"Now you don't see it, now you do" Time-Lock Puzzles and Fair Exchange

Ivo Maffei

Somerville College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Trinity 2024

To my family

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my supervisor, Bill Roscoe, for his invaluable guidance and unwavering support throughout my DPhil. His willingness to always be available at a moment's notice has been truly appreciated. I would also like to thank Michael Goldsmith for agreeing to act as my formal supervisor in the final month of my DPhil.

I am eternally grateful to my family for their unconditional love and support during these long years. A heartfelt thanks goes to my partner and my friends in Oxford, whose laughter and companionship have been a constant source of joy and strength, especially during the most challenging periods.

Abstract

Time-Release Encryption (TRE) is a cryptographic scheme designed to conceal messages for a predetermined duration. Constructing practical TRE schemes without relying on external parties typically involves Time-Lock Puzzles (TLPs) – unparallelisable computational challenges with predictable solving times. The solution to a TLP reveals the hidden message. This thesis investigates the potential of TLPs in two key areas.

First, we address a gap in the literature by constructing an efficient, practical, and quantum-secure TLP. We build upon the widely used concept of repeated squaring in RSA moduli, but modify the approach by performing these operations modulo a prime number. To further enhance security, we chain together multiple exponentiations using random permutations. The resulting TLP retains most of the efficiency of existing repeated-squaring-based puzzles while offering stronger security guarantees, particularly against attacks from quantum computers.

Second, we apply TLPs to the problem of fair exchange. This problem involves two or more parties, each holding a secret item, and explores how they can exchange these items while ensuring that no one cheats. Traditionally, exchanging secrets without a trusted neutral party is considered impossible. However, the field of partial fairness aims to design protocols that achieve some probability of fairness in all situations. By leveraging TLPs, we overcome previous limitations and significantly improve the round efficiency of fair exchange protocols. Specifically, we present 2-party and multi-party protocols that achieve optimal fairness, meaning no shorter protocol can offer a higher chance of a successful and fair exchange.

Contents

1 Introduction

The concept of Timed-Release Cryptography was first introduced by Timothy May on the Cypherpunks mailing list in 1993 [114]. May's revolutionary idea was to "send a message to the future". He envisioned a network of escrow services where anyone could send a message, guaranteeing delivery to the intended recipient only at a specified point in the future.

The academic community was introduced to this concept in 1996 by Rivest, Shamir, and Wagner's seminal paper "Time-Lock Puzzles and Timed-Release Encryption" [137], which paved the way for all subsequent schemes. Their interpretation of May's idea is more akin to the digital equivalent to a lock with a timer: once locked, it can only be opened after the timer expires. As Rivest *et al.* showed, there are two main approaches to constructing such a lock: relying on external parties to track time and release a secret key when appropriate, or modeling wall-clock time with computation through Time-Lock Puzzles (TLPs), which once solved reveal the locked message.

TLPs are central to this thesis since they have the advantage of not relying on trust. Our study focuses on the power that the existence of TLPs provides. The thesis addresses two goals: Is the assumption of TLPs realistic, even in a world with quantum computers? How can we leverage this power?

We answer the first question by designing a quantum-safe TLP that does not sacrifice all the efficiency of puzzles based on classical assumptions, addressing the inefficiency [107] or impracticality [33] of other quantum-safe puzzles.

To tackle the second question, we consider the use of TLPs in the general setting of fair exchange, a famously impossible task without external entities to govern the exchange [128]. TLPs allow us to achieve partial fairness, where the probability of unfairness can be made arbitrarily small. We will also show how TLPs can achieve optimality, meaning no other cryptographic primitive can further improve the chances of a fair exchange.

This remarkable result follows an idea proposed by Roscoe and Ryan [139]: using TLPs to guarantee the delivery of information while taking advantage of its inaccessibility until the puzzle is solved. All partially-fair protocols in the literature follow a similar blueprint: the items to be exchanged are hidden and shuffled among useless ones, then each item is exchanged without revealing the important secrets. With proper shuffling, unfairness is only possible through a lucky guess.

While partially-fair protocols without TLPs exist, TLPs allow for simpler and more private shufflings, leading to better secret distributions. Although we demonstrate how to push partial fairness to its limits, the study of TLPs in protocols is far from complete. We hope the results presented here will spark further exciting research.

1.1 Our Contributions

There are three main contributions of this thesis: the design of an efficient quantum-secure TLP, and the design of optimally-fair 2-party and multi-party protocols. Additionally, we construct a commutative encryption scheme which satisfies stronger security requirements than is usually discussed in the literature.

In terms of quantum security, the current state of affairs regarding TLPs is unsatisfactory. As we will delve into in great detail in Section 2.1, the majority of

constructions presently available in the literature rely on the assumption that repeatedly squaring an element of an unknown order group is a sequential process. Specifically, given a random element $g \in G$, this assumption posits that computing g^{2^t} requires t consecutive steps unless the order of G is already known. By instantiating G as an RSA group $\mathbb{Z}_N^*$, determining the order of the group is equivalent to factoring N. Furthermore, the repeated squaring assumption has been shown to be well-founded in a classical setting, where it was reduced to the difficulty of factoring in certain abstract models [90, 140]. However, in a quantum world, Shor's algorithm can efficiently calculate the order of any abelian group, rendering these assumptions no longer tenable. We identify three approaches to construct quantum-secure TLPs: hash-based, randomised encodings, and known-order repeated squaring. The first approach, which solely utilizes hash functions is limited due to the computational effort required to solve the puzzle being equivalent to that needed to construct it [5]. As a result, this method is deemed inefficient. The availability of efficient constructions for randomized encodings is scarce and generally relies on non-standard assumptions such as indistinguishability obfuscation. The construction we will propose in Chapter 3 is the only candidate from the third class. More specifically, we leverage the simple fact that computing a cube root in a finite filed requires more sequential computation than performing a simple cubing. Thus, our TLP stands out as the most pragmatic option to attain quantum security while preserving the majority of efficiency inherent in classical puzzles. From a theoretical standpoint, our main contribution is the proof of the sequentiality of repeated squaring in a group of *known* order. Additionally, we provide an in-depth analysis of the integer modular exponentiation process, shedding light on its limitations.

Endowed with a robust TLP, we design fair-exchange protocols that achieve optimal round complexity. In the context of distributed computation, particularly multi-party computation, researchers have attained remarkable outcomes by proving that any function can be computed amongst multiple parties without compromising the privacy of their inputs. Nevertheless, the primary disadvantage of this area

lies in ensuring fairness. In general, it is inherently impossible to guarantee that all parties will receive the outcome of the computation. More specifically, it is impracticable for a group of parties to exchange items fairly, as there will always be means by which a dishonest majority can prevent some honest party from receiving their items. To address this limitation, research has focused on two approaches: weakening the adversarial model or relaxing the fairness guarantee. It has been demonstrated that for any $\varepsilon > 0$, there exists a protocol that is fair with probability $1 - \varepsilon$ [78, 18]. However, these protocols are only applicable to exchanges where the items meet specific criteria. Additionally, the round complexity is impractical, being linear in relation to the size of the set of all possible items and exponential concerning the number of parties involved.[1] By leveraging TLPs, we have designed protocols that overcome these limitations. Specifically, our fair-exchange protocols are applicable to any scenario, and their round complexity is optimal, i.e., we prove that no shorter protocol can achieve the same probability of fairness. To design these protocols while maintaining relatively efficient computational and message complexity, we introduce a new primitive called *commutative blinding*. This is a variation of commutative encryption that meets more stringent security definitions. We present constructions that achieve these novel security definitions and rely on standard classical or post-quantum assumptions.

1.2 Thesis Structure

The thesis begins with an overview of different methods used to construct Time-Lock Puzzles (TLPs) in Chapter 2. Additionally, we classify all constructions based on their underlying assumptions for guaranteeing sequentiality. The chapter also covers related works in fair exchange and multi-party computations.

Chapter 3 presents our novel TLP construction based on cube root extraction, analyzing this operation and proving its sequentiality in the strongly algebraic group model. We show how to use pseudo-random permutations to chain together

[1] See Section 2.2 for precise quantification.

cube root extractions efficiently, resulting in a TLP that combines the strengths of hash-chain puzzles and puzzles based on repeated squaring. This construction achieves strong security guarantees, even against quantum adversaries, without sacrificing the efficiency of the latter type. The underlying ideas of this chapter have been published on arXiv as [104].

Chapter 4 demonstrates three methods for constructing secure commutative encryption that meets the requirements for instantiating the fair exchange protocols in the following chapters. We name this enhanced primitive *commutative blinding*. These constructions include one based on ElGamal, another based on a fully-homomorphic encryption scheme, and a general framework.

Chapter 5 discusses how to leverage TLPs to design a 2-party partially-fair exchange protocol, providing the protocol and a security proof against covert adversaries, entirely thanks to the TLP. The chapter demonstrates the protocol's optimality, as no shorter protocol can achieve better fairness. It also explores the scenario of repeated exchanges and outlines a construction based on nested oblivious transfers to enhance flexibility. The 2-party protocol using commutative blinding and its optimality were first presented in [106].

Chapter 6 extends the ideas from Chapter 5 to the multi-party context, designing a fair exchange protocol that accommodates any exchange of secrets among an arbitrary number of parties. In contrast to the 2-party setting, the natural notion of optimality leads to complex shufflings of secrets, necessitating a more flexible protocol. The chapter concludes with a high-level analysis of all partially-fair protocols based on similar underlying ideas, demonstrating that optimality can be achieved with our protocol. This protocol and analysis were first presented in [105].

2

Background and Related Work

Contents

In this chapter, we present an overview of the current state of the art for both Time-Lock Puzzles and Fair Exchange protocols. Moreover, we provide all the background knowledge, definitions and classical results that are required across different chapters of this thesis.

2.1 Time-Lock Puzzles

As previously mentioned, the purpose of Timed-Release Encryption (TRE) is to conceal a message for a predetermined period. When May introduced this concept

in 1993 [114], he envisioned a network of trusted parties that would release secret keys at specified times, allowing the recipient to decrypt the message only after receiving the corresponding key. Rivest *et al.* [137] formalised this concept in 1996 and proposed Time-Lock Puzzles (TLPs) as a method to implement TRE without relying on trusted parties.

The fundamental concept behind all TLPs is that Alice, the encrypter, can generate a puzzle that Bob, the decrypter, must solve before he can decrypt Alice's message, and analysis has shown that this process will take a predetermined time. However, TLPs lack the precision and granularity of Trusted Third Party (TTP)-based TRE, as they only ensure a "delay period" between encryption and decryption. Consequently, the receiver may not access the message immediately after the sender's designated time.

We refer to the time Bob must invest to solve the puzzle as the *delay* induced by the puzzle. It is also useful to discuss the ratio between the time Bob spends solving the puzzle and the time Alice requires to generate it, which we term the *delay factor*. Ideally, we aim to use puzzles with high delay factors, which we will call *efficient*. That is, in an efficient puzzle, Bob performs much more work than Alice.

Table 2.1 summarizes known TLPs and other relevant time-related primitives, such as Verifiable Delay Functions, Proofs of Sequential Work, and Timed Commitments.

In this section, we analyse these constructions by highlighting their advantages and drawbacks. We will demonstrate that no practical, efficient, and quantum-secure construction is known beyond the work presented in this thesis. Specifically, only the first three assumptions in Table 2.1 lead to practical constructions that can be implemented on everyday hardware. However, instantiations with random oracles are inefficient, and the order of an abelian group can be computed efficiently using quantum algorithms. We assert that this leaves constructions based on repeated squaring in a group of known order as the only valid candidate.

Assumption	Construction	Literature
Repeated squarings (unknown order group)	RSW	[137] [109] [112] [90] [31] [16] [100] [99] [150] [98] [30] [68] [13] [82]
	RSW chaining	[1] [133] [157] [61]
	Large exponent RSA	[86] [42]
Repeated squarings (known order group)	Root extraction	[138] [54] [87]
	Root extraction chaining	**this thesis** [95] [29] [149] [92]
Random oracles	Hash chains	[132] [107] [8] [154]
	Memory-bound functions	[53] [3] [55] [131] [102]
	Graph labelling	[108] [47]
	Lattice-based	[94]
Sequential language	Randomised encodings	[23] [146]
Ad-hoc	Subset sum	[152]
Ad-hoc	Discrete logarithm	[110] [111]
Ad-hoc	Isogeny walks	[33] [164] [41]

Table 2.1: Summary of different TLPs and related delay-inducing primitives. Underlined citations indicate works on TLPs as opposed to other primitives.

Furthermore, chaining constructions are required to ensure security for moderate and long delays. Therefore, our work represents the only TLP that satisfies all these requirements. We will now describe all the constructions in Table 2.1, deferring a more in-depth comparison of our TLP until after we detail its construction.

2.1.1 Repeated Squaring in a Group of Unknown Order

As Table 2.1 illustrates, most TLPs in the literature closely resemble the original RSW puzzle proposed by Rivest *et al.* [137]. In this scheme, Alice selects an RSA modulus $N = pq$, publishes N, and keeps p and q secret. To encrypt a message m with a delay parameter T, Alice computes $c = m + r^{2^T} \mod N$, where r is a random element in $\mathbb{Z}_N$.[1]

Alice, knowing p and q, can efficiently compute $r^{2^T} \mod N$ by raising r to the power of $\alpha = 2^T \mod \phi(N)$. Bob, given the ciphertext (c, r, N), must retrieve m by repeatedly squaring r for T iterations. This TLP's popularity stems from Alice's ability to easily increase the delay parameter T without incurring significant computational costs.

The underlying assumption is that repeated squaring in a group of unknown order is inherently sequential, meaning Bob is forced to perform T squarings sequentially due to not knowing the factorisation of N. While unproven in the standard model, this assumption has been linked to the RSA problem in theoretical models [90, 140]. Rotem *et al.* [141] also showed that hidden-order groups are somewhat necessary for TLPs, although Chapter 3 will demonstrate that their statement is not strong enough to exclude less efficient or slightly limited puzzles.

Numerous researchers have designed schemes similar to the RSW puzzle, adding extra features. For example, Mao [112] designed a proof for the puzzle's correct construction, Loe *et al.* [99] added some form of integrity, while Malavolta *et al.* [109] proposed the first homomorphic construction. Other research has focused

[1] Rivest *et al.* actually propose encrypting m using a symmetric cipher with a random key k, then computing $c = k + r^{2^T} \mod N$. This optimisation does not affect the overall security of the TLP, assuming the symmetric cipher is secure.

on translating the RSW puzzle to different groups of unknown order, such as class groups [157, 150], groups from integral recurrence relations [82], or Pell conics [13].

A notable deviation from the RSW puzzle is the scheme proposed by Jerschow and Mauve [86], which can be viewed as a standard RSA scheme with slow encryption. Alice runs the RSA setup to obtain $(N = pq, e, d, \phi(N))$, computes $r = 2^T \mod \phi(N)$ and $\hat{e} = 2^T + \phi(N) - r + e$, and constructs a puzzle with solution m by computing $c = m^d \mod N$. Bob receives $(c, \hat{e})$ and solves the puzzle by computing $c^{\hat{e}} \mod N$, which requires at least T squarings modulo N. This TLP can also be used to delay RSA signature verifications and encryptions, but relies on the same assumption as the RSW puzzle.

In 2018, Boneh *et al.* [29] introduced Verifiable Delay Functions (VDFs): functions slow to compute but fast to verify. The similarities with TLPs led to the exploration of using RSW in VDFs. Pietrzak [133] and Wesolowski [157] concurrently proposed using RSW for delay generation in VDFs, with the key contribution of providing non-interactive proofs of computation correctness. Moreover, Wesolowski [157] propose a method to "chain" RSW puzzles using hash functions, although this method cannot be directly applied to TLPs. The next chapter will discuss how to securely chain repeated squaring puzzles using pseudo-random permutations.

Compared to our work, all these RSW-based constructions are more efficient, i.e., they obtain higher delay factors for long delays. On the other hand, they are all insecure against quantum computation.

2.1.2 Repeated Squaring in a Group of Known Order

The first use of repeated squaring in a group of *known* order can be traced back to the seminal work of Dwork *et al.* [55] in 1993. They proposed computing square roots modulo a prime as a method to combat junk email. This idea was later employed by Jerschow and Mauve [87] to construct puzzles to mitigate denial-of-service attacks. In these applications, repeated squaring acts as a "computational price" that users pay to access a service.

It was not until 2017 that this idea was suggested in the context of TLPs, mentioned in a footnote by Roscoe [138] with the conjectured motivation of avoiding quantum vulnerabilities. The next chapter will fully develop the concept of constructing a TLP by combining this technique with random oracles.

In 2015, Lenstra and Wesolowski [95] demonstrated how to use square-root extraction to construct a "slow hash function" they named Sloth. Their approach involved alternating square root extraction with non-algebraic permutations. This idea was later revisited by Boneh *et al.* [29] in developing (weak) VDFs (Sloth++), and more recently by Souvik [149]. Concurrently to our work, a report from the Ethereum Foundation [92] proposed the use of cube-root extraction in conjunction with weak algebraic permutations to instantiate a VDF. This represents to most similar construction in the literature, but it was recently shown insecure by Biryukov *et al.* [22]. We remark that these precomputation-based attacks do not apply to our puzzle as they take advantage of the small modulus used in MinRoot. From a theoretical standpoint, as we will see in Chapter 3, precomputation attacks are equivalent to computing the discrete logarithm and the work of Biryukov *et al.* is no exception.[2] The 256-bit modulus used in MinRoot is insufficiently large to provide classical security against the discrete logarithm problem.[3]

In the next chapter, we will present our own variant of this approach, utilising cubing and incorporating secure non-algebraic pseudo-random permutations between cubing operations.

2.1.3 Random Oracles

The first formal proposal for using hash chains in TLPs was made by Pieprzyk in 1999 [132]. This straightforward TLP leverages verifiable secret sharing to be split among multiple parties. The core mechanism involves Alice selecting a random vector r and computing the secret $s = \mathsf{H}(\dots \mathsf{H}(r) \dots)$ by repeatedly applying a

[2]The similarities to computing the discrete logarithm were noted by the authors of [22] as well.

[3]NIST recommends using moduli of at least 2048 bits for Diffie-Hellman exchanges [14].

hash function H to r, T times. Bob receives r and can obtain s through the same computation. If Alice has access to parallelisation, she can compute n "chains" in parallel, obtaining pairs (r_i, h_i). These can be linked into a single puzzle $(r_1, h_1 \oplus r_2, \ldots, h_{n-1} \oplus r_n)$ with secret h_n [107].

Hash chains offer the advantage that the assumption of sequential chained hashing is likely weaker than that of repeated squaring. Security can be proven in the random oracle model [107]. However, the main drawback is that Alice spends almost as much time encrypting as Bob spends decrypting, unlike the RSW puzzle where Bob's computation is exponentially more complex than Alice's.

In 2011, Mahmoody *et al.* [107] investigated TLPs using hash functions and concluded that hash-based puzzles cannot guarantee a large gap between puzzle generation and solution time. They proved that if puzzle generation requires n hash computations, whether in parallel or not, a parallel algorithm can solve the puzzle with only n sequential hash computations. While not entirely ruling out hash-based TLPs, this result suggests that good TLPs are unlikely to be solely based on them. They proposed an optimal puzzle where generation involves n parallel hashes, but solving requires n sequential hashes. The puzzle works as follows: Alice picks n random seeds $x_1, \ldots, x_n$ and delays a secret s by sending Bob $(x_1, x_2 \oplus \mathsf{H}(x_1), \ldots, x_n \oplus \mathsf{H}(x_{n-1}), s \oplus \mathsf{H}(x_n))$.

Afshar *et al.* [5] recently extended these results to a post-quantum setting, demonstrating the impossibility of "efficient" TLPs based on random oracles when one or both parties can perform quantum computation. However, they showed that Mahmoody *et al.*'s puzzle remains secure against quantum solvers, requiring at least n sequential hashes even with quantum access to the random oracle. This supports the quantum-safety of random oracle-based TLPs, albeit at the cost of more expensive puzzle generation.

Another application of random oracles in time-related primitives is proofs of sequential work. Mahmoody *et al.* [108] introduced a proof of sequential work (PoSW) based on labelling a directed acyclic graph, where each vertex's label is generated

by hashing the labels of its neighbours. The graph's depth enforces sequential computation. Cohen and Pietrzak [47] refined this by using a tree structure to reduce memory requirements.

Lai and Malavolta [94] proposed a novel approach using a lattice-based permutation to instantiate a random oracle. Their motivation was to generate proofs of correctness in VDFs. While they prove that their permutation is "hash-like", this construction is less suitable for TLPs due to the easy parallelisation of matrix-vector multiplication.

In 2003, Dwork *et al.* [53] argued that memory access speed disparities are smaller than CPU speed differences, making puzzles based on memory-intensive functions more balanced.[4] Thus, they introduce the concept of "memory-bound functions". Their puzzle involved finding a path in a random array such that the hash of the last step has a specific number of trailing zeros. The chain of hashes generated heavily depends on the given array, thus forcing memory reads due to cache misses. Lysyanskaya and Segal [102] adapted this for time-lock purposes, making each puzzle solution unique. Although the delay is determined by the path length, parallel exploration of multiple paths is still possible. Thus access to parallelisation is a considerable advantage, rendering this technique less desirable for constructing TLPs. However, combining the idea of memory-bound functions with Mahmoody *et al.*'s construction could yield viable TLPs.

Abadi *et al.* [3] also explored memory-bound functions, suggesting using a function with a small domain to make repeated computation of its inverse best achieved through table lookups and memory reads. A puzzle is constructed by iteratively applying the function, requiring, on average, T memory reads to solve.

2.1.4 Randomised Encodings

The literature presents a few constructions that primarily rely on the existence of indistinguishability obfuscation (iO). While these constructions are not practical

[4] We should note that this argument has not been deeply analysed and the advent of "system on a chip" architectures might challenge it.

at the time of writing, they could represent the future of TLPs. At a high level, iO allows masking a circuit so that circuits computing the same function are indistinguishable. This is a powerful tool, as it would enable trivial slow circuits to be indistinguishable from genuinely un-parallelisable slow circuits.

Bitansky *et al.* [23] were the first to apply this idea to TLPs, constructing them from randomised encodings. A randomised encoding is essentially an indistinguishability obfuscator; that is, a randomised encoding $\widehat{C(x)}$ of a circuit C and input x is indistinguishable from the encoding of a dummy circuit hard-coded to output $C(x)$. Assuming the existence of a non-parallelising language and randomised encodings, one can construct a TLP by encoding a "dummy" circuit that "waits" for a delay period before outputting the solution. Notably, iO is only required for "efficient" TLPs, where puzzle generation is much faster than solving the puzzle. Non-efficient TLPs can be constructed using non-succinct randomised encodings such as Yao's garbled circuits [163].

Ameri *et al.* [7] revisited Bitansky *et al.*'s construction and adapted it to memory-bound functions. Using randomised encodings and assuming the existence of memory-hard problems, they constructed a puzzle with a guaranteed lower bound on memory requirements for a solution. While not directly constructing TLPs, their work can be seen as the memory-hard equivalent of Bitansky *et al.*'s. Combining the two, one could design TLPs by assuming a lower bound on memory read time, similar to research on memory-hard functions discussed earlier.

2.1.5 Subset Sum

In 2007, Tritilanunt *et al.* [152] designed a client-puzzle based on the subset sum problem. To the best of our knowledge, this is the only subset sum-based puzzle. Their construction relies on the idea that solving the subset sum problem is most efficiently done by constructing a lattice basis and performing LLL-reduction, a relatively sequential process. The puzzle involves generating weights by repeatedly hashing an initial random weight and selecting a random subset whose sum forms

the puzzle instance. The solver then uses LLL reduction to find the matching subset.

This puzzle could theoretically be adapted into a TLP by constructing a set of weights where the subset sum problem always has a unique solution. However, such sets can become quite large due to the size of the largest element being $\Omega\left(\frac{2^n}{\sqrt{n}}\right)$ [147]. Furthermore, the assumption of sequentiality in finding a solution is less established than those based on repeated squaring. Therefore, we are not aware of any concrete TLP based on Tritilanunt *et al.*'s work.

2.1.6 Discrete Logarithm

In 1999, Mao [110, 111] proposed the first TLP scheme to deviate from the RSW construction. Alice selects an RSA modulus $N = pq$ with Jacobi symbol $\left(\frac{-1}{N}\right) = 1$ and a random element e of hidden order t satisfying $\left(\frac{e}{N}\right) = -1$. Alice encrypts her message m using RSA with modulus N. Bob, given the ciphertext, N, and e, solves the puzzle by computing $t = \log_e 1 \mod N$. With t, Bob can factor N using $(e^{t/2} - 1)(e^{t/2} + 1) = 0 \mod N$ and decrypt the RSA ciphertext.

Mao's motivation was to enable Alice to efficiently prove the size of t to Bob without revealing it. However, the main issue is that computing the discrete logarithm is not inherently sequential. While small exponents can be easily extracted by exponentiation, parallelisation can generally aid Bob in solving the puzzle.

Loe *et al.* [100] proposed a similar construction in 2022, factoring the RSA modulus using $(x_2^t - x_1)(x_2^t + x_1) = 0 \mod N$, where x_1 and x_2 are given to Bob. However, the delay is generated by computing $x_2^t \mod N$, relying on the sequentiality of repeated squaring in an unknown order group.

Probably due to the potential for parallelisation in computing discrete logarithms, no other constructions resemble Mao's or rely heavily on the discrete logarithm problem.

2.1.7 Isogeny Walks

Without delving into the technical details of elliptic curves and isogenies, we will attempt to explain the underlying construction of TLPs based on isogeny graphs. An elliptic curve can be conceptualised as a set of points in a finite field. An isogeny is a map between elliptic curves that preserves certain properties, including supersingularity. By identifying isomorphic elliptic curves as a single vertex, an isogeny graph can be constructed by linking vertices where specific classes of isogenies exist.

Let $\phi : E \to E'$ be an isogeny between two elliptic curves, and let $e(\_, \_)$ be the Weil bilinear pairing.[5] There exists another unique isogeny $\hat{\phi} : E' \to E$, called the *dual* of ϕ, such that $e(\phi(P), Q) = e(P, \hat{\phi}(Q))$ for any two points P, Q.

Burdges and De Feo [33] utilise this to design a puzzle. A random elliptic curve E is selected, and a walk of length t is performed in the isogeny graph to reach another curve E'. The puzzle consists of the isogeny $\phi : E \to E'$, the curves E and E', a random point $Q \in E'$, and rP, where r is an integer and $P \in E$. Solving the puzzle requires tracing the walk back from E' to E to compute $\hat{\phi}(Q)$. The "shared secret" obtained by both the puzzle solver and generator is $e(\phi(P), Q)^r = e(rP, \hat{\phi}(Q))$, where the left-hand side can be computed by the generator and the right-hand side by the solver. This secret can be used to encrypt a message, effectively delaying its decryption.

Sequentiality is ensured by the assumption that, given the long walk ϕ, one must retrace all t steps to obtain $\hat{\phi}$. Concretely, Burdges and De Feo propose identifying each vertex of the graph with a value $\alpha \in \mathbb{F}_{p^2}$. Each step in the graph results in a new vertex with the value $(\alpha + \sqrt{\alpha^2 - 1})^2$.[6] The final walk $\phi : E \to E'$ can be represented by a list of encountered α values. To reverse the path, the

[5] The specifics of the pairing are not crucial here. We only require the property $e(\phi(P), Q)^r = e(rP, \hat{\phi}(Q))$, which will be explained shortly.

[6] The choice of a root for $\sqrt{\alpha^2 - 1}$ depends on the specific graph being traversed.

single-step isogeny $\phi_\alpha(X : Z)$ has a dual $\hat{\phi}_\alpha(X : Z) = ((X + Z)^2 : 4\alpha XZ)$. Therefore, computing $\hat{\phi}(Q)$ essentially involves t sequential squarings and $2t$ sequential multiplications.

While isogeny-based TLPs hold promise for quantum resistance, further research is needed. Puzzle generation is inefficient, and there are no theoretical guarantees that inverting the walk as described is optimal. Additionally, the authors of [33] note that a 1-hour delay would require a path length on the order of 10^{10}, resulting in a puzzle needing roughly 12 TiB of storage.

