}[\chi]$ is defined, add (P_j, Step, sid, χ, $\chi' := \mathbf{ChronoSteps}[\chi]$) to $\mathbf{L}_{in}$ and skip the next step. (ii) If $\mathbf{ChronoSteps}[\chi]$ is not defined, sample $\chi' \leftarrow_{\$} \mathcal{C}_{\mathrm{S}}$, set $\mathbf{ChronoSteps}[\chi] = \chi'$ and add (P_j, Step, sid, χ, χ') to $\mathbf{L}_{in}$.

Prove: Upon receiving (Prove, sid, χ, χ') from P_j, add (P_j, Prove, sid, REJECT) to $\mathbf{L}_{in}$ if $(\chi, \chi') \neq (\chi_0, \chi_T)$, and skip next. A non-corrupt P_S: If there is no $(\mathsf{P}_j, \pi) \in \pi_{\mathbf{list}}$, append $(\mathsf{P}_j, \pi \leftarrow_{\$} \Pi)$ to $\pi_{\mathbf{list}}$, then add (P_j, Prove, sid, π) to $\mathbf{L}_{in}$. A corrupt P_S: Forward the query to $\tilde{\mathcal{A}}$ and wait to receive π'. If $\pi' \notin \Pi$, halt; otherwise add (P_j, Prove, sid, π') to $\mathbf{L}_{in}$.

Verify: Upon receiving (Verify, sid, χ, χ', π') from P_j, add (P_j, Verify, sid, REJECT) to $\mathbf{L}_{in}$ if $(\chi, \chi') \neq (\chi_0, \chi_T)$ or $(-, \pi') \notin \pi_{\mathbf{list}}$; otherwise add ($\mathsf{P}_j$, Verify, sid, ACCEPT) to $\mathbf{L}_{in}$.

Open: Upon receiving (Open, sid, χ, χ', π') from P_R, send P_R the response (Open, sid, REJECT) if $(\chi, \chi') \neq (\chi_0, \chi_T)$ or $(-, \pi') \notin \pi_{\mathbf{list}}$; if not, add ($\mathsf{P}_R$, Open, sid, m_i) to $\mathbf{L}_{in}$.

Tick: At every tick, set $\mathbf{L}_{out} \leftarrow \mathbf{L}_{in}$ and $\mathbf{L}_{in} \leftarrow \varnothing$.

Elicitate: Upon receiving (Elicitate, sid) from P_j, pop all entries in $\mathbf{L}_{out}$ with (P_j, $\cdots$) and send (Elicitate, sid, **resp**) to P_j, where **resp** denotes the popped entries. If (P_j, Prove, sid, π') $\in$ **resp**, send (P_j, (χ_T, π')) to P_R.

Figure 5.5: The ideal functionality of ChronoCloak, F_{CC}.

5.5 Generic ChronoCloak

In this section, we present a generic protocol built on the game-based definition of ChronoCloak, as outlined in Definition 8. The protocol uses a non-committing technique to implement the ideal functionality F_{CC} in a hybrid model with functionalities F_{RO}, F_{OU}, and $\mathsf{F}_{\mathsf{clk}}$. We then prove that the protocol securely realizes F_{CC}, showing that Definition 8 of ChronoCloak, together with the non-committing technique, is simulatable, thus realizing F_{CC} in the random oracle model. Let $\Gamma_{\mathcal{E}} =$(KGen, En, De) be a non-committing encryption

scheme as defined in Definition 2 with a message space $\mathcal{M}$, a key space $\mathcal{K}$, and a ciphertext space $\mathcal{CT} := \mathcal{M}$. Let $\mathsf{Hash}_1 : \{0,1\}^* \to \mathcal{K}$ and $\mathsf{Hash}_2 : \{0,1\}^* \to \{0,1\}^{2\lambda}$ be hash functions modeled by the random oracle F_{RO}, where Hash_1 is defined as $\mathsf{F}_{\mathsf{RO}}(\mathsf{NECS} \,\|\, \cdot)$ and Hash_2 as $\mathsf{F}_{\mathsf{RO}}(\mathsf{CM} \,\|\, \cdot)$. Both hash functions take arbitrary inputs and serialize them into strings of bits.

5.5.1 The Protocol

In Figure 5.6, we present a generic protocol $\Gamma_{\mathsf{F_{CC}}}$ based on Definition 8, which is defined by the parameters ($\mathcal{M}$, $\mathcal{S}$, $\mathcal{C}_{\mathrm{R}}$, $\mathcal{C}_{\mathrm{S}}$, $\mathcal{Y}$, Π).

Initialization [Setup]. A trusted party $\mathsf{P}_{\mathrm{TTP}}$ generates a set of public parameters $pp \leftarrow_{\$} \mathsf{Setup}(\lambda, T)$. $\mathsf{P}_{\mathrm{TTP}}$ sends (Setup, $\mathsf{P}_{\mathrm{TTP}}$, pp) to all parties.

Preparation [Enrollment]. Initially, P_S and P_R share a common input pp and $sid \in \{0,1\}^*$. Using a private input $i \in \{0,1\}$, the receiver (P_R) decides which message to obtain. In response to a receiver's request, the sender (P_S) prepares the relevant message set $(m_0, m_1) \in \mathcal{M}^2$.

Message 1 [Commit]. First, P_R computes $(\mathbf{c}', [sk_r]) \leftarrow_{\$} \mathsf{Commit}(pp, [i])$. Then, P_R sends ($\mathsf{Commit}, sid, \mathsf{P}_R, \mathbf{c}'$) to P_S, and stores sk_r internally.

Message 2 [Challenge]. Upon receiving (Commit, sid, P_R, $\mathbf{c}'$) from P_R, P_S performs the following

- computes $\hat{m}_0 = \mathsf{En}_{sm}(m_0)$, $\hat{m}_1 = \mathsf{En}_{sm}(m_1)$, where $sm \leftarrow_{\$} \mathsf{KGen}(1^\lambda)$.
- computes $(\mathbf{c}, [\mathbf{a}]) \leftarrow_{\$} \mathsf{Challenge}(pp, \mathbf{c}', [\hat{m}_0, \hat{m}_1])$.
- sends (Query, sid, $\mathsf{NECS} \,\|\, \mathbf{a}$) to F_{RO}, and waits to receive (Query, sid, k).
- computes $\mathsf{ct} := \mathsf{En}_k(sm)$, and sends ($\mathsf{Query}$, sid, $\mathsf{CM} \,\|\, \mathsf{ct} \,\|\, sm \,\|\, k$) to F_{RO}, and waits to receive (Query, sid, cm_{ct}).

P_S sends ($\mathsf{Challenge}$, sid, P_S, $\hat{\mathbf{c}} := (\mathbf{c}, \mathsf{ct}, cm_{\mathsf{ct}})$) to P_R.

Message 3 [Solve]. Upon receiving ($\mathsf{Challenge}$, sid, P_S, $\hat{\mathbf{c}}$) from P_S, P_R sends ($\mathsf{Evaluate}$, sid, Solve, $\hat{in} := (pp, \hat{\mathbf{c}}.\mathbf{c})$ to F_{OU} and awaits to receive ($\mathsf{Evaluate}$, sid, $\mathbf{Inst}_{\mathsf{Solve}}$, T). To

continued on the next page

continued from the previous page

obtain $(\mathbf{a}, \pi \in \Pi)$, P_R sends T consecutive (Elicitate, sid) queries to the F_{OU}, stopping upon receiving the T^{th} response (Elicitate, sid, **resp**) with a non-empty **resp** $\neq \varnothing$.

Output [Open]. Upon obtaining $(\mathbf{a}, \pi)$, P_R performs the following

- parses $\hat{\mathbf{c}}$ into $(\mathbf{c}, \mathsf{ct}, cm_{\mathsf{ct}})$.
- checks that $\mathsf{Verify}(pp, \mathbf{c}, (\mathbf{a}, \pi)) = \text{ACCEPT}$; if no, it outputs $\perp$.
- sends (Query, sid, NECS $\|$ **a**) to F_{RO}, and waits to receive (Query, sid, k).
- computes $sm := \mathsf{De}_k(\mathsf{ct})$; then it sends (Query, sid, CM $\|$ ct $\|$ sm $\|$ k) to F_{RO}, and waits to receive (Query, sid, cm'_{ct}).
- checks the following conditions: $cm'_{\mathsf{ct}} \stackrel{?}{=} cm_{\mathsf{ct}}$. If this condition fails, the procedure outputs $\perp$. Otherwise, it proceeds to the next step.
- computes $\hat{m}_i \leftarrow \mathsf{Open}(pp, [sk_r], \mathbf{c}, (\mathbf{a}, \pi))$. If $\hat{m}_i = \perp$, it outputs $\perp$; otherwise, it proceeds to the next step.

Finally, P_R computes $m_i = \mathsf{De}_{sm}(\hat{m}_i)$.

Figure 5.6: Proposed ChronoCloak protocol $\Gamma_{\mathsf{F}_{\mathsf{CC}}}$ implementing F_{CC}.

5.5.2 Security

We will now provide a proof that $\Gamma_{\mathsf{F}_{\mathsf{CC}}}$ securely implements F_{CC}.

Theorem 6. *Let $\Gamma_{\mathcal{E}}$ =(KGen, En, De) be a non-committing encryption scheme as defined in Definition 2. Let (Setup, Commit, Challenge, Solve, Verify, Open) be a simulatable ChronoCloak outlined in Definition 8. The protocol $\Gamma_{\mathsf{F}_{\mathsf{CC}}}$ in Section 5.5.1 securely UC-implements F_{CC} in the (F_{RO}, F_{OU}, $\mathsf{F}_{\mathsf{clk}}$)-hybrid model under a malicious adversary (A) with static corruption, for any PPT environment Env.*

Proof. To prove the theorem, we construct the simulators $\tilde{\mathcal{A}}'_S$ and $\tilde{\mathcal{A}}'_R$ (see Figures 5.7 and 5.8). Env produces inputs sent to the appropriate parties (i.e., $\mathsf{P}_S \xleftarrow{send} (m_0, m_1), \mathsf{P}_R \xleftarrow{send} i, \mathsf{A} \xleftarrow{send}$ aux). The simulator provides *the malicious adversary* with the matching inputs. $\tilde{\mathcal{A}}'_S$ (resp. $\tilde{\mathcal{A}}'_R$) simulates $\Gamma_{\mathsf{F}_{\mathsf{CC}}}$ in the presence of a copy of A_S (resp. A_R), corrupting the

sender (resp. receiver). The simulations emulate the random oracle F_{RO} with empty state. Hence, it responds to any query from/to F_{RO} and keeps track of all queries and responses.

Security with Corrupted Sender and Honest Receiver. We present $\tilde{\mathcal{A}}'_S$ who passes the inputs $([m_0, m_1], \mathsf{aux})$ from Env to A_S.

The simulation $\tilde{\mathcal{A}}'_S$.

1. *Initialization and Preparation*: On Env's inputs (λ, T), $\tilde{\mathcal{A}}'_S$ simulates Setup honestly. Then, A_S is invoked with inputs (m_0, m_1, aux) from Env.

2. *Message 1 [Commit]*: As an honest receiver, $\tilde{\mathcal{A}}'_S$ computes $(\mathbf{c}', [sk_r]) \leftarrow_{\$} \mathsf{Commit}(pp, [i'])$ with a dummy input (e.g., $i' = 0$); then it sends $(\mathsf{Commit}, sid, \mathsf{A}_S, \mathbf{c}')$ to A_S and stores sk_r internally.

3. *Message 2 [Challenge]*: Upon receiving $(\mathsf{Challenge}, sid, \mathsf{A}_S, \hat{\mathbf{c}} := (\mathbf{c}, \mathsf{ct}, cm_{\mathsf{ct}}))$ from A_S, $\tilde{\mathcal{A}}'_S$ observes all queries (i.e., inString $\in \mathsf{F}_{\mathsf{RO}}.\mathbf{H}$) made to F_{RO}. Accordingly, $\tilde{\mathcal{A}}'_S$ identifies two records in $\mathsf{F}_{\mathsf{RO}}.\mathbf{H}$ of the form $(cm_{\mathsf{ct}}, \mathsf{CM} \parallel \mathsf{ct} \parallel str_1 \parallel str_2) \in \mathsf{F}_{\mathsf{RO}}.\mathbf{H}$ and $(str_2, \mathsf{NECS} \parallel str_3) \in \mathsf{F}_{\mathsf{RO}}.\mathbf{H}$, which allows it to obtain $sm \parallel k \leftarrow str_1 \parallel str_2$ and $\mathbf{a}' \leftarrow str_3$. Following that, $\tilde{\mathcal{A}}'_S$ checks if $\mathsf{ct} \stackrel{?}{=} \mathsf{En}_k(sm)$. If no such records exist or the validation fails, $\tilde{\mathcal{A}}'_S$ sets $(m_0, m_1) \leftarrow_{\$} \mathcal{M}^2$ and skips next step. Otherwise, $\tilde{\mathcal{A}}'_S$ calls $\mathcal{E}_{\mathsf{Challenge}}$ to extract $(\hat{m}_0, \hat{m}_1)$ from $\mathbf{c}$. Then, $\tilde{\mathcal{A}}'_S$ computes $m_0 = \mathsf{De}_{sm}(\hat{m}_0)$, and $m_1 = \mathsf{De}_{sm}(\hat{m}_1)$.

4. *Message 3 [Solve]*: After above, $\tilde{\mathcal{A}}'_S$ sends $(\mathsf{Challenge}, sid, \mathsf{P}_R, T, m_0, m_1)$ to F_{CC}, and waits for $(\mathsf{Challenge}, sid, \chi_0)$. Then, $\tilde{\mathcal{A}}'_S$ sends $(\mathsf{Evaluate}, sid, \mathbf{Inst}_{\mathsf{Solve}} \leftarrow \tilde{\mathcal{A}}'_R(\mathsf{Solve}), in \leftarrow \tilde{\mathcal{A}}'_R(pp, \hat{\mathbf{c}}.\mathbf{c}))$ to F_{OU}, and receives $(\mathsf{Evaluate}, sid, \mathbf{Inst}_{\mathsf{Solve}}, \tau)$. In the event that A_S did not query either or both of $(\mathsf{Query}, sid, \mathsf{NECS} \parallel \mathbf{a})$ and $(\mathsf{Query}, sid, \mathsf{CM} \parallel \mathsf{ct} \parallel sm \parallel k)$, $\tilde{\mathcal{A}}'_S$ tracks all queries to F_{RO}, and if query(ies) passes the validation in Step 3, $\tilde{\mathcal{A}}'_S$ outputs ABORT.

5. *Output [Open]*: After τ ticks, $\tilde{\mathcal{A}}'_S$ responds to all Prove queries from F_{CC} with $\perp$ if $\mathbf{a}'$, obtained from Step 3, is not equal to $\mathbf{a}$, obtained from F_{OU} (and skip next). Similarly, if the challenge $\hat{\mathbf{c}}$ is not well-formed, i.e., $cm'_{\mathsf{ct}} \neq \hat{\mathbf{c}}.cm_{\mathsf{ct}}$ for cm'_{ct} obtained using $\mathbf{a}$ and $\hat{\mathbf{c}}.\mathsf{ct}$ (and skip next). Otherwise, it responds with $\pi \in \Pi$, obtained from the solving process in Step 4.

   Lastly, once P_R, in the ideal execution, receives m_i, $\tilde{\mathcal{A}}'_S$ returns A_S's output to Env.

Figure 5.7: A simulation $\tilde{\mathcal{A}}'_S$ for a corrupted sender and an honest receiver executing Γ_{Fcc}.

Below, we show that $\tilde{\mathcal{A}}'_S$'s output is indistinguishable from the real execution and aligns with the ideal execution. $\tilde{\mathcal{A}}'_S$ operates honestly within the simulation by generating pp and acting as a trusted receiver, rendering it indistinguishable from a real execution. Furthermore, in Step 2, $\tilde{\mathcal{A}}'_S$ does not know P_R's input (in the ideal case), so it uses a fixed choice, i.e., $i' = 0$. Despite this, Env cannot distinguish whether i' matches P_R's input or not. This is due to ChronoCloak's 'privacy' assumption. As such, the two executions remain identical.

Additionally, to participate in the ideal execution, $\tilde{\mathcal{A}}'_S$ must send m_0 and m_1 to the F_{CC} at an early stage (i.e., before obtaining $\hat{y}$). However, since $\tilde{\mathcal{A}}'_S$ does not possess both secret messages, m_0 and m_1, it cannot participate in the ideal execution in a way that is consistent with the real execution. Fortunately, in Step 3, $\tilde{\mathcal{A}}'_S$ can extract and forward both secret messages m_0 and m_1 using ct, cm_{ct} and through F_{RO}. If it cannot extract both messages, $\tilde{\mathcal{A}}'_S$ will select a random message(s) of the same length (i.e., representing the case in which the adversary did not inquire F_{RO}). This indicates that A_S sends a random string(s) (i.e., $\hat{m}_0$, and $\hat{m}_1$), which implies sending a message(s) that are unknown to A_S (since A_S did not acquire the corresponding key k). Later on, if A_S makes additional query(ies) for the missing key, $\tilde{\mathcal{A}}'_S$ outputs ABORT or returns a suitable key that corresponds to the message sent to F_{CC} (in Step 4). However, the probability of A_S forming a valid challenge that passes in Step 5 is negligible since it has to predict the output of the query to F_{RO} in such a way that $k = \mathsf{Ex}_{sk_c}(\mathsf{ct})$ (same for cm_{ct}). In this case, the simulator sends a null proof message. Accordingly, the $\tilde{\mathcal{A}}'_S$ can participate in ideal execution with interaction consistent with input from Env and the adversary's strategy. As a result, both executions are indistinguishable.

Security with Honest Sender and Corrupted Receiver. We present $\tilde{\mathcal{A}}'_R$ who passes the inputs $([i], \mathsf{aux})$ from Env to A_R.

The simulation $\tilde{\mathcal{A}}'_R$.

1. *Initialization and Preparation*: The simulator obtains (λ, T) from the environment as a common public input. Then, A_R is invoked with inputs (i, aux) from Env.
2. *Message 1 [Commit]*: Upon receiving $(\mathsf{Commit}, sid, \mathsf{A}_R, \mathbf{c}')$ from A_R, $\tilde{\mathcal{A}}'_R$ calls $\mathcal{E}_{\mathsf{Commit}}$ to extract A_R's choice i' from $\mathbf{c}'$ (which may differ from i).
3. *Message 2 [Challenge]*: After above, $\tilde{\mathcal{A}}'_S$ samples two bogus messages $(\hat{m}_0, \hat{m}_1) \leftarrow_{\$} \mathcal{M}^2$. Then, it computes $(\mathbf{c}, [\mathbf{a}]) \leftarrow_{\$} \mathsf{Challenge}(pp, \mathbf{c}', [\hat{m}_0, \hat{m}_1])$, and samples $\mathsf{ct} \leftarrow_{\$}$

continued on the next page

continued from the previous page

$\mathcal{CT}$, and $cm_{\mathsf{ct}} \leftarrow_{\$} \mathsf{F}_{\mathsf{RO}}.\mathcal{H}_2 = \{0,1\}^{2\lambda}$. Finally, it sends ($\mathsf{Challenge}$, sid, A_R, $\hat{\mathbf{c}} := (\mathbf{c}$, ct, $cm_{\mathsf{ct}})$) to A_R.