2.1.8 TLPs in a Post-Quantum World

Designing TLPs that can withstand the advent of quantum computation is crucial. In this regard, the current state of the art is unsatisfactory. Most TLP constructions rely on repeated squaring in a group of unknown order, a process vulnerable to Shor's algorithm, which computes the order of any abelian group. While changing the underlying group with each puzzle could mitigate this issue, it would likely sacrifice efficiency. As detailed in the next chapter, the repeated squaring process can also be compromised using an oracle for discrete logarithms, although the efficiency of such an oracle remains an open question.

In contrast, constructions based on random oracles are secure, as proven by Afshar *et al.* [5]. Whether the random oracle is instantiated with standard hash functions, memory-hard functions, or lattice-based hashes is irrelevant to the security proof. However, care must be taken to ensure the instantiation is secure, and to the best of our knowledge, the mentioned constructions would likely be secure against quantum computation.

The security of TLPs based on less common assumptions, such as subset sum, or isogenies, is uncertain. While the isogeny Diffie-Hellman-like protocol (SIDH) was recently proven insecure [38], isogeny-based constructions may still be viable, as claimed by Chen and Zhang [41].

Bitansky *et al.*'s TLP, based on randomized encodings, lacks concrete efficient instantiations. Therefore, its security in a post-quantum world remains unclear. However, the inefficient construction based on garbled circuits can be built using oblivious transfer and standard cryptography, potentially allowing for instantiation with quantum-secure primitives.

2.2 Fair Exchange

A (multi-party) fair exchange protocol enables two (or more) parties to exchange secrets fairly without mutual trust. Given its practical significance, this problem has been extensively studied in various contexts.

Informally, the fair exchange problem involves parties exchanging digital items while guaranteeing that everyone receives what they want. Let $P_1, \ldots, P_k$ be the parties involved, we informally define *fairness* with the following statement: if an honest party P_i discloses any useful information about their secret for P_j, then P_i learns all secrets meant for them. This fairness requirement is the strongest possible: if an honest party P_i discloses any information, they are guaranteed to receive everything they expect.

The most straightforward approach to fair exchange is using a trusted mediator who would collect and re-distribute the secrets. However, this is not ideal, as the mediator becomes a security and computational bottleneck. Franklin and Reiter [66] sought to mitigate these issues by introducing the "semi-trusted third party" concept, where the third party may misbehave but cannot collude with other parties. Their protocol ensures fairness against a malicious semi-trusted third party if both parties are honest. Franklin and Tsudik [65] soon extended this idea to the multi-party context, proposing a protocol guaranteeing fairness with at most one dishonest party. The protocol was later improved by Mukhamedov *et al.* [120].

Despite improvements with semi-trusted neutral parties, relying on an online entity remains a computational bottleneck and often an unrealistic assumption. To address this, researchers designed "optimistic" protocols where the TNP is offline

and only acts if parties misbehave. Asokan *et al.* [9] first formalised this concept in multi-party fair exchange in 1996. Subsequently, optimistic protocols gained popularity (e.g. [64, 91, 93, 97]).

In the absence of trust, fair exchange was first shown to be impossible by Even and Yacobi [62] in 1980, who demonstrated the impossibility of exchanging signatures. Six years later, Cleve [46] proved fair coin flipping impossible due to adversaries introducing bias by aborting the protocol at strategic times. However, the first direct proof of fair exchange's impossibility came in 1999 from Pagnia and Gärtner [128], who showed that a fair exchange protocol would imply a fault-tolerant consensus protocol. Garbinato and Rickebusch [70] examined the multi-party setting with partial trust, demonstrating that fairness can be achieved with trusted neutral parties (TNPs) if and only if all honest parties can reach a majority of TNPs without interference from corrupted parties. More generally, assuming secure channels, a complete network topology, and the presence of a broadcast channel, fairness can only be guaranteed if there is a majority of honest parties [75]. Alternatively, fairness can be ensured if malicious parties are compelled to follow the protocol and cannot abort the communication. In this scenario, they are only permitted to perform additional computations to attempt to uncover secrets not intended for them.

These strong impossibility results lead to limited research towards designing fair exchange without trusted entities. However, a few attempts rely on other parties' honesty. For instance, Zhang *et al.* [165] proposed a protocol using secret sharing, guaranteeing fairness if most parties are honest. Avoine and Vaudenay [11] studied a different approach, where two parties fairly exchange secrets assuming the presence of external passive participants whose majority is honest. Later, Avoine *et al.* [10] proposed a model where each party has a tamper-proof "security module", with malicious parties controlling their connections but not the modules' inner workings. Their protocol achieves fairness with an honest majority.

A more pragmatic approach is to weaken the fairness requirement. This lead to the design of protocols that fall mainly into two distinct classes: gradual release and probabilistic fairness. Gradual release aims for fairness by incrementally revealing secrets, limiting any party's advantage. For example, Damgård [50] designed a commitment scheme allowing one-bit-at-a-time release with verification against the commitment, limiting the knowledge advantage to 1 bit. Boneh and Naor's [30] timed commitments, refined by Garay and Jakobsson [68] and Gary and Pomerance [69], use TLPs for gradual release. Parties commit to a hidden value and gradually reveal it by sending intermediate steps in solving the TLP, ensuring similar retrieval time assuming similar sequential computational power.

An alternative is partial fairness [78] or stochastic fairness [139], which allows a small but non-negligible chance of unfairness. Ben-Or *et al.* [20] presented an early example where contracts themselves are probabilistic, with parties exchanging signatures on contracts of the form "with probability p this contract is valid", and increasing p throughout the protocol. More recent work in this vein includes protocols similar to ours, initially by Gordon and Katz [78] and later in [18, 48, 139]. These protocols were often defined as a way to compute a function f over some private inputs. Informally, they say that a protocol computing a function f is $\frac{1}{p}$-secure if it can correctly compute f and guarantee that all parties receive the output with probability $1 - \frac{1}{p}$. Translating to fair exchange, a $\frac{1}{p}$-fair protocol can only guarantee fairness with probability $1 - \frac{1}{p}$.

Gordon and Katz [78] showed that in the 2-party context, any function with polynomial-sized input set X (or output set Y) can be $\frac{1}{p}$-securely computed in $\mathcal{O}(p|X|)$ (or $\mathcal{O}(p^2|Y|)$) rounds. They also proved their protocol optimal: functionalities not meeting the polynomial size bounds cannot be $\frac{1}{p}$-securely computed with arbitrary p. Thus, they show that $\frac{1}{p}$-fairness is possible whenever secrets are sampled from a polynomial-sized set X, and that these protocols require $\mathcal{O}(p|X|)$ communication rounds.

Roscoe and Ryan [139] and Couteau *et al.* [48], using TLPs, achieve a slightly different fairness notion: in a p-round protocol, the probability of a party getting the other's secret is at most $\frac{1}{p}$ higher than vice versa. While their partial fairness is bounded above by $\frac{1}{4}$, p can be chosen arbitrarily regardless of the secret type, i.e., secrets can be sampled from a non-polynomial-sized set.

Our Chapter 5 protocol achieves $\frac{1}{p}$-partial fairness in only $p+1$ rounds without restrictions on the secrets exchanged. By integrating this with generic Multi-Party Computation (MPC) techniques[7], we can $\frac{1}{p}$-securely compute any function in $p+4$ rounds. This demonstrates the power of TLPs in going beyond Gordon and Katz's impossibility result. Moreover, our protocol is shown to be "optimal" in the sense that no protocol can be $\frac{1}{p}$-fair with less than $p+1$ rounds. Thus, we almost completely solve the partial fairness problem in the two-party scenario.

In the multi-party context, Beimel *et al.* [18] construct protocols achieving partial fairness under different adversarial assumptions. With no limit on the number of malicious parties, any m-party function f with polynomial-size range Y can be $\frac{1}{p}$-securely computed in $\mathcal{O}(m)(2mp^2|Y|)^{2^m+1}$ rounds. Interestingly, if f also has a polynomial-size domain X, $\frac{1}{p}$-partial fairness is achievable in $\mathcal{O}(m)(p|Y||X|^{\mathcal{O}(m)})^{2^m+2}$ rounds, with full security under an honest majority.

Our Chapter 6 protocol achieves $\frac{1}{p}$-fairness in p exchange rounds after a $(p+2)$-round preparation phase. Once again, we provide an "optimality" result by showing that no protocol can be $\frac{1}{p}$-fair in less than p exchange rounds. While vastly improving the round complexity of Beimel *et al.*'s solutions, our result lacks full fairness with an honest majority. This "best of both worlds" approach remains an open research question.

2.3 Preliminaries

In this section, we introduce the formal definitions and classical results that will be used throughout this thesis. In particular, we define TLPs and provide the security

[7] See Section 2.3.2.

definitions for both TLPs and fair exchange protocols. We also discuss classical results from the field of multi-party computation (MPC).

2.3.1 Time-Lock Puzzles

Having reviewed the various constructions available in the literature, we can now formally define TRE using TLPs. Our definition aims to be general enough to encompass most constructions, particularly those based on random oracles.

Definition 2.1 (Time-Lock Puzzle)**.** A TLP scheme consists of three algorithms $(\mathsf{Pgen}, \mathsf{Delay}, \mathsf{Open})$:

1. $\mathsf{Pgen}(1^\lambda, T)$: Given the security parameter and time parameter, it outputs public-private parameters $(\mathsf{pp}, \mathsf{ps})$.
2. $\mathsf{Delay}(m, \mathsf{ps})$: Using the secret parameters ps, this algorithm constructs a puzzle with solution m.
3. $\mathsf{Open}(c, \mathsf{pp})$: Solves the puzzle c.

The correctness of the scheme is defined as:

$$\Pr\Big[s = s' \;\Big|\; (\mathsf{pp}, \mathsf{ps}) \xleftarrow{\$} \mathsf{Pgen}(1^\lambda, T); c \xleftarrow{\$} \mathsf{Delay}(s, \mathsf{ps}); s' \xleftarrow{\$} \mathsf{Open}(c, \mathsf{pp})\Big] \geq 1 - \mathsf{negl}(\lambda)$$

for some negligible function $\mathsf{negl}(\lambda)$ and all time parameters T and secrets s.

Remark. The literature often presents similar definitions where Pgen and Delay are restricted to being efficient (i.e. running in $\mathsf{poly}(\log T)$ time). We intentionally avoid this restriction to allow Delay and Open to have similar time complexities, enabling constructions based on random oracles to be classified as TLPs due to their stronger security guarantees.

Our definition differs from others by introducing the Pgen function. This provides more granularity in TLP analysis, as parameters generated by Pgen can be reused multiple times. For instance, in the RSW puzzle, Pgen involves constructing the RSA modulus N and computing $2^T \mod \phi(N)$, which can be expensive but is not required for every puzzle.

There are various approaches to defining TLP security, such as requiring the adversary to be unable to solve the puzzle before the designated time [109] or designing an IND-CPA-like game [2]. For our purposes, we require the puzzle to hide *all* information about its message, but not full CPA indistinguishability. Thus, we adopt the definition from [23], which is stronger than [109] and weaker than [2].

Definition 2.2 (TLP security)**.** We say that $(\mathsf{Pgen}, \mathsf{Delay}, \mathsf{Open})$ is a TLP with gap $\varepsilon \geq 1$ if there exists a polynomial $\widehat{T}(\_)$ such that for all polynomials $T(\_) \geq \widehat{T}(\_)$, any adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, and any pair of messages m_1, m_2, there is a negligible function $\mathsf{negl}(\lambda)$ such that

$$\Pr\left[b \leftarrow \mathcal{A}_2(\mathsf{state}, c) \;\middle|\; \begin{array}{c}(\mathsf{pp}, \mathsf{ps}) \xleftarrow{\$} \mathsf{Pgen}(1^\lambda, T(\lambda)); \mathsf{state} \leftarrow \mathcal{A}_1(\mathsf{pp}); \\ b \xleftarrow{\$} \{0,1\}; c \xleftarrow{\$} \mathsf{Delay}(m_b, \mathsf{ps})\end{array}\right] < \frac{1}{2} + \mathsf{negl}(\lambda)$$

where the adversary satisfies:

1. $\mathcal{A}_1$ runs in time $\mathsf{poly}(\lambda, T)$.
2. $\mathcal{A}_2$ runs in time at most $T^{\frac{1}{\varepsilon}}$ with at most $\mathsf{poly}(\lambda)$ parallel processes.

Intuitively, this definition states that for sufficiently large security parameters and delays, no adversary can solve the puzzle with an exponential speedup factor greater than $\frac{1}{\varepsilon}$. We use $\frac{1}{\varepsilon}$ instead of ε differing from previous literature to reinforce the idea that a larger gap TLP is more parallelisable. Therefore, a TLP with a large gap of 100 allows adversaries to solve the puzzle in $T^{0.01}$ time, while a small gap of 2 requires at least $\sqrt{T}$ time.

2.3.2 Multi-Party Computation

The fair exchange problem can be viewed as a specific instance of the broader Secure Multi-Party Computation (SMPC or MPC) problem. In this setting, N players $P_1, \ldots, P_N$ each possess an input x_i, and they aim to compute $y = \mathcal{F}(x_1, \ldots, x_N)$ [8] for an agreed-upon, possibly randomised, *functionality* $\mathcal{F}$, without revealing their inputs. For the fair exchange problem, we can define $\mathcal{F}$ as in Figure 2.1. It collects

[8] Sometimes it's easier to define the function $\mathcal{F}$ to output an N-vector where the i^{th} component is meant for the i^{th} party.

Fair Exchange Functionality
Fair exchange between N parties $P_1, \ldots, P_N$. 1. Upon receipt of message $(s_1, \ldots, s_N)$ from party P_i where s_i are secrets, store the secrets as the i^{th} row of a $N \times N$ matrix M. 2. If all parties have sent their secrets as opposed to some abort signal, send the i^{th}column of M to P_i. Otherwise, send $\perp$ to all parties.

Figure 2.1: Fair Exchange Functionality

secrets s, aborts with $\perp$ if exchange is invalid, otherwise sends each party their intended secrets.

To define security of a MPC protocol, we immagine an *ideal* world where there exists a central trusted entity with secure channels for every party. In such scenario, all parties can send their inputs to this entity which will execute the required functionality and re-distribute the outputs. In contrast, the *real* world is defined with the usual Dolev-Yao model where the N parties interact with each others. Intuitively, in the real world, one cannot design a protocol achieving "better" security than the ideal world. Thus, a protocol is said to be secure if it *simulates* the ideal world. In particular, it is easy to check that Figure 2.1 satisfies all the fair-exchange requirements we described earlier in Section 2.2. Thus, a protocol solves the fair exchange problem if it achieves the same result as the ideal world protocol.

Formally, $\text{IDEAL}_{\mathcal{S}}^{\mathcal{F}}(X)$ describes the outcomes of the N parties running the functionality $\mathcal{F}$ using inputs X under attack by $\mathcal{S}$. Similarly, $\text{REAL}_{\mathcal{A}}^{\Pi}(X)$ describes outcomes in the real world using protocol Π under attack by $\mathcal{A}$.

Definition 2.3 (Simulatability)**.** A protocol Π simulates functionality $\mathcal{F}$ if for every adversary $\mathcal{A}$, there exists an adversary $\mathcal{S}$ such that for all inputs X:

$$\left\{\text{IDEAL}_{\mathcal{S}}^{\mathcal{F}}(X)\right\} \overset{c}{\approx} \left\{\text{REAL}_{\mathcal{A}}^{\Pi}(X)\right\}$$

A protocol is said to compute a function f *securely* if it simulates the natural functionality that computes f on the parties inputs and return the result to everyone.

On the other hand, *security-with-abort* is defined using a functionality that sends the output to the corrupted parties first. These can decide to abort the protocol and thus prevent the honest parties to receive their results. This weakened definition is required in context where fair exchange is impossible.

An alternative way of weakening the fairness requirement comes from the concept of partial fairness. We have introduced this before as it is central to our research, but we provide here the formal definition.

Definition 2.4 (Partial Fairness)**.** A protocol Π $\frac{1}{p}$-securely computes a functionality $\mathcal{F}$ if for every PPT adversary $\mathcal{A}$ in the real world, there exists an ideal-world adversary $\mathcal{S}$ such that

$$\left\{\text{IDEAL}_{\mathcal{S}}^{\mathcal{F}}(x)\right\} \overset{1/p}{\approx} \left\{\text{REAL}_{\mathcal{A}}^{\Pi}(x)\right\}$$

where $X \overset{1/p}{\approx} Y$ means X and Y are computationally indistinguishable with probability $1 - \frac{1}{p}$.

Intuitively, protocol Π is $\frac{1}{p}$-fair if the probability of true fairness in each run is $1 - \frac{1}{p}$. In this thesis, we focus on fair exchange, using the definition with the fixed exchange functionality.

There are other ways of defining security, either by modifying the ideal world scenario or using other paradigms. For instance, Universal Composability [34] modifies the settings to include an "environment" controlling protocol execution in both worlds. Security requires that for every environment and real-world adversary, there exists an ideal-world adversary making the environment unable to distinguish between the worlds. While UC guarantees the secure composition of any two UC protocols, this thesis focuses on the traditional simulatability definition.

Adversarial Model

As usual in this field, we assume that the network topology is complete, i.e. any party can send messages to any other party. However, we work in the Dolev-Yao model where the adversary controls the communication channels. In particular,

the adversary can read all messages and tamper with them. As an immediate result, there are no delivery guarantees, and more generally we do not assume synchronous channels. However, we will often rely on the existence of confidential and authenticated channels.

It is common to distinguish adversaries based on their computational power, and in this thesis we always assume computational bounded adversaries. That is, we discuss computational security as opposed to information-theoretical guarantees. Moreover, we always consider static corruptions. That is, the identities of the malicious parties are fixed, i.e., no honest party will be corrupted and become dishonest during the execution of a protocol.

Under the above assumptions, we distinguish between three different classes of adversaries.

1. *Passive*: the adversary will follow the protocol description, but potentially run computations on the side or interrupt communication;
2. *Covert*: the adversary misbehaves only if they cannot be caught doing so;
3. *Active*: the adversary has no restrictions on their behaviour.

Oblivious Transfer

A key component for constructing MPCs, is the Oblivious Transfer (OT) primitive. While OT can be considered a cryptographic primitive like encryption (as we do in Chapter 5 when designing an augmented OT), it's easier to view it as a functionality or protocol.

A 1-out-of-k OT protocol involves a sender with k secrets $x_1, \ldots, x_k$ and a receiver with an index $1 \leq i \leq k$. The goal is for the receiver to obtain x_i, while the sender learns nothing. This can be formalised as the function $(i, (x_1, \ldots, x_k)) \mapsto (x_i, \perp)$, or the functionality of Figure 2.2, where the receiver learns nothing about other secrets x_j where $j \neq i$, and the sender learns nothing about i.

1-out-of-k Oblivious Transfer
The sender $\mathcal{S}$ has inputs: $x_1, \ldots x_k$. The receiver $\mathcal{R}$ has input: $i \in \{1, \ldots, k\}$.
1. Upon receiving $(x_1, \ldots, x_k)$ from $\mathcal{S}$, store these values.
2. On receipt of i from $\mathcal{R}$, send x_i to $\mathcal{R}$ and a token indicating protocol termination to $\mathcal{S}$.

Figure 2.2: 1-out-of-k Oblivious Transfer

GMW compiler

A key classical result in the study of MPC is a "compiler" that can enhance the security of a protocol. In particular, Goldreich, Micali and Wigderson [74] showed how to turn a protocol secure against passive adversaries that cannot abort communication to a protocol secure-with-abort even in the presence of a majority of active adversaries. While other techniques are available in the literature, this "GMW compiler" is the most renowned, and we briefly describe it here.

The GMW compiler involves three steps:

1. Parties commit to their inputs.
2. A random tape for each party is generated collectively.
3. The original protocol is executed, with each message accompanied by a zero-knowledge proof of its correctness.

This structure forces active adversaries to follow the protocol, as deviations are detected and treated as aborts. Consequently, active adversaries are limited to behaving like passive adversaries, performing side computations and interrupting communication. Formally, we have the following result (see e.g., Chapter 7 in [75].

Theorem 2.1. *Let* Π *be any protocol and let* Π' *be its* compiled *version according to the construction above. For any* active *adversary* $\mathcal{A}$ *to* Π'*, there is a* passive *adversary* $\mathcal{B}$ *for* Π *such that for every input* x

$$\{\text{REAL}^{\Pi'}_{\mathcal{A}}(x)\} \overset{c}{\approx} \{\text{REAL}^{\Pi}_{\mathcal{B}}(x)\}$$

A key aspect of this compiler is that it also applies to protocols that are safe against passive adversaries not aborting communication. Therefore, it applies to an adversarial model weaker than what we consider in this thesis. However, note that fairness is possible when considering this weaker model, but it is lost in the compiled version of the protocol. Nevertheless, if we have a protocol achieving partial fairness against our passive adversaries, then its compiled version will achieve the same partial fairness against malicious adversaries. Therefore, we will often describe our fair-exchange protocols assuming only passive adversaries. Achieving security against active adversaries derives immediately from this compiler. More details will be provided in the related chapters.

2.3.3 Commutative Encryption

Commutative encryption is a cornerstone of our fair exchange protocols. In Chapter 4, we will construct our own scheme to meet our protocols' specific requirements. Our need arises from shuffling items, a problem first addressed by the well-known SRA scheme [142] in the context of "mental poker".

Commutativity, as used in [142], means the order of key application doesn't affect the result:

Definition 2.5 (Commutative Encryption)**.** An encryption scheme $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ is commutative if for all key pairs k_1, k_2 and all messages m:

$$\mathsf{Enc}_{k_1}(\mathsf{Enc}_{k_2}(m)) = \mathsf{Enc}_{k_2}(\mathsf{Enc}_{k_1}(m))$$

The original SRA construction is similar to RSA, except the modulus doesn't need to be a product of two primes, generalising the Pohlig-Hellman cipher [134] with composite moduli.

Other schemes, like ElGamal and stream ciphers, also exhibit this commutativity property. Formalisation's based on ElGamal can be found in [83, 84].

In our exchange protocols, we require an encryption scheme with weaker commutativity properties and stronger security requirements than what is usually described

in the literature. We call this primitive *commutative blinding*, and report here its more general definition needed for a 2-party protocol.

Definition 2.6 (Commutative Blinding)**.** A commutative blinding scheme is a tuple of algorithms $(\mathsf{KGen}_1, \mathsf{KGen}_2, \mathsf{Blind}_1, \mathsf{Blind}_2, \mathsf{Unblind}_1, \mathsf{Unblind}_2)$ with sets $(\mathcal{M}, \mathcal{K}_1, \mathcal{K}_2, \mathcal{C}_{\text{int}}, \mathcal{C})$ such that

$$\mathsf{KGen}_1 \colon \{1\}^* \to \mathcal{K}_1 \qquad \mathsf{Blind}_1 \colon \mathcal{M} \times \mathcal{K}_1 \to \mathcal{C}_{\text{int}} \qquad \mathsf{Unblind}_1 \colon \mathcal{C} \times \mathcal{K}_1 \to \mathcal{C}_{\text{int}}$$

$$\mathsf{KGen}_2 \colon \{1\}^* \to \mathcal{K}_2 \qquad \mathsf{Blind}_2 \colon \mathcal{C}_{\text{int}} \times \mathcal{K}_2 \to \mathcal{C} \qquad \mathsf{Unblind}_2 \colon \mathcal{C}_{\text{int}} \times \mathcal{K}_2 \to \mathcal{M}$$

In terms of security, we require Blind_1 to satisfy the standard IND-CPA game. Similarly, we define an IND-CPA game for Blind_2, where the adversary chooses ciphertexts generated by Blind_1. Finally, we require a ciphertext outputted by $\mathsf{Unblind}_1$ to be indistinguishable from a ciphertext generated by Blind_1 when using the same message-key pair.

Since the focus of this thesis is on TLPs and fair exchange, we defer deeper discussions on commutative blinding to Chapter 4, where we highlight suitable constructions for our protocols.

3

Time-Lock Puzzles via Cubing

Contents

Currently, the most widely known Time-Lock Puzzle (TLP) relies on the hardness of integer factorisation. This approach is increasingly unsatisfactory as quantum computing advances. Despite efforts, research on post-quantum TLPs has seen lim-

ited success. Afshar *et al.* [5] developed TLPs based on hashing and demonstrated that the computation required to solve a hash-based puzzle is always equivalent to the effort invested in generating it. Consequently, hash-based puzzles are not efficient TLPs. Other quantum-secure alternatives involve randomised encodings [23] or isogeny walks [33]. However, constructing randomised encodings based on standard assumptions is inefficient, and isogeny-based TLPs demand enormous memory resources.[1] Therefore, developing an efficient post-quantum TLP is the next crucial step in this field. This chapter presents our research in this context.

We introduce and analyse a novel TLP based on iterating cube root extractions. This new approach sacrifices some efficiency of the classical RSW puzzle to eliminate reliance on integer factorisation. Thus, we achieve a balance between the quantum-secure but inefficient hash-based TLPs and the quantum-insecure but efficient RSW. A detailed comparison of our construction is provided in Section 3.3, following a thorough description of our TLP.

Our main contributions include proving the sequentiality of the cubing operation (Theorem 3.3) and conducting an extensive analysis of various attack vectors on repeated-squaring-based TLPs (Section 3.5). We also offer several practical suggestions for instantiating this puzzle in $\mathbb{Z}_p^*$.

The chapter begins with an analysis of the cubing operation and its sequentiality proof. We then discuss how to link these operations to form our TLP. Security in the Random Oracle Model (ROM) is demonstrated using existing literature, and we explore options for securely instantiating the random oracle. Once our puzzle is fully defined, we compare it in-depth with RSW and hash-based puzzles in Section 3.3. The latter half of the chapter focuses on practical aspects, including a proof-of-concept implementation of our construction (Section 3.4) and a comprehensive analysis of all known attack vectors (Section 3.5). This analysis is crucial for studying *all* puzzles based on repeated squaring, including the RSW puzzle.

[1] Refer to Section 2.1 for a more detailed discussion on the current state of the art.

3.1 Delay-Inducing Function

In this section, we present our construction based on modular cubing, focusing solely on the cubing aspect and neglecting the chaining feature for now. Recall the definition of TLPs from Definition 2.1 in Chapter 2:

A TLP scheme consists of three algorithms $(\mathsf{Pgen}, \mathsf{Delay}, \mathsf{Open})$:

1. $\mathsf{Pgen}(1^\lambda, T)$: Given the security parameter 1^λ and time parameter T, it outputs public-private parameters $(\mathsf{pp}, \mathsf{ps})$.
2. $\mathsf{Delay}(m, \mathsf{ps})$: Using the secret parameters ps, this algorithm constructs a puzzle with solution m.
3. $\mathsf{Open}(c, \mathsf{pp})$: Solves the puzzle c.

Our construction defines Delay as the iterative application of a "delay-inducing" function $d(x)$ alternated with a "chaining" function $h(x)$. Abusing notation, $\mathsf{Delay}(x) = (d \circ h)^t(x)$ for some parameter t. This section will primarily focus on the function $d(x) = x^3 \bmod p$ for a safe prime p.

Such a function is invertible for any prime $p \equiv 2 \mod 3$, and safe primes larger than 7 satisfy this condition. The inverse is $d^{-1}(x) = x^b \mod p$ for any b such that $3b \equiv 1 \mod \mathsf{ord}(x)$. Choosing $b = \frac{2p-1}{3}$ satisfies this congruence for any $x \in \mathbb{Z}_p^*$.

Intuitively, the function d introduces a delay because computing $d(x)$ requires two modular multiplications, while computing $d^{-1}(x)$ necessitates approximately $\log(b) \approx \log(p)$ sequential squarings. In Section 3.1.2, we prove that any algorithm inverting $d(x)$ will require approximately $\log(p)$ sequential squarings. Therefore, by selecting an appropriate p, we can control the ratio between the time spent creating the puzzle and the time spent solving it. Larger primes result in more efficient puzzle generation; however, as discussed in Section 3.5, larger moduli can also enable more effective parallelisation. Thus, it is crucial to choose a modulus that is as large as possible while still ensuring the required level of sequentiality for

Delay	Open
input: $: m, p, C$	**input:** $: c, p, C$
$s \xleftarrow{\$} \{0,1\}^\lambda$	$b \leftarrow \frac{2p-1}{3}$
$x \leftarrow \mathsf{pad}\,(m, s)$	**repeat** C **times**
repeat C **times**	$x \leftarrow c^b \bmod p$
$c \leftarrow h(x)$	$c \leftarrow h^{-1}(x)$
$x \leftarrow c^3 \bmod p$	$m \leftarrow \mathsf{pad}^{-1}\,(x)$
return c	**return** m

Figure 3.1: Encryption and decryption process.

the application.[2] Additionally, another parameter, C, is introduced to adjust the overall delay of the puzzle independently of the modulus choice.

Specifically, the parameter generation process $\mathsf{Pgen}(1^\lambda, T)$ selects a large safe prime p and an integer C such that the delay of computing d^{-1} is $\frac{T}{C}$ (i.e. we ignore the time spent computing h). Both parameters can be public. Figure 3.1 depicts the operations of Delay and Open. Note that the order of applying cubing and chaining affects the indistinguishability of the scheme and is therefore not arbitrary.

We work in the group $\mathbb{Z}_p^*$, but the message space might be a strict subset depending on the padding scheme. The correctness of the TLP follows directly from the observation that $x^{3b} = x^{2(p-1)+1} = x \mod p$ for any x. In Figure 3.1, we utilise a padding scheme pad to introduce randomness into the encryption process. We require the padding to satisfy two conditions:

1. For any message m, $\mathsf{pad}(m, \_)$ is an injective function.
2. $\mathsf{pad}(m, s)$ is sufficiently large so that its cube is much greater than p.

Padding is crucial for ensuring security when the TLP is used on a small message space. By introducing randomness into an otherwise deterministic algorithm, it increases the ciphertext space by a factor of 2^λ.

[2] In Section 3.4, we provide concrete measurements on how p affects the cube root computation.

We can also explicitly compute the probability of Bob correctly guessing a message before the delay opens, given the size of the seed. Suppose Alice and Bob are using a prime p of n bits, and Alice's seed has length l bits. To decrypt a message, Bob needs at least $n-1$ sequential modular multiplications. To encrypt a message, they need 2 sequential modular multiplications (neglecting the cost of the padding scheme).

Assume Bob has access to C concurrent computers. During the delay period (i.e. the time it takes to perform $n-1$ modular multiplications), Bob can perform $\frac{n-1}{2}C$ encryptions and thus guess the same number of random seeds.

If the seed is drawn uniformly at random from the set of all l-bit strings, then Bob has a probability of $\frac{n-1}{2^{l+1}}C$ of guessing the correct seed. If Alice wants to bound this probability by some value ε, she obtains the following inequality:

$$l > \log C + \log(n-1) - \log \varepsilon - 1$$

From a theoretical perspective, we fix the seed length as the security parameter λ, ensuring that the probability of Bob guessing it with polynomially many parallel processes is negligible: $\frac{\mathsf{poly}(\lambda)}{2^{\lambda}}$.

3.1.1 Using a Safe Modulus

Let p be the modulus used in our scheme. We previously specified that p should be a safe prime. In this section, we justify this choice by demonstrating that safe primes are optimal for maximising decryption time. Specifically, we aim for all values in $\mathbb{Z}_p^*$ to have the same decryption time, meaning that computing a cube root should take the same amount of time regardless of the input. If this condition is not met, puzzles generated with the same parameters p and C could result in varying delays, as the cube-root computation might be easier for some inputs than others.

Let $\mathcal{B}_p = \big\{(\lfloor \log b \rfloor + 1, x) \mid b \equiv 3^{-1} \bmod \mathsf{ord}(x) \wedge x \in \mathbb{Z}_p^* \wedge b \leq \mathsf{ord}(x)\big\}$,
$H_p = \max\{d \mid (d,x) \in \mathcal{B}_p \text{ for some } x\}$, and $\mathcal{H}_p^{\varepsilon} = \{x \mid (d,x) \in \mathcal{B}_p \wedge H_p - d < \varepsilon\}$.

- The set $\mathcal{B}_p$ contains pairs (d, x), where d is the bit-length of the exponent needed to invert $x^3 \bmod p$. One can think of this set as assigning a "difficulty parameter" d to each possible message x.
- H_p represents the maximum difficulty (i.e. the maximum bit-length of an exponent required for inversion).
- The set $\mathcal{H}_p^\varepsilon$ is the subset of $\mathbb{Z}_p^*$ for which decryption requires at most ε fewer squarings than the maximum value H_p. Thus, $\mathcal{H}_p^\varepsilon$ contains the elements for which decryption is at most ε steps faster than the hardest case.

We aim to minimise $K_p^\varepsilon = |\mathbb{Z}_p^*| - |\mathcal{H}_p^\varepsilon|$ among all primes p of the same bit-size. Intuitively, K_p^ε is the number of messages that can be decrypted with at least ε fewer steps than the hardest element. Hence, K_p^ε represents the number of elements for which our delay is shorter. Safe primes minimise this quantity because all elements in $\mathbb{Z}_p^* \setminus \{1, -1\}$ have an order of either p or $\frac{p-1}{2}$. Since these two values differ by only one bit, the exponents needed to invert the cubing operation have bit sizes differing by at most one. We formalise this in the following theorem.

Theorem 3.1. *If $p > 7$ is a safe prime, then for $2 \leq \varepsilon < H_p$, K_p^ε is minimal.*

Proof. First, it is clear that $\mathcal{H}_p^a \subseteq \mathcal{H}_p^b$ if $a \leq b$. Thus, we only need to prove the statement for K_p^2. To invert $c = x^3 \bmod p$, one needs to compute $c^b \bmod p$ such that $3b \equiv 1 \mod \mathsf{ord}(x)$:

$$b = 3^{-1} \bmod \mathsf{ord}(x) = \begin{cases} \frac{2\,\mathsf{ord}(x)+1}{3} & \text{if } \mathsf{ord}(x) \equiv 1 \bmod 3 \\ \frac{\mathsf{ord}(x)+1}{3} & \text{if } \mathsf{ord}(x) \equiv 2 \bmod 3 \\ \text{non-existent} & \text{if } \mathsf{ord}(x) \equiv 0 \bmod 3 \end{cases}$$

For every number p (not necessarily prime), $1, -1 \in \mathbb{Z}_p^*$. So there are at least two elements where $x^3 \equiv x \mod p$. Hence, for all $\varepsilon < H_p$, $K_p^\varepsilon \geq 2$.

Let $p = 2q + 1$ be the safe prime corresponding to a Sophie Germain prime $q > 3$. Then the group $\mathbb{Z}_p^*$ has order $2q$ and is isomorphic to $\mathbb{Z}_2 \oplus \mathbb{Z}_q$ under addition. Therefore, there is only 1 identity $(0, 0)$, 1 element of order 2 $(1, 0)$, $q - 1$ elements of order q $(0, \_)$, and $q - 1$ generators $(1, \_)$. For x of order $2q$, $b = \frac{4q+1}{3}$, while for

x of order q, $b = \frac{q+1}{3}$. Since the difference in bit-length between these two fractions is at most 1, we have $K_p^2 = 2$. □

The theorem demonstrates that when using safe primes, all but two elements of $\mathbb{Z}_p^*$ require essentially the same time to decrypt. Furthermore, we have hinted that computing the cube root appears to require an exponentiation whose exponent's bit-size is very close to the bit-size of the modulus. This is formalised and proved in Section 3.1.2.

Generating Safe Primes

Although our scheme remains secure when the same modulus is used repeatedly, changing the safe prime used prevents adversaries from performing meaningful precomputation. Therefore, the ability to generate safe primes quickly is a crucial challenge. To facilitate the adoption of our puzzle, we must address the practicality of this issue.

Generating large primes can be time-consuming, and generating safe primes is even more so. It is conjectured [144] that the number of safe primes less than x asymptotically approaches $1.32\frac{x}{(\ln x)^2}$, compared to the $\frac{x}{\ln x}$ density of any prime. Thus, safe primes are substantially more sparse.

In this section, we briefly discuss preliminary research results aimed at identifying other moduli that guarantee similar properties to safe primes while being more easily generated. Specifically, we advocate for the use of primes of the form $mq+1$, where $m \in \mathcal{O}(\log q)$. We refer to this class of primes as *almost safe*.

There has been limited theoretical research on efficient algorithms for generating safe primes. The most relevant work to our discussion is by Von zur Gathen and Shparlinski [155]. They designed a polynomial-time algorithm to generate primes of the form $2q+1$ or $2q_1q_2+1$, where q, q_1, q_2 are primes with $x^\alpha \leq q_1 < q_2 \leq q_1^{1/\alpha-1}$ for $\alpha \in [0.25, 0.276)$ and input x.

Using $p = 2q_1q_2 + 1$ in our scheme is relatively safe. The probability of a random element having a "small" order (i.e. not q_1q_2 or $2q_1q_2$) is $\frac{2q_1+2q_2-2}{2q_1q_2}$. If $n_1 = \log q_1$ and $n_2 = \log q_2$, this probability is roughly $\frac{2^{n_2+3}}{2^{n_1+n_2+1}} \approx \frac{1}{2^{n_1}}$, which is negligible in the bit-size of p. However, we believe that almost safe primes have more desirable properties.

Let $p = mq + 1$ be a prime, where q is also prime and m is a small integer (e.g. $m \in \mathcal{O}(\log q)$). In this case, elements of $\mathbb{Z}_p^*$ have order dividing mq, and there are only m elements with order dividing m (since $\mathbb{Z}_p^* \cong (\mathbb{Z}_m, +) \oplus (\mathbb{Z}_q, +)$). The probability of a random element having a small order is $\frac{m}{p}$, negligible in the bit-size of p and much lower than for primes discussed above.

Generating such primes is relatively easy. After finding a prime q, we can search for a prime $p \equiv 2 \mod 3$ in the sequence $\langle (6k + 4(q \bmod 3))q + 1 \mid k \in \mathbb{N} \rangle$. Primality of $p = mq + 1$ can be efficiently checked using Pocklington's test [145]: if we find an a such that $a^{mq} \equiv 1 \mod p$ and $\gcd(a^m - 1, p) = 1$, then p is prime.

The question is how many m values we need to try before finding a prime. We rewrite the prime-candidate sequence as $kd+a$, where $d = 6q$ and $a = 4(q \bmod 3)q+1$. The number of primes less than x in this arithmetic progression is given by $\pi(x; a, d)$ (see e.g. [21]). The prime number theorem for arithmetic progressions states that $\pi(x; a, d) \sim \frac{\mathrm{Li}(x)}{\phi(d)}$, where $\mathrm{Li}(x) = \int_2^x \frac{dt}{\ln t}$ and ϕ is the Euler's totient function. For large enough x, we have the lower bound $\mathrm{Li}(x) > \frac{x}{\ln x} - \gamma$, where γ is a constant close to 1. Therefore, $\pi(mq; a, d) \sim \frac{\mathrm{Li}(mq)}{2(q-1)} > \frac{1}{2(q-1)} \left(\frac{mq}{\ln(mq)} - 2 \right)$.

The inequality $\frac{1}{2(q-1)} \left(\frac{mq}{\ln(mq)} - 2 \right) > 1$ can be rearranged to $m > (\ln m + \ln q) \left(\frac{2q+1}{q} \right)$. Setting $m = 3 \ln q$, the inequality holds. Therefore, asymptotically, we expect to find a suitable prime less than $3q \ln q$.

We conducted a preliminary test using the GMP library to generate primes and then applied the method described above to find "almost safe" primes. Table 3.1 presents the mean and standard deviation of the measured timings over 10 trials, along with the number of m values tested before a prime was found. It is important

Bit-size	Prime Generation	Safe-Prime Generation	Attempts
1000	10.74 (8.480)	68.39 (42.73)	259.1 (159.8)
2000	109.6 (85.49)	685.6 (723.4)	373.1 (396.9)
3000	400.1 (331.1)	3895 (3225)	719.4 (591.3)
4000	1078 (1780)	11506 (13330)	970.5 (1121)

Table 3.1: Timings in milliseconds to generate "almost safe" primes.

to note that we employed the Pocklington primality test exclusively with the base $a = 2$. Consequently, it is possible that some primes were inadvertently excluded.

3.1.2 Sequentiality of the Cube-Root Process

In this section, we discuss the sequentiality of the decryption process. In particular, we show that decryption requires at least $\lfloor \log b \rfloor - 1$ sequential steps.

The Strong Algebraic Group Model (SAGM) was introduced by Katz *et al.* [90] as a tool to prove the sequentiality of the RSW puzzle. This abstract model strengthens the well-known Algebraic Group Model [67] by allowing one to measure the number of steps an algorithm performs.

A *strongly algebraic* algorithm receives efficient (i.e. bit-string) representations of group elements as input. However, it is essentially required to output a computational tree, consisting of group elements, that demonstrates how the final result is computed. This allows one to measure the running time of a strongly algebraic algorithm by examining the depth of such a tree. We present the formal definition from [90] below.

Definition 3.1. An algorithm $\mathcal{A}$ over a group $\mathbb{G}$ is called *strongly algebraic* if, in each (algebraic) step, $\mathcal{A}$ performs arbitrary local computations and then outputs one or more tuples of the following form:

1. $(x, x_1, x_2) \in \mathbb{G}^3$ where $x = x_1 \cdot x_2$ and x_1, x_2 were either provided as input to $\mathcal{A}$ or were outputted by $\mathcal{A}$ in some previous step(s).

2. $(x, x_1) \in \mathbb{G}^2$ where $x = {x_1}^{-1}$ and x_1 was either provided as input to $\mathcal{A}$ or was outputted by $\mathcal{A}$ in some previous step.

Definition 3.2 (Katz *et al.* [90])**.** Let $\mathcal{A}$ be a strongly algebraic algorithm over a group $\mathbb{G}$. Assume $\mathcal{A}$ is given only g as input. We define $\mathsf{DLOG}_g(x)$ recursively on each output of $\mathcal{A}$:

- $\mathsf{DLOG}_g(g) = 1$.
- $\mathsf{DLOG}_g(x) = \mathsf{DLOG}_g(x_1) + \mathsf{DLOG}_g(x_2)$ for any (x, x_1, x_2) output of $\mathcal{A}$.
- $\mathsf{DLOG}_g(x) = -\mathsf{DLOG}_g(x_1)$ for any (x, x_1) output of $\mathcal{A}$.

Intuitively, in such a restrictive model, t steps are required to compute g^{2^t}. Specifically, consider an algorithm that receives g as input. During any algebraic steps, it can only output a value g^i for some integer i. Additionally, at each step, the algorithm can either flip the sign of i or increase its absolute value by adding another $y \leq i$. This immediately imposes a bound on $|i|$.

We formalise this intuition with the following lemma.

Lemma 3.2. *Let $\mathcal{A}$ be a strongly algebraic algorithm over a group $\mathbb{G}$. Assume $\mathcal{A}$ is given only $g \in \mathbb{G}$ as input and runs for t algebraic steps. Then, for any output x of $\mathcal{A}$, we have $g^{\mathsf{DLOG}_g(x)} = x$ and $|\mathsf{DLOG}_g(x)| \leq 2^t$.*

Katz *et al.* [90] proved a slightly weaker version of this lemma without the constraint $g^{\mathsf{DLOG}_g(x)} = x$. We augment their proof to include this stronger requirement.

Proof of Lemma 3.2. We proceed by induction on t, the number of algebraic steps of $\mathcal{A}$.

Base Case ($t = 1$): In this case, $\mathcal{A}$ can only output $x \in \{g^{-1}, g^2\}$. For both of these values, $g^{\mathsf{DLOG}_g(x)} = x$ and $|\mathsf{DLOG}_g(x)| \leq 2$.

Inductive Step: Assume the lemma holds for all $t \leq n - 1$. Consider an algorithm $\mathcal{A}$ that runs for n steps. At step n, $\mathcal{A}$ outputs either:

Case 1: A Triple (x, x_1, x_2): By definition, $\mathsf{DLOG}_g(x) = \mathsf{DLOG}_g(x_1) + \mathsf{DLOG}_g(x_2)$. Since x_1 and x_2 were either input or produced in previous steps, the induction

hypothesis applies. Thus, $g^{\mathsf{DLOG}_g(x_i)} = x_i$ and $|\mathsf{DLOG}_g(x_i)| \leq 2^{n-1}$ for $i = 1, 2$. Therefore:

- $g^{\mathsf{DLOG}_g(x)} = g^{\mathsf{DLOG}_g(x_1)+\mathsf{DLOG}_g(x_2)} = x_1 x_2 = x$
- $|\mathsf{DLOG}_g(x)| = |\mathsf{DLOG}_g(x_1) + \mathsf{DLOG}_g(x_2)| \leq |\mathsf{DLOG}_g(x_1)| + |\mathsf{DLOG}_g(x_2)| \leq 2^{n-1} + 2^{n-1} = 2^n$

Case 2: A Pair (x, x_1): By definition, $\mathsf{DLOG}_g(x) = -\mathsf{DLOG}_g(x_1)$. Applying the induction hypothesis to x_1, we get $g^{\mathsf{DLOG}_g(x_1)} = x_1$ and $|\mathsf{DLOG}_g(x_1)| \leq 2^{n-1}$. Therefore:

- $g^{\mathsf{DLOG}_g(x)} = g^{-\mathsf{DLOG}_g(x_1)} = (x_1)^{-1} = x$
- $|\mathsf{DLOG}_g(x)| = |-\mathsf{DLOG}_g(x_1)| = |\mathsf{DLOG}_g(x_1)| \leq 2^n$

This completes the proof by induction. □

As an immediate result of this lemma is a bound on the number of algebraic steps required by an algorithm computing cube roots.

Theorem 3.3. *Let $\mathcal{A}$ be a strongly algebraic algorithm working in the group $\mathbb{Z}_p^*$ with input x^3 and output x. Then $\mathcal{A}$ runs for at least $\lfloor \log p \rfloor - 2$ (which is $\leq \lfloor \log b \rfloor - 1$) sequential steps.*

Proof. Assume for contradiction that $\mathcal{A}$ runs in $t \leq \lfloor \log p \rfloor - 3$ steps and outputs x. Let $c = x^3$. We know $\mathsf{DLOG}_c(x) = a + k\,\mathsf{ord}(x)$, where a is either $\frac{4q+1}{3}$ or $\frac{q+1}{3}$ (from Section 3.1.1) and k is an integer. By Lemma 3.2, $\frac{q+1}{3} \leq |\mathsf{DLOG}_c(x)| < 2^t \leq 2^{\lfloor \log p \rfloor - 3} \leq \frac{p}{8}$. However, $\frac{q+1}{3} > \frac{p}{8} = \frac{2q+1}{8}$, leading to a contradiction. □

The theorem above demonstrates that decrypting our delay-inducing function necessitates a number of sequential steps approximately equal to the modulus' bit-size. However, the SAGM and our discussion consider each multiplication as a single step.

In Section 3.5, we will show that this assumption is not entirely accurate, as modular multiplication can be parallelised. We now establish that sequentiality also depends on the discrete logarithm problem.

Specifically, let $a \leftarrow \mathcal{A}(b)$ for some strongly algebraic algorithm $\mathcal{A}$. Using DLOG, we can compute $\log_b(a)$ in a time comparable to the running time of $\mathcal{A}$. Consider an algorithm $\mathcal{A}$ that, after some precomputation with a generator g, can compute $\sqrt[3]{x}$ "too fast". This means $\mathcal{A}$ does not have sufficient time to compute $\sqrt[3]{x}$ using the standard repeated squaring method.

Consequently, the output x can be expressed as $g^k x^{3y}$ for some k and y, with the crucial condition that $k \neq 0$. Given the algebraic outputs of $\mathcal{A}$, we can efficiently compute k and y, allowing us to express x as $g^{k/(3y-1)}$ and thereby obtain the discrete logarithm of x.

Theorem 3.4. *Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be a strongly algebraic algorithm over $\mathbb{Z}_p^*$. Assume $\mathcal{A}_1$, given a generator g, runs in time t_p with t'_p algebraic steps, outputting an internal state state and group elements $h_1, \ldots, h_k$.*[3] *Assume $\mathcal{A}_2$, given state, $h_1, \ldots, h_k$, and x^3, outputs x in time t_o with $t'_o < \lfloor \log p \rfloor - 2$ algebraic steps. Then an algorithm $\mathcal{B}$ exists that computes $\log_g x$ given g and x in time $\mathcal{O}\big(t_p + t'_p + t_o + t'_o\big)$.*

Proof. We construct $\mathcal{B}$ to use $\mathcal{A}$. Firstly, $\mathcal{B}$ runs $\mathcal{A}_1$ on g to obtain state and $h_1, \ldots, h_k$. This takes t_p time. Using Definition 3.2, $\mathcal{B}$ can compute $\alpha_i = \mathsf{DLOG}_g(h_i) \mod p-1$ in time $\mathcal{O}\big(t'_p\big)$. Note that $g^{\alpha_i} = h_i$ by Lemma 3.2 and that α_i is minimal since we took the modulo. Thus, $\alpha_i = \log_g h_i$. $\mathcal{B}$ can then proceed to run $\mathcal{A}_2$ on $(\mathsf{state}, h_1, \ldots, h_k, x^3)$ to obtain x. This runs in time t_o. Using the algebraic steps of $\mathcal{A}_2$, $\mathcal{B}$ can express x as $(x^3)^\gamma \prod_{i=1}^k h_i^{\beta_i}$, where computing the exponents β_i and γ takes $\mathcal{O}(t'_o)$. Therefore, $\log_g x = 3\gamma \log_g x + \sum_{i=1}^k \beta_i \log_g h_i \mod p-1$. Hence,

$$\log_g x = \frac{1}{1-3\gamma} \sum_{i=1}^{k} \beta_i \alpha_i \mod p-1 \tag{3.1}$$

We have the guarantee that $3\gamma \neq 1 \mod p-1$ since $3\gamma = 1 \mod p \implies \gamma$ is a multiple of $\frac{2p-1}{3}$ with overwhelming probability.[4] But then $\gamma \geq 2^{\lfloor \log p \rfloor - 2}$ which

[3] Formally, $\mathcal{A}$ would accompany each step with a flag indicating output status.
[4] Recall our discussion in Section 3.1.1.

is impossible by Lemma 3.2 since $\mathcal{A}_2$ runs in $t'_o < \lfloor \log p \rfloor - 2$ algebraic steps. It follows that $\mathcal{B}$ can compute $\log_g x$ using Equation 3.1 in time $\mathcal{O}(t'_o)$. □

While this seems inconsequential, as the discrete logarithm is more complex than the cube root, we'll see in Section 3.5 that we often rely on the hardness of the discrete log.

3.2 Chaining Functions

As we will discuss in Section 3.5, the larger the prime p, the easier it is to parallelise the underlying integer multiplication. This suggests an upper bound on the delays safely achievable through cubing alone. The purpose of this section is to circumvent this limitation by reliably chaining together multiple delays. The techniques described here apply to other TLPs as well, but we maintain our focus on the cubing scheme from Section 3.1.

Let p be a safe prime, and define the function $d_p \colon \mathbb{Z}_p \to \mathbb{Z}_p$ by $d_p(x) = x^3 \bmod p$. Our goal is to use d_p repeatedly to achieve a long delay. Specifically, we want to find an efficient and efficiently invertible function $h_p \colon \mathbb{Z}_p \to \mathbb{Z}_p$ to interleave with d_p, allowing us to multiply the delay introduced by d_p.

Our final TLP scheme is represented by the function $\mathsf{Delay}_p^n(x)$:

$$\mathsf{Delay}_p^n(x) = (d_p \circ h_p \circ \cdots \circ d_p \circ h_p)(x)$$

To guide our choice of h_p, we note that inverting a polynomial $m(x)$ in $\mathbb{Z}_p$ is relatively easy. Given $b = m(a)$, we have $\gcd(m(x) - b, x^p - x) = x - a$. Therefore, we need to select h_p such that Delay_p^n cannot be easily represented as a polynomial over $\mathbb{Z}_p$.

In the context of VDFs, a similar problem was studied by Lenstra and Wesolowski [95] and Boneh *et al.* [29]. In particular, [95] proves that the construction Delay_p^n is secure when h_p is modelled as a random oracle. More precisely, they define a "slow" hash function, *Sloth*, as an alternating sequence of modular

square roots and a simple algebraic permutation (see Section 3.2.1), all followed by a hash computation. This primitive is used in a coin-toss protocol, and its security is proven.

Specifically, they prove the following theorem:

Theorem 3.5 ([95]). *Assume Sloth consists of l square-root computations and a random permutation $\sigma\colon \mathbb{Z}_p^* \to \mathbb{Z}_p^*$, followed by a hash H with an output of λ bits. Let $P\colon \{0,1\}^\lambda \to \{0,1\}$ be a predicate describing a desired property of Sloth's output. Assume $\log p, l \in \mathsf{poly}(\lambda)$ and that a single square root cannot be computed faster than time Δ. For any adversary $\mathcal{A}$ making at most $q \in \mathsf{poly}(\lambda)$ queries to σ and H and running in time at most $l\Delta$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that:*

$$\Pr\left[P(y) = 1 \;\middle|\; y \leftarrow \mathsf{Sloth}(x), x \leftarrow \mathcal{A}^{\sigma,\mathsf{H}}\right] < (2q + |P^{-1}(\{1\})|)2^{-\lambda} + \mathsf{negl}(\lambda)$$

Lenstra and Wesolowski conclude that "to find a valid output for Sloth on a random input, there is no faster strategy than computing l square roots sequentially" [95]. Consequently, our chain of repeated squarings and random permutations benefits from the same result: all cube roots must be computed sequentially.

Intuitively, this security stems from the same reason that hash-based TLPs are secure: to compute a chain of random oracle calls, one needs to perform all the calls sequentially. Since $d_p(x)$ is a bijection, $d_p \circ h_p$ is a pseudo-random permutation (PRP) if h_p is a PRP as well. Thus, Delay_p^n can be viewed as a chain of PRPs. When modelled as true black-box functions, sequentiality is immediate, provided the chain is at most polynomially long in the bit-size of p. Unruh [154] was the first to formally prove the sequentiality of a chain of random oracles. Therefore, our construction can also be proved secured using Unruh's work.

Theorem 3.6 (iterated hashing [154]). *Let $\mathcal{A}$ be any (quantum) algorithm $\mathcal{A}$ with oracle access to a random function $\mathsf{H} : \{0,1\}^\lambda \to \{0,1\}^\lambda$. Restrict $\mathcal{A}$ to perform at most polynomially many oracle calls to H and in less than T sequential adaptive*

rounds, then there is a negligible function $\mathsf{negl}(\lambda)$ *such that*

$$\Pr_{\mathsf{H},x}\left[\mathsf{H}^T(x) \leftarrow \mathcal{A}^{\mathsf{H}}(x)\right] < \mathsf{negl}(\lambda)$$

To quantify the exact delay obtained by our TLP, we note that to obtain the message m, one needs to invert the PRP on $h_p(m)$. Until the oracle modelling h_p is called, the value $h_p(m)$ is indistinguishable from $h_p(m')$ for any m' for which $h_p(m')$ has not been seen before. If the chain is polynomially long, the probability that $h_p(m)$ has been seen before is negligible, as the domain of h_p is exponentially large. Hence, $\mathsf{Delay}_p^n(m)$ is indistinguishable from $\mathsf{Delay}_p^n(m')$ until all the internal delays given by $d_p(x)$ are opened.

Given that the size of p will likely be large, constructing a truly random permutation is impractical. Therefore, we aim to use suitable pseudo-random permutations. In this regard, we diverge from previous literature, where VDFs are often constructed by choosing h_p to be efficiently representable as a fixed permutation. Our construction achieves stronger security guarantees by adhering more closely to abstract models where security is proven.

3.2.1 Algebraic Methods

For the sake of completeness, we present two algebraic methods used in the context of Verifiable Delay Functions (VDFs), although they lack any formal security guarantees.