Simultaneously, $\tilde{\mathcal{A}}'_R$ sends (Commit, sid, P_S, i') to F_{CC}. After that, it awaits F_{CC}'s ($\mathsf{Challenge}$, sid, χ_0) messages. Lastly. $\tilde{\mathcal{A}}'_R$ sends ($\mathsf{Evaluate}$, sid, $\mathbf{Inst}_{\mathsf{SeqEval}} \leftarrow \tilde{\mathcal{A}}'_R(\mathsf{Solve.SeqEval})$, $in \leftarrow \tilde{\mathcal{A}}'_R(\hat{\mathbf{c}}.\mathbf{c})$) to F_{OU}, and receives ($\mathsf{Evaluate}$, sid, $\mathbf{Inst}_{\mathsf{SeqEval}}$, τ).

4. *Message 3 [Solve]*: At every tick, $\tilde{\mathcal{A}}'_R$ obtains the current sequential step χ' by sending ($\mathsf{Elicitate}$, sid) to F_{CC}. Then, $\tilde{\mathcal{A}}'_R$ updates $\tau \leftarrow \tau - 1$ and sends (Step, sid, χ') to F_{CC}. Also, $\tilde{\mathcal{A}}'_R$ continues tracking all queries to F_{RO} from A_R. $\tilde{\mathcal{A}}'_R$ outputs ABORT, if $\tau > 0$ and $\tilde{\mathcal{A}}'_R$ observes a query sent to F_{RO} by A_R of form (Query, sid, $\mathsf{NECS} \parallel \mathbf{a}$).

5. *Output [Open]*: When $\tau = 0$, $\tilde{\mathcal{A}}'_R$ obtains $m_{i'}$ from F_{CC} by sending Prove and Open calls. Upon receiving (Query, sid, $\mathsf{NECS} \parallel \mathbf{a}$) from A_R, $\tilde{\mathcal{A}}'_R$ computes $sm = \mathsf{Ex}_{m_{i'}}(\hat{m}_{i'})$, $k = \mathsf{Ex}_{sm}(\mathsf{ct})$; then sends ($\mathsf{Query}$, sid, k) to A_R. Also, for any query of form (Query, sid, $\mathsf{CM} \parallel \mathsf{ct} \parallel sm \parallel k$), $\tilde{\mathcal{A}}'_R$ responds by (Query, sid, cm_{ct}).

Figure 5.8: A simulation $\tilde{\mathcal{A}}'_R$ for an honest sender and a corrupted receiver executing $\Gamma_{\mathsf{F_{CC}}}$.

Below, we show that the view generated by $\tilde{\mathcal{A}}'_R$ cannot be distinguished from the real execution and is consistent with the ideal execution. While in the simulation, $\tilde{\mathcal{A}}'_R$ behaves primarily as an honest sender, it deviates from honest execution in Step 3 (as it does not have P_S's inputs m_0 and m_1, in the ideal case). In more detail, $\tilde{\mathcal{A}}'_R$ sends to F_{OT} bogus challenge (i.e., $(\hat{m}_0, \hat{m}_1) \leftarrow_{\$} \mathcal{M}^2$, $\mathsf{ct} \leftarrow_{\$} \mathcal{CT}$, and $cm_{\mathsf{ct}} \leftarrow_{\$} \mathsf{F}_{\mathsf{RO}}.\mathcal{H} = \{0,1\}^{2\lambda}$) hoping to receive $m_{i'}$ from F_{CC} later on (i.e., after τ ticks). For this reason, $\tilde{\mathcal{A}}'_R$ must first extract A_R's choice, i.e., i', which it can with the help of $\mathcal{E}_{\mathsf{Commit}}$ (see Step 2). So far, $\tilde{\mathcal{A}}'_R$ has sent (through $\mathsf{Challenge}$) two random messages, which is equivalent to sending true messages encrypted with an unknown key (i.e., $\hat{m}_i = \mathsf{En}_{sm}(m_i)$ for $sm \in \mathcal{K}$, and $\forall i \in \{0,1\}$). Also, $\tilde{\mathcal{A}}'_R$ differs from the real execution in that it uniformly samples ct and cm_{ct}. However, without the knowledge of $\mathbf{a}$, all above mentioned are indistinguishable from real ones due to the indistinguishability property of the non-committing encryption scheme $\Gamma_{\mathcal{E}}$, and Hash_1 and Hash_2 are assumed to be modeled by the random oracle F_{RO}. Thus, the probability of distinguishing between executions is negligible without the knowledge of $\mathbf{a}$.

Further, $\tilde{\mathcal{A}}'_R$ in Step 4 advances step by step at each tick, enabling it to obtain $m_{i'}$ from F_{CC}. However, $\tilde{\mathcal{A}}'_R$ differs from the real execution in that it aborts when A_R sends a query of form (Query, sid, $\mathsf{NECS} \| \mathbf{a}$) (to F_{RO}) before $\tilde{\mathcal{A}}'_R$. The probability of this abort is equal to the

probability of A_R breaking the ChronoCloak's 'sequentiality' assumption (i.e., obtaining the answer $\mathbf{a}$ before time τ). After τ ticks, $\tilde{\mathcal{A}}'_R$ can only know one of the two messages (i.e., only $m_{i'}$), whereas Env knows both (i.e., $m_{i'}$ and $m_{1-i'}$). Env would, however, require to obtain the other message $\hat{m}_{1-i'}$ sent through $\mathsf{Challenge}$ (by $\tilde{\mathcal{A}}'_R$) to distinguish $m_{1-i'}$ from $\overline{m}_{1-i'} := \mathsf{De}_k(\hat{m}_{1-i'})$. Therefore, this is infeasible due to ChronoCloak's 'secrecy' assumption. In Step 5, $\tilde{\mathcal{A}}'_R$ can answer A_R's query to F_{RO} with the appropriate key k in accordance with $m_{i'}$. Yet, both executions will become distinguishable in case A_R obtains an alternative valid answer $\mathbf{a}'$ ($\mathbf{a}' \neq \mathbf{a}$) such that Verify outputs ACCEPT and $\tilde{\mathcal{A}}'_R$'s challenge $\hat{\mathbf{c}}$ is well-formed. The former contradicts ChronoCloak's 'uniqueness' assumption, while the latter is infeasible under the assumptions of Hash_1 and Hash_2. Therefore, such distinguishability is infeasible. As a result, $\tilde{\mathcal{A}}'_R$ can simulate the protocol with interactions that are consistent with inputs from the environment and the strategy of the adversary, whereby the two executions are indistinguishable.

□

5.6 ChronoCloak from Delayed Oblivious Transfer

In this section, we present a construction of ChronoCloak based on Definition 8.

Construction 1. Let $\Gamma_{\mathtt{OT}} = (\mathsf{OT}_{\text{setup}}, \mathsf{OT}_R, \mathsf{OT}_S, \mathsf{OT}_{Dec})$ represent a tuple of four algorithms that construct an oblivious transfer protocol, as defined in Definition 3. Let Γ_{tVDF} = (Setup, $\mathsf{Challenger}$, Solver, Verify) denote a tuple of algorithms constructing a trapdoor-VDF as defined in Definition 7 with $\mathcal{S}$, $\mathcal{C}_{\text{S}}$, $\mathcal{Y}$, Π, and $\mathcal{F}$ represent the secret space, challenge space, answer space, proof space, and trapdoor ensemble, respectively. Also, let $\Gamma_{\mathcal{E}} = (\mathsf{KGen}, \mathsf{En}, \mathsf{De})$ denote an encryption scheme as defined in Definition 1 with a message space $\mathcal{M}$, key space $\mathcal{K}$, and ciphertext space $\mathcal{CT} := \mathcal{M}$. Finally, let $\mathsf{H} : \{0,1\}^* \to \{0,1\}^{2\lambda}$ be a hash function modeled by a random oracle F_{RO}. The construction of ChronoCloak is derived from $\Gamma_{\mathtt{OT}}$, Γ_{tVDF}, and $\Gamma_{\mathcal{E}}$ is outlined as follows:

- $pp \leftarrow \mathsf{Setup}(1^\lambda, T)$: Outputs $pp := (pk, \mathsf{crs})$, where $pk \leftarrow_{\$} \Gamma_{\text{tVDF}}.\mathsf{Setup}(1^\lambda, T)$ and $\mathsf{crs} \leftarrow_{\$} \mathsf{OT}_{\text{setup}}(1^\lambda)$.
- $(\mathbf{c}', [sk_r]) \leftarrow \mathsf{Commit}(pp, [i])$: Outputs $(\mathbf{c}', [sk_r])$, where $(\mathbf{c}', [sk_r]) \leftarrow_{\$} \mathsf{OT}_R(pp.\mathsf{crs}, [i])$.
- Compute $(\mathbf{c}, [\mathbf{a}]) \leftarrow \mathsf{Challenge}(pp, \mathbf{c}', [m_0, m_1] \in \mathcal{M}^2)$:
  - compute $\mathbf{c}_1 \leftarrow_{\$} \Gamma_{\text{tVDF}}.\mathsf{Challenge}(pp.pk)$ with private internal state $[\mathsf{st}_{\mathbf{c}_1}]$ generated during this computation includes $(\mathbf{a} \in \mathcal{Y}, sk_c \in \mathcal{S})$ such that $\mathbf{a}, sk_c \in \mathsf{st}_{\mathbf{c}_1}$.

- compute $k^{\text{MSB}} \parallel k^{\text{LSB}} = k \leftarrow \mathsf{H}(\mathbf{c}_1 \parallel \mathbf{a})$, where k^{MSB} and $k^{\text{LSB}} \in \{0,1\}^\lambda$.
- compute $\hat{m}_0 \leftarrow \mathsf{En}_{k^{\text{MSB}}}(m_0)$, $\hat{m}_1 \leftarrow \mathsf{En}_{k^{\text{MSB}}}(m_1)$, and $\mathsf{ct} \leftarrow \mathsf{En}_{k^{\text{LSB}}}(sk_c)$.
- compute $cm_{\mathsf{ct}} \leftarrow \mathsf{H}(\mathsf{ct} \parallel sk_c \parallel k)$.
- compute $\mathbf{c}_2 \leftarrow_{\$} \mathsf{OT}_S(pp.\mathsf{crs}, \mathbf{c}', \hat{m}_0, \hat{m}_1)$.
- return $(\mathbf{c} := (\mathbf{c}_1, \mathbf{c}_2, \mathsf{ct}, cm_{\mathsf{ct}}), [\mathbf{a}])$.

- $(\mathbf{a}, \pi) \leftarrow \mathsf{Solve}(pp, \mathbf{c})$: outputs $(\mathbf{a}, \pi) \leftarrow \Gamma_{\text{tVDF}}.\mathsf{Solve}(pp.pk, \mathbf{c}.\mathbf{c}_1)$.

- $(\text{ACCEPT}/\text{REJECT}) \leftarrow \mathsf{Verify}(pp, \mathbf{c}, \mathbf{a}, \pi)$:
  - if $\Gamma_{\text{tVDF}}.\mathsf{Verify}(pp.pk, \mathbf{c}.\mathbf{c}_1, \mathbf{a}, \pi) = \text{REJECT}$, return REJECT; otherwise, continue.
  - compute $k^{\text{MSB}} \parallel k^{\text{LSB}} = k \leftarrow \mathsf{H}(\mathbf{c}_1 \parallel \mathbf{a})$, where k^{MSB} and $k^{\text{LSB}} \in \{0,1\}^\lambda$.
  - compute $sk_c \leftarrow \mathsf{De}_{k^{\text{LSB}}}(\mathbf{c}.\mathsf{ct})$ and return REJECT if $sk_c \notin \mathcal{S}$, where sk_c defines a trapdoor $\mathsf{tr}_{sk_c} \in \mathcal{F}$.
  - if $\mathsf{H}(\mathsf{ct} \parallel sk_c \parallel k) \neq \mathbf{c}.cm_{\mathsf{ct}}$ or $\mathsf{tr}_{sk_c}(pp.pk) \neq (\mathbf{c}_1, \mathbf{a})$ return REJECT; otherwise, continue.
  - return (ACCEPT).

- $m_i \leftarrow \mathsf{Open}(pp, [sk_r], \mathbf{c}, \mathbf{a}, \pi)$:
  - if $\Gamma_{\text{tVDF}}.\mathsf{Verify}(pp.pk, \mathbf{c}.\mathbf{c}_1, \mathbf{a}, \pi)$ outputs REJECT, return $\perp$; otherwise, continue.
  - compute $\hat{m}_i \leftarrow \mathsf{OT}_{Dec}(pp.\mathsf{crs}, [sk_r], \mathbf{c}.\mathbf{c}_2)$.
  - compute $m_i := \mathsf{De}_{k^{\text{MSB}}}(\hat{m}_i)$ if $\mathsf{H}(\mathbf{c}.\mathsf{ct} \parallel sk_c \parallel k) = \mathbf{c}.cm_{\mathsf{ct}}$, $sk_c \in \mathcal{S}$, and $\mathsf{tr}_{sk_c}(pp.pk) = (\mathbf{c}_1, \mathbf{a})$. Otherwise $m_i := \perp$, where $k^{\text{MSB}} \parallel k^{\text{LSB}} = k \leftarrow \mathsf{H}(\mathbf{c}.\mathbf{c}_1 \parallel \mathbf{a})$ and $sk_c := \mathsf{De}_{k^{\text{LSB}}}(\mathbf{c}.\mathsf{ct})$.
  - return (m_i)

Theorem 7. *Let $\Gamma_{OT} = (OT_{setup}, OT_R, OT_S, OT_{Dec})$ be a UC-secure oblivious transfer protocol (modeled by F_{OT}), Γ_{tVDF} a secure trapdoor-VDF as defined in Definition 7, and $\Gamma_{\mathcal{E}}$ a semantically secure encryption scheme as defined in Definition 1. Then, the construction presented in Construction 1 constitutes a secure ChronoCloak (as defined in Definition 8), assuming that H is modeled as a random oracle F_{RO}.*

Proof. To verify the security of Construction 1, we show that all ChronoClack properties hold, as outlined below.

Correctness. It follows by construction.

Sequentiality. The construction is sequential, as it follows from assuming Γ_{tVDF} is sequential, H is modeled as a random oracle, and $\Gamma_{\mathcal{E}}$ is a semantically secure encryption scheme. Let $\mathcal{A} := (\mathsf{A}_1, \mathsf{A}_2, \mathsf{A}_{\mathrm{short}})$ represent an adversary defined by a set of polynomially bounded algorithms. Construction 1 satisfies sequentiality if no adversary $\mathcal{A}$ can win the following game.

- A trusted process computes and publishes $pp \leftarrow_{\$} \mathsf{Setup}(1^{\lambda}, T)$. Then, $\mathcal{A}$ performs a pre-computation in time $\mathsf{Poly}(T, \lambda)$ and outputs $\mathsf{pc} \leftarrow \mathsf{A}_1(pp)$.
- $\mathcal{A}$ computes and outputs $\mathbf{c}'$ (i.e., $(\mathbf{c}', [sk_r]) \leftarrow_{\$} \mathsf{A}_2(\mathsf{Commit}, pp, [i])$).
- A trusted process aborts, if $\mathbf{c}' \notin \mathcal{C}_{\mathrm{R}}$. Otherwise, it computes and publishes the challenge $\mathbf{c}$, defined as $(\mathbf{c} = (\mathbf{c}_1, \mathbf{c}_2, \mathsf{ct}, cm_{\mathsf{ct}}), [\mathbf{a}])$, which involves:
  1. computing $\mathbf{c}_1 \leftarrow_{\$} \Gamma_{\mathrm{tVDF}}.\mathsf{Challenge}(pp.pk)$.
  2. computing $\mathsf{ct} \leftarrow \mathsf{En}_{k^{\mathrm{LSB}}}(sk_c)$ and $cm_{\mathsf{ct}} \leftarrow \mathsf{H}(\mathsf{ct} \,\|\, sk_c \,\|\, k)$ with $k^{\mathrm{MSB}} \,\|\, k^{\mathrm{LSB}} = k \leftarrow \mathsf{H}(\mathbf{c}_1 \,\|\, \mathbf{a})$.
  3. computing $\mathbf{c}_2 \leftarrow_{\$} \mathsf{OT}_S(pp.\mathsf{crs}, \mathbf{c}', \mathsf{En}_{k^{\mathrm{MSB}}}(m_0), \mathsf{En}_{k^{\mathrm{MSB}}}(m_1))$.
- $\mathcal{A}$ computes and outputs $\mathbf{a}' \leftarrow \mathsf{A}_{\mathrm{short}}(pp, \mathsf{pc}, \mathbf{c})$ in parallel time less than T, where $\mathbf{a}' \in \mathcal{Y}$.

To win the game, $\mathcal{A}$'s output $\mathbf{a}'$ must equal $\mathbf{a}$, i.e., $(\mathbf{a}, -) \leftarrow \mathsf{Solve}(pp, \mathbf{c})$, with non-negligible probability.

Suppose, for the sake of contradiction, that there exists an adversary $\mathcal{A} := (\mathsf{A}_1, \mathsf{A}_2, \mathsf{A}_{\mathrm{short}})$ that can compute (or find) $\mathbf{a}' = \mathbf{a}$ in less than time T with a non-negligible probability. Then, at least one of the following must hold:

- We can construct an algorithm $\mathcal{B}$ from $\mathcal{A}$ to break the sequentiality of Γ_{tVDF}. The algorithm $\mathcal{B}$ receives $pp.pk$ as input and invokes A_1 to compute pc. After obtaining $\mathbf{c}.\mathbf{c}_1$ (the output of $\Gamma_{\mathrm{tVDF}}.\mathsf{Challenge}$), $\mathcal{B}$ invokes $\mathsf{A}_{\mathrm{short}}$ to compute $\mathbf{a}' = \mathbf{a}$ in less than time T, which is faster than $\Gamma_{\mathrm{tVDF}}.\mathsf{Solve}$. Hence, $\mathcal{B}$ breaks the sequentiality of Γ_{tVDF} contradicting our assumption of Γ_{tVDF}.
- $\mathcal{A}$ can obtain sufficient information about sk_c from $\mathsf{ct} \leftarrow \mathsf{En}_{k^{\mathrm{LSB}}}(sk_c)$ without knowing k^{LSB}, which enables it to compute $\mathbf{a}$. This contradicts our assumption that $\Gamma_{\mathcal{E}}$ is a semantically secure encryption scheme.

- $\mathcal{A}$ can efficiently extract information about the preimage of cm_{ct} from H, which enables it to recover sk_c and/or k^{LSB}, and thus compute $\mathbf{a}$. This implies that H does not satisfy preimage resistance, which contradicts the assumption of H being modeled as a random oracle.

Therefore, assuming the correctness of Construction 1, the sequentiality of Γ_{tVDF}, the random oracle model for H, and the semantic security of $\Gamma_{\mathcal{E}}$, no adversary $\mathcal{A}$ wins the above game with non-negligible probability. As a result, Construction 1 satisfies the sequentiality property.