Swapping Neighbours

Lenstra and Wesolowski [95] proposed the following "swapping neighbours" function:

$$h_p(x) = \begin{cases} x+1 & \text{if } x \text{ is odd} \\ x-1 & \text{if } x \text{ is even} \end{cases}$$

Note that: $\mathsf{Delay}_p^2(x) = \left((x \pm 1)^3 \pm 1\right)^3 \bmod p$. Therefore, given a message m $\mathsf{Delay}_p^2(m)$ can be written as one of four possible polynomials in m. Computing a polynomial GCD as mentioned above would allow us to invert Delay_p^2 in roughly

the same time required to invert a single cubing. This suggests that the use of the "swapping neighbours" function should be avoided in short chains due to this potential vulnerability.

Linear Permutation

Boneh *et al.* [29] proposed a permutation h_p defined over $(\mathbb{Z}_p)^2$:

$$h_p(x, y) = (y + c_1, x + c_2)$$

for some constants c_1, c_2.

While we are not aware of specific security flaws in this construction, it does require defining d_p over $(\mathbb{Z}_p)^2$, and h_p is far from being a secure pseudo-random permutation.

3.2.2 Format-Preserving Encryption

A natural candidate for the function h_p is an encryption scheme. Format-Preserving Encryption (FPE) focuses on constructing PRPs on arbitrary domains, such as $\mathbb{Z}_p$ in our case. The NIST-approved FPE schemes [123] are based on Feistel Networks and encrypt elements of Σ^n, where Σ is an arbitrary alphabet and $n > 1$. The constraint $n > 1$ precludes the direct use of $\Sigma = \mathbb{Z}_p$, meaning none of these encryptions can be directly applied to the domain $\mathbb{Z}_p$.

However, these schemes can be used to encrypt messages in the domain $\{0,1\}^n$, where n is the bit-size of p. Combining this with the "cycle walking" technique described by Black and Rogaway [24] provides an encryption from $\mathbb{Z}_p \to \mathbb{Z}_p$. Given an encryption $\mathsf{Enc}\colon \{0,1\}^n \to \{0,1\}^n$ and a message space $\mathcal{M} \subset \{0,1\}^n$, the construction in Figure 3.2 results in an encryption $\mathcal{M} \to \mathcal{M}$.

If Enc is a permutation of $\{0,1\}^n$, this method traverses the orbit of x under Enc in $\{0,1\}^n$ until it reaches an element in $\mathcal{M}$. This approach is effective when the ratio $\frac{|\mathcal{M}|}{2^n}$ is high enough to avoid long encryption chains. Moreover, if Enc is a truly random permutation of $\{0,1\}^n$, this method yields a truly random permutation of $\mathcal{M}$. Cycle walking can be used with any cipher whose domain includes $\mathbb{Z}_p$.

Cycle Walking-($\mathsf{Enc}, \mathcal{M}$)

input: m, k
$c \leftarrow \mathsf{Enc}_k(m)$
while $c \notin \mathcal{M}$:
  $c \leftarrow \mathsf{Enc}_k(c)$
return c

Figure 3.2: Cycle walking construction.

Both-Ends Encryption

input: k, s
$\mathsf{Enc} \leftarrow$ block cipher with block size b
$l \leftarrow$ bit-size of s
$x \leftarrow l \bmod b$
$t \leftarrow \mathsf{Enc}_k(s[0, l-x)) \| s[l-x, l)$
$r \leftarrow t[0, x) \| \mathsf{Enc}_k(t[x, l))$
return r

Figure 3.3: FPE from both ends.

Using stream ciphers would essentially allow constructing an encryption with domain $\{0,1\}^{\lfloor \log p \rfloor + 1}$ (i.e. the bit-size of p). Such an encryption would be "optimal" in terms of binary string encryptions, as no other encryption would require fewer cycles in Figure 3.2. However, this approach necessitates tracking initial seeds throughout the chaining process and, importantly, the operation would be easily parallelisable.

Therefore, we propose a novel and efficient FPE method to use in conjunction with cycle walking (Figure 3.3). While we provide a brief intuition regarding the security of such scheme, a formal proof is left as future work since it is outside our area of expertise and would fall beyond the scope of this thesis. Essentially, we encrypt the message twice with a non-parallelisable mode but without padding. Since each call to $h(x)$ is modelled as the same random permutation, we can fix one initialisation vector for all applications.

Intuitively, this construction is a PRP if the underlying encryption is. More specif-

ically, every part of the input is encrypted, and truncation of a PRP is also a PRP. If AES is used as the underlying PRP, composition should not pose problems, as it is roughly equivalent to increasing the number of rounds within a single AES call.

To find the most efficient FPE method, we computed the number of single-block encryption calls required by the following four constructions:

1. AES with cycle walking: An n-bit prime is treated as a string of $\lceil \frac{n}{128} \rceil$ bits.
2. AES-based stream cipher (e.g. counter mode).
3. Both-Ends Encryption with cycle walking.
4. FF1 method from NIST standards with cycle walking.[5]

Let p be an n-bit prime and $k = \frac{n}{128}$. Modelling each encryption as a random permutation, the expected number of cycle walking iterations is $\frac{2^b-1}{p}$, where b is the encryption domain's bit-length. Specifically:

- For stream cipher, Both-Ends Encryption, and FF1: $b = n$
- For plain AES: $b = 128\lceil k \rceil$

Thus, the expected number of encryptions is:

- AES: $\lceil k \rceil \frac{2^{128\lceil k \rceil}-1}{p}$
- Stream cipher: $\lceil k \rceil \frac{2^n-1}{p}$
- Both-Ends Encryption: $2\lfloor k \rfloor \frac{2^n-1}{p}$
- FF1 (at least)[6]: $10 \left(\left\lceil \frac{\lceil \lceil \lceil n/2 \rceil /8 \rceil /4 \rceil +1}{4} \right\rceil + \left\lfloor \frac{\lceil \lceil n/2 \rceil /8 \rceil +1}{16} \right\rfloor \right) \frac{2^n-1}{p}$

Table 3.2 illustrates these formulas using two safe primes of similar size, one fitting the AES block size almost perfectly and the other not: $1030710193 \cdot 2^{44001} + 3$ and $178787695 \cdot 2^{44001} + 1$. All numbers are rounded to the nearest integer.

[5] We do not consider FF3-1, as its specifications don't support large domains like those needed for primes of 4000 or more bits.

[6] The specifications allow for the definition of an extra input "tweak" which could increase the number of block encryptions.

Prime bit-size	AES only	FF1	Both-Ends	Stream Cipher
$344 \cdot 128 - 1$	717	3604	715	358
$344 \cdot 128 - 3$	4132	5195	1030	516

Table 3.2: Comparison of FPE methods in terms of number of AES block encryptions.

The stream cipher approach is clearly the most efficient, requiring the fewest AES applications to "cover" all n bits. However, it lacks sequentiality, as previously noted. Our Both-Ends encryption, requiring roughly twice the operations, outperforms the remaining methods while providing some sequentiality.

3.2.3 Card Shuffling

Another approach to generating pseudo-random permutations on arbitrary domains is the design of "card shuffling" algorithms. Assume the message space is $\{0, \ldots, N-1\}$, then message $i-1$ is represented by the i^{th} card in a deck of N cards. After shuffling the deck, that card will end up in some position $j \in \{0, \ldots, N-1\}$, and we say that the encryption of i is j.

An *oblivious* shuffle is a shuffle where the trajectory of a card can be computed without calculating the paths of the other cards. Thus, computing the encryption of i does not require computing the encryption of any other message. In the literature, we found only a few FPE schemes based on oblivious card shuffling [81, 118, 119, 136]. We present two options that we will benchmark in Section 3.4.

Thorp Shuffle

The earliest card shuffle in the literature is the Thorp shuffle (Figure 3.4). Following the description by Morris *et al.* [119], the shuffle is performed as follows: cut the deck in half and pick up each half; according to a coin flip, drop the last card from either the left or right deck. Figure 3.4 shows the algorithm for the shuffle, computing only the trace of the input. To reverse the shuffle, the sampling of the coin flip (variable b in Figure 3.4) must be deterministic.

Thorp

input: N, x
repeat R **times** :
  $y \leftarrow x + \frac{N}{2} \mod N$
  $b \xleftarrow{\$} \{0,1\}$
  if $x > y$: $x \leftarrow 2x + 1 - b$
  else : $x \leftarrow 2x + b$
return x

Figure 3.4: Thorp shuffle.

Swap-or-Not Shuffle

The swap-or-not shuffle by Hoang *et al.* [81] follows a similar structure: pair cards x and $K - x \mod N$ and swap them according to a coin flip. Figure 3.5 shows the algorithm for this shuffle, and again, the coin flip must be deterministic. Hoang *et al.* suggest simulating the coin flip using round functions F_i applied to $\max(x, y)$. As a result, the sequence of coin flips cannot be precomputed, as each flip depends on the current state of the algorithm.

Swap-Or-Not

input: N, x
repeat R **times** :
  $K \xleftarrow{\$} \{0, \ldots, N-1\}$
  $y \leftarrow K - x \mod N$
  $b \xleftarrow{\$} \{0,1\}$
  if $b = 1$: $x \leftarrow y$
return x

Figure 3.5: Swap-Or-Not shuffle.

Limitations of Card Shuffling for Our Purposes

While both of these methods appear highly sequential and proven to be secure, they are not practical for our purposes. The number of iterations required to achieve good security is in $\mathcal{O}(\log p)$. Since we work with very large primes, this approach

is far less efficient than the FPE methods discussed earlier. However, we presented both schemes as they more closely embody the random permutation assumption, which is used to guarantee the security of our TLP.

3.3 Comparison to Other TLPs

We have concluded the explanation of our construction. We now consolidate everything, ensuring we state all the assumptions underpinning our work. Specifically, $\mathsf{Delay}(m) = (d \circ h)^C(x)$ for some parameter C, where $d(x) = x^3 \mod p$ for a safe prime p, and $h(x)$ is a PRP, such as the Both-Ends Encryption method explained above. The cubing operation acts in $\mathbb{Z}_p^*$, but our security proof works in any cyclic group of order $2q$ for a prime q. A slight modification of Theorem 3.3 would prove security in any (cyclic) group of prime order.

In this section, we clearly state all the assumptions required for the effective use of our TLP. Additionally, we compare our scheme to both the RSW and iterated hashing. Our findings are summarised in Table 3.3.

	RSW	Cubing	Hash chain
Puzzle Generation	$\mathcal{O}((\log T + N)M)$	$\mathcal{O}(2TM/N)$	$\mathcal{O}(T)$
Puzzle Solution	$\mathcal{O}(TM)$	$\mathcal{O}(TM)$	$\mathcal{O}(T)$
Practical Efficiency	✓	✓	✗
Quantum Annoying	✗	✓	✓
Quantum Secure	✗	✓*	✓

Table 3.3: Comparison between TLPs, where T is the time parameter, M the cost of modular multiplication and N the bit size of the modulus. *Under Assumption 3.4.

3.3.1 Assumptions

Assumption 3.1 (Repeated Squaring in a Group of Known Order)**.** *To compute* $x^{2^t} \mod p$*, one needs to perform at least* t *sequential squarings, where* $2^t < p$*.*

This assumption guarantees the sequentiality of repeated squaring in a group of known order. This is strictly weaker than the assumption of repeated squaring in

a group of unknown order. We have proven this assumption holds in the strongly algebraic group model (SAGM), modelling group operations as a single unit.

Assumption 3.2. *Given a safe prime p and a random element $x \in \mathbb{Z}_p^*$, computing $x^3 \bmod p$ sequentially is faster than computing $x^{\frac{2p-1}{3}} \bmod p$, even with access to polynomially many processors.*

This assumption ensures that the function d creates a delay. It's stronger than Assumption 3.1, as a single squaring might be highly parallelisable, making inverting the cubing faster. In Section 3.5, we'll see that modular multiplication is easily parallelisable, with an asymptotic complexity of $\mathcal{O}(\log \log p)$. However, we believe this assumption is plausible in practical scenarios.

Assumption 3.3. *The Both-Ends Encryption is a secure pseudo-random permutation.*

This assumption states that our choice of $h(x)$ is a secure PRP. One could choose another scheme analysed in Section 3.2. Any function safely modelled as a true permutation would satisfy our security requirements.

Note that Assumption 3.3 is the only one truly required for some security level. If it were the only assumption to hold, our construction could be implemented with a "small" prime, effectively becoming akin to a hash chain, but with the advantage of using block ciphers instead of hash functions. In particular, CPU instructions for AES are more widely supported than, say, SHA-256 instructions. Hence, we expect minimal improvements from software optimisations when working with AES.

Assumption 3.4. *Computing the discrete logarithm modulo p with non-negligible probability using only $\lfloor \log p \rfloor / s$ live (i.e. not precomputed) sequential squarings requires solving a Closest Vector Problem instance with dimension close to s.*

This assumption aims to rule out large speed-ups, as the CVP problem is impractical in large dimensions. Assuming this holds, quantum computers would only offer a performance boost comparable to better hardware. Thus, we argue this primitive would be quantum-safe.

3.3.2 Quantum Considerations

While our Assumption 3.4 is about the computation of a discrete logarithm, the discrete logarithm does not directly solve the TLP. However, in Section 3.5.5, we will see that it provides a way to compute cube roots using only $\mathcal{O}(\log \log p)$ sequential modular multiplications rather than $\mathcal{O}(\log p)$.

In the event that our quantum assumption does not hold, our scheme can be regarded as "quantum annoying". This concept, formalised by Eaton and Stebila [57] in the context of password-authenticated key exchanges, refers to protocols where each password guess requires a discrete logarithm computation. Thus, even with relatively practical quantum computers, attacks remain impractical due to the high cost of computation.

In our context, "quantum annoying" means that quantum computation is required for each use of $d(x)$. Therefore, quantum computation must be both efficient and cheap to break our scheme. Moreover, even if many puzzles use the same group, it provides no real advantage to a quantum adversary.

In the event that new, faster quantum algorithms are discovered for computing the discrete logarithm or solving the Closest Vector Problem, the level of quantum security offered by our scheme would need to be reevaluated. However, even in this scenario, the "quantum annoying" property would still hold, making attacks computationally costly even with efficient quantum algorithms.

3.3.3 Comparison with RSW

The most widely used and perhaps most similar TLP is the RSW [137] puzzle. While many variations exist in the literature, we focus on the original work, as all modifications fundamentally rely on the same assumptions. We refer the reader to Chapter 2 for a detailed explanation of the RSW puzzle.

The RSW puzzle relies on the assumption that repeated squaring in a group of unknown order is a sequential process. This is somewhat equivalent to our Assumption 3.1, although it is stronger. Similarly, our Assumption 3.2 is analogous

to the idea that RSW puzzle generation is faster than puzzle solution when the solver has access to polynomially many processors.

Delay and Efficiency

The key advantage of the RSW puzzle is that the induced delay is somewhat independent of the modulus size. Trivial puzzle generation takes $\mathcal{O}((\log T + \log N)(\log N \log\log N))$, as it requires $\mathcal{O}(\log T)$ modular multiplications to compute $a = 2^T \bmod \phi(N)$ and $\mathcal{O}(\log a) = \mathcal{O}(\log N)$ modular multiplications to compute $r^a \bmod N$. Solving the puzzle requires T modular multiplications, computable in $\mathcal{O}(T \log\log N)$ time by parallelising modular multiplication, or even $\mathcal{O}(T)$ in theoretical circuits (see Section 3.5.3).

In our puzzle, the additional work a solver performs compared to the puzzle generator is directly linked to the prime's bit-size. It takes two modular multiplications (i.e. $\mathcal{O}(\log p \log\log p)$) to create the puzzle and $\log p$ modular multiplications to solve it with a theoretical circuit of depth $\mathcal{O}(\log p)$. This loss of puzzle generation efficiency is the trade-off for removing the unknown order assumption.

Chaining and Information Leakage

Another notable characteristic is the use of chaining in our construction. This allows us to decouple the delay imposed on the solver from the prime's bit-size, at the cost of a more expensive puzzle generation step. Moreover, chaining with PRPs introduces security guarantees. Specifically, until all delays $d(x)$ are inverted, any two puzzle solutions are plausible. This is not the case in the RSW puzzle, where information about the final value $r^{2^T} \bmod N$ might "leak" before all T squarings are computed. This has led some RSW-related works (e.g. [30, 100]) to require assumptions about the "pseudo-randomness" of the squaring operation, similar to the Blum-Blum-Shub number generator [27].

Chaining RSW Puzzles

Chaining RSW puzzles has been proposed mainly in the context of VDFs. However, Abadi and Kiayias [1] studied this, aiming to release messages at different times. Their solution is to include the seed for the repeated squaring of a puzzle within the previous puzzle's solution. This is similar to choosing $h(x)$ as XOR masking, where after the repeated squaring you would add a number to obtain the next puzzle's starting point. In our context, where intermediate values are not of interest, the main downside is that the puzzle size grows linearly with the number of chainings.

Quantum Considerations

Shor's algorithm [143] can compute the order of any finite abelian group, rendering the repeated squaring assumption invalid in any such group. Additionally, the ability to compute discrete logarithms would lead to the same attack our puzzle suffers from (explained in Section 3.5.5).

While the latter attack fails to provide a large speed-up under Assumption 3.4, unlike our cubing TLP, the RSW puzzle cannot be considered quantum-safe due to Shor's algorithm. In other words, even with our assumption holding, a quantum adversary can break the core assumption of the RSW puzzle.

The RSW puzzle can be made "quantum annoying" by using fresh moduli for each puzzle, meaning that a quantum computation is required to break each puzzle instance. However, this significantly increases the cost of puzzle generation, making it less practical for many applications.

3.3.4 Comparison with Hash Chains

As noted earlier, $d \circ h$ can be viewed as a single permutation. Thus, our construction can be seen as a hash chain, and hash-chain puzzles could potentially be adapted to our construction. Moreover, by choosing large primes, the function $d \circ h$ can be considered memory-hard, introducing the benefits of ASIC resistance [131].

A key difference between our scheme and standard hash chains is that, under Assumptions 3.1 and 3.2, our puzzle generation is faster than puzzle solution. In contrast, for traditional hash chains, an adversary with better hardware can solve the puzzle in less wall clock time than it took the honest party to construct it.

Assumptions and Parallelism

While Assumptions 3.1 and 3.2 don't have direct equivalents in hash-chain puzzles, hash-chain puzzles still rely on the assumption that the puzzle solver cannot parallelise each individual hash computation.

We note that our choice of AES-based PRPs might be advantageous over using standard hashes like SHA-3. Modern CPUs are built with AES instructions, making optimisations on the software level less effective. While SHA-256 instruction sets are available for some CPU architectures, they are not as consistently supported (e.g. only the last three generations of Intel Core).

Quantum Security

The key advantage of hash-based TLPs is their security against quantum computation. Afshar *et al.* [5] proved the security of hash chains in the random oracle model when the adversary has access to quantum computation. In contrast, our construction requires certain assumptions about the speed of quantum discrete logarithm algorithms to achieve quantum safety.

3.4 Proof of Concept Implementation

To evaluate the performance of our TLP, we implemented it in C, utilising the GMP library [151] and OpenSSL [127]. The GMP library focuses on high performance through its low-level assembly implementations of various primitives, and the GMP community invests significant effort in tailoring code for diverse processor pipelines. Similarly, OpenSSL is a highly optimised library that leverages AES-specific CPU instructions.[7]

[7] All the code is available online at https://github.com/Ivo-Maffei/TreCubing/tree/main.

3.4.1 Random Number Generation

For random number generation, we used either a 64-bit Xorshift RNG from the Marsaglia family [113] or GMP's default Mersenne-Twister-based RNG. Xorshift was used for generating AES keys and simulating coin flips in card shuffling algorithms, where its relatively weak randomness is sufficient, and its speed is advantageous. GMP's RNG was used to pick random messages for testing primitives and generating round keys for the Swap-Or-Not algorithm (Figure 3.5). We neglected the padding scheme in all tests, as its cost is negligible compared to the other primitives.

3.4.2 Testing Methodology and Results

Each primitive was tested with 100 different messages, and we report the mean of the samples. The standard error (standard deviation over the square root of the number of samples) is so small that it would not be visible in the graphs. Tests were conducted on both a 2016 laptop (Intel i7-6920HQ) and an Oracle Cloud VM.Standard3.Flex VM (latest-generation Intel Xeon Platinum 8358).

Figure 3.6 shows that the time required to compute the cube root grows quadratically with the modulus bit-size. Interestingly, as shown in Table 3.4, the speed difference between the two machines is bounded by 5%, with the older laptop often faster.

Figure 3.7 shows the performance of various chaining methods. Timings are computed by running Thorp and Swap-Or-Not with n iterations, where n is the prime modulus' bit-size. The Both-Ends Encryption method is significantly more efficient than any card-shuffling technique, averaging below 0.02 ms on both machines across all bit-sizes. The Oracle VM performed better in these tests, achieving speed-ups between 30% and 70%. However, since all timings are minimal, this does not significantly impact the efficiency or effectiveness of our TLP.

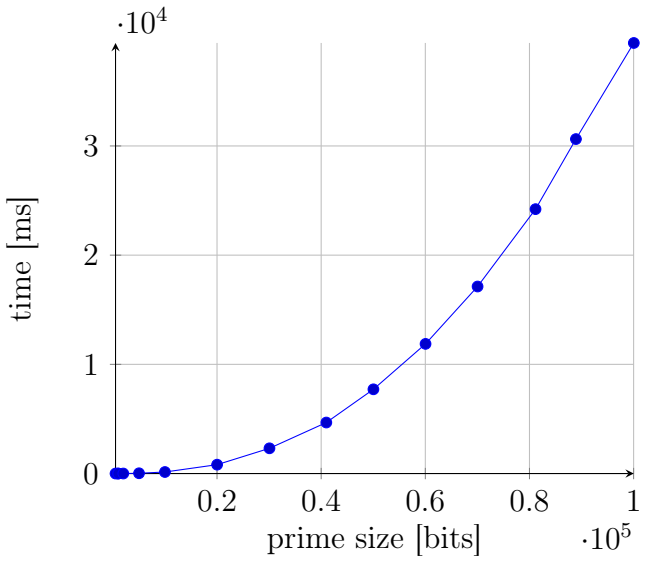

Figure 3.6: Time required to compute the cube-root.

bit-size	Oracle VM time difference (%)
512	-5.5
1001	0.73
1025	0.59
2001	2.8
5024	-2.1
9999	3.6
20016	4.0
30072	3.1
40991	4.8
50034	3.4
60043	1.3
70034	-0.58
81163	0.49
88921	-0.31
100037	1.0

Table 3.4: Time difference for computing the cube root.

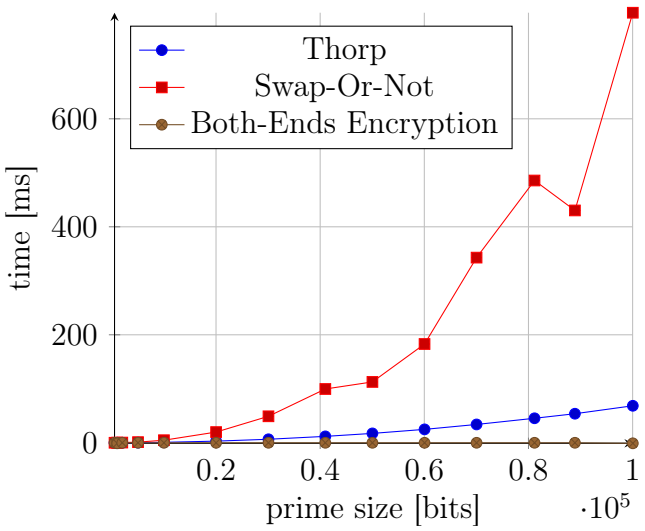

Figure 3.7: Time required to compute Thorp and Swap-Or-Not with bit-size rounds.

3.5 Security Analysis

While our TLP's sequentiality is proven within the strongly algebraic group model, it is essential to discuss how this translates to real-world applications. Theoretical proofs on the sequentiality of repeated squaring, such as ours and [90, 140], assume that multiplication is an atomic operation and are always proven in abstract theoretical models.

In this section, we explore whether it is possible to accelerate cube-root extraction in modular arithmetic. First, we demonstrate that modular exponentiation reduces to repeated squaring and examine whether the entire exponentiation or a single multiplication can be parallelised. We then consider enhancing performance through faster hardware and leveraging quantum computation.

The final section will motivate our Assumption 3.4 on post-quantum security. Table 3.5 provides a brief summary of the different avenues to attack our scheme. Apart from the first approach, our discussion applies to any TLP based on the repeated squaring assumption. Importantly, all these techniques can be used to challenge constructions based on RSW.

Approach	Practical	Gains
Reduction to repeated squarings 3.5.1	Yes	< 2x
Parallelisation of integer multiplication 3.5.2	For very large moduli	Limited
Implementation of theoretical circuits 3.5.3	For small moduli	< 30x
Faster hardware 3.5.4	Yes	< 20x
Leveraging quantum computation 3.5.5	For small moduli	Moderate

Table 3.5: Approaches to speedup cube-root extraction.

3.5.1 Repeated Squarings

We briefly explain why the sequentiality of modular exponentiation depends entirely on the repeated squaring assumption. Since the introduction of the RSA cryptosystem, researchers have sought to optimise modular exponentiation. Let a, b, and m be integers, with b having n bits. All optimisations in the literature aim to reduce the number of modular multiplications needed to compute $a^b \bmod m$. However, there seems to be an implicit lower bound of $n-1$ multiplications.

As an example, consider the "sliding window" algorithm. Interpreting b as an n-bit string and assuming $n = Nw$ for some "window size" w, we can write $b = \sum_{i=0}^{N-1} b[iw, iw+w)2^{iw}$, and thus $a^b = \prod_{i=0}^{N-1} a^{2^{iw} b[iw,iw+w)}$. Figure 3.8 shows a high-level overview of the sliding window algorithm, as implemented in the GMP library.

Like other methods, this technique aims to reduce the multiplications required beyond repeated squaring. Moreover, these extra multiplications can be parallelised. Regardless of the values of $k_i = T[b[i, i+w)]$, for any $i \neq j$, $a^{k_i 2^{iw}}$ and $a^{k_j 2^{jw}}$ can be computed independently and multiplied together as they become available. As a result, the average number of sequential multiplications needed is $n + \frac{w-1}{2}$, with an implicit lower bound of $n-1$.

Therefore, for the rest of this analysis, we focus entirely on the repeated squaring process, but without restrictions on the modulus. This analysis covers both our construction and any other based on repeated squaring, whether the underlying group order is known or not. We note that we focus on integers; some results

Sliding Window Exponentiation

input: a, b, w
output: a^b
construct table T of size 2^w
foreach $x \in \{0,1\}^w$:
  $T[x] = a^x$
$c \leftarrow T[b[0, w)]$
$i \leftarrow w$
repeat $\frac{n}{w} - 1$ **times** :
  repeat w **times** :
    $c \leftarrow c^2$
  $c \leftarrow c \cdot T[b[i, i + w)]$
  $i \leftarrow i + w$
return c

Figure 3.8: Sliding window exponentiation assuming $n = 0 \mod w$.

might not directly apply to other groups like elliptic curves or class groups. The question of applying these attacks to generic groups is left for future research.

Another frequent operation in solving repeated-squaring TLPs is modular reduction. The most common algorithms are Montgomery's [117] and Barrett's [15] reductions. Both compute modular reduction using two integer multiplications plus a few additions/subtractions. Despite similar time complexity, Montgomery's reduction is more widespread, being the default in GMP and used in recent VDF research ([115, 167]).

Figure 3.9 shows the pseudocode for Montgomery's reduction. Given a and b, it computes $aR^{-1} \mod b$ for a fixed R. To use this algorithm, the modular exponentiation must be in "Montgomery form". That is, to compute $c^2 \mod b$, convert the base c to $c_M := cR \mod b$. Then, $\text{REDC}(c_M^2) = c^2 R \mod b$. The final result must also be converted back from Montgomery form, which can be done using REDC again. For REDC to work correctly, we need $a < Rb$ and $\gcd(R, b) = 1$. Moreover, R should be a power of 2 for efficiency.

REDC

input: a, b, R

output: $aR^{-1} \bmod b$

$b' \leftarrow b^{-1} \bmod R$ (can be precomputed)

$m \leftarrow (a \bmod R) \cdot b' \bmod R$

$t \leftarrow (a + m \cdot b)/R$

if $t > b$:

  $t \leftarrow t - b$

return t

Figure 3.9: Montgomery's reduction algorithm.

Barrett's reduction is based on the observation that $a \bmod b = a - \left\lfloor \frac{a}{b} \right\rfloor b$. Computing $\left\lfloor \frac{a}{b} \right\rfloor$ can be done efficiently using formulas like $\frac{a}{2^k} \cdot \frac{2^{2k}}{b} \cdot \frac{1}{2^k}$, where the middle term can be precomputed. With careful exponent choices, $\left\lfloor \frac{a}{b} \right\rfloor$ can be approximated using bit shifts and a single multiplication. Thus, the theoretical complexity (sequential and circuit depth) is similar to Montgomery reduction.

Other approaches involve lookup tables, as seen in recent works by Cao *et al.* [37], Will and Ko [160], and Öztürk [167]. These methods precompute $x \mapsto x \bmod b$ for specific x values and represent a with a suitable vector of x's, leading to an addition-based algorithm. However, they have worse complexity on sequential Turing machines and achieve the same depth in the circuit model.

We have demonstrated that the repeated squaring problem can be reduced to computing integer multiplications. Furthermore, all theoretical results supporting the sequential nature of the repeated squaring assumption depend on multiplication being an atomic operation. Therefore, we will examine this assumption in the next section.

3.5.2 Parallelisation of Integer Multiplication

In this section, we analyse known integer multiplication algorithms to show how they can be parallelised. Let a and b be two n-bit numbers; we aim to compute ab.

The Toom-Cook algorithms [28] are a family of multiplication algorithms, where the well-known Karatsuba-Ofman multiplication is equivalent to Toom-2. The asymptotic complexity of the Toom-k algorithm is $\mathcal{O}\left(n^{\frac{\log(2k-1)}{\log k}}\right)$. Figure 3.10 describes this algorithm.

Toom-k

input: a, b both n bits long
output: ab
$\lambda \leftarrow 2^{\lfloor \frac{n}{k} \rfloor}$
compute a_i's such that $a = \sum_{i=0}^{k-1} a_i \lambda^i$
compute b_i's such that $b = \sum_{i=0}^{k-1} b_i \lambda^i$
$A(x) \coloneqq \sum_{i=0}^{k-1} a_i x^i$
$B(x) \coloneqq \sum_{i=0}^{k-1} b_i x^i$
pick a set $\mathcal{P} = \{p_1, \ldots, p_{2k-1}\}$
$\mathcal{V} \leftarrow \{v_i \mid v_i = A(p_i)\, B(p_i) \quad \forall\, p_i \in \mathcal{P}\}$
$P(x) \leftarrow$ polynomial such that $P(p_i) = v_i$ for $1 \leq i \leq 2k-1$
return $P(\lambda)$

Figure 3.10: Toom-Cook algorithm.

The choice of the set of points $\mathcal{P}$ significantly impacts the algorithm's performance. The values p_i are chosen to make computing $A(p_i)$ and $B(p_i)$ efficient. Additionally, the interpolation to obtain $P(x)$ is often hardcoded to minimise the number of operations.

This algorithm can be parallelised by computing each v_i on concurrent threads. However, our attempts to parallelise GMP's implementation using OpenMP [126] did not yield performance improvements for primes up to 100000 bits. Thus, this low-level parallelisation is far from trivial.

Another important family of multiplication algorithms is based on Fast Fourier Transforms. These algorithms share a similar principle with the Toom-Cook family:

the multiplicands a and b are interpreted as polynomials $A(x)$ and $B(x)$. The FFT is used to evaluate these polynomials at the n^{th} roots of unity in a chosen field. The algorithm recursively computes the multiplication of these points and then interpolates these to obtain the polynomial $P(x) = A(x)B(x)$.

Notable FFT algorithms include the Schönhage-Strassen algorithm [71] with a time complexity of $\mathcal{O}(n \log n \log \log n)$, and the Harvey and van der Hoeven algorithm [80] with a time complexity of $\mathcal{O}(n \log n)$.

Like the Toom-Cook family, FFT algorithms can be parallelised. However, on consumer-grade hardware, they are only worthwhile for multiplicand sizes well beyond the 100000-bit limit considered in our implementation.

In summary, while integer multiplication can be parallelised, software parallelisation is not feasible for the number sizes we consider. As we will see in the following sections, significant performance improvements require the use of ad-hoc circuits or GPUs.

3.5.3 Theoretical Circuits

We have observed that modular exponentiation reduces to the repeated squaring assumption and that integer multiplication can be parallelised. Although this parallelisation appears practical only for very large numbers, we discuss the theoretical results from complexity theory. We present findings on the sequential nature of multiplication and exponentiation within the standard circuit model, where each logic gate has a maximum of two input bits.

Standard textbooks [156] provide the following results:

Theorem 3.7. *Let x, y be two n-bit numbers. There exists a circuit of depth $\mathcal{O}(\log n)$ and size $\mathcal{O}(n)$ that computes $x + y$.*

Theorem 3.8 (Carry-Save Adders [56])**.** *Let $x_1, \ldots, x_k$ be k n-bit numbers. There exists a circuit of depth $\mathcal{O}(\log(kn))$ and size $\mathcal{O}(n \log k)$ that computes $\sum_{i=1}^{k} x_i$.*

Theorem 3.9. *Let x, y be two n-bit numbers. There exists a circuit of depth $\mathcal{O}(\log n)$ and size $\mathcal{O}(n^2)$ that computes xy.*

Beame

input: n-bits numbers $x_1, \ldots, x_k$

output: $\prod_{i=1}^{k} x_i$

$\mathcal{P} \leftarrow \left\{ p_i \text{ prime} \;\middle|\; \prod_{i=1}^{h} p_i > \prod_{i=1}^{k} x_i \right\}$

foreach p **in** $\mathcal{P}$:

  $g_p \leftarrow$ generator of $\mathbb{Z}_p^*$

  $T_p \leftarrow$ a table such that $T_p[i] = g_p^i \mod p$

foreach p **in** $\mathcal{P}$:

  foreach i **in** $\{1, \ldots, k\}$:

    $r_i \leftarrow \log_{g_p}(x_i \mod p)$ using table T_p

  $r \leftarrow \sum_{i=1}^{k} r_i \mod (p-1)$

  $y_p \leftarrow T_p[r]$

using CRT find y so that $\forall p \in \mathcal{P} \;\; y \equiv y_p \mod p$

return y

Figure 3.11: Beame *et al.*'s iterated multiplication circuit.

These bounds are tight. Any function on n bits that uses all of them requires a circuit of depth $\Omega(\log n)$ just to "read" the input. Recent work also proved a precise depth lower bound for multiplication: $2\log(n-1) - 2\log(\log(n-1)-1) - 4$ [158]. This contrasts with the recently discovered $\mathcal{O}(n \log n)$ sequential multiplication algorithm.[80] Regarding lower bounds for sequential multiplication, apart from the trivial $\Omega(n)$, no tighter bounds have been proven. Afshani *et al.* showed that any circuit computing integer multiplication must have size $\Omega(n \log n)$, assuming an unproven conjecture. Thus, multiplication is widely believed to be $\Theta(n \log n)$ when run sequentially and $\Theta(\log n)$ with polynomial parallelism.

Perhaps more surprisingly, the following theorems demonstrate that integer exponentiation can also be parallelised using the Chinese Remainder Theorem (CRT).

Theorem 3.10 (Beame *et al.* [17])**.** *Let $x_1, \ldots, x_k$ be k n-bit numbers. There exists a circuit of depth $\mathcal{O}(\log n + \log k)$ that computes $\prod_{i=1}^{k} x_i$.*

Figure 3.11 shows Beame *et al.*'s proposed circuit. The idea is to break the problem into smaller instances using the CRT. Given coprime a and b such that $ab > x^2$, we can compute $x^2 \bmod a$ and $x^2 \bmod b$ in parallel, then use the CRT to retrieve $(x^2 \bmod ab) = x^2$. The key is that we can choose primes such that the largest one is only logarithmically sized compared to the final result.

Specifically, the bit-size of the product of the first k primes is approximated by the first Chebyshev function $\vartheta(x) = \sum_{p \text{ prime}}^{p \leq x} \ln p$. For p_k denoting the k^{th} prime, $\vartheta(p_k) \in \Theta(k \log k)$. To compute the k^{th} power of an n-bit number (or the product of k n-bit numbers), we can work modulo roughly the first nk primes, as their product will have bit-size $\mathcal{O}(nk \log(nk))$, accommodating the result. The nk^{th} prime is in $\Theta(nk \log(nk))$, thus having bit-size $\Theta(\log(nk))$. By precomputing exponentiation and reduction tables for primes of that size in time $\mathsf{poly}(nk)$, all operations modulo the primes can be performed in depth $\mathcal{O}(\log(nk))$. Thus, the overall circuit depth complexity is $\mathcal{O}(\log(nk))$. This proves that x^k can be computed in depth $\mathcal{O}(\log(nk))$ when x is an n-bit number.

Furthermore, reducing a kn-bit number modulo an n-bit number (i.e. $x^k \bmod y$) can also be done in depth $\mathcal{O}(\log n)$, provided k is polynomial in n, by precomputing tables for the values $2^i \bmod y$.

Interestingly, this shows that $x^k \bmod y$ can be computed without repeated squaring. The trick lies in the ability to compute the discrete logarithm in the prime moduli and to compute exponentiation in those moduli with a simple lookup.

By iterating this approach, we can compute $x^{k^a} \bmod y$ in depth $\mathcal{O}(a \log(nk))$. Choosing $a = \frac{t}{\log k}$ and $k = n$, we can compute $x^{2^t} \bmod y$ in depth $\mathcal{O}(t)$ with precomputation and circuit size polynomial in n.

Hamano *et al.* [79] went a step further by providing an explicit circuit to compute modular exponentiation, not just repeated squaring.

Theorem 3.11 (Hamano *et al.* [79]). *Let x, y, and m be three n-bit numbers. For any $\alpha > \frac{1}{\log n}$, there exists a circuit of depth $\mathcal{O}\left(n\frac{\alpha+1}{\alpha}\right)$ and size $\mathcal{O}(n^{3+2\alpha}/(\alpha \log n))$ that computes $x^y \bmod m$.*

Hamano

input: x, y, m with $x, y \leq m$ and m of n bits
output: $x^y \mod m$
fix parameter α
$k \leftarrow \lceil \alpha \log n \rceil$
create empty table T of size n
$z \leftarrow x$
for r **in** $\left\{1, 2, \ldots, \frac{n}{k}\right\}$:
  for i **in** $\{1, 2, \ldots k\}$: (in parallel)
    $T\left[(r-1)k + i\right] \leftarrow z^{2^i} \mod m$
  $z \leftarrow T\left[rk\right] \mod m$
let $y = \sum_{i=0}^{n-1} y_i 2^i$
return $\prod_{i=1}^{n} y_i T[i] \mod m$

Figure 3.12: Hamano *et al.*'s modular exponentiation circuit.

Hamano *et al.* construct a circuit for Theorem 3.11 using the high-level structure shown in Figure 3.12. The key step is the computation of $z^{2^k} \bmod m$ in depth $\mathcal{O}(k)$. While they suggest using a circuit from Okabe *et al.* [125], the circuit by Beame *et al.* [17] (Figure 3.11) offers greater memory efficiency while maintaining the same depth.

Practical Considerations

Theoretically, these results show that the entire repeated squaring process can be carried out in depth linear in the exponent's bit-size t, a significant improvement over the trivial sequential complexity of $\mathcal{O}(tn \log n)$ which is also believed in $\Omega(tn \log n)$. However, the practicality of these results, especially for large bit-sizes, is questionable.

α	N. Primes Used	log Tables	Reduction Tables	Circuit Size
0.062	9367	766 MB	1 GB	10^{15}
0.75	21462282	13 PB	4 TB	10^{20}

Table 3.6: Memory requirements to construct the circuit of Figure 3.12 for $n = 70034$.

There are two main challenges in realising the circuit proposed by Hamano *et al.*: circuit size and memory requirements for precomputation. For instance, to compute $x \bmod p$, the algorithm in Figure 3.11 uses a table of the values $2^i \bmod p$ for $i \in \{1, \ldots, n\}$, where n is the bit-size of x. Additionally, tables for computing the discrete logarithm in each prime modulus must be calculated and stored.

As an example, choosing $\alpha = 1$ and $n = 70034$ leads to a circuit of size (abusing notation) $\mathcal{O}(10^{23})$. According to Wikipedia, the highest transistor count on a single processor is 2.6×10^{12} [159]. Moreover, the fastest supercomputer to date [148] has 9472 CPUs and 37888 GPUs with a total transistor count of 2×10^{15}. Lowering α reduces circuit size, but picking $\alpha = 0.062 \approx \frac{1}{\log n}$ (which would still result in an infeasible circuit size of $\mathcal{O}(10^{15})$) leads to a depth of $\mathcal{O}(n \log n)$. Table 3.6 shows the memory requirements for two α values when $n = 70034$. Memory becomes prohibitive as α approaches values yielding an effective speed-up.

Although achieving the theoretical complexity discussed in the previous section is not feasible, many researchers have focused on designing custom circuits to speed up repeated squaring.

The most acclaimed result is Öztürk's multiplier [167]. This algorithm represents integers as polynomials, similar to those in Section 3.5.2. However, Öztürk's representation is redundant: each polynomial coefficient is stored with an extra bit of memory, and the polynomial itself has a degree one higher than necessary. This redundancy reduces the number of bit carries that must be computed sequentially, resulting in an $\mathcal{O}(\log n)$ sequential time complexity for multiplying two n-bit numbers.

bit-size	Our	Öztürk	Mert	VDF competition
512	70	26	7	N/A
1024	257	N/A	8*	25*

Table 3.7: Comparison of modular squaring latencies obtained by FPGA's implementations. (* simulated not synthesised)

Mert *et al.* [115] further improved this design by integrating modular reduction within the same circuit. Their circuit performs a modular multiplication in depth at most $5 \log n$ [92], very close to the lower bound of $2 \log(n-1) - 2 \log(\log(n-1)-1) - 4$ [158].

The performance of these algorithms shows small latencies compared to our implementation in Section 3.4. Table 3.7 compares the single modular squaring latency of our naive implementation on an older laptop to the simulated performance of Mert *et al.* [115], Öztürk [167], and the winning algorithm from a VDF Alliance competition [6] aimed at minimising circuit latency for solving the RSW TLP.

The table suggests a speed-up factor of around 30 between our implementation and Mert *et al.*'s, and around 10 for the other simulations. While these factors seem large, they are within expectations. It's important to note that these fast implementations come at a high cost in circuit area and memory, making them impractical for even moderately large bit-sizes. For example, Mert *et al.*'s most area-efficient design requires 17 million AND gates and as many (3×2) full adders when operating with 4096-bit numbers. Given that the number of AND gates grows quadratically with the bit-size, working with 40000-bit numbers would require at least 10^9 AND gates.

Therefore, we do not expect to see very large performance speed-ups from circuit implementations, even when performing modular multiplication with relatively large bit-sizes.

Alternatively, Field Programmable Gate Arrays (FPGAs) are often suggested as an alternative way to synthesise theoretical circuits, but we believe commercially available FPGAs wouldn't significantly impact our TLP. Apart from the algorithms

of Table 3.7, other research on FPGA-based multiplication is far from being any concern to us. For example, Woo *et al.* [161] implemented multiplication algorithms on FPGAs, multiplying two 8-bit numbers with a 5-cycle latency. Kakacak *et al.* [89] claimed better timings than the state of the art, but their implementations still require 7 ns to multiply two 64-bit numbers.

In contrast, modern architectures like AMD Zen 2 using GMP require around 1.75 clock cycles per 64 bits for multiplication. With a 3.9 GHz base clock speed, an AMD Ryzen 7 3800X can multiply two 64-bit numbers in roughly 0.4 ns. This demonstrates that current FPGA results in the literature are far from competitive with traditional hardware. Moreover, high-end FPGAs cost two orders of magnitude more than a consumer CPU.

3.5.4 Improvements through hardware

In this section, we discuss how different hardware might lead to performance improvements. Regarding CPUs, single-core performance differences do not appear to be vast. Smartphone CPUs typically have a clock speed around 2 GHz, while top-tier desktop processors can reach up to 6 GHz, bounding the clock speed gap by a factor of 3.

However, different CPU architectures can lead to varying performance, even with equal clock speed. To estimate this, we used a table published on the GMP library website [73], which lists the average clock cycles required for various low-level tasks on different architectures. For example, summing two $64n$-bit numbers takes n cycles on AMD Zen 3 but $8.66n$ cycles on ARM A5 neon. Similarly, squaring a number with the trivial algorithm on an Intel Atom takes roughly 9.7 cycles per 32 bits, compared to 1.41 cycles per 64 bits on an Apple M1 chip. This suggests that the performance gap between older and newer architectures can be much larger than the clock speed difference. Therefore, we conservatively assume that malicious entities could achieve a 15x speed-up compared to mid-range CPUs.

GPUs are another common hardware type used in the literature for parallelising multiplication. However, we could not find a comprehensive comparison of different GPU and CPU architectures. Chang *et al.* [40] multiplied two 192K-bit numbers on a GTX 1070 in 1.12 ms, while GMP on an Intel i7-6920HQ took roughly 5 ms. Emeliyanenko [60] achieved a 10x speed-up with a GeForce GTX 280 compared to a quad-core Intel Xeon E5420 on numbers with tens of thousands of bits. However, Ochoa-Jiménez *et al.* [124] found that GPU implementations of RSA were considerably slower than CPU counterparts for key sizes up to 3072 bits.

Based on available results, we conservatively assume a 20x speed-up when using a GPU compared to mid-range CPUs.

3.5.5 Quantum Computation

In Section 3.1.2, we showed a link between repeated squaring and the discrete logarithm problem. In Section 3.5.3, we presented a circuit that breaks the repeated squaring assumption by using the discrete logarithm in small prime moduli. Now, we present an even simpler attack, assuming the existence of a discrete logarithm oracle.[8] Crucially, this attack can be carried out in any cyclic group for which we know a generator.

Assume we are given an element $c \in \mathbb{G}$ and need to compute c^t. Let g be a generator of $\mathbb{G}$ (or perhaps of a subgroup containing c). Compute $x = \log_g c$, then $g^{tx} = c^t$. While this doesn't provide a direct advantage, as x likely has a large bit-size, the adversary knows the base g before receiving x. Therefore, they could precompute a table of values g^{2^i} for enough indices to ensure $2^i \geq tx$. This would allow computing g^{tx} as a binary tree, requiring only $\mathcal{O}(\log\log(tx))$ sequential multiplications. Essentially, the repeated squaring computation is performed "offline" before receiving the puzzle. For a 100000-bit prime p, the precomputed table would be roughly 1 GB, making this attack feasible.

[8] This attack would still somewhat work even if we could not compute the discrete logarithm directly, but only a multiple of it.

Thus, the ability to compute the discrete logarithm quickly would break the repeated squaring process. In Section 3.3, we discussed how our chaining structure makes our TLP "quantum annoying". In the rest of this section, we estimate the efficiency of discrete logarithm computation, leading to the formulation of Assumption 3.4. After briefly reviewing the state-of-the-art in the classical setting, we dive into quantum algorithms. Since the seminal work by Shor, quantum algorithms for the discrete logarithm problem followed Shor's blueprint. The latest and greatest of these can be found in a recent work by Ekerå. We analyse their algorithm to highlight its limitation and motivate our Assumption 3.4. Our discussion on quantum computation is concluded with an estimate of the hardware needed for a quantum computer capable of challenging our TLP.

Classical Algorithms for the Discrete Logarithm

The fastest classical algorithm for computing the discrete logarithm is the number field sieve. Although it's an exponential algorithm, Adrian *et al.* [4] demonstrate how sufficient precomputation can be used to compute the discrete logarithm in a Diffie-Hellman Key Exchange (DHKE) group in approximately one minute.

The core idea behind their work is as follows. Let the DHKE protocol have parameters p and g, where p is the prime modulus and g is the generator. To compute $\log_g b$ for some b, we first determine a number $l \in [1, p-1]$ such that $g^l b = q_1 q_2 \dots q_t$, where all q_i are "small" primes. Then, $\log_g b \bmod p$ can be quickly deduced from the $\log_g q_i \bmod p$ values. The precomputation phase is spent constructing a database of these $\log_g q_i \bmod p$ values.

While this precomputation can be parallelised, it still demands substantial computational resources. The authors estimate that computing the discrete logarithm in a 1024-bit DHKE group in near real-time would require a multi-billion dollar investment and several years of precomputation.

Given that we work with much larger primes in our setting, we believe that classical attacks would not pose a threat, even from state-level adversaries.

Ekerå's Algorithm

In a quantum world, Shor's algorithm [143] can solve the discrete logarithm problem in polynomial time. However, Shor's algorithm requires computing exponentiation similar to our TLP, rendering it impractical for breaking our TLP. Ekerå [58] showed how to improve Shor's algorithm by reducing the required group operations. Like Shor's, Ekerå's algorithm can be split into two parts: a quantum experiment that produces pairs, and a classical post-processing phase yielding the result. The quantum experiment is shown in Figure 3.13 and is used to compute $\log_g x \bmod p$.

Quantum experiment

input: x, g, p, s
output: j, k
pick m so that $2^{m-1} \leq p < 2^m$
$l \leftarrow \left\lceil \frac{m}{s} \right\rceil$
$a \leftarrow$ superposition of 0 to $2^{m+l} - 1$
$b \leftarrow$ superposition of 0 to $2^l - 1$
$c \leftarrow g^a x^{-b} \mod p$
apply quantum Fourier transform of size 2^{m+l} and 2^l to (a, b, c)
measure the result (j, k, y)
return j, k

Figure 3.13: Ekerå's quantum experiment.

The crucial detail allowing Ekerå's algorithm to be relevant here is that the size of the quantum register b can be bounded using the trade-off parameter s. While the overall number of exponentiations is at least $n + 1$ (where n is the bit-size of p), if we ignore long coherence times, the value of $g^a \bmod p$ can be computed before receiving the encrypted message x. Thus, only the computation on x and the Quantum Fourier Transform (QFT) must be performed during the "delay period". By choosing s appropriately, the computation of $x^{-b} \bmod p$ can be made much smaller than decrypting x through our TLP.

The classical post-processing part of the algorithm (Figure 3.14) is a simplified version suitable for our needs; the full algorithm is in [58]. Computing u from v

Post-processing

input: $\{(j_1, k_1), \ldots, (j_n, k_n)\}, m, s$

output: d

$l \leftarrow \left\lceil \frac{m}{s} \right\rceil$

$f(x) \coloneqq (x \bmod 2^{m+l}) - 2^{m+l} \left\lfloor \frac{x \bmod 2^{m+l}}{2^{m+l-1}} \right\rfloor$

$v \leftarrow (f(-2^m k_1), \ldots, f(-2^m k_n), 0)$

$L \leftarrow$ lattice with basis $\{(j_1, \ldots, j_n, 1)\} \cup \left\{ 2^{m+l} e_i \middle| \ i \in \{1, \ldots, n\} \right\}$

$u \leftarrow$ vector in L closest to v

return $u[n]$ (last entry in the vector)

Figure 3.14: Ekerå's post-processing to compute the discrete logarithm.

is an instance of the Closest Vector Problem (CVP), considered infeasible in high dimensions, so n must be chosen carefully. The dominant step in CVP solvers is the LLL reduction of the lattice basis, which Nguyen and Stehlé [122] proposed a method for with a time complexity of $\mathcal{O}(d^2 n(d + \log B) d \log d \log B)$, where d is the lattice dimension, n is the underlying vector space dimension, and B is the largest vector norm in the basis. Assuming a small n, we believe the post-processing could be computed quickly.

Ekerå [58] analyses the relation between s and n, stating that "a run of the quantum algorithm yields $\approx l \approx m/s$ bits of information on d". Thus, roughly s trials are needed to obtain d with reasonable probability. These trials could be run in parallel, but the post-processing still requires solving the CVP in a lattice of dimension $n + 1$. This limits the speed-up factor due to the potentially high cost of the post-processing for moderately large n values. Remember that during the puzzle's delay period, one must compute the discrete logarithm and then evaluate the binary tree of modular multiplications. Solving the lattice problem is only one step in the entire process.

Gidney and Ekerå [72] analysed real-world requirements for running quantum algorithms like this. They examined various algorithms (including Shor's and Ekerå's)

bit-size	failure chance	size [Mqb]	expected running time [hours]	tiffoli count [$\times 10^9$]
1024	99%	1.5	66	0.3
2048	79%	3.2	120	9.0
3072	88%	4.5	180	7.8
4096	97%	6.4	590	12
8192	67%	15	2100	250
12288	88%	21	5200	240
16384	13%	35	6300	2000
32768	25%	69	8100	4200
65536	45%	190	8800	40000
131072	6%	830	8500	500000

Table 3.8: Minimal size to run the precomputation phase with maximal trade-offs.

for efficiency in size and speed. They found that the cost of implementing the QFT is negligible, and most of the effort is spent computing the modular exponentiation.

Using their code, we estimated the fastest single-run parameters to perform one modular multiplication and the cost of our precomputation phase (computing $g^a \bmod p$ from Figure 3.13). Tables 3.8 and 3.9 show our estimates, indicating how much quantum computation needs to improve before posing a threat to our TLP.

Table 3.8 is computed by minimising the quantum computer size while keeping the expected running time below one year. We used the largest possible trade-off factor $s = n - 1$ for each bit-size n, representing the cost of n modular multiplications on n-bit numbers.

Table 3.9 estimates the cost of a single modular multiplication, minimising latency regardless of failure probability. This differs from the expected running time, as quantum computation may fail due to physical factors, requiring a restart.

3.6 Guidelines for Secure Instantiations

In this chapter, we presented a Time-Lock Puzzle based on cube-root extraction interleaved with pseudo-random permutations. We proved the time-delaying prop-

bit-size	failure chance	size [Mqb]	latency [mins]	expected running time [mins]	tiffoli count [$\times 10^9$]
1024	25%	1.3	0.02	0.03	0.0
2048	68%	2.5	0.04	0.1	0.0
3072	66%	4.9	0.06	0.2	0.0
4096	5%	7.6	0.09	0.1	0.0
8192	8%	16	0.2	0.2	0.01
12288	15%	23	0.3	0.3	0.03
16384	22%	31	0.4	0.5	0.05
32768	75%	61	0.7	3.0	0.2
65536	10%	170	1.6	1.8	0.7
131072	26%	340	3.2	4.3	3.0

Table 3.9: Minimal 1-shot running time to compute a single modular multiplication.

erties of cubing within the Strongly Algebraic Group Model and highlighted the indistinguishability security, even against quantum computation, of the overall scheme. However, we also dedicated significant effort to identifying attacks on the repeated squaring process. We demonstrated that, with precomputation, integer repeated squaring can be massively parallelised using residue number systems or efficient discrete logarithm computation. Moderate speed-ups can also be achieved by parallelising modular multiplication or using high-performance hardware.

This raises the question of selecting secure parameters for instantiating the puzzle. We can choose the bit-size of the modulus and the seed used in the padding. We provided concrete inequalities for the latter, suggesting a rule-of-thumb value of 256 bits. The choice of modulus size is more nuanced.

3.6.1 Choosing the Modulus Size

Two primary approaches exist when selecting the modulus size for our TLP. The first involves using relatively small primes, as in MinRoot [92] or the standard RSW [137], mimicking the assumption that modular multiplication is a single step. By distributing fast and cost-effective ASICs for modular multiplication, we could reasonably assume that honest parties perform this operation almost as quickly as

Attack Type	Small Modulus	Large Modulus
software parallelisation	✗	✓
ASIC for multiplication	✓	✗
ASIC for exponentiation	✓	✗
faster CPU	✗	✓
GPUs	✗	✓

Table 3.10: Comparison of successful attack classes.

the most resourceful adversary. However, a recent Ethereum Foundation report [96] showed that even in this scenario, moderate speed-ups are possible.