Uniqueness. This follows directly from the soundness of Γ_{tVDF} and the assumption that H is modeled as a random oracle. Assume, for the sake of contradiction, that there exists a challenge $\mathbf{c} \in \mathcal{C}_{\text{S}}$ of $\mathbf{c}' \in \mathcal{C}_{\text{R}}$ and two distinct answers $\mathbf{a} \neq \mathbf{a}'$ with corresponding proofs π and π' such that either $\mathsf{Verify}(pp, \mathbf{c}, \mathbf{a}', \pi') = \text{ACCEPT}$ or $\mathsf{Open}(pp, sk_r, \mathbf{c}, \mathbf{a}', \pi') \neq \perp$. First, consider the case where $\mathsf{Verify}(pp, \mathbf{c}, \mathbf{a}', \pi') = \text{ACCEPT}$. The verification algorithm involves a number of steps, where the first step is to check whether $\Gamma_{\text{tVDF}}.\mathsf{Verify}(pp.pk, \mathbf{c}.\mathbf{c}_1, \mathbf{a}', \pi') = \text{ACCEPT}$. If $\mathbf{a}' \neq \mathbf{a}$ also satisfies $\Gamma_{\text{tVDF}}.\mathsf{Verify}(pp.pk, \mathbf{c}.\mathbf{c}_1, \mathbf{a}', \pi') = \text{ACCEPT}$, then this contradicts the soundness property of Γ_{tVDF}. As a result, the probability of such an event is negligible in λ. However, in the event that $\mathbf{a}'$ passes $\Gamma_{\text{tVDF}}.\mathsf{Verify}$, the verifier is required to recompute $k^{\text{MSB}} \parallel k^{\text{LSB}} = k = \mathsf{H}(\mathbf{c}_1 \parallel \mathbf{a}')$. Since $\mathbf{a}' \neq \mathbf{a}$ and H is modeled as a random oracle, the probability that $k = k'$ (i.e., $\mathsf{H}(\mathbf{c}_1 \parallel \mathbf{a}') = \mathsf{H}(\mathbf{c}_1 \parallel \mathbf{a})$) is negligible. Accordingly, the decryption of ct will return an incorrect sk_c', and the verification of cm_{ct} (checking if $\mathsf{H}(\mathsf{ct} \parallel sk_c' \parallel k) = \mathbf{c}.cm_{\mathsf{ct}}$) will fail. As a result, $\mathsf{Verify}(pp, \mathbf{c}, \mathbf{a}', \pi')$ will output $\perp$, which contradicts our assumption that it outputs ACCEPT.

Next, consider the case where $\mathsf{Verify}(pp, \mathbf{c}, \mathbf{a}', \pi') \neq \text{ACCEPT}$ but $\mathsf{Open}(pp, sk_r, \mathbf{c}, \mathbf{a}', \pi') \neq \perp$. In Open's description, we verify for $(\mathbf{a}', \pi')$ using $\Gamma_{\text{tVDF}}.\mathsf{Verify}(pp.pk, \mathbf{c}.\mathbf{c}_1, \mathbf{a}', \pi')$ and then check for cm_{ct}. For the former check, if $\Gamma_{\text{tVDF}}.\mathsf{Verify}$ returns REJECT, then Open outputs $\perp$ by definition. Therefore, $\mathsf{Open}(pp, sk_r, \mathbf{c}, \mathbf{a}', \pi') \neq \perp$ implies that $\Gamma_{\text{tVDF}}.\mathsf{Verify}(pp.pk, \mathbf{c}.\mathbf{c}_1, \mathbf{a}', \pi') = \text{ACCEPT}$, which brings us back to the first case and the same contradiction. For the latter check, an adversary must find a collision in the hash function H to construct a valid alternative answer $\mathbf{a}'$ that yields the same cm_{ct}. As a result, by definition, $\mathsf{Open}(pp, sk_r, \mathbf{c}, \mathbf{a}', \pi')$ will also output $\perp$, which contradicts our assumption of such case (i.e., $\mathsf{Verify}(pp, \mathbf{c}, \mathbf{a}', \pi') \neq \text{ACCEPT}$ but $\mathsf{Open}(pp, sk_r, \mathbf{c}, \mathbf{a}', \pi') \neq \perp$). Thus, the probability that there exists a distinct $\mathbf{a}' \neq \mathbf{a}$ satisfying either $\mathsf{Verify}(pp, \mathbf{c}, \mathbf{a}', \pi') = \text{ACCEPT}$ or $\mathsf{Open}(pp, sk_r, \mathbf{c}, \mathbf{a}', \pi') \neq \perp$ is negligible, contradicting our initial assumption. As a result, Construction 1 satisfies uniqueness, which follows from the assumption that Γ_{tVDF} is sound and the hash function H is modeled by a random oracle.

Hiding. Construction 1 satisfies the hiding property under the assumption that the OT protocol Γ_{OT} is UC-secure. Let $\mathcal{A}$ be a PPT adversary. Our goal is to prove that $\mathcal{A}$ cannot distinguish whether a challenge $\mathbf{c}$ is constructed with $\mathcal{A}$'s own message pair (m_0, m_1) or a random message pair (m'_0, m'_1), even when given the challenge's answer $\mathbf{a}$. Specifically, the adversary's view, parameterized by a secret bit $[b]$, is defined as follows:

- The public parameters pp, generated by $\mathsf{Setup}(1^\lambda, T)$.
- A pre-computation pc on pp and a message pair (m_0, m_1), provided by $\mathcal{A}$.
- The commitment $\mathbf{c}'$, generated using $\mathsf{Commit}(pp, [i])$.
- The challenge $\mathbf{c}$, constructed as $\mathsf{Challenge}(pp, \mathbf{c}', \begin{bmatrix} (m_0, m_1) & \text{for } [b] = 0 \\ [m'_0, m'_1] \leftarrow_\$ \mathcal{M}^2 & \text{for } [b] = 1 \end{bmatrix})$.
- The answer $\mathbf{a}$ to the challenge $\mathbf{c}$, obtained by $\mathsf{Solve}(pp, \mathbf{c})$.

We define two hybrid games, G_0 and G_1, parameterized by a secret bit $b \in \{0, 1\}$:

- **Game** G_0: The challenger uses the adversary's messages (m_0, m_1) to construct $\mathbf{c}$, which generates the adversary's view with $b = 0$.
- **Game** G_1: The challenger uses secret random messages (m'_0, m'_1), sampled uniformly from $\mathcal{M}^2$, to construct $\mathbf{c}$, which generates the adversary's view with $b = 1$.

In both games, the adversary $\mathcal{A}$ outputs a guess $b' \in \{0, 1\}$. Suppose, for the sake of contradiction, that the adversary $\mathcal{A}$ can distinguish between G_0 and G_1 with non-negligible probability, i.e., $|\mathsf{Pr}(b' = b \,|\, G_b) - 1/2|$ is non-negligible. Then, at least one of the following must hold:

Distinguishability of Challenge. In Construction 1, the challenge $\mathbf{c}$ is defined as $(\mathbf{c}_1, \mathbf{c}_2, \mathsf{ct}, cm_{\mathsf{ct}})$, where $\mathbf{c}_1$, ct, and cm_{ct} are constructed independently of the secret bit b (i.e., independently of both $\mathbf{c}'$ and the message pair (m_0, m_1)). The only component of $\mathbf{c}$ that depends on b is $\mathbf{c}_2$, which is generated by OT_S. If $\mathcal{A}$ can distinguish between G_0 and G_1 based on $\mathbf{c}$, it implies that $\mathbf{c}_2$ leaks some information about whether $b = 0$ or $b = 1$. However, the OT protocol is assumed to be UC-secure, meaning that the output $\mathbf{c}_2$ (of OT_S) should reveal no information about the sender's messages. Thus, the assumption that $\mathcal{A}$ can distinguish between G_0 and G_1 based on $\mathbf{c}$ (via $\mathbf{c}_2$) directly contradicts the UC-security assumption of the OT protocol.

Distinguishability by extraction. The adversary $\mathcal{A}$ might attempt to distinguish between G_0 and G_1 by extracting one or both of the sender's secret messages from $\mathbf{c}$ (via $\mathbf{c}_2$) and verifying if they match the original provided message pair. However, this leads to a contradiction with the OT protocol, which guarantees that any party without the appropriate secret key cannot extract any information about the secret messages. Therefore, $\mathcal{A}$ cannot distinguish between G_0 and G_1 through extraction.

Distinguishability by verification. In Construction 1, the verification procedure checks $\mathbf{a}$ against $\mathbf{c}_1$, ct, and cm_{ct}, all of which are constructed independently of b. Therefore, $\mathcal{A}$ cannot distinguish between G_0 and G_1 through verification process.

Since none of the above hold, this contradicts the assumption that $|\mathsf{Pr}(b' = b \,|\, G_b) - 1/2|$ is non-negligible. Therefore, the hiding property is established, assuming the underlying OT protocol is UC-secure.

Secrecy. The secrecy of Construction 1 is maintained under the assumption that the OT protocol is UC-secure. This is because the input messages are exclusively transmitted through the OT protocol. In particular, only $\mathbf{c}_2$, which is derived via $\mathsf{OT}_S(pp.\mathsf{crs},\ \mathbf{c}',\ \mathsf{En}_{k^{\mathrm{MSB}}}(m_0),\ \mathsf{En}_{k^{\mathrm{MSB}}}(m_1))$, is constructed using the input messages. With $\mathbf{c}_2$, only one of the transferred messages, m_i, can be retrieved via OT_{Dec}.

Privacy. The privacy of Construction 1 relies on the OT protocol being UC-secure. This is because the output $\mathbf{c}'$ of $\mathsf{Commit}(pp, [i])$ is generated directly using $\mathsf{OT}_R(pp.\mathsf{crs}, [i])$.

Simulatability of Commit. The $\mathcal{E}_{\mathsf{Commit}}$ simply uses the UC-secure OT protocol's extraction mechanism to determine the receiver's choice i' and returns it.

Simulatability of Challenge. The $\mathcal{E}_{\mathsf{Challenge}}$ works as follows: First, it calls the UC-secure OT protocol's extraction mechanism to extract the ciphertexts $\hat{m}_0$ and $\hat{m}_1$, which are encrypted versions of the target messages m_0 and m_1. Next, $\mathcal{E}_{\mathsf{Challenge}}$ monitors all queries (the record of queries) made to the random oracle used during the challenge generation. It identifies records of the form $str_1 \leftarrow \mathsf{H}(\mathbf{c}_1 \parallel str_2)$, which allows it to obtain $k^{\mathrm{MSB}} \parallel k^{\mathrm{LSB}} = k \leftarrow str_1$, the answer $\mathbf{a} \leftarrow str_2$, and the secret $sk_c := \mathsf{De}_{k^{\mathrm{LSB}}}(\mathbf{c}.\mathsf{ct})$ (note that there might be multiple records with the same prefix, i.e., $\mathbf{c}_1 \parallel str_2$). To verify correctness, $\mathcal{E}_{\mathsf{Challenge}}$ checks that $cm_{\mathsf{ct}} \stackrel{?}{=} \mathsf{H}(\mathsf{ct} \parallel sk_c \parallel k)$ and $(\mathbf{c}_1, \mathbf{a}) \stackrel{?}{=} \mathsf{tr}_{sk_c}(pp.pk)$, where $\mathsf{tr}_{sk_c} \in \mathcal{F}$. If one record passes these validations, $\mathcal{E}_{\mathsf{Challenge}}$ decrypts $\hat{m}_0$ and $\hat{m}_1$, recovering the input messages as $m_0 = \mathsf{De}_{k^{\mathrm{MSB}}}(\hat{m}_0)$ and $m_1 = \mathsf{De}_{k^{\mathrm{MSB}}}(\hat{m}_1)$; otherwise, $\tilde{\mathcal{A}}'_S$ sets $(m_0, m_1) \leftarrow_{\$} \mathcal{M}^2$. Lastly, $\mathcal{E}_{\mathsf{Challenge}}$ returns m_0 and m_1.

□

By Theorem 7, Construction 1 is a secure ChronoCloak as defined in Definition 8. Toward UC security, combining this result with Theorem 6 implies that Construction 1 can be directly integrated into the generic protocol from Section 5.5, resulting in a UC-secure implementation of the ideal functionality of ChronoCloak ($\mathsf{F_{CC}}$) defined in Figure 5.5.

5.7 Chapter Summary

In this chapter, we have presented a new construct called ChronoCloak, intended to address the issue of premature exposure of sender's data and privacy-preserving of receiver's interaction. In the context of oblivious digital dissemination, ChronoCloak has the potential to be a viable integrated solution to mitigate premature disclosure. In contrast to its existing counterparts, the proposed ChronoCloak is based on an integrated model and does not require a time server. Essentially, ChronoCloak enables a sender to transmit secrets through a puzzle that must be solved via lengthy computation. By solving the puzzle, only a subset of the secrets is revealed to the receiver, which is oblivious to the sender. Moreover, the receiver may outsource the computation (with public verification), but only the intended recipient is able to retrieve a subset of secrets using the solution to the puzzle. In this work, we have defined an ideal functionality for ChronoCloak and presented a generic protocol to implement this functionality in the random oracle model by integrating a ChronoCloak instance constructed from an OT protocol and a trapdoor-VDF.

Additionally, Construction 1 appears to scale well for multiple receivers, since it allows a sender to transmit multiple contents to multiple receivers using a single challenge with one OT operation per receiver, which is reasonable. However, the protocol does not scale well in scenarios involving multiple senders, as each sender must create its own challenge and corresponding solution, which impacts scalability. Addressing this limitation is left for future work. One straightforward solution could involve building our work on delay primitives such as delay encryption. However, to our knowledge, there is one construction of delay encryption that faces some practicality issues. Therefore, it would be interesting to develop a delay encryption method that suits our application.

Chapter 6

Streamlining CSIDH: Cost-Effective Strategies for Group Actions Evaluation

This chapter introduces a new batching strategy for computing multiple group actions in Commutative Supersingular Isogeny Diffie-Hellman (CSIDH), which is essential for constructing cryptographic schemes such as zero-knowledge proofs and signature schemes. There are two variants of the batching strategy introduced in this work, each of which is suitable for different security requirements and application contexts. For the first variant, we focus on situations where group actions are public, aiming to reduce CSIDH's computational cost. Using our strategy, we show that computational costs can be effectively reduced, making it a viable option for schemes where efficiency is critical. This variant reduces the computational cost of group actions by roughly up to 14%, when compared to non-batched action computation. The second variant is designed for constant-time group actions, which reduces computational costs while maintaining resilience to side-channel timing attacks. This chapter also introduces a new constant-time implementation of CSIDH that, when combined with the second variant, reduces the computational costs of secret action sets by roughly up to 8% compared to individual computation using the state-of-the-art constant-time algorithm, while the new constant time alone reduces computation by approximately up to 4%. The work presented in this chapter appeared in [161].

6.1 Introduction

With the advent of quantum computing, attention has turned to post-quantum cryptography, a quantum resistant solution. A viable candidate in this arena is the CSIDH protocol, first proposed by Castryck et al. in Asiacrypt 2018 [41]. While not a contender for post-quantum standardization by the National Institute of Standards and Technology, CSIDH has received significant attention within the post-quantum cryptography research community. Building upon the hardness of computing isogenies between supersingular elliptic curves, CSIDH offers a promising foundation for various cryptographic constructions, including non-interactive key-exchanges, digital signatures, zero-knowledge proofs, and oblivious transfers.

However, a critical challenge in the practical deployment of CSIDH-based schemes is the efficiency of computing group actions (via an isogeny). Current implementations, while considered secure, suffer from high computational cost and latency, hindering their applicability in real-world scenarios, as explored in [35]. These efficiency concerns are not just theoretical but have practical implications, especially in the context of constructions such as signature schemes [7, 21, 54, 71, 73] or zero-knowledge proofs [71, 20, 7, 59]. As an example, a CSIDH-based Sigma protocol, which has a challenge space of size three and thus a soundness error of $1/3$, requires repeating its group actions in the commit-challenge-response format $\lceil \lambda / \log 3 \rceil$ times to obtain λ bits of security. In these settings, the computational overhead introduced by CSIDH's group action becomes magnified, potentially limiting their scalability and, thus, its adoption.

So, the question we address in this work is *how to strategically compute a set of group actions to minimize the overall computational cost*. In particular, this question is addressed in two contexts: one involves computing a set of private group actions (e.g., for secret commitments), which must be implemented in a constant-time manner to prevent information leakage, and the other involves computing a set of public group actions (e.g., for validating responses), which does not require security restrictions. Through our work, we aim to improve the practical usability of CSIDH-based constructions, making them viable for wider adoption.

Contribution. In this chapter, we propose a new batching strategy to address the above-mentioned question by reducing the number of finite field arithmetic operations required. An *action system*, as used in this chapter, refers to a collection of cryptographic group actions performed on *a set element*. This chapter proposes a practical approach to computing action systems applying two variants of our batching strategy. Our first variant, described in Section 6.3, reduces the computational cost of public action set by roughly

up to 14% compared to element-wise group action evaluation. As described in Section 6.4, we introduce a new constant-time implementation of CSIDH and a second variant that, when combined, reduce the computation cost of a secret action set by roughly up to 8% compared to the current state-of-the-art constant-time CSIDH. On its own, the new constant-time implementation of CSIDH reduces the computations by about 4%. Lastly, the source code for a proof-of-concept implementation of our work is available on https://github.com/ahzawia/Streamlining-CSIDH.

Prior Work. Since the CSIDH proposal in 2018, various attempts have been made to improve the efficiency of CSIDH operations. Most of these proposals focus on improving certain computations within the scheme, such as improving isogeny and scalar multiplication through various methods [39, 43, 120]. Additionally, many improvements aim to secure CSIDH implementations against side-channel attacks, as in [10, 35, 42, 119, 126]. However, most techniques designed for constant-time implementation increase the cost of CSIDH actions. Although many techniques have been proposed to reduce computational cost, they typically exploit the properties of a single isogeny evaluation. To the best of our knowledge, no work has focused on exploiting the properties of evaluating a set of CSIDH group actions. We note that the method proposed in [45] computes group actions in the Single Instruction/Multiple Data fashion on Intel processors, increasing execution throughput.

Organization. The remainder of this chapter is organized as follows. We provide a brief review of the necessary background and preliminaries in Section 6.2, which includes an overview of the state-of-the-art constant-time implementation of [10] and some speed-up techniques from the literature. In Section 6.3, we present our new evaluation strategy for a public group of CSIDH-actions. Then we present experimental results and provide a comparative analysis. In Section 6.4, we introduce a new constant-time CSIDH implementation, a constant-time evaluation of private groups of CSIDH-actions, and outline our speed-up techniques. We then provide our experimental results along with a comparison. Section 6.5 contains some concluding remarks.

6.2 Background and Preliminaries

This section introduces the relevant concepts and notations used in this work.

Notations. The 'cmov' is a conditional move function that takes three inputs: A, B, and C. cmov returns A if condition C is true; otherwise, it returns B.

6.2.1 CSIDH

Castryck et al. [5] proposed CSIDH as a post-quantum cryptographic scheme based on efficient commutative group action. CSIDH was originally introduced as a form of the Diffie-Hellman key exchange, with its security based on the hardness of finding isogenies between random elliptic curves.

Ideal Class Group Action. Let $\mathcal{O}$ be an order of an imaginary quadratic field $\mathbb{Q}(\sqrt{p})$, where p is a large prime. The ideal class group $\mathbf{cl}(\mathcal{O})$ is an abelian group of equivalence classes of invertible fractional $\mathcal{O}$-ideals under multiplication (see [143, Theorem 5.5]). Let $\mathcal{E}_{\mathbb{F}_p}(\mathcal{O})$ be the set of supersingular elliptic curves over a prime finite field $\mathbb{F}_p$ of characteristic p with $\mathcal{O}$. By theory of complex multiplication, the action of $\mathbf{cl}(\mathcal{O})$ on the set $\mathcal{E}_{\mathbb{F}_p}(\mathcal{O})$ defines a group action (i.e., $\star : \mathbf{cl}(\mathcal{O}) \times \mathcal{E}_{\mathbb{F}_p}(\mathcal{O}) \to \mathcal{E}_{\mathbb{F}_p}(\mathcal{O})$ that satisfies a set of properties). Relevant details can be found in [139] and [41, Theorem 7].