A notable advantage of using small primes is the ability to quickly generate safe (or almost safe) pseudo-primes, as discussed in Section 3.1.1. By increasing the frequency of modulus changes, we reduce the effectiveness of attacks reliant on precomputation, leaving faster hardware as the primary potential advantage for adversaries. More research is needed in this direction, but our intuition suggests that it should be possible to efficiently generated moduli where cube-root extraction is hard with overwhelming probability.

The second approach involves using moderately large primes. This strategy aims to increase the memory cost of our function, aligning with Percival's idea [131] that memory-hard functions are ASIC-resistant. This choice directly addresses the theoretical circuit presented in Section 3.5.3. However, it comes with the trade-off of easier parallelisation in software and the potential use of GPUs, leading to moderate speed-ups. An additional advantage of larger primes is the improved ratio between delay and opening times, resulting in more efficient puzzle construction.

Table 3.10 compares these two approaches in terms of the attack classes we believe could be successful.

Post-quantum security is a crucial factor. If the discrete logarithm can be broken, both approaches are vulnerable to the attack shown in Section 3.5.5. However, this attack is more feasible with smaller moduli, as computing the discrete logarithm

modulo a 1000-bit number is significantly easier than for a 60000-bit number. Consequently, larger moduli may offer a buffer period before quantum computers can realistically be an advantage, even though the attack itself would still only yield moderate speed-ups.

When considering classical attacks, frequent changes of the modulus are crucial if relying on small bit-sizes. As early as 2015, Adrian *et al.* [4] estimated that nation-state adversaries could potentially break the 1024-bit discrete logarithm problem with sufficient precomputation. Therefore, "catastrophic" attacks (i.e. those leading to extremely large speed-ups) are mitigated only by using large moduli or frequently changing the modulus.

As a result, until very efficient ways to construct safe moduli are uncovered, we suggest using large primes.

4

Commutative Blinding

Contents

In the remaining of this thesis, we will describe how TRE can be used in the context of fair exchange to improve previous bounds. In particular, we will design optimally-fair exchange protocols both in the 2-party and the multi-party setting. While TRE is the key component that allows our construction to achieve optimality, another important primitive is commutative encryption.

The efficiency of our protocols relies on an encryption scheme where the order of keys used to encrypt the same message multiple times is irrelevant. While any commutative encryption scheme satisfies this property, we require stronger security guarantees to safely implement our fair exchange protocols.

Commutative encryption, introduced by Shamir *et al.* [142], is an encryption scheme that satisfies the property $\mathsf{Enc}_{k_1}(\mathsf{Enc}_{k_2}(m)) = \mathsf{Enc}_{k_2}(\mathsf{Enc}_{k_1}(m))$ for any pair of keys (k_1, k_2) and any message m. The related notion of IND-CPA is designed for scenarios where all keys are honestly generated, and the adversary is only given a

ciphertext after all encryptions are applied. This is insufficient for our application since the adversary is responsible for part of the encryption, generating their own key and encrypting a partial ciphertext.

Therefore, we introduce a *new* primitive, which we call *commutative blinding*, that generalises the concept of commutative encryption and provides stronger security guarantees. Our construction can replace classical commutative encryption and be used in scenarios where partial ciphertexts must remain secure and not all keys are honestly generated. Additionally, we present a lattice-based construction, which we believe is the first discussion of a post-quantum commutative encryption scheme that does not rely on the hidden discrete logarithm problem.

We start the chapter by providing formal definitions of the commutative properties and security requirements. Section 4.3 will show an ElGamal-based construction that highlights the differences with standard commutative encryption. The next section generalises this construction in an abstract framework, opening the door to post-quantum instantiations. In Section 4.5 we provide an instantiation based on the Ring Learning With Error problem which doesn't fully fit our abstract framework.

For ease of presentation, our exposition will mainly focus on the two-party context, however we note that extending any of our constructions to the multi-party context is trivial.

4.1 Definition

Our goal is to construct functions: $\mathsf{Blind}_1, \mathsf{Blind}_2, \mathsf{Unblind}_1, \mathsf{Unblind}_2$ so that

$$\forall k_1, k_2, m \quad \mathsf{Unblind}_2\left(\mathsf{Unblind}_1\left(\mathsf{Blind}_2\left(\mathsf{Blind}_1\left(m, k_1\right), k_2\right), k_1\right), k_2\right) = m$$

In the above notation, we denote these functions as symmetric ciphers, but there is nothing stopping us from using asymmetric ciphers. Indeed, all the constructions we present are based on public-key ciphers.

In terms of security, we would like to obtain indistinguishability at each step: with Blind_1, Blind_2 and $\mathsf{Unblind}_1$. While weaker requirements could be used to prove the security of our protocols, we stick to the above since it will simplify the security proofs.

Definition 4.1 (Commutative Blinding)**.** A commutative blinding scheme is a tuple of algorithms $(\mathsf{KGen}_1, \mathsf{KGen}_2, \mathsf{Blind}_1, \mathsf{Blind}_2, \mathsf{Unblind}_1, \mathsf{Unblind}_2)$[1] with sets $(\mathcal{M}, \mathcal{K}_1, \mathcal{K}_2, \mathcal{C}_{\text{int}}, \mathcal{C})$ such that

$$\mathsf{KGen}_1 \colon \{1\}^* \to \mathcal{K}_1 \qquad \mathsf{Blind}_1 \colon \mathcal{M} \times \mathcal{K}_1 \to \mathcal{C}_{\text{int}} \qquad \mathsf{Unblind}_1 \colon \mathcal{C} \times \mathcal{K}_1 \to \mathcal{C}_{\text{int}}$$
$$\mathsf{KGen}_2 \colon \{1\}^* \to \mathcal{K}_2 \qquad \mathsf{Blind}_2 \colon \mathcal{C}_{\text{int}} \times \mathcal{K}_2 \to \mathcal{C} \qquad \mathsf{Unblind}_2 \colon \mathcal{C}_{\text{int}} \times \mathcal{K}_2 \to \mathcal{M}$$

In order for the scheme to make sense, we require

$$\forall k_1 \in \mathcal{K}_1 \; \forall k_2 \in \mathcal{K}_2 \; \forall m \in \mathcal{M}$$
$$\mathsf{Unblind}_2\left(\mathsf{Unblind}_1\left(\mathsf{Blind}_2\left(\mathsf{Blind}_1\left(m, k_1\right), k_2\right), k_1\right), k_2\right) = m$$

If these functions were used in other contexts, these names could be misleading. In particular, $\mathsf{Unblind}_1$ and Blind_1 are not necessarily inverses, nor are $\mathsf{Unblind}_2$ and Blind_2. In most practical scenarios, the blinding and unblinding will be inverses as well as $\mathsf{Blind}_1 = \mathsf{Blind}_2$ and $\mathsf{Unblind}_1 = \mathsf{Unblind}_2$. However, this is not required and the constructions we propose will technically require the whole generality of the definition.

Note that all of the above blinding/unblinding functions will most likely use some randomisation (or they would not be IND-CPA). Thus, we will often write $\mathsf{Blind}(m, k; r)$ for some "random seed" r and assume that Blind is deterministic. This allows us to explicitly keep track of the random nonces used.

Regarding security, we use the standard IND-CPA game with the Blind_1 function to obtain a notion of indistinguishability under CPA for Blind_1. We would like to

[1] For the multi-party context, we need $3N$ functions $(\mathsf{KGen}_1, \ldots, \mathsf{KGen}_N, \mathsf{Blind}_1, \ldots, \mathsf{Blind}_N, \mathsf{Unblind}_1, \ldots, \mathsf{Unblind}_N)$ where we require that $\mathsf{Unblind}_N(\ldots \mathsf{Unblind}_1(\mathsf{Blind}_N(\ldots \mathsf{Blind}_1(m, k_1) \ldots, k_N), k_1) \ldots, k_N) = m$.

obtain a similar notion for Blind_2, however, Blind_2 is defined only on the ciphertexts of Blind_1, so we obtain two different notions. The weaker notion is depicted in Figure 4.1, and it is a simple adaptation of the standard IND-CPA game where Blind_2 is only used over ciphertexts encrypted with the same key.[2] This is what we will require for the security of our protocol. While Figure 4.1 shows the adversary picking messages m_0, m_1, in reality we only require the ciphertexts sent to the oracle to be valid ciphertexts generated by the same key. Whether the adversary has knowledge of the key and the messages encrypted is not a concern to us. Another stronger requirement can be designed when the adversary picks ciphertexts encrypted under different keys. However, we will not discuss it in this thesis as it will not be to any use in the context of the fair exchange protocols.

Weak IND-CPA game for Blind_2

Oracle		**Adversary**
$k \leftarrow \mathsf{KGen}_2(1^\lambda)$		
		has oracle access to $\mathsf{Blind}_2(\_, k)$
		$m_0, m_1 \leftarrow \mathcal{M}; k' \leftarrow \mathsf{KGen}_1(1^\lambda)$
		$c_0 \leftarrow \mathsf{Blind}_1(m_0, k')$
		$c_1 \leftarrow \mathsf{Blind}_1(m_1, k')$
	$\xleftarrow{c_0, c_1}$	
$b \xleftarrow{\$} \{0,1\}$		
$c \leftarrow \mathsf{Blind}_2(c_b, k)$		
	$\xrightarrow{c}$	
		return b'

Figure 4.1: Weak CPA game for the Blind_2 scheme.

To guarantee fairness of the exchange protocol, we would like some indistinguishability property like the ones above for $\mathsf{Unblind}_1$, where we say that receiving $\mathsf{Unblind}_1(\mathsf{Blind}_2(\mathsf{Blind}_1(m, k_1), k_2), k_1)$ does not leak information about m or k_1. The

[2] In the multi-party setting, the IND-CPA games for $\mathsf{Blind}_3, \ldots, \mathsf{Blind}_N$ are defined in the similar manner where the adversary picks 2 ciphertexts generated by the previous blinding functions. The keys used in constructing such ciphertexts must be the same.

stronger requirement encapsulating this idea is the below Equation 4.1.[3]

$$\mathsf{Unblind}_1(\mathsf{Blind}_2(\mathsf{Blind}_1(m, k_1; r_1), k_2; r_2), k_1; r_3) \overset{c}{\approx} \mathsf{Blind}_1(m, k_2; r_4) \tag{4.1}$$

where the adversary can choose all parameters but r_3 and r_4, which are picked uniformly at random. Once more, the security of our protocols is guaranteed with a weaker requirement, yet we stick to the above for ease of presentation.

We do not define any notion of IND-CCA. The standard (adaptive) notion of IND-CCA implies various forms of non-malleability [19], which are unlikely to hold for our instantiations or any other instantiation of commutative encryption we are aware of. Since our goal in defining and constructing this primitive is its application in the fair-exchange protocols discussed in the following chapters, we only prove the IND-CPA requirements that are sufficient for this specific use case.

4.2 Related Constructions

In this chapter, we present a general framework for instantiating the described commutative blinding primitive, along with two concrete constructions: one based on ElGamal and the other on the RLWE problem. The ElGamal-based scheme integrates techniques from various fields to achieve the stronger security guarantees required by commutative blinding. The RLWE-based scheme goes a step further by ensuring security against quantum computation. We believe this to be the first discussion of post-quantum commutative encryption that does not rely on the hidden discrete logarithm problem.

It is important to note that these are not the only possible approaches. The primary reason for designing our own constructions was the need for stronger security requirements than those typically considered for commutative encryption. Specifically, the requirements for our commutative blinding scheme must allow the adversary to choose keys or ciphertexts within the chain of encryptions. To demonstrate that our requirements are strictly stronger than the usual definition, we analyse

[3] Similar equations can be designed in the multi-party context for $\mathsf{Unblind}_2, \ldots, \mathsf{Unblind}_{N-1}$.

the commutative encryption scheme by Huang and Tso [83]. This ElGamal-based scheme closely resembles our construction, with the crucial difference being the absence of a re-randomisation step during re-encryption. In our construction, Blind_2 would alter the random coins used in Blind_1, which is not the case in Huang and Tso's work [83]. This change would render Blind_2 insecure according to our definition, as the adversary could distinguish ciphertexts based on the random seeds used by Blind_1.

In the post-quantum research domain, commutative encryption is less studied, and all research we are aware of focuses on generalising the discrete logarithm problem [51, 116]. However, recent research [121, 162] suggests that this approach may be inherently flawed.

Commutative encryption seems to arise naturally from research on multi-key fully homomorphic encryption (MKFHE). To optimise MPC protocols, MKFHE aims to design FHE schemes where the plaintext can be encrypted by multiple parties, each using their own set of keys [101]. The commutativity property follows directly from the commutativity of the underlying LWE (or variant) construction (see [43]). However, MKFHE schemes are unlikely to satisfy our stronger security requirements, as their security analysis assumes that the adversary cannot choose the keys used in the encryption process. For instance, the re-encryption of a ciphertext in the scheme by Chen *et al.* [43] lacks any re-randomisation step.

Nevertheless, the field of lattice-based security is rich with ciphers based on the LWE (or variant) problem [32, 45], which could potentially lead to other post-quantum blinding schemes. Moreover, these constructions tend to be relatively similar, so we anticipate they could be used by adhering to the general framework of Section 4.4 or by slightly modifying it, as we do in this chapter with the BFV scheme.

4.3 ElGamal

The first construction we present is one based on the ElGamal asymmetric cipher [59]. Although this construction relies on the hardness of the discrete logarithm, its high-level structure is similar to our later constructions and serves as an introduction to them.

The ElGamal cipher can be described quickly as follows. Pick a group G of order q where the Decisional Diffie-Hellman assumption holds. Pick a generator g and a secret integer s. Let the public key be $h = g^s$ and s is the secret key. A message m is encrypted into (mh^y, g^y) for some randomly chosen $y \in \mathbb{Z}_q^*$. The ciphertext (c_1, c_2) is decrypted into $c_1 c_2^{-s}$. It's easy to spot that the step masking the message (e.g. mh^y) has commutative properties. If we were to pick a second key h', then $(mh^y)h'^{y'} = (mh'^{y'})h^y$, meaning that masking with h first is equivalent to masking with h' first. The second part of the cipher (i.e. g^y) is used to encapsulate the random nonce y. To achieve IND-CPA on the second encryption we cannot allow this "tag" to identify which y was picked. Hence, we introduce a "re-randomisation" step where the random nonce y is substituted by a new nonce unknown to both parties. This technique was introduced as a way to construct proxy re-encryption schemes [26] and it has been used before as a way to shuffle lists of ElGamal ciphertexts (e.g. [85]). Nevertheless, it appears to be the missing part of ElGamal-based commutative encryptions (see [83]).

The resulting scheme works as follows. Let G be a group of order q where the DDH assumption holds. Pick a generator g and two secret integers s_1, s_2. Let the two public keys be $h_1 = g^{s_1}, h_2 = g^{s_2}$. Define[4]

$$\begin{aligned}
\mathsf{Blind}_1(m, h_1) &= (mh_1^a, g^a) & a &\xleftarrow{\$} \mathbb{Z}_q^* \\
\mathsf{Blind}_2((c_1, c_2), h_2) &= (c_1 h_1^c h_2^b, c_2 g^c, g^b) & b, c &\xleftarrow{\$} \mathbb{Z}_q^* \\
\mathsf{Unblind}_1((c_1, c_2, c_3), s_1) &= (c_1 c_2^{-s_1} h_2^d, c_3 g^d) & d &\xleftarrow{\$} \mathbb{Z}_q^* \\
\mathsf{Unblind}_2((c_1, c_2), s_2) &= c_1 c_2^{-s_2}
\end{aligned}$$

[4] In the multi-party setting, we define $\mathsf{Blind}_3, \ldots, \mathsf{Blind}_N$ as Blind_2 by extending the tuple and re-randomising all previous parts.

4. Commutative Blinding

Firstly, we can show correctness by noting that

$$\mathsf{Unblind}_2(\mathsf{Unblind}_1(\mathsf{Blind}_2(\mathsf{Blind}_1(m, h_1), h_2), s_1), s_2) = \\ mh_1^a h_1^c h_2^b (g^a g^c)^{-s_1} h_2^d (g^b g^d)^{-s_2} = m$$

IND-CPA of Blind_1 follows directly by the fact that Blind_1 is the ElGamal encryption function. In particular, ElGamal's CPA indistinguishability can be reduced to the DDH assumption [153]. While commutative encryption schemes similar to our exist [83], their security is often proved assuming the honesty of all encrypters. Meaning that IND-CPA is shown to hold when the adversary selects two messages and receives back the result of the multiple encryptions (in our notation: $\mathsf{Blind}_2(\mathsf{Blind}_1(m, h_1), h_2)$). Our requirement that Blind_2 is IND-CPA is stronger as it allows the adversary to pick h_1 as well as the random nonces used in Blind_1. Below we adapt a standard reduction of ElGamal to the DDH problem [88] to prove that Blind_2 is CPA secure when used on ciphertexts generated by a single key and where the adversary generates their key. The detailed description of the game we consider is given in Figure 4.2. The main difference between this and the game of Figure 4.1 from the start of this chapter is that this scheme is based on a public-key cipher, hence we have both parties to "publish" their keys as soon as they are generated. We say that an adversary wins the game if $b' = b$.

Theorem 4.1. *Assuming the DDH problem is hard, then* Blind_2 *is* IND-CPA *as per Figure 4.2*

Proof. First recall that the DDH assumption states that there is no PPT decider $\mathcal{D}$ that can distinguish between a DH triple (g^a, g^b, g^{ab}) and a random triple (g^a, g^b, g^c). More formally, there is a negligible function $\mathsf{negl}(\lambda)$ such that

$$\left|\Pr\left[\mathcal{D}(g^a, g^b, g^{ab}) = 1\right] - \Pr\left[\mathcal{D}(g^a, g^b, g^c) = 1\right]\right| < \mathsf{negl}(\lambda)$$

where λ is used in the construction of the underlying group.

To prove the security of our scheme, we construct a decider $\mathcal{S}$ which takes advantage of an adversary $\mathcal{A}$ for Figure 4.2. In particular, $\mathcal{S}$ will work as follows:

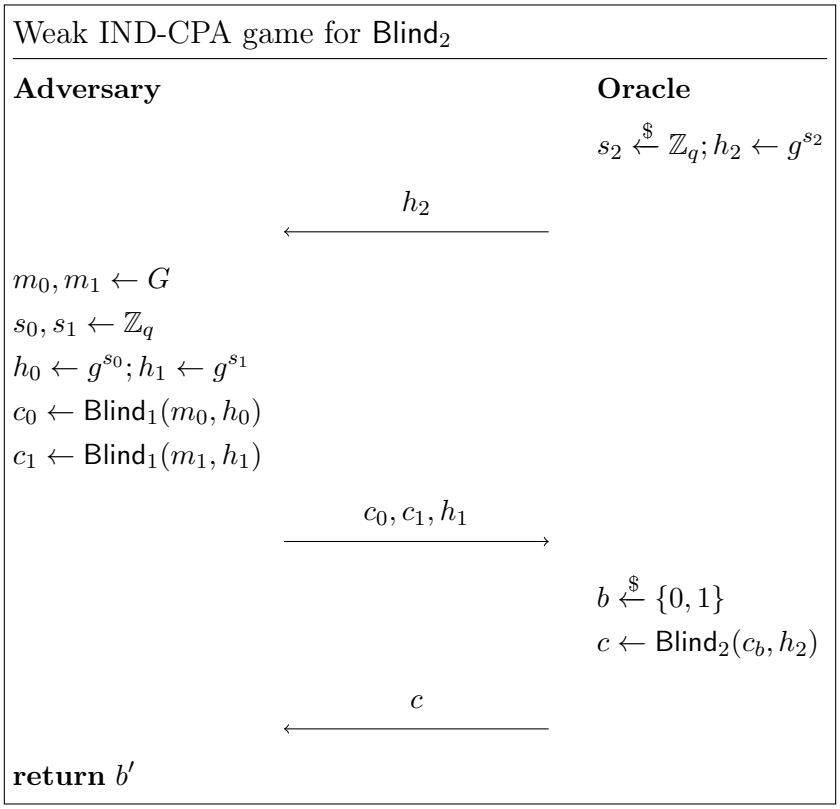

Figure 4.2: Weak CPA game for the Blind_2 scheme.

1. Receive the general parameters (G, q, g) and a triple (g_1, g_2, g_3).
2. Set $h_2 = g_1$ and send this to $\mathcal{A}$.
3. Receive ciphertexts $({c_1}^{(0)}, {c_2}^{(0)}), ({c_1}^{(1)}, {c_2}^{(1)})$ and their public key h_1.
4. Sample a random bit β, and a random $r \xleftarrow{\$} \mathbb{Z}_q^*$.
5. Send $\mathcal{A}$ the ciphertext $({c_1}^{(\beta)} h_1^r g_3, {c_2}^{(\beta)} g^r, g_2)$.
6. Receive b from $\mathcal{A}$ and return 1 if $b = \beta$, else 0.

Let us analyse how $\mathcal{S}$ behaves. If the triple (g_1, g_2, g_3) is a DH triple (g^a, g^b, g^{ab}), then note that the ciphertext that $\mathcal{A}$ receives is exactly $\mathsf{Blind}_2(({c_1}^{(\beta)}, {c_2}^{(\beta)}), h_2)$. Thus, $\mathcal{A}$'s view is exactly the IND-CPA game and $\mathcal{S}$ outputs 1 if $\mathcal{A}$ wins such a game. Let the probability of $\mathcal{A}$ beating the IND-CPA game be $\frac{1}{2} + \mu(\lambda)$. We will show that $\mu(\lambda)$ is negligible.

Now consider the case where (g_1, g_2, g_3) is not a DH triple. Hence, the three elements are independently and uniformly distributed over G. It follows that the ciphertext $({c_1}^{(\beta)} h_1^r g_3, {c_2}^{(\beta)} g^r, g_2)$, is also a triple of independent and uniformly distributed elements. Thus, $\mathcal{A}$'s view is indistinguishable from the game of Figure 4.3 where we say that the adversary wins the game if $b' = \beta$. Clearly, in such a game the probability that $\mathcal{A}$ guesses correctly is $\frac{1}{2}$. Thus, we have that

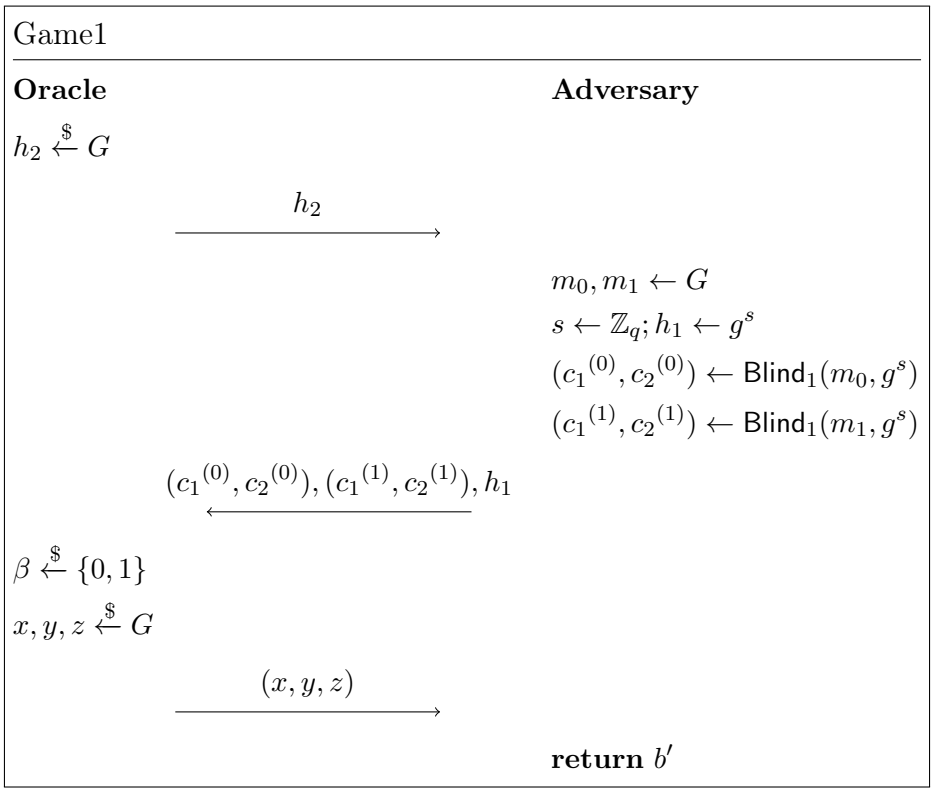

Figure 4.3: Game with pointless encryption.

$$\begin{aligned}\mathsf{negl}(\lambda) &> \left|\Pr\left[\mathcal{S}(g^a, g^b, g^{ab}) = 1\right] - \Pr\left[\mathcal{S}(g^a, g^b, g^c) = 1\right]\right| \\ &= \left|\frac{1}{2} + \mu(\lambda) - \frac{1}{2}\right| = |\mu(\lambda)|\end{aligned}$$

This shows that any PPT adversary $\mathcal{A}$ has negligible advantage in the IND-CPA game for Blind_2. □

The above proves that Blind_2 is IND-CPA, yet we still require $\mathsf{Unblind}_1$ to be "indistinguishable" (see Eq. 4.1). In our construction, Eq. 4.1 follows immediately by

noting that

$$\mathsf{Unblind}_1(\mathsf{Blind}_2(\mathsf{Blind}_1(m, k_1), k_2), k_1; r) = \mathsf{Blind}_1(m, k_2; r')$$

where r' is uniformly distributed given that r is. All in all, this ElGamal-based scheme satisfies all the required properties to be used in our fair exchange protocols.

4.4 General Construction

In this short section, we generalise the previous construction.

Assume the existence of an asymmetric encryption scheme with functions

$$\mathsf{Enc}_{\mathsf{pk}}(m, r) = (f(\mathsf{pk}, m, r), h(\mathsf{pk}, r))$$
$$\mathsf{Dec}_{\mathsf{sk}}(c)$$

where all the above functions are deterministic and the r represents the random seeds used. Crucially, we split the encryption function into two different functions: f and h. This models the ElGamal encryption where the message is hidden by $f(h, m, r) = mh^r$, while the second part of the pair is used to encapsulate the random seed r with $h(h, r) = g^r$. Assume further that f commutes: $f(\mathsf{pk}_1, f(\mathsf{pk}_2, m, r_2), r_1) = f(\mathsf{pk}_2, f(\mathsf{pk}_1, m, r_1), r_2)$, then there is a straightforward way to design a "classical" (e.g. [83]) commutative encryption scheme. However, recall that our indistinguishability for Blind_2 is much stronger. In the previous section we used a re-randomisation procedure. Thus, assume further that there is a function rerand such that

$$\{(\mathsf{pk}, m, r', \mathsf{rerand}\,(\mathsf{Enc}_{\mathsf{pk}}(m, r'), \mathsf{pk}, r))\}_r \overset{\mathrm{c}}{\approx} \{(\mathsf{pk}, m, r', \mathsf{Enc}_{\mathsf{pk}}(m, r))\}_r \tag{4.2}$$

That is, no PPT adversary can distinguish between a re-randomised cipher and one generated using an unknown random seed. Moreover, we require Eq. 4.2 to hold even when the distinguisher can choose pk, m and r'.

Given the above, we will prove that the following represents a commutative blinding scheme:

$$\begin{aligned}
\mathsf{Blind}_1(\mathsf{pk}_1, m, r) &= (f(\mathsf{pk}_1, m, r), h(\mathsf{pk}, r)) \\
\mathsf{Blind}_2(\mathsf{pk}_2, c, r_1, r_2) &= (f(\mathsf{pk}_2, d_1, r_1), d_2, h(\mathsf{pk}_2, r_1)) \\
&\quad \text{where } (d_1, d_2) = \mathsf{rerand}(c, r_2) \\
\mathsf{Unblind}_1(\mathsf{sk}_1, (c_1, c_2, c_3), r) &= \mathsf{rerand}((\mathsf{Dec}_{\mathsf{sk}_1}((c_1, c_2)), c_3), r) \\
\mathsf{Unblind}_2(\mathsf{sk}_2, c) &= \mathsf{Dec}_{\mathsf{sk}_2}(c)
\end{aligned}$$

To prove the security of our construction, we need to show that both Blind_1 and Blind_2 are IND-CPA as well as checking the indistinguishability of $\mathsf{Unblind}_1$. The security of Blind_1 comes directly from the security of the underlying encryption system. Note further that $\mathsf{Unblind}_1(\mathsf{sk}_1, c, r) \overset{c}{\approx} \mathsf{Enc}_{\mathsf{pk}_2}(m, r')$ for some fresh r' and the correct message m. Hence, all we need to prove is the indistinguishability of Blind_2.

Theorem 4.2. *Let* $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *be a public-key cipher satisfying* IND-CPA*, and construct a commutative blinding scheme as described above. Then* Blind_2 *is* IND-CPA*.*

Proof. Let $\mathcal{A}$ be an adversary to the Blind_2 (weak) IND-CPA game as described in Section 4.1 and 4.3. We construct an adversary $\mathcal{S}$ to the IND-CPA game for Enc. Consider the reduction of Figure 4.4 where we set $\beta = 1$.

It should be clear that $\mathcal{S}$ is playing the IND-CPA game for Enc. When $b = \gamma$, the last message to $\mathcal{A}$ is $(f(\mathsf{pk}_2, {d_1}^{(b)}, r'), {d_2}^{(b)}, h(\mathsf{pk}_2, r')) = \mathsf{Blind}_2(\mathsf{pk}_2, c^{(b)}, r', r)$. Thus $\mathcal{A}$'s view is exactly the IND-CPA game for Blind_2. Say that $\mathcal{A}$ guesses correctly with probability $\frac{1}{2} + \varepsilon(\lambda)$. Now consider the case where $b \neq \gamma$, then $\mathcal{A}$ receives $(f(\mathsf{pk}_2, {d_1}^{(b)}, r'), {d_2}^{(1-b)}, h(\mathsf{pk}_2, r'))$. Due to Equation 4.2, the above is indistinguishable from $(f(\mathsf{pk}_2, {d_1}^{(b)}, r'), h(\mathsf{pk}_1, r_2), h(\mathsf{pk}_2, r'))$ for some fresh r_2. Now consider the IND-CPA game for Blind_2, but alter it slightly so that the challenger instead of sending the adversary $(c_1, c_2, c_3) \leftarrow \mathsf{Blind}_2(\mathsf{pk}_2, c, r_1, r_2)$, they send them

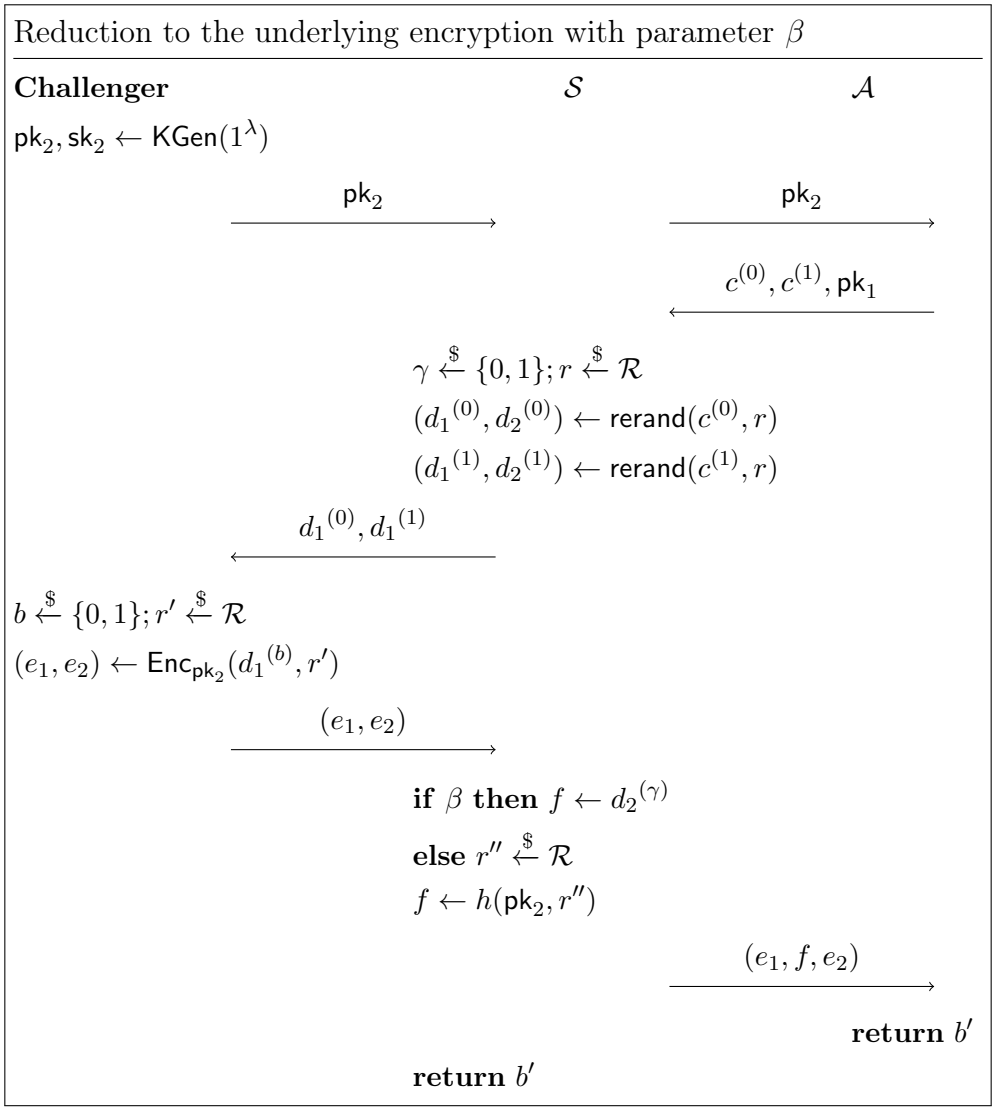

Figure 4.4: Reduction to underlying encryption system.

$(c_1, h(\mathsf{pk}_1, r_3), c_3)$ for some fresh r_3. Call this new game $\overline{\text{IND-CPA}}$. Note that this game is what $\mathcal{A}$ sees when, in the reduction of Figure 4.4, $b \neq \gamma$ (and $\beta = 1$). We want to prove that all PPT adversaries can win this new game only with negligible advantage. To do so, we construct a reduction to the IND-CPA game of the underlying encryption scheme. This can be seen in Figure 4.4 when $\beta = 0$. Thus, we have that all PPT algorithms win the $\overline{\text{IND-CPA}}$ game with at most negligible advantage. Since $\overline{\text{IND-CPA}}$ is computationally indistinguishable from $\mathcal{A}$'s view of our original reduction (Figure 4.4 with $\beta = 1$) when $b \neq \gamma$, it means that $\mathcal{A}$ guesses b correctly with negligible advantage (say $\mu(\lambda)$). Thus, we can express the probability of $\mathcal{S}$ winning the IND-CPA of Enc in terms of μ and ε, obtaining

$$\frac{1}{2}\left(\frac{1}{2} + \varepsilon(\lambda) + \frac{1}{2} + \mu(\lambda)\right) = \frac{1}{2} + \frac{\varepsilon(\lambda) + \mu(\lambda)}{2}$$

Since the encryption scheme $(\mathsf{Enc}, \mathsf{Dec})$ is IND-CPA, the above implies that $\frac{\varepsilon(\lambda)+\mu(\lambda)}{2}$ is negligible. Since μ is also negligible, it follows that so is ε. Hence, $\mathcal{A}$ wins the IND-CPA game for Blind_2 with negligible (that is, ε) advantage. □

The above theorem shows a different way to prove the security of our ElGamal-based construction as well as opening the door to other schemes.

4.5 Post-Quantum Commutative Encryption

In this section, we give a concrete construction of a commutative blinding scheme based on the Ring Learning With Error decisional problem. An efficient algorithm for the LWE problem implies the existence of a quantum algorithm for some worst-case lattice problems [135], which are believed to be quantum-secure. Thus, LWE (and its variants over rings and Torus) is likely to maintain its hardness in a quantum realm. More specifically, our construction is based on (possibly the most trivial) encryption scheme relying on the Ring LWE decisional problem [63]. Similarly to the LWE problem, the decisional RLWE problem was proved to be quantum reducible to some worst-case lattice problem [103].

In this section, we report a simpler the definition of the decisional RLWE as described in [32]. Let $f(x) = x^d + 1$ where d is a power of 2, $q \geq 2$ be an integer, $R = \mathbb{Z}[x]/\langle f(x)\rangle$ be a polynomial ring, and χ be a distribution over R. Write R_q for the ring R/qR and fix $s \in R_q$. The $\text{RLWE}_{d,q,\chi}$ problem consists of distinguishing between the distributions: uniform over R_q^2 and the one generated by sampling $a \xleftarrow{\$} R_q$, $e \xleftarrow{\$} \chi$ and outputting $(a, as + e)$.[5] Our construction is based on the BFV [63] scheme, yet all the fully homomorphic schemes we are aware of (e.g. [32, 45, 44]) are based on variants of the LWE problem, thus we would expect that similar blinding schemes can be constructed using other fully homomorphic ciphers.

The BFV scheme [63] can be described as follows. Let q, d, R_q and χ be as in the definition of the RLWE problem. Pick a random element $s \in R_q$, this constitutes the secret key sk. Pick another random element $a \in R_q$ and sample the error probability χ to get e. The public key is $\mathsf{pk} = (-a \cdot s + e, a)$. Pick an integer t, the message space is defined to be R_t, and definite $\Delta = \lfloor q/t \rfloor$.[6] The encryption/decryption functions are described as:

$$\mathsf{Enc}_{\mathsf{pk}}(m) = (\mathsf{pk}(1) \cdot u + e + \Delta m, \mathsf{pk}(2) \cdot u + e') \quad \text{where } u \xleftarrow{\$} R_q, e, e' \xleftarrow{\$} \chi$$
$$\mathsf{Dec}_{\mathsf{sk}}(c_1, c_2) = (c_1 + c_2 \cdot \mathsf{sk})/\Delta$$

where Dec will map the result, which is in R_q, to the closest element in R_t.

Assume the same setup as above and let $(\mathsf{pk}_1, \mathsf{sk}_1), (\mathsf{pk}_2, \mathsf{sk}_2)$ be two pairs of pub-

[5] Technically, e is defined over R instead of R_q. Hence, $as+e$ is an element of R, yet this can be easily embedded into R_q by taking the modulo q operation on each coefficient of the polynomial. We will always assume that all elements are mapped in R_q when needed.

[6] We refer to the original paper [63] for a more accurate description of how to pick these parameters.

lic/private keys. We define our commutative blinding scheme as follows:

$$\begin{aligned}
\mathsf{Blind}_1(\mathsf{pk}_1, m, u, e_1, e_2) &= (\mathsf{pk}_1(1) \cdot u + e_1 + \Delta m, \mathsf{pk}_1(2) \cdot u + e_2) \\
\mathsf{Blind}_2(\mathsf{pk}_2, (c_1, c_2), u, v, e_1, e_2) &= (c_1 + \mathsf{pk}_2(1) \cdot u + e_1 + \mathsf{pk}_1(1) \cdot v, \\
&\qquad c_2 + \mathsf{pk}_1(2) \cdot v, \mathsf{pk}_2(2) \cdot u + e_2) \\
\mathsf{Unblind}_1(\mathsf{sk}_1, (c_1, c_2, c_3), v, e_1, e_2) &= (c_1 + c_2 \cdot \mathsf{sk}_1 + \mathsf{pk}_2(1) \cdot v + e_1, \\
&\qquad c_3 + \mathsf{pk}_2(2) \cdot v + e_2) \\
\mathsf{Unblind}_2(\mathsf{sk}_2, (c_1, c_2)) &= \lfloor (c_1 + c_2 \cdot \mathsf{sk}_2)/\Delta \rceil
\end{aligned}$$

Note that Blind_1 and $\mathsf{Unblind}_2$ are the standard encryption-decryption. Blind_2 encrypts the first part of the ciphertext (the part hiding the message) and re-randomises the second item. This construction is very similar to the general construction of the previous section, yet it does not exactly follow it. In particular, the decomposition of the encryption function would lead to $f(m) = \mathsf{pk}(1) \cdot u + e + \Delta m$, thus Blind_2 would need to multiply its argument by Δ as well. Doing so would introduce unnecessary complications in the re-randomisation step as well as in $\mathsf{Unblind}_1$.

Correctness of the unblinding process depends on the choice of χ, q, t as well as other small optimisations one could pick (such as sampling the random vectors from χ or $R/2R$). In particular, one should prove that the "noise" introduced by the errors from χ do not exceed a critical level after which the closest plaintext to the decrypted value is no longer the original message. However, we can argue its correctness without a deep technical analysis by noting that the overall noise level of our scheme is bounded above by 3 times the noise of a simple BFV ciphertext. Thus, if the parameters are picked so that one can homomorphically compute 2 additions, then our construction will work as well since the noise of a ciphertext after 2 additions is (roughly) 3 times the noise of a plain ciphertext. We note that general instantiations of these parameters allow many more homomorphic operations.

The security of our scheme reduces to the underling RLWE problem. In particular, the indistinguishability of Blind_1 comes directly from the BFV encryption. Security of Blind_2 is proved in Theorem 4.4, while the requirement on $\mathsf{Unblind}_1$ is shown in Theorem 4.3. Regarding $\mathsf{Unblind}_1$, it would be impossible to prove Eq. 4.1 since the ciphertext obtained from $\mathsf{Unblind}_1$ has a higher level of noise. Moreover, if the adversary can pick the error vectors e_i at will without actually sampling from χ, they could easily choose large errors which would have negligible probabilities according to χ. Therefore, we prove Eq. 4.1 by forcing the adversary to pick the error vectors correctly from χ and with the extra relaxation that the RHS uses an error vector sampled with the appropriate noise level. In particular, χ is often picked as a Gaussian distribution, thus increasing the noise level is equivalent to "flattening" the bell curve.

Theorem 4.3. $\mathsf{Unblind}_1$ *as constructed above satisfies the requirement of Eq. 4.1 with the above modifications.*

Proof. Firstly note that $\mathsf{Unblind}_1(\mathsf{sk}_1, (c_1, c_2, c_3)) = (\Delta m + \mathsf{pk}_2(1) \cdot (v + z) + e_7 \cdot (u + w) + \mathsf{sk}_1 \cdot e_2 + e_1 + e_3 + e_5, \mathsf{pk}_2(2) \cdot (v + z) + e_4 + e_6)$ for some errors $e_1, \dots e_7$ and polynomials u, v, w, z. We need to show that this is indistinguishable from $\mathsf{Blind}_1(\mathsf{pk}_2, m)$ in the eyes of a party which controls everything but z, e_5 and e_6. We can rewrite the above as $(\Delta m + \mathsf{pk}_2(1) \cdot x + e, \mathsf{pk}_2(2) \cdot x + e')$ where $x \in R_q$ and e, e' are "small". Crucially, $x (= v + z)$ is uniformly distributed since z is, while e, e' are both affected by e_5, e_6. Since all error vectors are picked according to χ, the adversary has no control over them and the resulting distributions of e, e' are hidden thanks to e_5 and e_6. □

Theorem 4.4. *Assuming the hardness of the RLWE problem,* Blind_2 *is* IND-CPA.

Proof. This proof will resemble the one showing the correctness of the ElGamal-based commutative blinding. In particular, let $\mathcal{A}$ be any PPT adversary to the IND-CPA game for Blind_2, we construct a PPT distinguisher $\mathcal{S}$ for the RLWE problem. Let $\mathcal{S}$ behave as follows:

4. Commutative Blinding

1. Upon receipt of the pair (a, b) from the challenger of the RLWE problem, send (b, a) as pk_2 to $\mathcal{A}$.
2. $\mathcal{A}$ will reply with $\boldsymbol{c}^{(0)}, \boldsymbol{c}^{(1)}, \mathsf{pk}_1$.
3. Pick a uniform bit b and send $\mathcal{A}$ $\mathsf{Blind}_2(\mathsf{pk}_2, \boldsymbol{c}^{(b)})$.
4. If $\mathcal{A}$ guesses b, then return 1 otherwise 0

Assume that $\mathcal{S}$ is given an RLWE pair (a, b). That is, there is $s \in R_q$ such that $b = a \cdot s + e$ for some e sampled from χ. It follows that (b, a) is a public key of our commutative blinding construction with private key $-s$. Therefore, $\mathcal{A}$'s view is exactly the one of the IND-CPA game for Blind_2. Now consider the scenario where $\mathcal{S}$ is given a uniformly distributed pair (a, b). Assume that $\boldsymbol{c}^{(i)} = (\Delta m_i + \mathsf{pk}_1(1) \cdot v_i + e_i, \mathsf{pk}_1(2) \cdot v_i + e_i')$. Then, the last message that $\mathcal{A}$ receives is $(\Delta m_b + \mathsf{pk}_1(1) \cdot u + \mathsf{pk}_2(1) \cdot w + e_b + e_2, \mathsf{pk}_1(2) \cdot u + e_b', \mathsf{pk}_2(2) \cdot w + e_2')$ for some $u, w \in R_q$ and e_2, e_2' sampled from χ. Since u is a uniformly random element of R_q, the element $\mathsf{pk}_1(2) \cdot u + e_b'$ is also uniformly distributed over R_q even if the adversary can pick both e_b' and $\mathsf{pk}_1(2)$. Since $\mathsf{pk}_2(2)$ is uniformly distributed and independent of $\mathsf{pk}_2(1)$, the term $\mathsf{pk}_2(2) \cdot w + e_2'$ is also uniformly distributed and independent of the first part of the tuple. This independence means that w is still a "free" variable which is not influenced by this last term. Therefore, we see that by ranging w over R_q we can make the first term of the tuple be any element of R_q (assuming that $\mathsf{pk}_2(1) \neq 0$, which happens with overwhelming probability). Thus, $\mathcal{A}$'s view is indistinguishable from one where they receive a completely random triple in R_q^3. It should be clear, that any PPT algorithm playing the game defined by $\mathcal{A}$'s view will guess b with probability $\frac{1}{2}$. Say that $\mathcal{A}$ wins the IND-CPA game for Blind_2 with probability $\frac{1}{2} + \varepsilon(\lambda)$. Therefore, $\mathcal{S}$ outputs 1, when given a RLWE sample, with probability $\frac{1}{2} + \varepsilon(\lambda)$, while they output 1, given a uniform sample, with probability $\frac{1}{2}$. Thus, $\varepsilon(\lambda)$ must be negligible if RLWE problem is hard. □

With the above theorem, we proved that our RLWE-based construction of the commutative blinding can be used for our optimally-fair exchange protocol. Since

the RLWE assumption is believed to be secure against quantum computation, we designed a quantum-secure commutative blinding scheme. In conjunction with quantum-safe symmetric encryption and TRE, we can instantiate our protocols of the next chapters in a post-quantum world.

4.5.1 On the practicality of RLWE ciphers

A common misconception about homomorphic encryption is their inefficiency. While there is no denial that schemes as the one describe above are not as efficient as traditional encryption schemes, we need to understand whether its use in protocols would make these unpractical. To this end, we decided to test the real-world efficiency of the BFV scheme. We implemented a very simple program using the OpenFHE [12] library.[7] In both the protocols of Chapter 5 and Chapter 6, the commutative blinding scheme is used to encrypt keys of symmetric ciphers. Thus, we measured how long it would take to encrypt and decrypt 256-bit messages. To estimate the cost of the re-randomisation procedure, one should note that it is equivalent to homomorphically add a zero message. Although the implementation of BFV in the used library almost surely differs from the description above, our experiment shows that our construction is feasible, although not very efficient.

Table 4.1 presents the average time recorded over 100 trials on a late 2016 laptop equipped with an Intel i7-6920HQ CPU at 2.9 GHz. These values are compared to the performance of RSA and network latency.[8] We observe that the encryption procedure is significantly more time-consuming than decryption. The time spent on re-randomisation is predominantly influenced by the encryption of the zero message. Although homomorphic encryption is approximately one or two orders of magnitude slower than RSA, it is comparable to network latency.

In the protocol described in Chapter 5, exchanging $2N$ messages will result in roughly $15N$ms per party for computation related to blinding, while network la-

[7] The code is available in Appendix A.

[8] The OpenSSL "speed" tool was used to benchmark 1024-bit RSA, while network latency was measured by sending 100 ping packets to google.com from a home network over Wi-Fi.

encryption	re-randomisation	decryption	RSA enc/dec	network ping
3.8 ms (0.2)	3.6 ms (0.3)	0.5 ms (0.07)	7/99 μs	15 ms (10)

Table 4.1: Timings comparison measured over 100 trial. Standard deviation between parenthesis.

tency will impose a delay of at least $30N$ms. The multi-party protocol in Chapter 6 is less efficient due to its additional flexibility, with each party spending around $(8+7K)N$ms on blinding and at least $15(N+1)K$ms on network delay. Therefore, while our construction is not highly efficient, it is feasible. We also note that a more sophisticated implementation could compute several blindings in parallel, further reducing the overall latency associated with the blinding operation.

5

2-Party Fair Exchange

Contents

In this chapter, we initiate our study of fair exchange and illustrate how TRE can significantly enhance previous results. We begin with a concise introduction to the fair exchange problem, followed by the presentation of our protocol. Section 5.3 contains the security proofs that demonstrate the partial fairness of our protocol. Subsequently, we emphasise the optimality of our construction, showing that no

protocol with fewer exchange messages can achieve the same level of fairness. The chapter concludes with preliminary research findings on handling protocol failures. We examine whether it is possible to "restart" the exchange without compromising fairness. This inquiry leads us to consider alternative methods to instantiate our protocol, aiming to improve flexibility while maintaining optimality.

5.1 Introduction

Consider the scenario where two parties Alice and Bob want to exchange some digital items s_A, s_B (often referred as "secrets" in the literature), yet they do not trust each other. While this scenario is practically ubiquitous in our daily digital interactions (e.g. e-commerce), current techniques to solve the problem rely on Alice and Bob trusting some third entity to act as a mediator [65] or a judge [9]. Although the same result can be achieved with weaker trust assumptions, say using multiple mediators [70], it was proved impossible in the absence of any trust [128]. In particular, there is always an adversary which, by aborting at the correct moment, induces unfairness. However, researchers have found ways to circumvent this impossibility result, and the approach we will consider in this thesis is the concept of partial fairness introduced by Gordon and Katz [78]. Intuitively, we allow a small, yet non-negligible, probability of unfairness. Therefore, our focus will be on minimising this small chance of unfairness. While the formal definition of partial fairness is given in Chapter 2, we briefly recall that a protocol is said to achieve ε-partial fairness if it is unfair with probability ε.

Since the introduction of partial fairness in 2010, the technique used to achieve partial fairness is to create a list of messages where only one of them conveys something meaningful which will reveal the secret. With this approach, Gordon and Katz [78] design a protocol that achieves $\frac{1}{p}$-partial fairness in $O(p|\mathcal{D}|)$ exchange messages, where $\mathcal{D}$ is the set from which the secrets are picked. Roscoe and Ryan [139] were first to introduce TRE in this setting, and they achieve a slightly different concept

of partial fairness where the difference in the probabilities of each party obtaining the secrets is bounded. In this chapter, we show a protocol that improves on previous protocols by achieving $\frac{1}{N}$-partial fairness in $N + 1$ exchange messages. Moreover, we prove that no protocol can achieve lower probabilities of unfairness with the same number of exchange messages.

5.1.1 Preliminaries

To describe the fair exchange protocol that follows, we require a few different cryptographic primitives. Firstly, we will take advantage of a non-committing symmetric encryption scheme.

Definition 5.1 (Non-committing Encryption Scheme)**.** Let $(\mathsf{Enc}, \mathsf{Dec})$ represent a "standard" encryption scheme, with message space $\mathcal{M}$ and key space $\mathcal{K}$. We say that it is *non-committing* if

$$\forall m_1, m_2 \in \mathcal{M}, \forall k \in \mathcal{K}, \exists k' \in \mathcal{K} \;\; \mathsf{Dec}_{k'}(\mathsf{Enc}_k(m_1)) = m_2$$

The above property will be used in our security proof to allow simulatability. In particular, non-committing encryption was introduced by Canetti *et al.* [35] to allow simulatability of standard public-key encryption. Although, we do not require the asymmetric setting, we use it in a similar situation where a simulator needs to generate a ciphertext that can later be decrypted to any message. Moreover, we require that there is no PPT adversary $\mathcal{A}$ that, given the ciphertext $\mathsf{Enc}_k(x)$, can distinguish between k and k'. This means that $\mathcal{A}$ cannot tell which of the two keys was used to construct the ciphertext. Hence, they cannot distinguish between the interaction with the simulator or the real adversary. Formally, we require that no PPT adversary can distinguish between "correct" tuples (M, k, c) where $c \leftarrow \mathsf{Enc}_k(M)$ and "equivocated" tuples (M, k, c) where $M \leftarrow \mathsf{Dec}_k(c)$ yet $c \leftarrow \mathsf{Enc}_{k'}(M')$ for some other message-key pair. The simplest way to instantiate such an encryption scheme is to use a one-time pad. We have designed our protocol to accommodate this by using encryption keys only once.

Naturally, we will use a timed-release encryption scheme with encryption/decryption functions Delay / Open. While this scheme could rely on trusted parties, we do not want to introduce trust in the system. Thus, the choice of using Delay and Open emphasises the non-interactive nature of the cipher. Our security proof only requires Delay to hide its contents for the desired amount of time, hence we refer the reader to Chapter 2 for a formal definition of TRE.

The final building block for the protocol is commutative blinding. More specifically, we require the primitive defined and constructed in Chapter 4. That is, we need a scheme with functions $(\mathsf{KGen}_1, \mathsf{KGen}_2, \mathsf{Blind}_1, \mathsf{Blind}_2, \mathsf{Unblind}_1, \mathsf{Unblind}_2)$ so that for all messages m and keys $k_1 \xleftarrow{\$} \mathsf{KGen}_1, k_2 \xleftarrow{\$} \mathsf{KGen}_2$

$$\mathsf{Unblind}_2(\mathsf{Unblind}_1(\mathsf{Blind}_2(\mathsf{Blind}_1(m, k_1), k_2), k_1), k_2) = m$$

5.2 The Protocol

Firstly, we define a few permutations that will be used in the protocol.

$$\sigma_d : \mathbb{Z}_N \to \mathbb{Z}_N \qquad \sigma_d^e : \mathbb{Z}_{N+1} \to \mathbb{Z}_{N+1}$$

$$x \mapsto x + d \mod N \qquad x \mapsto \begin{cases} \sigma_d(x) & \text{if } 0 \leq x < N \\ N & \text{if } x = N \end{cases}$$

$$\tau_N^b : X^N \to X^N \qquad \pi^b : X^{N+1} \to X^{N+1}$$

$$\boldsymbol{v} \mapsto \langle \boldsymbol{v}[i(-1)^b + (N-1)b] \rangle_{i \in \mathbb{Z}_N} \qquad \boldsymbol{v} \mapsto \langle \boldsymbol{v}[(i-b) \bmod (N+1)] \rangle_{i \in \mathbb{Z}_{N+1}}$$

Figure 5.1 is the exchange protocol.