Evaluation of an Action. CSIDH operates by performing a group action, computed through isogenies, on supersingular elliptic curves in $\mathcal{E}_{\mathbb{F}_p}(\mathcal{O} = \mathbb{Z}[\sqrt{-p}])$, where p is congruent to -1 modulo a small odd prime for all small odd primes in a set $\mathcal{L}$. Essentially, for a given input curve E, the basic operation of CSIDH involves the secret $\mathfrak{a} \in \mathbf{cl}(\mathcal{O})$ acting on E, resulting in $E' = \mathfrak{a} \star E$.

The main advantage of CSIDH in terms of efficiency is the use of supersingular curves of order $\#E(\mathbb{F}_p) = p + 1$. This allows for an easy construction of $\mathbb{F}_p$-rational subgroups of $E(\mathbb{F}_p)$ with a small prime order (i.e., points of order $\ell_i \in \mathcal{L}$ exist in $\mathbb{F}_p$) by simply selecting $p = 4\ell_1\ell_2 \cdots \ell_n - 1$ with $\ell_i \in \mathcal{L}$ for all $i \in [n]$. These small primes are known as Elkies primes, where the ideal $\ell_i\mathcal{O}$ splits into $\mathfrak{l}_i = (\ell_i, \pi - 1)$ and $\mathfrak{l}_i^{-1} = (\ell_i, \pi + 1)$ for all $\ell_i \in \mathcal{L}$ (i.e., $\ell_i\mathcal{O} = \mathfrak{l}_i\mathfrak{l}_i^{-1}, \forall \ell_i \in \mathcal{L}$). This means that for every $E \in \mathcal{E}_{\mathbb{F}_p}(\mathcal{O})$, two actions (namely $\mathfrak{l}_i$ or $\mathfrak{l}_i^{-1}$) can be applied for each $\ell_i \in \mathcal{L}$. Each of these actions can be computed via an isogeny constructed using Vélu's formulas [149]. Given a sufficiently large set $\mathcal{L}$, we hope that the elements of $\mathbf{cl}(\mathcal{O})$ can be expressed as a product of small integral powers of elements of $\mathcal{L}$ (i.e., $\prod_{i=1}^{n} \mathfrak{l}_i^{e_i}$ for $e_i \in \mathbb{Z}$); this allows for an efficient evaluation of the class group action. Hence, CSIDH's action is defined by a vector of integers $(e_i)_{[n]}$ uniformly sampled from a set $\mathcal{S}^n$ (i.e., $e_i \in \mathcal{S} = [-b, b]$).

The supersingular elliptic curve in CSIDH is defined by its Montgomery form $E_A : y^2 = x^3 + Ax^2 + x$, with the Montgomery coefficient A mod p. Algorithm 1 presents the original CSIDH class group action evaluation.

Algorithm 1: CSIDH's original algorithm [41] for class group action

Input: A $\in \mathbb{F}_p$ such that $E_A : y^2 = x^3 + Ax^2 + x$ is supersingular, and an integer exponent vector $(e_1, \ldots, e_n)$

Output: B such that $E_B : y^2 = x^3 + Bx^2 + x$ is $[\mathfrak{l}_1^{e_1} \cdot \cdots \cdot \mathfrak{l}_n^{e_n}] * E_A$

$B \leftarrow A$
while some $e_i \neq 0$ **do**
  Sample a random $x \in \mathbb{F}_p$
  $s \leftarrow +1$ if $x^3 + Bx^2 + x$ is square in $\mathbb{F}_p$, else $s \leftarrow -1$
  $S \leftarrow \{i : e_i \neq 0, \text{sign}(e_i) = s\}$
  if $S \neq \varnothing$ **then**
    $k \leftarrow \prod_{i \in S} \ell_i$
    $Q \leftarrow [(p+1)/k]T$, `// T, a projective point with x-coordinate`
    for $i \in S$ **do**
      $P \leftarrow [k/\ell_k]Q$ `// Point to be used as kernel generator`
      if $P \neq \infty$ **then**
        $(B, Q) \leftarrow \text{Isogeny}(B, P, Q)$
        $(k, e_i) \leftarrow (k/\ell_i, e_i - s)$
return B

6.2.2 Constant-Time Implementation of CSIDH

In CSIDH, the secret action $\mathfrak{a}$ is represented by a vector of integers $(e_1, \ldots, e_n)$. For all $i \in [n]$, computing $\mathfrak{l}_i^{e_i}$ requires evaluating a degree-ℓ_i isogeny e_i times, which causes the key to directly influence the execution time of Algorithm 1 (i.e., a *variable time* algorithm). To eliminate this dependency, several constant-time methods and efficient implementations have been proposed [10, 35, 42, 119, 126].

Constant-Time Algorithm. A constant-time algorithm has an execution time that does not depend on input and remains the same for multiple executions. Hence, a constant-time version of Algorithm 1 has an execution time independent of the secret vector $(e_1, \ldots, e_n)$. In our context, one of the most important concepts for achieving constant time is the use of a dummy isogeny.

Dummy isogenies. To achieve a constant running time, this technique involves performing a fixed number of isogenies of each degree $\ell_i \in \mathcal{L}$ and only updating the results dictated by the private key (i.e., when $e_i \neq 0$). However, if used and unused isogenies are computed in the same loop, additional multiplications are necessary to push the torsion point for the

next isogeny evaluation. Meyer et al. [119] discussed the use of *tailored* dummy isogeny, where instead of computing the ℓ_i-isogeny and pushing Q through, $[\ell_i]Q$ is computed using the kernel computation of the dummy isogeny with two extra differential additions.

Although the dummy isogenies technique (or dummy computation) helps in hiding the secret's magnitude (i.e., $|e_i|$), it is not entirely sufficient. Certain parts of Algorithm 1 are closely linked to the secret key, such as the number of iterations for computing scalars in Lines 8 and 10, which can reveal the secret key's sign. To address this issue, Meyer et al. [119] suggest changing the secret key space by considering only positive elements (i.e., changing $\mathcal{S} = [-b, b]$ to $\mathcal{S} = [0, 2b]$), which increases the number of dummy isogenies. To enhance efficiency, Onuki et al. [126] proposed using two torsion points (instead of one) in the for-loop and pushing both points through. This approach reduces the magnitude of the key space (i.e., from $2b$ to b), significantly reducing the number of dummy computations.

Unfortunately, the constant-time version of Algorithm 1 performs worse in terms of execution time compared to the variable-time version. Hence, different techniques to speed up the computations have been proposed, such as splitting isogenies into multiple batches [119], multiplicative strategy [120]. A notable work, Constant-Time CSIDH (CTIDH) [10], has introduced a highly optimized constant-time algorithm that reduces the number of field operations, resulting in a faster execution time than its counterpart.

CTIDH: Faster Constant-Time CSIDH. CTIDH, introduced by Banegas et al., improves on previous constant-time CSIDH implementations by *batching* key space to achieve better constant-time performance.

Batching primes. In CTIDH, primes ℓ_i are grouped into B batches, each containing consecutive primes, defined by a vector $N = (N_1, \ldots, N_B)$ where $N_i > 0$ and $\sum_{i=1}^{B} N_i = n$. The set of n primes $\{\ell_1, \ldots, \ell_n\}$ are distributed and relabeled among these batches as: $(\ell_1, \ldots, \ell_{N_1}) := (\ell_{1,1}, \ldots, \ell_{1,N_1})$, $(\ell_{N_1+1}, \ldots, \ell_{N_1+N_2}) := (\ell_{2,1}, \ldots, \ell_{2,N_2})$ and so on.

Batching key space. Secret key elements e_k are also relabeled as $e_{i,j}$ representing j^{th} secret element in the i^{th} batch. For the i^{th} batch $(\ell_{i,1}, \ldots, \ell_{i,N_i})$, secret elements $e_{i,j}$ are sampled such that $\sum_{j=1}^{N_i} |e_{i,j}| \leq m_i$, where m_i is batch bound. For $N \in \mathbb{Z}_{>0}^{B}$ and $m \in \mathbb{Z}_{\geq 0}^{B}$, the CTIDH key space is defined as follows:

$$\mathcal{K}_{N,m} := \{(e_1, \ldots, e_n) \in \mathbb{Z}^n : \sum_{j=1}^{N_i} |e_{i,j}| \leq m_i \text{ for } 1 \leq i \leq B\}.$$

Constant time techniques. CTIDH addresses side-channel timing attacks by computing each isogeny in a batch at the cost of the highest degree isogeny in that batch. The

key insight is that isogeny formulas exhibit a "Matryoshka-doll" structure, where, for i^{th} batch, an $\ell_{i,j}$-isogeny can be computed at the cost of an ℓ_{i,N_i}-isogeny ($\ell_{i,j} \leq \ell_{i,N_i}$) by performing additional dummy operations. As a result, every isogeny computation within the batch has the same computational cost, thus hiding the actual isogeny degree. Finally, to maintain constant time, we compute all m_i isogenies for the i^{th} batch, for all $i \in [B]$. Other operations, such as constant-time scalar multiplication and kernel point generation success rate, are also adjusted to maintain constant-time behavior within the batch. These adjustments ensure that no timing information is leaked about the secret key. Despite the computational overhead, CTIDH improves the performance of constant-time CSIDH mainly by batching the secret key space, which reduces the number of isogenies required. In other words, CTIDH achieves a larger key space with fewer isogenies compared to CSIDH; for example, the key space of 2^{256} in CSIDH-512 requires 438 isogenies [35], while CTIDH needs only 208. The Algorithm 2 outlines the constant-time group action of CTIDH, which consists of the following subroutines:

- UniformRandomPoints takes as input a curve coefficient A of $E_A \in \mathcal{E}_{\mathbb{F}_p}(\mathcal{O})$, and returns a pair of projective points (T_0, T_1) of an order in $\#E(\mathbb{F}_p)$, where $(T_0, T_1) \in (E[p+1] \cap \ker(\pi - 1]), E[p+1] \cap \ker(\pi + 1]))$.
- PointAccept takes as input a kernel point P, batch number I, and an index $J \in [N_I]$. It returns 0 if P is a point at infinity and 1 otherwise, with a distribution based only on I.
- MatryoshkaIsogeny is a constant time isogeny subroutine, with running time depends only on I.

In addition, in this work, we define xMULConstTime as constant-time scalar multiplication, which takes as input a torsion point P, a scalar r, and a batch number I, and returns $[r]P$ in time dependent only on I. In Algorithm 2, it is used in Lines 9, 14, and 20. In contrast, xMUL is a variable-time scalar multiplication algorithm used in Line 8.

Algorithm 2: CTIDH's original algorithm for constant-time group action.

Parameters: $(B \in \mathbb{Z}_{>0}) \leq n, N \in \mathbb{Z}_{>0}^{B}, m \in \mathbb{Z}_{\geq 0}^{B}$
Require: $e \in \mathcal{K}_{N,m}, E_A \in \mathcal{E}_{\mathbb{F}_p}(\mathcal{O})$
Ensure: A with $E'_A = (\prod_i \ell_i^{e_i}) \star E_A$

$(\mu_1, \ldots, \mu_B) \leftarrow (m_1, \ldots, m_B)$
while $\mu_1, \ldots, \mu_B \neq (0, \ldots, 0)$ **do**
  Let $I := \{1 \leq i \leq B : \mu_i > 0\}$ of size k, sorted in ascending order
  for $1 \leq i \leq k$ **do**
    $J_i \leftarrow$ some j such that $e_{i,j} \neq 0$ if such a j exists, else $j \leftarrow_{\$} [N_{I_i}]$
    $\epsilon_i \leftarrow \text{sign}(e_{I_i,J_i})$; `// 1 if` $e_{I_i,J_i} > 0$`; 0 if` $e_{I_i,J_i} = 0$`; o.w. -1`
  $(T_0, T_1) \leftarrow \text{UniformRandomPoints}(A)$
  $(T_0, T_1) \leftarrow (([r]T_0, [r]T_1)$ where $r = 4 \prod_{i \notin I} \prod_{j=1}^{N_i} \ell_{i,j}$
  $(T_0, T_1) \leftarrow (([r']T_0, [r']T_1)$ where $r' = \prod_{i \in I} \prod_{\substack{j=1 \\ j \neq J_i}}^{N_i} \ell_{i,j}$
  $r' \leftarrow \prod_{i \in I} \ell_{i,J_i}$
  for $j = k$ **to** 1 **do**
    $s \leftarrow \text{SignBit}(\epsilon_j)$; `// 1 if` $\epsilon_j < 0$`; o.w. 0`
    $r' \leftarrow r'/\ell_{I_j,J_j}$
    $P \leftarrow [r']T_s$
    $f_j \leftarrow \text{PointAccept}(P, I_j, J_j)$
    if $f_j = 1$ **then**
      $(A', (T'_0, T'_1)) \leftarrow \mathsf{Matryoshkalsogeny}(A, P, (T_0, T_1), I_j, J_j)$
      if $\epsilon_j \neq 0$ **then**
        $(A, T_0, T_1) \leftarrow (A', T'_0, T'_1)$
    $(T_0, T_1) \leftarrow ([\ell_{I_j,J_j}]T_0, [\ell_{I_j,J_j}]T_1)$
  for $1 \leq i \leq k$ **do**
    $(\mu_{I_i}, e_{I_i,J_i}) \leftarrow (\mu_{I_i} - f_i, e_{I_i,J_i} - f_i\epsilon_i)$
return A

6.3 Public Action Set Evaluation

In this section, we propose a strategy for evaluating a set of public CSIDH actions with the aim of improving the evaluation of the set. Below, we will use either a vector of integers $\boldsymbol{e}^{(i)}$ of size n or a Gothic font to denote the i^{th} ideal $\mathfrak{e}_i$ from a set.

Action System. An execution set is a set of independent CSIDH-actions $\{\boldsymbol{e}^{(k)}\}_{k\in[c]}$ that need to be evaluated to run an application (e.g., a digital signature), where $c \in \mathbb{N}$ is the execution set size. The action system is a subset of the execution set. In particular, an action system is a set of actions $\{\boldsymbol{e}^{(k)}\}_{k\in[c']}$ with the same starting point E_A, where $c' \leq c$ is the system size. The difficulty of an action $\mathfrak{e}_i$ is the absolute sum of all $\boldsymbol{e}^{(i)}$'s elements (i.e., $\sum_{m=1}^{n} |\boldsymbol{e}_m^{(i)}|$).

Basic Idea

Our approach is centered around exploiting group actions' commutative property (i.e., $\mathfrak{ab} \star E = \mathfrak{a} \star (\mathfrak{b} \star E) = \mathfrak{b} \star (\mathfrak{a} \star E)$). By carefully arranging multiple actions in a specific order (i.e., a strategy), we take advantage of commutativity, which results in certain group actions overlapping during the evaluation process. The design of this strategy becomes crucial, allowing us to identify and eliminate redundant overlapping actions within the action system. In other words, the crucial aspect of strategically constructing an action system $\{\boldsymbol{e}^{(j)}, \boldsymbol{e}^{(i)}\}$ is to ensure that $\boldsymbol{e}^{(j)}$ and $\boldsymbol{e}^{(i)}$ have a significant amount of overlap. This optimization strategy is expected to result in around $\sim 20\%$ reduction in the computational cost. Additionally, it is necessary to recompute the strategy for each new execution set during the evaluation process, as the strategy depends on the input execution set; unfortunately, this introduces computational overhead. It is important to understand that the final outcome will depend on the method used to derive the strategy. To illustrate, for $\boldsymbol{e}^{(j)}$ and $\boldsymbol{e}^{(i)}$, we are looking for a common action $\boldsymbol{q}^{(ji)}$ such that $\boldsymbol{e}^{(j)} = \boldsymbol{q}^{(ji)} + \boldsymbol{r}^{(j)}$ and $\boldsymbol{e}^{(i)} = \boldsymbol{q}^{(ji)} + \boldsymbol{r}^{(i)}$. Hence, to obtain $\boldsymbol{e}^{(j)} \star E$ and $\boldsymbol{e}^{(i)} \star E$, we compute $E' = \boldsymbol{q}^{(ji)} \star E$ and then $\boldsymbol{r}^{(j)} \star E'$ and $\boldsymbol{r}^{(i)} \star E'$. The question to be answered here is how to find a common action $\boldsymbol{q}^{(ji)}$ efficiently, which can be formulated as follows.

Problem 1. *Given $\{\boldsymbol{e}^{(k)}\}_{k\in[c]}$, find distinct pairs $\boldsymbol{e}^{(j)}$ and $\boldsymbol{e}^{(i)}$ with a minimum distance, i.e., with $\boldsymbol{q}^{(ji)}$ that minimizes the difficulty of $\boldsymbol{r}^{(j)}$ and $\boldsymbol{r}^{(i)}$.*

To answer this question, one straightforward method involves finding a pair $(\boldsymbol{e}^{(j)}, \boldsymbol{e}^{(i)})$ with $\boldsymbol{q}^{(ji)}$ that has the most common values. However, such a method might not be the most efficient. Therefore, in this study, we propose using a more practical metric, referred to as M, to measure the proximity between $\boldsymbol{e}^{(j)}$ and $\boldsymbol{e}^{(i)}$. Here, the distance metric M quantifies the number of intersecting positive and negative (non-zero) elements in the pair, with a higher M indicating a smaller distance (i.e., greater overlapping). We then identify the pair with the highest M-value. Thus, the problem is redefined as follows: *find distinct pairs $\boldsymbol{e}^{(j)}$ and $\boldsymbol{e}^{(i)}$ from $\{\boldsymbol{e}^{(k)}\}_{k\in[c]}$ that maximize M.* The main advantage of this method is that it only involves determining the number of 1's in the output of an XOR of two n-bit

operands, which has a low computational cost. We then select the pairs $(\boldsymbol{e}^{(j)}, \boldsymbol{e}^{(i)})$ with the highest M-value, where $i, j \in [c]$ and $i \neq j$. Given this approach, it is easy to see that the complexity is at most equivalent to $\frac{(c-1)c}{2}(1+n)$ integer additions, which constitute an overhead. However, this overhead is negligibly small compared to other operations required by group action for the parameters used in our experiments. Figure 6.1 is an example implementation for identifying action systems from a public execution set. Finally, our empirical results indicate that our method offers a reasonable approximation to $\boldsymbol{q}^{(ji)}$.

```python
def identify_action_systems_for_public_strategy(execution_set):
    # execution set size
    c = len(execution_set)
    # This function works with public set, so no security restriction
    used_action = [False] * c
    # The output list of action systems, i.e., list of pair of actions
    action_systems = []
    # Iterate over all actions in the execution set.
    for i in range(c):
        # Skip if this action has already been paired
        if used_action[i]:
            continue
        best_j = None
        M = -1 # distance metric M
        for j in range(i + 1, c):
            # Skip if this action has already been paired
            if used_action[j]:
                continue
            # Initialize match score for the current pair (i, j)
            M_ij = 0
            # calculate match score (M_ij)
            for idx in range(n_primes):
                e_i_idx = execution_set[i][idx]
                e_j_idx = execution_set[j][idx]
                # sign(x) returns 1 if x>0, 0 if x = 0, else  -1
                M_ij += int(sign(e_i_idx) != 0 and sign(e_i_idx) == sign(e_j_idx))
            # Update the best match if the current pair has a higher score
            if M_ij > M:
                M = M_ij
                best_j = j
        if best_j is not None:
            action_systems.append((i, best_j))
            used_action[i] = True
            used_action[best_j] = True
    return action_systems
```

Figure 6.1: An example implementation for identifying public action systems.

Analysis of Our Strategy

Here, we discuss how we analyze the computational cost of variable-time CSIDH implementations using our strategy. In particular, and for comparability to previous work, we focus on CSIDH-512 [41].

Selected parameters. The implementation of CSIDH-512 is defined over the finite field $\mathbb{F}_p$, where $p = 4\prod_{i=1}^{n=74} \ell_i - 1$ with ℓ_1 to ℓ_{73} being all odd primes from 3 to 373 and $\ell_{74} = 587$. Depending on the application, there are two methods to sample class elements. The first method involves sampling an element from a defined key space of size 2^{256}, as in [10, 41, 126]. The second method involves sampling "directly" from the ideal class group, provided the class order is known. Although our initial assessment using the first method shows a computational reduction of between 17% and 22% (depending on the defined key space), we will analyze our strategy with the second method because it has been used in relevant applications that can benefit from our work, such as [7, 54, 59]. Thus, given that $\mathbf{cl}(\mathcal{O})$ is cyclic with generator $\mathfrak{l}$ and order O, we sample the action $\mathfrak{e}_i$ in a form of $\mathfrak{l}^e$, where $e \leftarrow_\$ \mathbb{Z}_O$ and $\ell \in \mathcal{L}$. We then use a reduction to obtain a short[1] vector $\boldsymbol{e}^{(i)} = (\boldsymbol{e}_1^{(i)}, \ldots, \boldsymbol{e}_n^{(i)})$ by solving the closest vector problem in the lattice of relations, as described in [19, 21].

Cost metric. The symbols $\mathbf{M}$, $\mathbf{S}$, and $\mathbf{A}$ are used to denote the operation count for multiplication, squaring, and addition (or subtraction) on the finite field $\mathbb{F}_p$, respectively. Similar to [10], we use two metrics to reflect the total number of operations, named Metric 1 and Metric 2, defined by $\mathsf{Sum}((\mathbf{M}+\mathbf{S}+\mathbf{A}) \odot (1,1,0))$ and $\mathsf{Sum}((\mathbf{M}+\mathbf{S}+\mathbf{A}) \odot (1,0.8,0.05))$, respectively. To evaluate a group action, we use the implementation of the variable-time CSIDH-action function presented in the most recent version of CTIDH software, which is similar to the implementation in CSI-FiSh [19]. The main difference is the use of an efficient point sampler subroutine, namely Elligator [18]. For cost measurement, we use the operation counter in the CTIDH's software to obtain the count for the number of $\mathbb{F}_p$-operations in the class group action of CSIDH-512. Because the costs vary based on the execution set size, we ran an experiment on various sizes, comparing the cost of two different metrics with an action system size of two.

Experimental results. In every experiment, the elements of execution set were randomly chosen from the key space, ensuring that any speedup reflects typical scenarios. Table 6.1 provides the cost for computing an execution set of size $c = 50$ with an action system of size $c' = 2$, which shows a roughly 14% reduction in computational cost in both Metric 1 and 2. Also, as depicted in Figure 6.2, we present our experimental results measuring the computational costs for action system of size two of execution sets ranging in size from 2

[1] A vector with small difficulty.

to 58 with an increment of four.

Table 6.1: The computational cost for computing a CSIDH-512 execution set of size 50 with an action system of size two, i.e., executing 25 action systems of size 2. The experiment was conducted using the source code from [11] and [19], with the latter being built using the makefile from its fork [142].

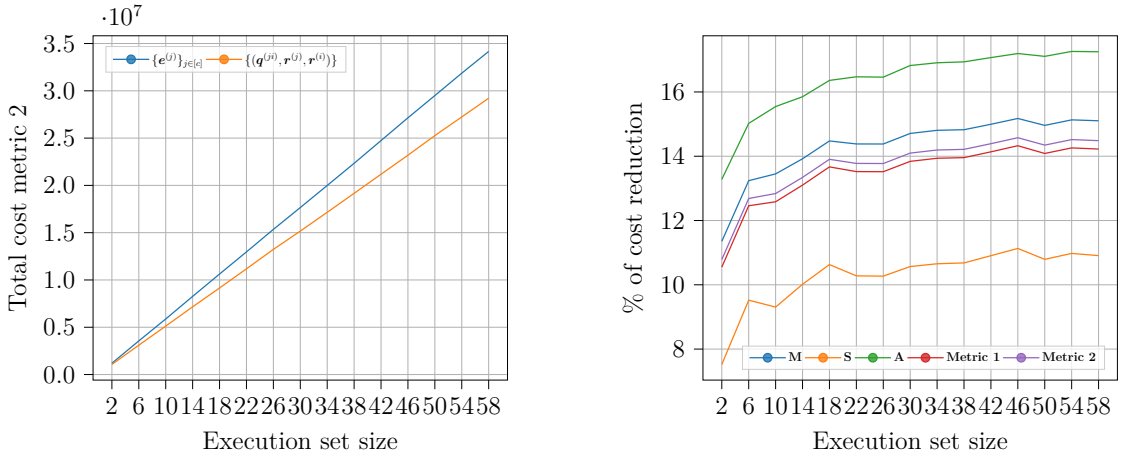
	M	**S**	**A**	Cost	
				Metric 1	Metric 2
Individual computation [11, 19, 41]	23,310,847	6,177,830	23,651,547	29,488,677	29,435,689
Strategic computation (this work, Section 6.3)	19,796,236	5,488,757	19,651,986	25,284,993	25,169,841
Reduction:	**15%**	**11.15%**	**16.91%**	**14.25%**	**14.49%**

6.4 Private Action Set Evaluation

This section presents methods for reducing the number of field arithmetic operations in the constant-time version of CSIDH, namely Algorithm 2 of CTIDH. In particular, we are here concerned with situations where the protocol requires the computation of a secret action system, such as those described in [7, 20, 21, 71, 59]. Hence, we need to consider the possibility of a physical attack vector (by a malicious observer) that may be used to obtain the secret set of group actions, i.e., $\{\boldsymbol{e}^{(k)}\}_{k\in[c]}$.

Basic Idea

At a high level, our technique consists of three parts. First, we compute the torsion points T_0 and T_1 that correspond to positive and negative secret exponents with *distinct* orders in each execution round, in contrast to previous work CTIDH [10] and [126]. With our proposed subroutine in Algorithm 4, GetTorsionPoint, we eliminate the inner-loop point multiplication at Line 20 in Algorithm 2. Hence, we present a new constant-time implementation of CSIDH in Algorithm 3. Secondly, we evaluate the execution set strategically by constructing action systems using an approach similar to that described in Section 6.3. This method helps reduce the number of torsion points computed and the isogeny evaluations required. Finally, since every system starts from the same elliptic curve E_A, we



(a) Total cost with Metric 2.

(b) Percentage of cost reduction.

Figure 6.2: Summary of our results for evaluating an execution set of size c ranging from 2 to 58 with a CSIDH-512's action system of size $c' = 2$.

implement a method to eliminate redundant computation to obtain torsion points T_0 and T_1.

New Constant-Time Implementation

Here we give a description of our constant-time implementation of CSIDH algorithm. First, we concentrate on the inner loop (Lines 7–20) of CTIDH Algorithm 2. In this algorithm, Line 9 computes the torsion points with the right order to compute all isogenies in a k-iteration. As a result, this computation is performed using constant-time xMULConstTime to conceal the secret isogeny degree. Further, at the end of each iteration, we need to push the other torsion point (i.e., T_{1-s}) to attain the required torsion degree for computing the subsequent isogeny; however, we are required to push both points when $\epsilon_i = 0$, which leaks this information. Hence, Line 20 always pushes both resulting in an extra overhead, which can be overcome by removing Line 20 and replacing Line 9 with the subroutine GetTorsionPoint as described in Algorithm 4.

Algorithm 3: New constant-time group action.

Parameters: $(B \in \mathbb{Z}_{>0}) \leq n, N \in \mathbb{Z}_{>0}^{B}, m \in \mathbb{Z}_{\geq 0}^{B}$
Require: $e \in \mathcal{K}_{N,m}, E_A \in \mathcal{ELL}_{\mathcal{O}}$
Ensure: A with $E'_A = (\prod_i \ell_i^{e_i}) \star E_A$

$(\mu_1, \ldots, \mu_B) \leftarrow (m_1, \ldots, m_B)$
while $\mu_1, \ldots, \mu_B \neq (0, \ldots, 0)$ **do**
  Let $I := \{1 \leq i \leq B : \mu_i > 0\}$ of size k, sorted in ascending order
  for $1 \leq i \leq k$ **do**
    $J_i \leftarrow$ some j so that $e_{i,j} \neq 0$ if such a j exists, else $j \leftarrow_{\$} [N_{I_i}]$
    $\epsilon_i \leftarrow \mathsf{sign}(e_{I_i,J_i})$ `// 1 if` $e_{I_i,J_i} > 0$`; 0 if` $e_{I_i,J_i} = 0$`; -1 o.w.`
    $\tilde{f}_i \leftarrow \mathsf{RandmCoinToss}(\epsilon_i, I_i, J_i)$ `//` $\tilde{f}_i \in \{0,1\}$ `is a random coin based on` ϵ_i, I_i`, and` J_i
  $(T_0, T_1) \leftarrow \mathsf{UniformRandomPoints}(A)$
  $(T_0, T_1) \leftarrow ([r]T_0, [r]T_1)$ where $r = 4 \prod_{i \notin I} \prod_{j=1}^{N_i} \ell_{i,j}$
  $(T_0, T_1) \leftarrow \mathsf{GetTorsionPoint}((T_0, T_1), \tilde{f}, \epsilon, I, J, N)$
  $\epsilon' \leftarrow 1$
  for $j = k$ **to** 1 **do**
    $\epsilon' \leftarrow \mathsf{cmov}(\epsilon_j, \epsilon', \epsilon_j \neq 0 \text{ and } \tilde{f}_j \neq 0)$
    $s \leftarrow \mathsf{SignBit}(\epsilon')$ `// 1 if` $\epsilon' < 0$`, o.w. 0`
    $P \leftarrow T_s$
    for $i = j - 1$ **to** 1 **do**
      $P \leftarrow \mathsf{xMULConstTime}(P, \ell_{I_i,J_i}, I_i)$
      if $i = j - 1$ **then**
        $T_s \leftarrow \mathsf{cmov}(P, T_s, \epsilon_i = 0 \text{ or } \tilde{f}_i = 0)$
    $(f_j, P) \leftarrow \mathsf{PointAccept}(P, T_s, \epsilon_j, \tilde{f}_j)$
    if $f_j = 1$ **then**
      $(A', (T'_0, T'_1)) \leftarrow \mathsf{MatryoshkaIsogeny}(A, P, (T_0, T_1), I_j, J_j)$
      $(A, T_0, T_1) \leftarrow \mathsf{cmov}((A', T'_0, T'_1), (A, T_0, T_1), \epsilon_j \neq 0)$
    if $j = k$ **then**
      $T_0 \leftarrow \mathsf{xMULConstTime}(T_0, \ell_{I_j,J_j}, I_j)$
  for $1 \leq i \leq k$ **do**
    $(\mu_{I_i}, e_{I_i,J_i}) \leftarrow (\mu_{I_i} - f_i, e_{I_i,J_i} - f_i \epsilon_i)$
return A

Our constant-time algorithm in Algorithm 3 utilizes the GetTorsionPoint subroutine, ensuring that both points T_0 and T_1 have the right order with respect to the sign ϵ of the secret vector. Hence, we do not have to push both torsion points, which eliminates the

Algorithm 4: Torsion point computation, GetTorsionPoint.

Input: (T_0, T_1), $\tilde{f} \in \{0,1\}^k$, $\epsilon \in \{-1,0,1\}^k$, $I \in \mathbb{Z}_{\geq 0}^{k \leq B}$, $J \in \mathbb{Z}_{>0}^{k}$, $N \in \mathbb{Z}_{>0}^{B}$
Output: Updated T_0 and T_1

1 $\epsilon' \leftarrow 1$;
2 **for** $i = k$ **to** 1 **do**
3 $\quad$ ClearSet $\leftarrow \{1, \ldots, N_{I_i}\}$;
4 $\quad$ $\epsilon' \leftarrow \mathsf{cmov}(\epsilon_i, \epsilon', \epsilon_i \neq 0 \text{ and } \tilde{f}_i \neq 0)$;
5 $\quad$ **for** $j \in$ ClearSet **do**
6 $\quad\quad$ **if** $\epsilon' = 1$ **then**
7 $\quad\quad\quad$ $T_1 \leftarrow \mathsf{xMUL}(T_1, \ell_{I_i,j})$;
8 $\quad\quad$ **else if** $\epsilon' = -1$ **then**
9 $\quad\quad\quad$ $T_0 \leftarrow \mathsf{xMUL}(T_0, \ell_{I_i,j})$;
10 $\quad$ $j_{target} \leftarrow J_i$;
11 $\quad$ ClearSet $\leftarrow$ ClearSet $\setminus \{j_{target}\}$;
12 $\quad$ **for** $j \in$ ClearSet **do**
13 $\quad\quad$ **if** $\epsilon' = 1$ **then**
14 $\quad\quad\quad$ $T_0 \leftarrow \mathsf{xMULConstTime}(T_0, \ell_{I_i,j}, I_i)$;
15 $\quad\quad$ **else if** $\epsilon' = -1$ **then**
16 $\quad\quad\quad$ $T_1 \leftarrow \mathsf{xMULConstTime}(T_1, \ell_{I_i,j}, I_i)$;
17 **return** (T_0, T_1);

extra overhead (i.e., the scalar multiplication at Line 20). In Algorithm 3, the subroutine RandmCoinToss takes a sign ϵ_i, batch number I_i, and position J_i, and performs a coin toss with a success probability $\alpha_{\epsilon_i,I_i,J_i}$, which will be defined later.

Implementation Reasoning and Security. In Algorithm 3, we compute one isogeny per batch in each round (Lines 12−25) using the torsion point with the correct order from the subroutine GetTorsionPoint. In GetTorsionPoint, the first for-loop operation (Lines 5-9 in Algorithm 4) clears the order of the point with the opposite sign by performing scalar multiplication for all primes within the batch for all target batches I. This process is independent of the secret sign, allowing the use of the inexpensive scalar multiplication xMUL instead of xMULConstTime. The second for-loop performs, for each batch in I, a scalar multiplication by $\prod_{i \in I} \prod_{j \in \mathsf{ClearSet} \setminus \{J_i\}} \ell_{i,j}$. Therefore, we use xMULConstTime to conceal ℓ_{i,J_i} that represents the selected secret isogeny. In this case, the number of multiplications is $\sum_{i \in I} N_i - 1$; therefore, the number of multiplications is independent of the secret ϵ_i and rather depends on the vector I, which is public information. Additionally, when $\epsilon_i = 0$,

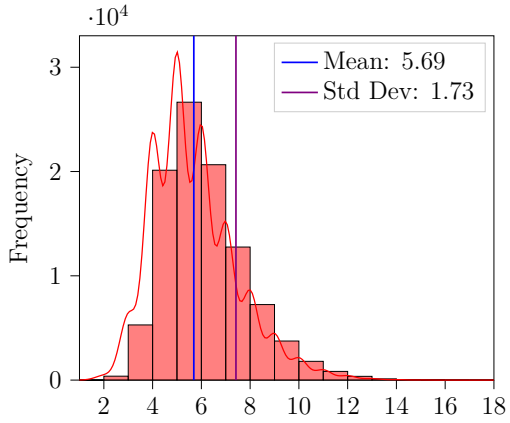
the subroutine GetTorsionPoint performs the multiplication using the sign of the previous non-zero ϵ'.

Kernel point failure. In Algorithm 3, Lines 16–17 produce a possible kernel generator, denoted as P. With probability $1 - 1/\ell_{I_j,J_j}$, P is a kernel of order ℓ_{I_j,J_j}, indicated by $f'_j = 1$, where ℓ_{I_j,J_j} is related to J_j, the secret isogeny degree. Conversely, with probability $1/\ell_{I_j,J_j}$, P is likely a point at infinity, indicated by $f'_j = 0$. As pointed out in [10], the value of f'_j depends on J_j. Therefore, as in [10], we deliberately increase the failure probability from $1/\ell_{I_j,J_j}$ to $1/\ell_{I_j,1}$, with $\ell_{I_j,1}$ being the first prime in the batch $(\ell_{I_j,1}, \ldots, \ell_{I_j,N_{I_j}})$. To achieve this, we use a biased coin flip variable $\tilde{f}_j$ with a success probability $\alpha_{\epsilon_j,I_j,J_j}$ (utilizing RandmCoinToss). We then set f_j given both f'_j and $\tilde{f}_j$ (employing PointAccept). Specifically, for $\epsilon_j \neq 0$, the subroutine PointAccept yields (f_j, P) with $f_j = 1$ exclusively when P is not at infinity *and* the coin flip $\tilde{f}_j$ succeeds with probability $\alpha_{\epsilon_j,I_j,J_j} = \frac{\ell_{I_j,J_j}(\ell_{I_j,1}-1)}{\ell_{I_j,1}(\ell_{I_j,J_j}-1)}$. In instances where $\epsilon_j = 0$, it returns (f_j, T_s) with f_j being based solely on the coin flip $\tilde{f}_j$, with probability $\alpha_{0,I_i,J_i} = 1 - 1/\ell_{I_j,1}$. As a result, the value of f_j is rendered independent of both J_j and ϵ_j.

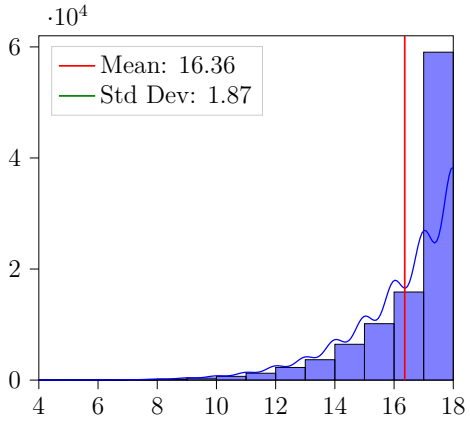
MatryoshkaIsogeny. As in CTIDH, MatryoshkaIsogeny performs a fixed number of operations based only on input $i \in I$, regardless of the input point order. In Algorithm 3, we thus feed a dummy input point to MatryoshkaIsogeny for secret with a value of zero ($\epsilon_j = 0$) and PointAccept outputs $f_j = 1$, so that the computational cost remains the same regardless of ϵ_j's value.

Strategic Computation of a Private Action System

In this section, we propose a constant-time strategy for evaluating a private set of actions, which is a variant to the strategy in Section 6.3. In this discussion, we define a *round of execution* as a single iteration that evaluates a group action, involving the computation of k-isogenies for the targeted batches I, as specified in Algorithm 3, Lines 12-25. To fully compute an action, it requires at least $\mathsf{max}(m_1, \ldots, m_B)$ rounds. To compute a private set of actions, we use our constant-time strategy to eliminate d rounds of execution, referred to as d-common round(s) $\boldsymbol{q}^{(ij)}$. When constructing the action system, the common round must execute a common (overlapped) action set *in all targeted batches*. This is essential to ensure that the execution of the common round involves a fixed number of isogeny computations, which is equal to the number of targeted batches, regardless of the overlapped action $\boldsymbol{q}^{(ij)}$. To fully utilize the strategy, it is essential that the d-common round(s) include d overlapping actions within each of the targeted batches (i.e., the difficulty of $\boldsymbol{q}^{(ij)} = \#$ of the target batches $\times d$). This contrasts with the strategy outlined in Section 6.3, which may



(a) The distribution of the maximum absolute value.



(b) The distribution of the sum of secret elements $(e_{11,1}, \ldots, e_{11,8})$, such that $m_{11} = 18$.

Figure 6.3: The distribution of the sum of secret elements $(e_{11,1}, \ldots, e_{11,8})$ and the distribution of the maximum absolute value in the secret vector (i.e., $\mathsf{max}(|e_{11,1}|, \ldots, |e_{11,8}|)$) for Batch 11 in the CTIDH-512 over 100 thousands random distinct keys.

not necessarily require d overlapping actions within each of the targeted batches. In other words, for each batch that does not overlap, we will be forced to use a dummy isogeny, not only negating the gain we are seeking but also possibly resulting in incorrect output. Fortunately, due to the distribution of keys in each batch's key space, overlapping is highly likely to occur, particularly in moderate size execution sets and with $d = 1$. For example, in batches 3 through 13 of CTIDH-512, the bulk of keys in the key space have a batch sum close to the batch sum bound m_i, and the largest element is less than half of the batch sum m_i, as shown in Figure 6.3. Hence, the vast majority of keys have a flat spectrum of small values, maximizing the likelihood of common round overlaps across batches within a key set.

In this work, we employ a single common round of execution (i.e., $d = 1$ with the difficulty of $\boldsymbol{q}^{(ij)} = \#$ of target batches) for targeted batches between 3 to 13 of CSIDH-512. However, and for further computational reduction, two or more common rounds can be utilized in hybrid form. For example, for an execution set of size c, 20% of the action systems can be executed with two common rounds, the remaining 80% with one round, and so forth.

In addition, the efficiency of our strategy depends critically on the torsion point being full order in the common execution round, since a torsion point that is not of full order will

prevent us from fully executing the common round. There are two ways to avoid this issue: i) select a targeted overlapped-batches with a low failure probability, such as batches with larger primes, and ii) use a deterministic seed for the UniformRandomPoints subroutine (in the first round of the execution) that guarantees full-torsion points to be generated.

Elligator. As in the previous works [10, 119, 126], we use Elligator : $E_A \to T_{0/1}$ [18] in Algorithm 3 to construct UniformRandomPoints. Also, given E_A, Elligator can initially be executed using a deterministic seed, stored as public parameters, that results in full-order points, i.e., T_0 and T_1 both have order $(p+1)/4$.

Reducing Torsion Point Computation

To evaluate group actions (e.g., $\boldsymbol{q}^{(ij)}$), a torsion point of a specific degree must be sampled for each round of execution. Duplicate point sampling and multiplication can be eliminated from redundant torsion point computations, as some or all action systems in the execution set share the same initial E_A. In this regard, this technique is applied in the first round of execution or whenever we have actions starting from the same E_A. Specifically, we adjust the computation of torsion points to determine two pairs of torsion points, one pair for each corresponding action, as detailed in Algorithm 5. The latter includes a subroutine shown in Algorithm 6, which is identical to Algorithm 4 except that it uses xMULConstTime instead of xMUL on lines 10–11.

Analysis of Our Strategy

In this section, we analyze the computational cost of constant-time CSIDH implemented using our strategy.

Selected parameters. In order to be able to provide a comparative analysis based on computational cost, we use parameters similar to those of CTIDH. In particular, we consider CSIDH-512 [41], which is characterized by the finite field $\mathbb{F}_p$, where $p = 4\prod_{i=1}^{74} \ell_i - 1$ with ℓ_1 to ℓ_{73} being all odd primes from 3 to 373 and $\ell_{74} = 587$. For the secret key space, we consider CTIDH-512's proposal $\mathcal{K}_{N,m}$ introduced in [10], which requires CTIDH 208 odd-isogeny evaluations and a minimum of 18 torsion point samplings and computations. Furthermore, our work does not make use of the improved version of CTIDH, namely SECSIDH [35], since the latter is focused on improvements based on optimization of field arithmetic (e.g., using the Karatsuba algorithm for field multiplication) and employs the same algorithm as CTIDH, whereas we aim to reduce the *number* of finite field arithmetic operations. In this work, we modify CTIDH's algorithms in combination with our previous

Algorithm 5: Initial round torsion point computation.

Input: $E_A \in \mathcal{E}_{\mathbb{F}_p}(\mathcal{O})$, $(\tilde{f}^1, \tilde{f}^2) \in \{0,1\}^{2k}$, $(\epsilon^1, \epsilon^2) \in \{-1,0,1\}^{2k}$, $I \in \mathbb{Z}_{\geq 0}^{k \leq B}$, $(J^1, J^2) \in \mathbb{Z}_{>0}^{2k}$

Output: $(T_0^1, T_1^1), (T_0^2, T_1^2)$

$(T_0, T_1) \leftarrow \mathsf{UniformRandomPoints}(A)$

$(T_0, T_1) \leftarrow ([r]T_0, [r]T_1)$ where $r = 4 \prod_{i \notin I} \prod_{j=1}^{N_i} \ell_{i,j}$

CommonTargets $\leftarrow$ list of B-empty lists

for $i = k$ **to** 1 **do**

  $j' \leftarrow_{\$} \{1, \ldots, N_{I_i}\} \setminus \{J_i^1, J_i^2\}$

  $\text{CommonTargets}_i \leftarrow \mathsf{cmov}((J_i^1, J_i^2), (J_i^1, j'), J_i^1 \neq J_i^2)$

  $\text{CommonTargets}_i \leftarrow \mathsf{cmov}([N_{I_i}], \text{CommonTargets}_i, N_{I_i} \leq 2)$

  $\mathsf{ClearSet} \leftarrow \{1, \ldots, N_{I_i}\} \setminus \{\text{CommonTargets}_i\}$

  for $j \in \mathsf{ClearSet}$ **do**

    $T_0 \leftarrow \mathsf{xMULConstTime}(T_0, \ell_{I_i,j}, I_i)$

    $T_1 \leftarrow \mathsf{xMULConstTime}(T_1, \ell_{I_i,j}, I_i)$

$(T_0^1, T_1^1) \leftarrow \mathsf{GetTorsionPoint2}((T_0, T_1), \tilde{f}^1, \epsilon^1, I, J^1, \text{CommonTargets})$

$(T_0^2, T_1^2) \leftarrow \mathsf{GetTorsionPoint2}((T_0, T_1), \tilde{f}^2, \epsilon^2, I, J^2, \text{CommonTargets})$

return $(T_0^1, T_1^1), (T_0^2, T_1^2)$

computation strategy, both of which are independent of field-arithmetic level optimizations. Furthermore, SECSIDH works on CSIDH with very large parameters from CSIDH-2048 to CSIDH-9216. While their techniques are more effective with these parameters, they appear to have limited practical applications, as the performance of the applications in question relies heavily on knowing the order of the ideal class $\mathbf{cl}(\mathcal{O})$.

Cost metric. To evaluate the action systems, we construct our algorithms using the constant-time CTIDH functions presented in the most recent version of the **high-ctidh** software [11] built with the makefile from its fork [142], including Elligator, xMULConstTime, and Matryoshkalsogeny. For cost measurement, we use the arithmetic operation counter in the **high-ctidh** software to obtain the count for the number of operations on $\mathbb{F}_p$ evaluating the execution set.

Results. Our results are compared with individual computations of the execution set carried out using the group action implemented in **high-ctidh** software [11, 142]. We initially benchmarked our constant-time Algorithm 3 (and its subroutine Algorithm 4) alone, without our strategic method, against the state-of-the-art CTIDH algorithm [11, 142]. For CTIDH-512, our results show a reduction of 2.75% for field multiplication, 11.08% for

Algorithm 6: GetTorsionPoint2 (revised from Algorithm 4)

Input: (T_0, T_1), $\tilde{f} \in \{0,1\}^k$, $\epsilon \in \{-1,0,1\}^k$, $I \in \mathbb{Z}_{\geq 0}^{k \leq B}$, $J \in \mathbb{Z}_{>0}^k$, $\mathsf{ClearSets} \in \mathbb{Z}_{>0}^k$
Output: Updated T_0 and T_1

$\epsilon' \leftarrow 1$;
for $i = k$ **to** 1 **do**
  $\mathsf{ClearSet} \leftarrow \mathsf{ClearSets}_i$;
  $\epsilon' \leftarrow \mathsf{cmov}(\epsilon_i, \epsilon', \epsilon_i \neq 0 \text{ and } \tilde{f}_i \neq 0)$;
  for $j \in \mathsf{ClearSet}$ **do**
    if $\epsilon' = 1$ **then**
      $T_1 \leftarrow \mathsf{xMULConstTime}(T_1, \ell_{I_i,j}, I_i)$;
    else if $\epsilon' = -1$ **then**
      $T_0 \leftarrow \mathsf{xMULConstTime}(T_0, \ell_{I_i,j}, I_i)$;
  $j_{target} \leftarrow J_i$;
  $\mathsf{ClearSet} \leftarrow \mathsf{ClearSet} \setminus \{j_{target}\}$;
  for $j \in \mathsf{ClearSet}$ **do**
    if $\epsilon' = 1$ **then**
      $T_0 \leftarrow \mathsf{xMULConstTime}(T_0, \ell_{I_i,j}, I_i)$;
    else if $\epsilon' = -1$ **then**
      $T_1 \leftarrow \mathsf{xMULConstTime}(T_1, \ell_{I_i,j}, I_i)$;
return (T_0, T_1);

squaring, 4.48% for Metric 1, and 3.96% for Metric 2; however, we observe a 1.12% increase in field addition. Furthermore, Table 6.2 illustrates roughly up to 8% reduction in computational cost for an execution set of size 4 with an action system of size 2.

Table 6.2: The computational cost for computing two action systems of size 2, i.e., $\{\{\boldsymbol{e}^{(1)}, \boldsymbol{e}^{(2)}\}, \{\boldsymbol{e}^{(3)}, \boldsymbol{e}^{(4)}\}\}$. The experiment was conducted using the source code from [11], which is built on the makefile from its fork [142].

	M	**S**	**A**	Cost	
				Metric 1	Metric 2
Individual computation (previous work [10, 11])	1,788,198	468,887	2,001,842	2,257,086	2,263,400
Strategic computation (this work, Section 6.4)	1,676,746	400,398	1,954,800	2,077,144	2,094,805
Reduction:	**6.23%**	**14.60%**	**2.35%**	**7.97%**	**7.45%**

6.5 Chapter Summary

In this chapter, we have proposed multiple practical techniques to reduce the number of field arithmetic operations required to evaluate ideal class group actions on a set element. Our first proposal is a batching strategy to efficiently evaluate public action systems. We have analyzed our strategy for action set sizes ranging between 2 and 58, where it shows a reduction of around 14% of field arithmetic operations. Secondly, we have proposed an improved constant-time algorithm that is merged with our batch strategy to evaluate a private set of action system. Our analysis indicates an improvement of roughly up to 8%, with the improved implementation of constant time CSIDH alone reducing the computation by around 4%.

Chapter 7

Concluding Remarks and Future Works

In this thesis, we explored some aspects of the field of cryptography in which *time* plays a crucial role in its security. To expand the scope of applications in this field, we extended an existing theoretical notion, introduced a new notion, and improved the efficiency of one of the few building blocks used to construct delay-based primitives.

7.1 Concluding Remarks

Our work for this thesis began by expanding on the notion of trapdoor VDF, which was initially defined in [156]. In Chapter 4, we introduced a new class of trapdoor verifiable delay functions (trapdoor-VDFs) that features a large ensemble of trapdoors for each VDF instance. Within this framework, the challenger randomly samples a secret trapdoor from the ensemble to construct a challenge. Using the secret trapdoor, the challenger can solve the challenge fast, whereas the solver, who does not know the trapdoor, must perform lengthy sequential computations, taking a long, predetermined period of time. The solver's computation is publicly verifiable, which does not require knowledge of the secret trapdoor. We also proposed a framework and a proof-of-concept instance for trapdoor-VDF. The proposed framework allows reuse of public parameters in constructions with costly public setups (e.g., isogeny-based constructions such as [33] and [72]). In short, our initial notion addresses the sender's (i.e., the challenger's) needs to release digital content after a predetermined delay while retaining the reusability of the underlying setup's parameters, which we expect to be beneficial in distributed systems.

We then further developed such notion and asked the question of the receiver's privacy, specifically in the context of digital content distribution. In particular, we addressed the scenario where clients (i.e., a receiver who may also act as a solver) are concerned about their interests in certain content being exposed. Hence, in Chapter 5, we introduced a new construct called ChronoCloak, intended to address the issue of premature exposure of sender's digital content and privacy-preserving of receiver's interaction. In the context of oblivious digital dissemination, ChronoCloak offers a viable integrated solution to mitigate premature disclosure. In contrast to its existing solutions, the proposed ChronoCloak is based on an integrated model and does not require a time server. Moreover, the receiver may outsource the computation (with public verification), but only the intended recipient will be able to retrieve the subset of secrets using the solution to the sender's challenge. We then proposed a definition of ChronoCloak's ideal functionality. We next presented a generic protocol that implements this functionality within the random oracle model. The protocol builds on any ChronoCloak instance proven secure under Definition 8, uplifting its security to a UC-secure implementation. In this regard, we also propose a secure Chrono-Cloak instance based on Definition 8, constructed using an oblivious transfer protocol and a trapdoor-VDF.

One of the discussions we had on ChronoCloak focused on its scalability. We noted that ChronoCloak is scalable in situations where the sender interacts with multiple receivers. For example, in our proposed construction, the sender shares a single challenge but must engage in multiple OT interactions, one for each receiver. In Chapter 6, we proposed a strategy of such scenario that is implemented within the CSIDH setting, e.g., CSIDH-based OT protocol as in [9]. As illustrated in Chapter 6, the benefit of our work extends to other constructs such as signature schemes [7, 21, 54, 71, 73] or zero-knowledge proofs [71, 20, 7, 59]. In particular, we proposed a new batching strategy that reduces the number of field arithmetic operations required to perform multiple executions of a protocol (or repeated actions within a protocol) within CSIDH settings. Our first batching strategy targets actions without security restrictions (that is, not considering side-channel threats), which reduces the computational costs of a public action set by roughly up to 14% compared to individual execution. Then, we proposed a new constant-time implementation of CSIDH, and a constant-time batching strategy. When combined, both reduce the computational costs of a secret action set by roughly up to 8% compared to the individual execution of current state-of-the-art constant-time CSIDH; whereas the new constant time by itself reduces the computations by approximately up to 4%. Lastly, to help the reader understand our work, we have included the source code for a proof-of-concept implementation, which is available at `https://github.com/ahzawia/Streamlining-CSIDH`.

7.2 Future Work

Looking back at our work, we can identify potential avenues for future research that could produce interesting results. Here are a few examples of possible future research.

Regarding our trapdoor-VDF, Chapter 4 provides a formal definition along with a proof-of-concept construction. However, our construction is not quantum secure, and to our knowledge, there are no trapdoor-VDFs that provide this security guarantee. The development of such a construction is of great interest.

In another aspect, scalability remains a significant challenge when adopting delay primitives, such as trapdoor-VDF and ChronoCloak, particularly in applications requiring high throughput. They struggle to meet such a requirement since they typically require lengthy sequential computations (and verifications) per challenge, which consumes substantial computational resources. As a result, they scale poorly as the size of the application increases. One possible solution is to introduce them with batching capabilities; an example would be to introduce trapdoor-VDF that performs lengthy sequential evaluation (and verification) for an entire batch of challenges rather than for each challenge.

Another possible solution to address the scalability issue is to propose efficient primitives based on a concept similar to delay encryption. Regarding our work in Chapter 5, to enhance ChronoCloak's scalability, multiple senders can encapsulate their challenges using a randomly generated session ID, similar to the delay encryption construct. However, to our knowledge, there is only one example of delay encryption, which is inefficient. Therefore, it would be interesting to explore a more efficient delay encryption construction.

As of now, in the private strategy discussed in Chapter 6, we execute actions using an action system *of size two* to avoid the complexity overhead associated with a larger system. However, it would be interesting to explore the execution with an action system of a size *larger than two*. In a constant-time setting, with actions drawn from the CTIDH-512 key space, the expected number of execution rounds (or isogeny evaluations) is at least 208, while the average number of rounds needed to correctly execute an action is about 180. This implies that on average 13.46% of the rounds are dummy rounds. In our current approach, we only took advantage of one-third of these, which leaves about two-thirds open to future improvement. Therefore, a larger action system, combined with a scheduling algorithm, may enhance the efficiency of the constant-time strategy.

References

[1] Simon Abelard. *Counting points on hyperelliptic curves in large characteristic : algorithms and complexity. (Comptage de points de courbes hyperelliptiques en grande caractéristique : algorithmes et complexité)*. PhD thesis, University of Lorraine, Nancy, France, 2018.

[2] Hamza Abusalah, Chethan Kamath, Karen Klein, Krzysztof Pietrzak, and Michael Walter. Reversible proofs of sequential work. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part II*, volume 11477 of *Lecture Notes in Computer Science*, pages 277–291. Springer, 2019.

[3] Navid Alamati, Luca De Feo, Hart Montgomery, and Sikhar Patranabis. Cryptographic group actions and applications. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part II*, volume 12492 of *Lecture Notes in Computer Science*, pages 411–439. Springer, 2020.

[4] Orestis Alpos, Zhipeng Wang, Alireza Kavousi, Sze Yiu Chau, Duc Viet Le, and Christian Cachin. DSKE: digital signature with key extraction. *IACR Cryptol. ePrint Arch.*, page 1753, 2022.

[5] Myrto Arapinis, Nikolaos Lamprou, and Thomas Zacharias. Astrolabous: A universally composable time-lock encryption scheme. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part II*, volume 13091 of *Lecture Notes in Computer Science*, pages 398–426. Springer, 2021.

[6] Arasu Arun, Joseph Bonneau, and Jeremy Clark. Short-lived zero-knowledge proofs and signatures. In Shweta Agrawal and Dongdai Lin, editors, *Advances in Cryptology - ASIACRYPT 2022 - 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5-9, 2022, Proceedings, Part III*, volume 13793 of *Lecture Notes in Computer Science*, pages 487–516. Springer, 2022.

[7] Shahla Atapoor, Karim Baghery, Daniele Cozzo, and Robi Pedersen. CSI-SharK: CSI-FiSh with sharing-friendly keys. In Leonie Simpson and Mir Ali Rezazadeh Baee, editors, *Information Security and Privacy - 28th Australasian Conference, ACISP 2023, Brisbane, QLD, Australia, July 5-7, 2023, Proceedings*, volume 13915 of *Lecture Notes in Computer Science*, pages 471–502. Springer, 2023.

[8] Saikrishna Badrinarayanan, Daniel Masny, and Pratyay Mukherjee. Efficient and tight oblivious transfer from PKE with tight multi-user security. In Giuseppe Ateniese and Daniele Venturi, editors, *Applied Cryptography and Network Security - 20th International Conference, ACNS 2022, Rome, Italy, June 20-23, 2022, Proceedings*, volume 13269 of *Lecture Notes in Computer Science*, pages 626–642. Springer, 2022.

[9] Saikrishna Badrinarayanan, Daniel Masny, Pratyay Mukherjee, Sikhar Patranabis, Srinivasan Raghuraman, and Pratik Sarkar. Round-optimal oblivious transfer and MPC from computational CSIDH. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *Public-Key Cryptography - PKC 2023 - 26th IACR International Conference on Practice and Theory of Public-Key Cryptography, Atlanta, GA, USA, May 7-10, 2023, Proceedings, Part I*, volume 13940 of *Lecture Notes in Computer Science*, pages 376–405. Springer, 2023.

[10] Gustavo Banegas, Daniel J. Bernstein, Fabio Campos, Tung Chou, Tanja Lange, Michael Meyer, Benjamin Smith, and Jana Sotáková. CTIDH: faster constant-time CSIDH. *IACR Cryptol. ePrint Arch.*, page 633, 2021.

[11] Gustavo Banegas, Daniel J. Bernstein, Fabio Campos, Tung Chou, Tanja Lange, Michael Meyer, Benjamin Smith, and Jana Sotáková. CTIDH: Faster constant-time CSIDH, software. https://ctidh.isogeny.org/software.html, 2024.

[12] Paulo Barreto, Glaucio Oliveira, and Waldyr Benits. Supersingular isogeny oblivious transfer. *IACR Cryptol. ePrint Arch.*, page 459, 2018.

[13] Paulo S. L. M. Barreto, Bernardo David, Rafael Dowsley, Kirill Morozov, and Anderson C. A. Nascimento. A framework for efficient adaptively secure composable oblivious transfer in the ROM. *CoRR*, abs/1710.08256, 2017.

[14] Paulo S. L. M. Barreto, Ben Lynn, and Michael Scott. On the selection of pairing-friendly groups. In Mitsuru Matsui and Robert J. Zuccherato, editors, *Selected Areas in Cryptography, 10th Annual International Workshop, SAC 2003, Ottawa, Canada, August 14-15, 2003, Revised Papers*, volume 3006 of *Lecture Notes in Computer Science*, pages 17–25. Springer, 2003.

[15] Carsten Baum, Bernardo David, Rafael Dowsley, Ravi Kishore, Jesper Buus Nielsen, and Sabine Oechsner. CRAFT: composable randomness beacons and output-independent abort MPC from time. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *Public-Key Cryptography - PKC 2023 - 26th IACR International Conference on Practice and Theory of Public-Key Cryptography, Atlanta, GA, USA, May 7-10, 2023, Proceedings, Part I*, volume 13940 of *Lecture Notes in Computer Science*, pages 439–470. Springer, 2023.

[16] Carsten Baum, Bernardo David, Rafael Dowsley, Jesper Buus Nielsen, and Sabine Oechsner. TARDIS: A foundation of time-lock puzzles in UC. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part III*, volume 12698 of *Lecture Notes in Computer Science*, pages 429–459. Springer, 2021.

[17] Mihir Bellare, Shai Halevi, Amit Sahai, and Salil P. Vadhan. Many-to-one trapdoor functions and their relation to public-key cryptosystems. *IACR Cryptol. ePrint Arch.*, page 19, 1998.

[18] Daniel J. Bernstein, Mike Hamburg, Anna Krasnova, and Tanja Lange. Elligator: elliptic-curve points indistinguishable from uniform random strings. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 967–980. ACM, 2013.

[19] Ward Beullens. CSI-FiSh: github repository available at. https://github.com/KULeuven-COSIC/CSI-FiSh/.

[20] Ward Beullens, Lucas Disson, Robi Pedersen, and Frederik Vercauteren. CSI-RAShi: Distributed key generation for CSIDH. In Jung Hee Cheon and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 12th International Workshop, PQCrypto 2021, Daejeon, South Korea, July 20-22, 2021, Proceedings*, volume 12841 of *Lecture Notes in Computer Science*, pages 257–276. Springer, 2021.

[21] Ward Beullens, Thorsten Kleinjung, and Frederik Vercauteren. CSI-FiSh: Efficient isogeny based signatures through class group computations. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part I*, volume 11921 of *Lecture Notes in Computer Science*, pages 227–247. Springer, 2019.

[22] Nir Bitansky, Shafi Goldwasser, Abhishek Jain, Omer Paneth, Vinod Vaikuntanathan, and Brent Waters. Time-lock puzzles from randomized encodings. In Madhu Sudan, editor, *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, Cambridge, MA, USA, January 14-16, 2016*, pages 345–356. ACM, 2016.

[23] Ian F Blake, Gadiel Seroussi, and Nigel P Smart. *Advances in Elliptic Curve Cryptography*. London Mathematical Society Lecture Note Series. Cambridge University Press, 2005.

[24] Alexander R. Block, Justin Holmgren, Alon Rosen, Ron D. Rothblum, and Pratik Soni. Time- and space-efficient arguments from groups of unknown order. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part IV*, volume 12828 of *Lecture Notes in Computer Science*, pages 123–152. Springer, 2021.

[25] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 757–788. Springer, 2018.

[26] Dan Boneh, Benedikt Bünz, and Ben Fisch. A survey of two verifiable delay functions using proof of exponentiation. *IACR Commun. Cryptol.*, 1(1):7, 2024.

[27] Dan Boneh and Matthew K. Franklin. Efficient generation of shared RSA keys. *J. ACM*, 48(4):702–722, 2001.

[28] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security,*

Gold Coast, Australia, December 9-13, 2001, Proceedings, volume 2248 of *Lecture Notes in Computer Science*, pages 514–532. Springer, 2001.

[29] Zvika Brakerski, Nico Döttling, Sanjam Garg, and Giulio Malavolta. Leveraging linear decryption: Rate-1 fully-homomorphic encryption and time-lock puzzles. In Dennis Hofheinz and Alon Rosen, editors, *Theory of Cryptography - 17th International Conference, TCC 2019, Nuremberg, Germany, December 1-5, 2019, Proceedings, Part II*, volume 11892 of *Lecture Notes in Computer Science*, pages 407–437. Springer, 2019.

[30] Pedro Branco, Nico Döttling, and Akshayaram Srinivasan. A framework for statistically sender private OT with optimal rate. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part I*, volume 14081 of *Lecture Notes in Computer Science*, pages 548–576. Springer, 2023.

[31] Johannes Buchmann and Hugh C. Williams. A key-exchange system based on imaginary quadratic fields. *J. Cryptol.*, 1(2):107–118, 1988.

[32] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent snarks from DARK compilers. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 677–706. Springer, 2020.

[33] Jeffrey Burdges and Luca De Feo. Delay encryption. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part I*, volume 12696 of *Lecture Notes in Computer Science*, pages 302–326. Springer, 2021.

[34] Jan Camenisch, Anja Lehmann, Gregory Neven, and Kai Samelin. UC-secure non-interactive public-key encryption. In *30th IEEE Computer Security Foundations Symposium, CSF 2017, Santa Barbara, CA, USA, August 21-25, 2017*, pages 217–233. IEEE Computer Society, 2017.

[35] Fabio Campos, Jorge Chávez-Saab, Jesús-Javier Chi-Domínguez, Michael Meyer, Krijn Reijnders, Francisco Rodríguez-Henríquez, Peter Schwabe, and Thom Wig-

gers. On the practicality of post-quantum TLS using large-parameter CSIDH. *IACR Cryptol. ePrint Arch.*, page 793, 2023.

[36] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 136–145. IEEE Computer Society, 2001.

[37] Ran Canetti, Eyal Kushilevitz, and Yehuda Lindell. On the limitations of universally composable two-party computation without set-up assumptions. *J. Cryptol.*, 19(2):135–167, 2006.

[38] Ran Canetti, Pratik Sarkar, and Xiao Wang. Efficient and round-optimal oblivious transfer and commitment with adaptive security. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part III*, volume 12493 of *Lecture Notes in Computer Science*, pages 277–308. Springer, 2020.

[39] Wouter Castryck and Thomas Decru. CSIDH on the surface. In Jintai Ding and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020, Paris, France, April 15-17, 2020, Proceedings*, volume 12100 of *Lecture Notes in Computer Science*, pages 111–129. Springer, 2020.

[40] Wouter Castryck and Thomas Decru. An efficient key recovery attack on SIDH. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part V*, volume 14008 of *Lecture Notes in Computer Science*, pages 423–447. Springer, 2023.

[41] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. CSIDH: an efficient post-quantum commutative group action. In Thomas Peyrin and Steven D. Galbraith, editors, *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part III*, volume 11274 of *Lecture Notes in Computer Science*, pages 395–427. Springer, 2018.

[42] Daniel Cervantes-Vázquez, Mathilde Chenu, Jesús-Javier Chi-Domínguez, Luca De Feo, Francisco Rodríguez-Henríquez, and Benjamin Smith. Stronger and faster side-channel protections for CSIDH. In Peter Schwabe and Nicolas Thériault, editors,

Progress in Cryptology - LATINCRYPT 2019 - 6th International Conference on Cryptology and Information Security in Latin America, Santiago de Chile, Chile, October 2-4, 2019, Proceedings, volume 11774 of *Lecture Notes in Computer Science*, pages 173–193. Springer, 2019.

[43] Jorge Chávez-Saab, Jesús-Javier Chi-Domínguez, Samuel Jaques, and Francisco Rodríguez-Henríquez. The SQALE of CSIDH: sublinear vélu quantum-resistant isogeny action with low exponents. *J. Cryptogr. Eng.*, 12(3):349–368, 2022.

[44] Jorge Chávez-Saab, Francisco Rodríguez-Henríquez, and Mehdi Tibouchi. Verifiable isogeny walks: Towards an isogeny-based postquantum VDF. In Riham AlTawy and Andreas Hülsing, editors, *Selected Areas in Cryptography - 28th International Conference, SAC 2021, Virtual Event, September 29 - October 1, 2021, Revised Selected Papers*, volume 13203 of *Lecture Notes in Computer Science*, pages 441–460. Springer, 2021.

[45] Hao Cheng, Georgios Fotiadis, Johann Großschädl, Peter Y. A. Ryan, and Peter B. Rønne. Batching CSIDH group actions using AVX-512. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(4):618–649, 2021.

[46] Seung Geol Choi, Jonathan Katz, Alex J. Malozemoff, and Vassilis Zikas. Efficient three-party computation from cut-and-choose. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 513–530. Springer, 2014.

[47] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, USA, 23-25 October 1995*, pages 41–50. IEEE Computer Society, 1995.

[48] Tung Chou and Claudio Orlandi. The simplest protocol for oblivious transfer. In Kristin E. Lauter and Francisco Rodríguez-Henríquez, editors, *Progress in Cryptology - LATINCRYPT 2015 - 4th International Conference on Cryptology and Information Security in Latin America, Guadalajara, Mexico, August 23-26, 2015, Proceedings*, volume 9230 of *Lecture Notes in Computer Science*, pages 40–58. Springer, 2015.

[49] Peter Chvojka, Tibor Jager, Daniel Slamanig, and Christoph Striecks. Versatile and sustainable timed-release encryption and sequential time-lock puzzles (extended abstract). In Elisa Bertino, Haya Schulmann, and Michael Waidner, editors, *Computer*

Security - ESORICS 2021 - 26th European Symposium on Research in Computer Security, Darmstadt, Germany, October 4-8, 2021, Proceedings, Part II, volume 12973 of *Lecture Notes in Computer Science*, pages 64–85. Springer, 2021.

[50] Dan Cline, Thaddeus Dryja, and Neha Narula. Clockwork: An exchange protocol for proofs of non front-running. The Stanford Blockchain Conference., 2020.

[51] Bram Cohen and Krzysztof Pietrzak. Simple proofs of sequential work. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 451–467. Springer, 2018.

[52] Henri Cohen. A course in computational algebraic number theory. *Graduate texts in Math.*, 138:88, 1993.

[53] David A Cox. Primes of the form x2+ ny2 John Wiley & Sons. *New York*, 1989.

[54] Daniele Cozzo and Nigel P. Smart. Sashimi: Cutting up csi-fish secret keys to produce an actively secure distributed signing protocol. In Jintai Ding and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020, Paris, France, April 15-17, 2020, Proceedings*, volume 12100 of *Lecture Notes in Computer Science*, pages 169–186. Springer, 2020.

[55] Giovanni Di Crescenzo, Rafail Ostrovsky, and Sivaramakrishnan Rajagopalan. Conditional oblivious transfer and timed-release encryption. In Jacques Stern, editor, *Advances in Cryptology - EUROCRYPT 99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, volume 1592 of *Lecture Notes in Computer Science*, pages 74–89. Springer, 1999.

[56] Bernardo David, Rafael Dowsley, and Anderson C. A. Nascimento. Universally composable oblivious transfer based on a variant of LPN. In Dimitris Gritzalis, Aggelos Kiayias, and Ioannis G. Askoxylakis, editors, *Cryptology and Network Security - 13th International Conference, CANS 2014, Heraklion, Crete, Greece, October 22-24, 2014. Proceedings*, volume 8813 of *Lecture Notes in Computer Science*, pages 143–158. Springer, 2014.

[57] Cyprien Delpech de Saint Guilhem, Péter Kutas, Christophe Petit, and Javier Silva. SÉTA: Supersingular encryption from torsion attacks. *IACR Cryptol. ePrint Arch.*, page 1291, 2019.

[58] Cyprien Delpech de Saint Guilhem, Emmanuela Orsini, Christophe Petit, and Nigel P. Smart. Semi-commutative masking: A framework for isogeny-based protocols, with an application to fully secure two-round isogeny-based OT. In Stephan Krenn, Haya Shulman, and Serge Vaudenay, editors, ., volume 12579 of *Lecture Notes in Computer Science*, pages 235–258. Springer, 2020.

[59] Cyprien Delpech de Saint Guilhem and Robi Pedersen. New proof systems and an OPRF from CSIDH. In Qiang Tang and Vanessa Teague, editors, *Public-Key Cryptography - PKC 2024 - 27th IACR International Conference on Practice and Theory of Public-Key Cryptography, Sydney, NSW, Australia, April 15-17, 2024, Proceedings, Part III*, volume 14603 of *Lecture Notes in Computer Science*, pages 217–251. Springer, 2024.

[60] Thomas Decru, Luciano Maino, and Antonio Sanso. Towards a quantum-resistant weak verifiable delay function. In Abdelrahaman Aly and Mehdi Tibouchi, editors, *Progress in Cryptology - LATINCRYPT 2023 - 8th International Conference on Cryptology and Information Security in Latin America, LATINCRYPT 2023, Quito, Ecuador, October 3-6, 2023, Proceedings*, volume 14168 of *Lecture Notes in Computer Science*, pages 149–168. Springer, 2023.

[61] Christina Delfs and Steven D. Galbraith. Computing isogenies between supersingular elliptic curves over $\mathfrak{O}_{\mathrm{p}}$. *Des. Codes Cryptogr.*, 78(2):425–440, 2016.

[62] Samuel Dobson and Steven D. Galbraith. Trustless groups of unknown order with hyperelliptic curves. *IACR Cryptol. ePrint Arch.*, page 196, 2020.

[63] Samuel Dobson, Steven D. Galbraith, and Benjamin Smith. Trustless unknown-order groups. *CoRR*, abs/2211.16128, 2022.

[64] Nico Döttling, Sanjam Garg, Mohammad Hajiabadi, Daniel Masny, and Daniel Wichs. Two-round oblivious transfer from CDH or LPN. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part II*, volume 12106 of *Lecture Notes in Computer Science*, pages 768–797. Springer, 2020.

[65] Nico Döttling, Sanjam Garg, Giulio Malavolta, and Prashant Nalini Vasudevan. Tight verifiable delay functions. In Clemente Galdi and Vladimir Kolesnikov, editors, *Security and Cryptography for Networks - 12th International Conference, SCN 2020, Amalfi, Italy, September 14-16, 2020, Proceedings*, volume 12238 of *Lecture Notes in Computer Science*, pages 65–84. Springer, 2020.

[66] Nico Döttling, Russell W. F. Lai, and Giulio Malavolta. Incremental proofs of sequential work. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part II*, volume 11477 of *Lecture Notes in Computer Science*, pages 292–323. Springer, 2019.

[67] Kirsten Eisenträger, Sean Hallgren, Kristin E. Lauter, Travis Morrison, and Christophe Petit. Supersingular isogeny graphs and endomorphism rings: reductions and solutions. *IACR Cryptol. ePrint Arch.*, 2018:371, 2018.

[68] Karim Eldefrawy, Sashidhar Jakkamsetti, Ben Terner, and Moti Yung. Standard model time-lock puzzles: Defining security and constructing via composition. *IACR Cryptol. ePrint Arch.*, page 439, 2023.

[69] Naomi Ephraim, Cody Freitag, Ilan Komargodski, and Rafael Pass. Continuous verifiable delay functions. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part III*, volume 12107 of *Lecture Notes in Computer Science*, pages 125–154. Springer, 2020.

[70] Shayan Eskandari, Seyedehmahsa Moosavi, and Jeremy Clark. Sok: Transparent dishonesty: Front-running attacks on blockchain. In Andrea Bracciali, Jeremy Clark, Federico Pintore, Peter B. Rønne, and Massimiliano Sala, editors, *Financial Cryptography and Data Security - FC 2019 International Workshops, VOTING and WTSC, St. Kitts, St. Kitts and Nevis, February 18-22, 2019, Revised Selected Papers*, volume 11599 of *Lecture Notes in Computer Science*, pages 170–189. Springer, 2019.

[71] Luca De Feo and Steven D. Galbraith. Seasign: Compact isogeny signatures from class group actions. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019,*

Proceedings, Part III, volume 11478 of *Lecture Notes in Computer Science*, pages 759–789. Springer, 2019.

[72] Luca De Feo, Simon Masson, Christophe Petit, and Antonio Sanso. Verifiable delay functions from supersingular isogenies and pairings. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part I*, volume 11921 of *Lecture Notes in Computer Science*, pages 248–277. Springer, 2019.

[73] Luca De Feo and Michael Meyer. Threshold schemes from isogeny assumptions. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *Public-Key Cryptography - PKC 2020 - 23rd IACR International Conference on Practice and Theory of Public-Key Cryptography, Edinburgh, UK, May 4-7, 2020, Proceedings, Part II*, volume 12111 of *Lecture Notes in Computer Science*, pages 187–212. Springer, 2020.

[74] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986.

[75] Tako Boris Fouotsa. SIDH with masked torsion point images. *IACR Cryptol. ePrint Arch.*, page 1054, 2022.

[76] Tako Boris Fouotsa, Tomoki Moriya, and Christophe Petit. M-SIDH and MD-SIDH: countering SIDH attacks by masking information. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part V*, volume 14008 of *Lecture Notes in Computer Science*, pages 282–309. Springer, 2023.

[77] Cody Freitag, Ilan Komargodski, Rafael Pass, and Naomi Sirkin. Non-malleable time-lock puzzles and applications. In Kobbi Nissim and Brent Waters, editors, *Theory of Cryptography - 19th International Conference, TCC 2021, Raleigh, NC, USA, November 8-11, 2021, Proceedings, Part III*, volume 13044 of *Lecture Notes in Computer Science*, pages 447–479. Springer, 2021.

[78] Cody Freitag, Ilan Komargodski, Rafael Pass, and Naomi Sirkin. Non-malleable time-lock puzzles and applications. In Kobbi Nissim and Brent Waters, editors,

Theory of Cryptography - 19th International Conference, TCC 2021, Raleigh, NC, USA, November 8-11, 2021, Proceedings, Part III, volume 13044 of *Lecture Notes in Computer Science*, pages 447–479. Springer, 2021.

[79] Steven D. Galbraith. *Mathematics of Public Key Cryptography*. Cambridge University Press, 2012.

[80] Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for cryptographers. *IACR Cryptol. ePrint Arch.*, 2006:165, 2006.

[81] Juan A. Garay, Philip D. MacKenzie, and Ke Yang. Strengthening zero-knowledge protocols using signatures. In Eli Biham, editor, *Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings*, volume 2656 of *Lecture Notes in Computer Science*, pages 177–194. Springer, 2003.

[82] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 467–476. ACM, 2013.

[83] Sanjam Garg and Akshayaram Srinivasan. Two-round multiparty secure computation from minimal assumptions. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 468–499. Springer, 2018.

[84] Ziya Alper Genç, Vincenzo Iovino, and Alfredo Rial. "The simplest protocol for oblivious transfer" revisited. *Inf. Process. Lett.*, 161:105975, 2020.

[85] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. Zkboo: Faster zero-knowledge for boolean circuits. In Thorsten Holz and Stefan Savage, editors, *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, pages 1069–1083. USENIX Association, 2016.

[86] James L Hafner and Kevin S McCurley. A rigorous subexponential algorithm for computation of class groups. *Journal of the American mathematical society*, 2(4):837–850, 1989.

[87] Darrel Hankerson, Alfred J Menezes, and Scott Vanstone. *Guide to elliptic curve cryptography*. Springer Science & Business Media, 2006.

[88] Carmit Hazay and Yehuda Lindell. *Efficient Secure Two-Party Protocols - Techniques and Constructions*. Information Security and Cryptography. Springer, 2010.

[89] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In David A. Wagner, editor, *Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings*, volume 5157 of *Lecture Notes in Computer Science*, pages 572–591. Springer, 2008.

[90] David Jao and Luca De Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In Bo-Yin Yang, editor, *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011, Taipei, Taiwan, November 29 - December 2, 2011. Proceedings*, volume 7071 of *Lecture Notes in Computer Science*, pages 19–34. Springer, 2011.

[91] David Jao and Vladimir Soukharev. A subexponential algorithm for evaluating large degree isogenies. In Guillaume Hanrot, François Morain, and Emmanuel Thomé, editors, *Algorithmic Number Theory, 9th International Symposium, ANTS-IX, Nancy, France, July 19-23, 2010. Proceedings*, volume 6197 of *Lecture Notes in Computer Science*, pages 219–233. Springer, 2010.

[92] Samuel Jaques, Hart Montgomery, Razvan Rosie, and Arnab Roy. Time-release cryptography from minimal circuit assumptions. In Avishek Adhikari, Ralf Küsters, and Bart Preneel, editors, *Progress in Cryptology - INDOCRYPT 2021 - 22nd International Conference on Cryptology in India, Jaipur, India, December 12-15, 2021, Proceedings*, volume 13143 of *Lecture Notes in Computer Science*, pages 584–606. Springer, 2021.

[93] Jonathan Katz, Julian Loss, and Jiayu Xu. On the security of time-lock puzzles and timed commitments. In Rafael Pass and Krzysztof Pietrzak, editors, *Theory of Cryptography - 18th International Conference, TCC 2020, Durham, NC, USA, November 16-19, 2020, Proceedings, Part III*, volume 12552 of *Lecture Notes in Computer Science*, pages 390–413. Springer, 2020.

[94] Alireza Kavousi, Aydin Abadi, and Philipp Jovanovic. Timed secret sharing. *IACR Cryptol. ePrint Arch.*, page 1024, 2023.

[95] Janis Klaise. Orders in quadratic imaginary fields of small class number, 2012. Undergraduate report, University of Warwick.

[96] Jodie Knapp and Elizabeth A. Quaglia. Fair and sound secret sharing from homomorphic time-lock puzzles. In Khoa Nguyen, Wenling Wu, Kwok-Yan Lam, and Huaxiong Wang, editors, *Provable and Practical Security - 14th International Conference, ProvSec 2020, Singapore, November 29 - December 1, 2020, Proceedings*, volume 12505 of *Lecture Notes in Computer Science*, pages 341–360. Springer, 2020.

[97] David Russell Kohel. *Endomorphism rings of elliptic curves over finite fields*. PhD thesis, University of California, Berkeley, 1996.

[98] James Kraft and Lawrence Washington. *An introduction to number theory with cryptography*. CRC press, 2018.

[99] Yi-Fu Lai, Steven D. Galbraith, and Cyprien Delpech de Saint Guilhem. Compact, efficient and uc-secure isogeny-based oblivious transfer, 2020.

[100] Yi-Fu Lai, Steven D. Galbraith, and Cyprien Delpech de Saint Guilhem. Compact, efficient and uc-secure isogeny-based oblivious transfer. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part I*, volume 12696 of *Lecture Notes in Computer Science*, pages 213–241. Springer, 2021.

[101] Kim Laine and Kristin E. Lauter. Time-memory trade-offs for index calculus in genus 3. *J. Math. Cryptol.*, 9(2):95–114, 2015.

[102] Claiborne G Latimer. Quaternion algebras. *Duke Mathematical Journal*, 15(2):357–366, 1948.

[103] Jonathan Lee. The security of groups of unknown order based on jacobians of hyperelliptic curves. *IACR Cryptol. ePrint Arch.*, page 289, 2020.

[104] Arjen K. Lenstra and Benjamin Wesolowski. Trustworthy public randomness with sloth, unicorn, and trx. *Int. J. Appl. Cryptogr.*, 3(4):330–343, 2017.

[105] Christopher Leonardi. A note on the ending elliptic curve in SIDH. *IACR Cryptol. ePrint Arch.*, 2020:262, 2020.

[106] Baiyu Li and Daniele Micciancio. Equational security proofs of oblivious transfer protocols. In Michel Abdalla and Ricardo Dahab, editors, *Public-Key Cryptography - PKC 2018 - 21st IACR International Conference on Practice and Theory of Public-Key Cryptography, Rio de Janeiro, Brazil, March 25-29, 2018, Proceedings, Part I*, volume 10769 of *Lecture Notes in Computer Science*, pages 527–553. Springer, 2018.

[107] Yehuda Lindell, editor. *Tutorials on the Foundations of Cryptography*. Springer International Publishing, 2017.

[108] Jia Liu, Tibor Jager, Saqib A. Kakvi, and Bogdan Warinschi. How to build time-lock encryption. *Des. Codes Cryptogr.*, 86(11):2549–2586, 2018.

[109] Yi Liu, Qi Wang, and Siu-Ming Yiu. Towards practical homomorphic time-lock puzzles: Applicability and verifiability. In Vijayalakshmi Atluri, Roberto Di Pietro, Christian Damsgaard Jensen, and Weizhi Meng, editors, *Computer Security - ESORICS 2022 - 27th European Symposium on Research in Computer Security, Copenhagen, Denmark, September 26-30, 2022, Proceedings, Part I*, volume 13554 of *Lecture Notes in Computer Science*, pages 424–443. Springer, 2022.

[110] Angelique Faye Loe, Liam Medley, Christian O'Connell, and Elizabeth A. Quaglia. A practical verifiable delay function and delay encryption scheme. *IACR Cryptol. ePrint Arch.*, page 1293, 2021.

[111] Angelique Faye Loe, Liam Medley, Christian O'Connell, and Elizabeth A. Quaglia. TIDE: A novel approach to constructing timed-release encryption. In Khoa Nguyen, Guomin Yang, Fuchun Guo, and Willy Susilo, editors, *Information Security and Privacy - 27th Australasian Conference, ACISP 2022, Wollongong, NSW, Australia, November 28-30, 2022, Proceedings*, volume 13494 of *Lecture Notes in Computer Science*, pages 244–264. Springer, 2022.

[112] Xu Ma, Lingling Xu, and Fangguo Zhang. Oblivious transfer with timed-release receiver's privacy. *J. Syst. Softw.*, 84(3):460–464, 2011.

[113] Mohammad Mahmoody, Tal Moran, and Salil P. Vadhan. Time-lock puzzles in the random oracle model. In Phillip Rogaway, editor, *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*, pages 39–50. Springer, 2011.

[114] Mohammad Mahmoody, Tal Moran, and Salil P. Vadhan. Publicly verifiable proofs of sequential work. In Robert D. Kleinberg, editor, *Innovations in Theoretical Computer*

Science, ITCS '13, Berkeley, CA, USA, January 9-12, 2013, pages 373–388. ACM, 2013.

[115] Mohammad Mahmoody, Caleb Smith, and David J. Wu. Can verifiable delay functions be based on random oracles? In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 83:1–83:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[116] Giulio Malavolta and Sri Aravinda Krishnan Thyagarajan. Homomorphic time-lock puzzles and applications. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part I*, volume 11692 of *Lecture Notes in Computer Science*, pages 620–649. Springer, 2019.

[117] Ueli M. Maurer. Abstract models of computation in cryptography. In Nigel P. Smart, editor, *Cryptography and Coding, 10th IMA International Conference, Cirencester, UK, December 19-21, 2005, Proceedings*, volume 3796 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2005.

[118] Timothy C. May. Timed release crypto. February 1993. http//www.hks.net/cpunks/cpunks-0/1460.html.

[119] Michael Meyer, Fabio Campos, and Steffen Reith. On lions and elligators: An efficient constant-time implementation of CSIDH. In Jintai Ding and Rainer Steinwandt, editors, *Post-Quantum Cryptography - 10th International Conference, PQCrypto 2019, Chongqing, China, May 8-10, 2019 Revised Selected Papers*, volume 11505 of *Lecture Notes in Computer Science*, pages 307–325. Springer, 2019.

[120] Michael Meyer and Steffen Reith. A faster way to the CSIDH. In Debrup Chakraborty and Tetsu Iwata, editors, *Progress in Cryptology - INDOCRYPT 2018 - 19th International Conference on Cryptology in India, New Delhi, India, December 9-12, 2018, Proceedings*, volume 11356 of *Lecture Notes in Computer Science*, pages 137–152. Springer, 2018.

[121] Daniele Micciancio and Jessica Sorrell. Simpler statistically sender private oblivious transfer from ideals of cyclotomic integers. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon,*

South Korea, December 7-11, 2020, Proceedings, Part II, volume 12492 of *Lecture Notes in Computer Science*, pages 381–407. Springer, 2020.

[122] Gwyneth Moreland. Class field theory for number fields and complex multiplication, 2016. University of Chicago REU Program Paper.

[123] Tomoki Moriya. Masked-degree SIDH. *IACR Cryptol. ePrint Arch.*, page 1019, 2022.

[124] Satoshi Nakamoto. Bitcoin: a peer-to-peer electronic cash system, 2009. http://dx.doi.org/10.2139/ssrn.3440802.

[125] Jesper Buus Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In Moti Yung, editor, *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, volume 2442 of *Lecture Notes in Computer Science*, pages 111–126. Springer, 2002.

[126] Hiroshi Onuki, Yusuke Aikawa, Tsutomu Yamazaki, and Tsuyoshi Takagi. (short paper) A faster constant-time algorithm of CSIDH keeping two points. In Nuttapong Attrapadung and Takeshi Yagi, editors, *Advances in Information and Computer Security - 14th International Workshop on Security, IWSEC 2019, Tokyo, Japan, August 28-30, 2019, Proceedings*, volume 11689 of *Lecture Notes in Computer Science*, pages 23–33. Springer, 2019.

[127] Christophe Petit and Kristin E. Lauter. Hard and easy problems for supersingular isogeny graphs. *IACR Cryptol. ePrint Arch.*, page 962, 2017.

[128] Krzysztof Pietrzak. Simple verifiable delay functions. In Avrim Blum, editor, *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA*, volume 124 of *LIPIcs*, pages 60:1–60:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[129] Michael O. Rabin. How to exchange secrets with oblivious transfer. *IACR Cryptol. ePrint Arch.*, page 187, 2005.

[130] Joost Renes. Computing isogenies between montgomery curves using the action of (0, 0). In Tanja Lange and Rainer Steinwandt, editors, *Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018, Fort Lauderdale, FL, USA, April 9-11, 2018, Proceedings*, volume 10786 of *Lecture Notes in Computer Science*, pages 229–247. Springer, 2018.

[131] Ronald L Rivest, Adi Shamir, and David A Wagner. Time-lock puzzles and timed-release crypto. 1996. https://people.csail.mit.edu/rivest/pubs/RSW96.pdf.

[132] Alexander Rostovtsev and Anton Stolbunov. Public-key cryptosystem based on isogenies. *IACR Cryptol. ePrint Arch.*, page 145, 2006.

[133] Lior Rotem. Simple and efficient batch verification techniques for verifiable delay functions. In *TCC*, volume 13044 of *LNCS*, pages 382–414. Springer, 2021.

[134] Lior Rotem, Gil Segev, and Ido Shahaf. Generic-group delay functions require hidden-order groups. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part III*, volume 12107 of *Lecture Notes in Computer Science*, pages 155–180. Springer, 2020.

[135] Hans-Georg Rück. A note on elliptic curves over finite fields. *Mathematics of Computation*, 49(179):301–304, 1987.

[136] René Schoof. Nonsingular plane cubic curves over finite fields. *J. Comb. Theory A*, 46(2):183–211, 1987.

[137] Barak Shani. A note on isogeny-based hybrid verifiable delay functions. *IACR Cryptol. ePrint Arch.*, page 205, 2019.

[138] Daniel Shumow. Isogenies of elliptic curves: A computational approach. *IACR Cryptol. ePrint Arch.*, 2009:522, 2009.

[139] Joseph H. Silverman. *Advanced Topics in the Arithmetic of Elliptic Curves*. Springer New York, NY, 1994.

[140] Joseph H Silverman. *The arithmetic of elliptic curves*, volume 106. Springer Science & Business Media, 2009.

[141] Michael A. Specter, Sunoo Park, and Matthew Green. Keyforge: Non-attributable email from forward-forgeable signatures. In Michael D. Bailey and Rachel Greenstadt, editors, *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 1755–1773. USENIX Association, 2021.

[142] David Stainton and Jacob Appelbaum. high-ctidh (Version fix_private_constructor_rng). https://git.xx.network/elixxir/high-ctidh/.

[143] Ian Stewart and David Tall. *Algebraic number theory and Fermat's last theorem*. CRC Press, 2015.

[144] Anton Stolbunov. Constructing public-key cryptographic schemes based on class group action on a set of isogenous elliptic curves. *Adv. Math. Commun.*, 4(2):215–235, 2010.

[145] Teik Guan Tan, Vishal Sharma, Zengpeng Li, Pawel Szalachowski, and Jianying Zhou. Zkbdf: A zkboo-based quantum-secure verifiable delay function with prover-secret. In Jianying Zhou, Lejla Batina, Zengpeng Li, Jingqiang Lin, Eleonora Losiouk, Suryadipta Majumdar, Daisuke Mashima, Weizhi Meng, Stjepan Picek, Mohammad Ashiqur Rahman, Jun Shao, Masaki Shimaoka, Ezekiel O. Soremekun, Chunhua Su, Je Sen Teh, Aleksei Udovenko, Cong Wang, Leo Yu Zhang, and Yury Zhauniarovich, editors, *Applied Cryptography and Network Security Workshops - ACNS 2023 Satellite Workshops, ADSC, AIBlock, AIHWS, AIoTS, CIMSS, Cloud S&P, SCI, SecMT, SiMLA, Kyoto, Japan, June 19-22, 2023, Proceedings*, volume 13907 of *Lecture Notes in Computer Science*, pages 530–550. Springer, 2023.

[146] John Tate. Endomorphisms of abelian varieties over finite fields. *Inventiones mathematicae*, 2(2):134–144, 1966.

[147] Steve Thakur. Constructing hidden order groups using genus three jacobians. *IACR Cryptol. ePrint Arch.*, page 348, 2020.

[148] Sri Aravinda Krishnan Thyagarajan, Adithya Bhat, Giulio Malavolta, Nico Döttling, Aniket Kate, and Dominique Schröder. Verifiable timed signatures made practical. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, pages 1733–1750. ACM, 2020.

[149] Jacques Vélu. Isogénies entre courbes elliptiques. *CR Acad. Sci. Paris, Séries A*, 273:305–347, 1971.

[150] Vanessa Vitse. Simple oblivious transfer protocols compatible with supersingular isogenies. In Johannes Buchmann, Abderrahmane Nitaj, and Tajje-eddine Rachidi, editors, *Progress in Cryptology - AFRICACRYPT 2019 - 11th International Conference on Cryptology in Africa, Rabat, Morocco, July 9-11, 2019, Proceedings*, volume 11627 of *Lecture Notes in Computer Science*, pages 56–78. Springer, 2019.

[151] Ulrich Vollmer. Asymptotically fast discrete logarithms in quadratic number fields. In Wieb Bosma, editor, *Algorithmic Number Theory, 4th International Symposium,*

ANTS-IV, Leiden, The Netherlands, July 2-7, 2000, Proceedings, volume 1838 of *Lecture Notes in Computer Science*, pages 581–594. Springer, 2000.

[152] Yohei Watanabe and Junji Shikata. Timed-release secret sharing scheme with information theoretic security. *CoRR*, abs/1401.5895, 2014.

[153] Yohei Watanabe and Junji Shikata. Information-theoretically secure timed-release secret sharing schemes. *J. Inf. Process.*, 24(4):680–689, 2016.

[154] Yohei Watanabe and Junji Shikata. Timed-release computational secret sharing and threshold encryption. *Des. Codes Cryptogr.*, 86(1):17–54, 2018.

[155] William C Waterhouse. Abelian varieties over finite fields. In *Annales scientifiques de l'École normale supérieure*, volume 2, pages 521–560, 1969.

[156] Benjamin Wesolowski. Efficient verifiable delay functions. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part III*, volume 11478 of *Lecture Notes in Computer Science*, pages 379–407. Springer, 2019.

[157] Lingling Xu, Fangguo Zhang, and Shaohua Tang. Timed-release oblivious transfer. *Secur. Commun. Networks*, 7(7):1138–1149, 2014.

[158] Ke Yuan, Ziwei Cheng, Keyan Chen, Bozhen Wang, Junyang Sun, Sufang Zhou, and Chunfu Jia. Multiple time servers timed-release encryption based on shamir secret sharing for EHR cloud system. *J. Cloud Comput.*, 13(1):116, 2024.

[159] Ahmed Zawia and M. Anwar Hasan. A new class of trapdoor verifiable delay functions. In Guy-Vincent Jourdan, Laurent Mounier, Carlisle M. Adams, Florence Sèdes, and Joaquín García-Alfaro, editors, *Foundations and Practice of Security - 15th International Symposium, FPS 2022, Ottawa, ON, Canada, December 12-14, 2022, Revised Selected Papers*, volume 13877 of *Lecture Notes in Computer Science*, pages 71–87. Springer, 2022.

[160] Ahmed Zawia and M. Anwar Hasan. Chronocloak: An integrated solution for mitigating premature disclosure in oblivious digital dissemination. In Nicky Mouha and Nick Nikiforakis, editors, *Information Security - 27th International Conference, ISC 2024, Arlington, VA, USA, October 23-25, 2024, Proceedings, Part I*, volume 15257 of *Lecture Notes in Computer Science*, pages 232–251. Springer, 2024.

[161] Ahmed Zawia and M. Anwar Hasan. Streamlining CSIDH: cost-effective strategies for group actions evaluation. In Nicky Mouha and Nick Nikiforakis, editors, *Information Security - 27th International Conference, ISC 2024, Arlington, VA, USA, October 23-25, 2024, Proceedings, Part II*, volume 15258 of *Lecture Notes in Computer Science*, pages 101–118. Springer, 2024.

[162] Xiaotong Zhou, Debiao He, Jianting Ning, Min Luo, and Xinyi Huang. Efficient construction of verifiable timed signatures and its application in scalable payments. *IEEE Trans. Inf. Forensics Secur.*, 18:5345–5358, 2023.