5.2.1 Protocol Overview

Our protocol follows the idea used by Roscoe and Ryan [139], i.e. Alice and Bob each hide their secret among a set of dummy messages. The aim is to prevent any party from predicting when the secrets will be exchanged or distinguish when they have sent or received a secret until all exchanges should have finished. A

Optimally Fair Exchange Protocol

Alice		**Bob**
	Setup phase	
$d_A \xleftarrow{\$} \mathbb{Z}_N; \boldsymbol{k^A} \xleftarrow{\$} \mathcal{K}^{N+1}$		
$B_1^A \xleftarrow{\$} \mathsf{KGen}_1(1^\lambda); B_2^A \xleftarrow{\$} \mathsf{KGen}_2(1^\lambda)$		
$\boldsymbol{c^A} \leftarrow \langle \mathsf{Blind}_1(\boldsymbol{k^A}[\sigma^e_{d_A}(i)], B_1^A)\rangle_{i\in\mathbb{Z}_{N+1}}$		
$D_A \leftarrow \mathsf{Delay}\left(\mathsf{Enc}_{\boldsymbol{k^A}[0]}(s_A)\|B_2^A\|d_A\|b_A\right)$		
$\mathbf{S_1}$:	$D_A, \boldsymbol{c^A} \longrightarrow$	
		$d_B \xleftarrow{\$} \mathbb{Z}_N; \boldsymbol{k^B} \xleftarrow{\$} \mathcal{K}^N$
		$B_1^B \xleftarrow{\$} \mathsf{KGen}_1(1^\lambda); B_2^B \xleftarrow{\$} \mathsf{KGen}_2(1^\lambda)$
		$\boldsymbol{c^B} \leftarrow \langle \mathsf{Blind}_1(\boldsymbol{k^B}[\sigma_{d_B}(i)], B_1^B)\rangle_{i\in\mathbb{Z}_N}$
		$\boldsymbol{d} \leftarrow \pi^{b_B}\left(\langle \mathsf{Blind}_2(\boldsymbol{c^A}[\sigma^e_{d_B}(i)], B_2^B)\rangle_{i\in\mathbb{Z}_{N+1}}\right)$
		$D_B \leftarrow \mathsf{Delay}\left(\mathsf{Enc}_{\boldsymbol{k^B}[0]}(s_B)\|B_2^B\|d_B\|b_B\right)$
$\mathbf{S_2}$:	$\longleftarrow D_B, \boldsymbol{c^B}, \boldsymbol{d}$	
$\boldsymbol{c^{BA}} \leftarrow \langle \mathsf{Blind}_2(\boldsymbol{c^B}[\sigma_{d_A}(i)], B_2^A)\rangle_{i\in\mathbb{Z}_N}$		
$\boldsymbol{e^B} \leftarrow \tau_N^{b_A}\left(\boldsymbol{c^{BA}}\right)$		
$\boldsymbol{e^A} \leftarrow \tau_{N+1}^{b_A}(\boldsymbol{d})$		
$\boldsymbol{m^A} \leftarrow \langle \mathsf{Unblind}_1(\boldsymbol{e^A}[i], B_1^A)\rangle_{i\in\mathbb{Z}_{N+1}}$		
$\mathbf{S_3}$:	$\boldsymbol{e^B} \longrightarrow$	
		$\boldsymbol{m^B} \leftarrow \langle \mathsf{Unblind}_1(\boldsymbol{e^B}[i], B_1^B)\rangle_{i\in\mathbb{Z}_N}$
	Exchange phase	
$\mathbf{E_{2r+1}}$:	$\boldsymbol{m^A}[r] \longrightarrow$	
$\mathbf{E_{2r+2}}$:	$\longleftarrow \boldsymbol{m^B}[r]$	

Figure 5.1: Main protocol.

key component to achieve this is timed-release encryption. The delayed messages D_A, D_B fulfil two different purposes. Firstly, they guarantee the exchanged secrets cannot be computed as soon as the keys $\boldsymbol{k^A}[0], \boldsymbol{k^B}[0]$ are received, but only at the end of the protocol. This deprives the agents of information that lets them know when to abort. Secondly, they behave as a timed commitment to each player's strategy. This is used to detect any active cheater[1]. Most of the complexity of the protocol lies in the three setup messages that are used to hide $\boldsymbol{k^A}[0]$ and $\boldsymbol{k^B}[0]$ among dummy keys. The shuffling process proceeds as follows:

1. Each party places their secret among a list of N values and permutes it. Alice places an extra dummy value at the end of her list.
2. After exchanging these lists, each party permutes the received list.
3. Bob decides whether to place Alice's extra dummy at the start or end of Alice's list.
4. Alice decides whether to reflect each list around their midpoint or not.

In the first two steps, we restrict the permutations to rotations. This makes the permutations commute without affecting the probability distribution of where the secrets are. After step 2, we obtain two lists where the secrets are in the same position. This in not enough to achieve optimality since the relative order of the secrets is fixed. If the messages were exchanged in their form after step 2, the secrets were in position r and Alice were the first at sending the r^{th} blinded value, then Bob's secret would follow Alice's. This would achieve a sub-optimal fairness probability of roughly $1 - \frac{1}{N}$.

The last two steps have the effect of shuffling the relative order of the secrets. If Bob places the extra dummy at the start of Alice's list, then he is effectively moving Alice's secret after his. Alice's action changes the relative order of any pair of messages. At the end of step 4, the relative order of the secrets is uniformly

[1]For example, if its initially declared and committed strategy obliges A to send its own secret in message x, then not doing so would be a detectable abuse even if the protocol is terminated before its end.

random and unknown to the parties. In particular, this represents the optimal distribution, i.e. no probability distribution of the 2 secret messages leads to a fairer exchange.

One could conclude the protocol by asking each party to release the content of the delayed message. This would constitute a substantial optimisation because the time-lock puzzle is computationally expensive, and it would only be solved when the protocol is not terminated. However, from a security perspective, there is no need to have extra messages, therefore, in our exposition and analysis, we keep the protocol as short and simple as possible.

5.2.2 Security Requirements

After explaining the protocol, we can justify the need for the security parameters we set. The requirements on the timed-release encryption and symmetric encryption should be straight forward. These two primitives are only meant to hide the messages encrypted.

The commutative blinding scheme must satisfy more complex requirements since it is the cornerstone of the setup phase. The scheme is required to hide the blinded values as well as hiding the shuffling. As a result, to require that nothing can be extracted from $\boldsymbol{c^A}$ or $\boldsymbol{c^B}$, the IND-CPA security we ask is more than sufficient. In particular, KPA security could be enough as when the protocol is aborted early, but some messages are sent, then the adversary has knowledge of some plaintexts. The (weak) CPA requirement on Blind_2 is needed to guarantee that Alice cannot guess d_B, b_B from $\boldsymbol{d}$, and prevents Bob from guessing d_A, b_A from $\boldsymbol{e^B}$.

In the protocol of Figure 5.1, we only use two keys for the commutative blinding. While Blind_1 must use only one key to allow each party to use $\mathsf{Unblind}_1$ without knowing the permutation on the list of message, Blind_2 could be used with multiple keys. The only advantage of this approach is allowing more constructions of the commutative blinding, potentially allowing the use of simple xor masking as Blind_2. However, in this circumstance, one needs to set an additional requirement

to guarantee that no information on the permutations is leaked by the keys used. In particular, note that $\boldsymbol{m}^{\boldsymbol{B}}[i]$ would be blinded using a key whose index depends entirely upon on b_A. Therefore, Alice can unblind messages as she receives them. However, she cannot check if the message received contain the correct key (since the encryption of the secret is delayed) and the IND-CPA requirement prevents her from obtaining information from $\boldsymbol{c}^{\boldsymbol{B}}$. On the other side, $\boldsymbol{m}^{\boldsymbol{A}}[i]$ is blinded using a key dependent on both b_A and b_B. Therefore, Bob does not know a priori which key is used. If he were to discover which key is used to blind $\boldsymbol{m}^{\boldsymbol{A}}[i]$, then he would learn b_A. This gives Bob knowledge of whether $\boldsymbol{k}_{\boldsymbol{A}}[0]$ precedes or succeeds $\boldsymbol{k}_{\boldsymbol{B}}[0]$, which constitutes an unfair advantage.

As a result, we present the protocol with this stricter definition of the commutative blinding and give implementations satisfying the requirements in Section 4. Nevertheless, the same issue arises when considering the random tokens that a pseudo-random encryption function would use. Forcing Blind_2 to use the same random tokens in each encryption is not a viable option as it would make any candidate function fail the (weak and strong) IND-CPA requirement. Hence, the need for Equation 4.1 to hold, which asks $\mathsf{Unblind}_1$ to completely hide which random tokens were used in the previous encryptions.

5.2.3 Commutative Blinding and why it is needed

The commutative blinding scheme is the most complex primitive used in our protocol. Quite a few options are available for commutative encryptions: ElGamal, SRA, Pohlig-Hellman, Massay-Omura, etc.. However, when used as is, most do not satisfy our requirements. Chapter 4 is entirely dedicated to the construction of suitable schemes. Crucially, we propose two concrete constructions, one of which is supposed to be quantum-secure.

Given the complexity of commutative blinding, a natural question is whether we can achieve the same results without its use. While this topic is discussed in depth in Section 5.6, we give here a hint on the complexity of finding a solution.

Our aim of hiding the secret messages among a list of other dummy messages is similar to the issue of shuffling a deck of cards for an online game of poker. No single party must know the whole shuffle and both need to contribute to it. This "mental poker" problem is often solved using commutative blinding, oblivious transfer or by generating cards on the fly. [74, 76, 77, 142] The last option is clearly not an option in our context. On the other hand, the use of oblivious transfer is appealing. Standard results [75] imply that there is a protocol using OT to compute any functionality securely with abort. Thus, our setup can be instantiated using this method. However, these techniques require expressing the functionality as a circuit which is later "garbled" [166, 75]. Thus, the complexity of the setup phase will greatly increase.

A more efficient protocol relying on OT could be constructed similarly to [48] where one OT protocol is used to allow both parties to perform a joint shuffle on a list. However, this technique can only be translated to our setting if the shuffling phase could be split into two sub-shuffles each performed independently by the two parties. Unfortunately, we could not find such shuffle. In Section 5.6, we suggest a different approach which relies on nesting OT within each other and setting stronger requirements on the OT primitives themselves. However, we do not provide a full security proof as this would go well beyond the purpose of establishing that there is a working 2-party optimally-fair exchange protocol using TRE.

5.3 Security Analysis

A protocol Π^{M+1} is a sequence of messages $\langle m_0, \ldots, m_M \rangle$. A time point t in the protocol Π^M is an integer $t \in \mathbb{Z}_M$. We say that messages m_i with $i \leq t$ happens before t. We write T_A (and T_B) for the time point after which Alice (respectively Bob) is guaranteed to obtain s_B (respectively s_A). For instance, $m_0 = \mathsf{Enc}_k(s_A) \wedge m_3 = k \implies T_B \leq 3$. A subtler example is: $m_0 = \mathsf{Enc}_k(s_A) \wedge m_4 = \mathsf{Delay}(k) \implies T_B \leq 4$. Using this simple model, we can analyse the fairness of our

protocol against passive adversaries. In this context, we define a passive adversary as one that is only allowed to go offline unexpectedly.

In this section, we write $s_A \in \boldsymbol{m}^{\boldsymbol{A}}[i]$ to mean that $\boldsymbol{m}^{\boldsymbol{A}}[i]$ is the blinding of $\boldsymbol{k}^{\boldsymbol{A}}[0]$. Essentially, $\boldsymbol{m}^{\boldsymbol{A}}[i]$ is the only important message in $\boldsymbol{m}^{\boldsymbol{A}}$. The analogue notation $s_B \in \boldsymbol{m}^{\boldsymbol{B}}[i]$ will also be used.

5.3.1 Adversarial Model

In our adversarial model, corruptions are static, i.e., the adversary decides whether to control Alice or Bob before the protocol begins. We assume communication channels are authenticated and guarantee confidentiality. Additionally, we assume that replaying messages between different exchanges is not possible. Any malformed message is ignored, treating external tampering as a network failure. However, receiving a malformed message signed by the adversary constitutes proof of their misbehaviour, though aborting communication is not considered a malicious action as in many circumstances it can happen through chance.

Recall our definitions of adversarial behaviour from Section 2.3.2. A *passive* adversary is only allowed to perform side computations and abort communication. A *covert* adversary misbehaves only if they can avoid detection. Finally, an *active* adversary has unrestricted behaviour.

We first prove security against passive adversaries, then demonstrate how the protocol is also secure against covert adversaries in Section 5.3.3. Finally, we show how to enhance the protocol to achieve security against active adversaries in the following section. Formally, all adversaries we consider are non-uniform, meaning that in addition to the protocol input, they can receive extra auxiliary information.

5.3.2 Fairness

We will show that the protocol of Figure 5.1 achieves $\frac{1}{2N}$-partial fairness. The key idea is that T_A and T_B are kept next to each other so that there is only one successful early-termination attack among the $2N$ possible. Due to the use of

time-lock puzzles, we see that $T_A = i + 2$ if and only if E_i is the blinding of $\boldsymbol{k}^{\boldsymbol{B}}[0]$. Similarly, $T_B = i + 2$ if and only if E_i is the blinded $\boldsymbol{k}^{\boldsymbol{A}}[0]$. Therefore, we compute the distribution of T_A, T_B by looking at the distribution of $\boldsymbol{k}^{\boldsymbol{A}}[0]$ and $\boldsymbol{k}^{\boldsymbol{B}}[0]$.

Theorem 5.1. *Let Π be the fair exchange protocol from Figure 5.1. Then*

$$\Pr\left[s_A \in \boldsymbol{m}^{\boldsymbol{A}}[i] \wedge s_B \in \boldsymbol{m}^{\boldsymbol{B}}[j]\right] = \begin{cases} \frac{1}{2N} & \text{if } i = j \vee i = j + 1 \\ 0 & \text{otherwise} \end{cases} \tag{5.1}$$

Proof. Note that

$$\boldsymbol{m}^{\boldsymbol{A}}[i] = \mathsf{Blind}_2\left(\boldsymbol{k}^{\boldsymbol{A}}[\sigma^e_{d_A+d_B}((i(-1)^{b_A} + Nb_A - b_B) \bmod (N+1))], B_2^B\right) \tag{5.2}$$

$$\boldsymbol{m}^{\boldsymbol{B}}[i] = \mathsf{Blind}_2\left(\boldsymbol{k}^{\boldsymbol{B}}[(i(-1)^{b_A} + (N-1)b_A + d_A + d_B) \bmod N], B_2^A\right) \tag{5.3}$$

So

$$s_A \in \boldsymbol{m}^{\boldsymbol{A}}[i] \iff 0 = \sigma^e_{d_A+d_B}((i(-1)^{b_A} + Nb_A - b_B) \bmod (N+1))$$
$$s_B \in \boldsymbol{m}^{\boldsymbol{B}}[i] \iff 0 = (i(-1)^{b_A} + (N-1)b_A + d_A + d_B) \bmod N$$

Hence

$$\begin{aligned}\Pr\left[s_B \in \boldsymbol{m}^{\boldsymbol{B}}[i]\right] &= \Pr_{\substack{b_a \xleftarrow{\$} \{0,1\} \\ d_A, d_B \xleftarrow{\$} \mathbb{Z}_N}}\left[0 = (i(-1)^{b_A} + (N-1)b_A + d_A + d_B) \bmod N\right] \\ &= \frac{1}{2}\Big(\Pr_{d \xleftarrow{\$} \mathbb{Z}_N}[0 = (i+d) \bmod N] + \\ &\qquad \Pr_{d \xleftarrow{\$} \mathbb{Z}_N}[0 = (N-1-i+d) \bmod N]\Big) = \frac{1}{N}\end{aligned}$$

Assume $s_B \in \boldsymbol{m}^{\boldsymbol{B}}[j]$, that is:

$$0 = (j(-1)^{b_A} + (N-1)b_A + d_A + d_B) \bmod N$$

We will show that s_A is either in $\boldsymbol{m}^{\boldsymbol{A}}[j]$ or $\boldsymbol{m}^{\boldsymbol{A}}[j+1]$ by analysing the four cases $(b_A, b_B) \in \{0,1\}^2$

1. $\mathbf{b_A = b_B = 0}$

$$s_B \in \boldsymbol{m}^{\boldsymbol{B}}[j] \implies \sigma^e_{d_A+d_B}(j) = (j + d_A + d_B) \bmod N = 0 \implies s_A \in \boldsymbol{m}^{\boldsymbol{A}}[j]$$

2. $\mathbf{b_A = b_B = 1}$

$$s_B \in \boldsymbol{m^B}[j] \implies \sigma^e_{d_A+d_B}(N-1-j) = (N-1-j+d_A+d_B) \bmod N = 0$$
$$\implies s_A \in \boldsymbol{m^A}[j]$$

3. $\mathbf{b_A = 0 \wedge b_B = 1}$

$$s_B \in \boldsymbol{m^B}[j] \implies \sigma^e_{d_A+d_B}(j+1-b_B) = (j+d_A+d_B) \bmod N = 0$$
$$\implies s_A \in \boldsymbol{m^A}[j+1]$$

4. $\mathbf{b_A = 1 \wedge b_B = 0}$

$$s_B \in \boldsymbol{m^B}[j] \implies$$
$$\sigma^e_{d_A+d_B}(N-(j+1)) = (N-j-1+d_A+d_B) \bmod N = 0$$
$$\implies s_A \in \boldsymbol{m^A}[j+1]$$

As a result,

$$\Pr_{b_A,b_B \xleftarrow{\$} \{0,1\}}\left[s_A \in \boldsymbol{m^A}[i] \;\middle|\; s_B \in \boldsymbol{m^B}[j]\right] = \begin{cases} \frac{1}{2} & \text{if } i = j \vee i = j+1 \\ 0 & \text{otherwise} \end{cases}$$

The statement of the theorem follows directly from the equation

$$\Pr\left[s_A \in \boldsymbol{m^A}[i] \wedge s_B \in \boldsymbol{m^B}[j]\right] =$$
$$\Pr\left[s_A \in \boldsymbol{m^A}[i] \;\middle|\; s_B \in \boldsymbol{m^B}[j]\right] \Pr\left[s_B \in \boldsymbol{m^B}[j]\right]$$

□

While the above proves the distribution of the secrets, we still have to prove that the protocol is secure and achieves $\frac{1}{2N}$-partial fairness. Recall the formal definition of partial fairness from Chapter 2, we need to prove that for each (computationally bounded) adversary $\mathcal{A}$ there is a simulator $\mathcal{S}$ for which the outcome of the real-world protocol attacked by $\mathcal{A}$ is indistinguishable with probability $1 - \frac{1}{2N}$ from the outcome of the ideal-world functionality attacked by $\mathcal{S}$. We prove all the above

considering a passive adversary. Dealing with an active adversary will require some more assumptions on the protocol and these will be described later in this chapter.

Before diving into the simulatability proof, we provide more intuitive sketch proofs to ease the reader into understanding our protocol. Firstly, we can prove that the protocol does not disclose the permutation used in shuffling the secrets while the delayed messages are still unknown. If we further prove that, once the delays are opened, you can only retrieve the secrets if you have received the corresponding message in the exchange phase, then our protocol achieves its aim. In light of Theorem 5.1, the above shows how partial fairness is obtained: an adversary can only blindly pick a round where to abort, hence the chance of them guessing the correct point is at most $\frac{1}{2N}$.

Lemma 5.2. *In the protocol of Figure 5.1, while the delays are not revealed, B's view is independent of d_A, b_A and similarly, A's view is independent of d_B, b_B.*

Proof. This proof was machine-checked using CryptoVerif [25] and the script is shown in Appendix B. Consider the case where Bob is corrupt. Let Π_0 be the game where Bob is running the protocol of Figure 5.1 and must guess d_A, b_A. Firstly, since the delayed messages are not revealed, the protocol is indistinguishable from one where the delayed messages contain a string of zeros of the appropriate length. Thus, Π_0 is indistinguishable from the protocol Π_1 where D_A contains only zeros. From Bob's point of view, $\boldsymbol{e^B}$ is indistinguishable from any other N-vector of ciphertexts of Blind_2, due to the semantic security of Blind_2. Technically, by using the IND-CPA of Blind_2 on each item of $\boldsymbol{c^{BA}}$, $\boldsymbol{e^B}$ is (computationally) indistinguishable from $\tau_N^{b_A}(\langle \mathsf{Blind}_2(c_0, B_2^A)\rangle_{i\in\mathbb{Z}_N})$ where c_0 is any fixed valid ciphertext. Since c_0 is fixed, then permuting this list is the same as simply permuting the randomness tokens used by Blind_2. These are picked uniformly at random, hence the lists are statistically indistinguishable. As a result, Π_1 is computationally indistinguishable from Π_2 where $\boldsymbol{e^B}$ is replaced by $\langle \mathsf{Blind}_2(c_0, B_2^A)\rangle_{i\in\mathbb{Z}_N}$. Using the requirement on $\mathsf{Unblind}_1$ (Eq. 4.1), $\boldsymbol{m^A}$ is equivalent to $(\tau_{N+1}^{b_A} \circ \pi^{b_B})(\langle \mathsf{Blind}_1(\boldsymbol{k^A}[(\sigma_{d_B}^e \circ \sigma_{d_A}^e)(i)], B_2^B)\rangle)$, where Blind_1 is using fresh

randomness tokens. Since $\boldsymbol{k^A}$ is picked at random by Alice, the above vector is indistinguishable from $\langle \mathsf{Blind}_1(\boldsymbol{k'^A}, B_2^B)\rangle$ where $\boldsymbol{k'^A}$ is a new randomly sampled vector from $\mathcal{K}^{N+1}$. Thus Π_2 is indistinguishable from Π_3 where $\boldsymbol{m^A}$ is constructed as above without using d_A or b_A. Finally, $\boldsymbol{c^A}$ is indistinguishable from any other vector of ciphertexts of Blind_1 due to the semantic security of Blind_1. Thus, Π_3 is indistinguishable from Π_4 where $\boldsymbol{c^A} \leftarrow \langle \mathsf{Blind}_1(0, B_1^A)\rangle_{i\in\mathbb{Z}_{N+1}}$. Since Π_4 has no trace of d_A nor b_A, the overall protocol hides these parameters.

Similarly, we can show that Alice's view is independent of d_B, b_B. The key steps are: the indistinguishability of Blind_2 on $\boldsymbol{d}$, Eq. 4.1 for $\boldsymbol{m^B}$, and the indistinguishability of Blind_1 on $\boldsymbol{c^B}$. □

Lemma 5.3. *In the protocol of Figure 5.1, Alice can obtain s_B if and only if they receive $\boldsymbol{m^B}[x_A]$ where $x_A(-1)^{b_A} + (N-1)b_A + d_A + d_B = 0 \mod N$. Likewise, Bob can obtain s_A if and only if they receive $\boldsymbol{m^A}[x_B]$ where $\sigma^e_{d_A+d_B}((x_B(-1)^{b_A} + Nb_A - b_B) \bmod (N+1)) = 0$.*

Proof. First note that the values x_A and x_B are taken from Theorem 5.1 and their exact values are not important in the dynamic of this proof. Assume Alice is given a transcript of the protocol as well as the content of D_B (i.e. $b_B, d_B, \mathsf{Enc}_{\boldsymbol{k^B}[0]}(s_B)$). Due to the security of the encryption scheme, s_B can only be retrieved if $\boldsymbol{k^B}[0]$ can be retrieved. If $\boldsymbol{m^B}[x_A]$ has not been received, then $\boldsymbol{k^B}[0]$ always appears blinded in the ciphertext $\mathsf{Blind}_1(\boldsymbol{k^B}[0], B_c^B)$ since the introduction of $\boldsymbol{c^B}$. Since Blind_1 is semantically secure, $\mathsf{Blind}_1(\boldsymbol{k^B}[0], B_c^B)$ is indistinguishable from any other ciphertexts of Blind_1. Thus, Alice's view is indistinguishable from one where $\boldsymbol{k^B}[0]$ does not appear at all.

Likewise, in Bob's view of the protocol, $\boldsymbol{k^A}[0]$ always appear in the ciphertext $\mathsf{Blind}_1(\boldsymbol{k^A}[0], B_c^A)$ introduced by $\boldsymbol{c^A}$. The IND-CPA property of Blind_1 guarantees that $\boldsymbol{k^A}[0]$ remains secret, hence s_A cannot be retrieved due to the COA security of Enc. □

The above lemmas together with Theorem 5.1 gives the intuition on why our protocol achieves partial fairness in a secure way. However, we report below the formal simulatability proof.

Theorem 5.4. *For any (passive) PPT adversary $\mathcal{A}$ in the real-world (Figure 5.1), there is a corresponding simulator $\mathcal{S}$ in the ideal world so that*

$$\text{IDEAL}_{\mathcal{S}}(s_A, s_B) \overset{1/2N}{\approx} \text{REAL}_{\mathcal{A}}(s_A, s_B)$$

We prove this theorem in the "classical" (e.g. [75]) ideal/real-world paradigm where the simulator will *rewind* the real-word adversary. Formally, this is done by treating the adversary as deterministic with any required random coins as auxiliary inputs. This drastically differs from the Universal Composability framework [34] where the simulator cannot rewind the environment.[2]

Proof of Theorem 5.4. Fix the adversary $\mathcal{A}$, we describe the simulator $\mathcal{S}$. For ease of presentation, we assume that $\mathcal{A}$ is impersonating Alice. Note that the scenario in which Bob is malicious is essentially the same, hence omitted. $\mathcal{S}$ with secret s_A will run as follows:

1. start a local instance of $\mathcal{A}$ with the same inputs (both s_A and any auxiliary ones) as themselves;
2. pick a random s'_B and run the setup phase with $\mathcal{A}$;
3. after the setup phase is run, $\mathcal{S}$ computes $\mathsf{Open}(D_A)$ to extract the values d_A and b_A chosen by $\mathcal{A}$;
4. thus, $\mathcal{S}$ can compute the rounds R_A and R_B where the secrets will be exchanged;

[2] In the proof, the rewinding ability of the simulator is used to "get around" the hiding properties of the timed commitment D_A. In the UC framework, no commitment scheme can be instantiated in the plain model [36] (i.e. without any trust). In this context, one need to use equivocable and extractable commitment schemes in the common reference string model: that is, commitment schemes where the simulator, by carefully picking the common string, can break both the hiding and binding properties.

5. during the opening of the delayed message, $\mathcal{A}$ would have terminated the protocol, hence rewind it to the start of the exchange phase;
6. acting as Bob, perform the exchange phase with $\mathcal{A}$ up to the message $\boldsymbol{m}^{\boldsymbol{A}}[R_A]$;
7. upon receipt of that message, using the knowledge of $\mathsf{Open}(D_A)$, extract the secret s'_A used by $\mathcal{A}$ in the exchange;
8. if $\mathcal{S}$ cannot extract s'_A (e.g. R_A is never reached), assume $s'_A = s_A$;
9. if $\boldsymbol{m}^{\boldsymbol{B}}[R_B]$ was already sent, rewind $\mathcal{A}$ to immediately before $\mathcal{S}$ sends it;
10. send s'_A to the fair exchange TTP and receive back s_B;
11. using the non-committing property of the encryption scheme obtain the key k' so that $\mathcal{A}$ would decrypt $\mathsf{Enc}_{\boldsymbol{k}^{\boldsymbol{B}}[0]}(s'_B)$ as s_B;
12. construct the blinded message m' using k' and send it to $\mathcal{A}$ in place of $\boldsymbol{m}^{\boldsymbol{B}}[R_B]$;
13. continue the exchange phase until the end;
14. output whatever $\mathcal{A}$ outputs.

Whenever $\mathcal{A}$ sends a malformed message or aborts, $\mathcal{S}$ will abort the fair exchange. We now need to show that $\mathcal{A}$'s view in the simulation is indistinguishable from the real-world as well as proving that the output of the ideal world with $\mathcal{S}$ is $\frac{1}{2N}$-indistinguishable from the output of the real-world. Note that the only difference between (the final run without rewinding of) the protocol simulated by $\mathcal{S}$ and the real-world is the use of the non-committing property of the encryption scheme. By definition of such property, $\mathcal{A}$ would not be able to distinguish between m' and $\boldsymbol{m}^{\boldsymbol{B}}[R_B]$. In particular, Lemma 5.3 guarantees that $\mathcal{A}$ never obtains s'_B. Thus $\mathcal{A}$'s view in the simulation is indistinguishable from the view in the real-world. It follows that $\mathcal{A}$'s output in the simulation is indistinguishable from their output in the real-world. However, partial fairness arises from the differences in the output of the honest party. In particular, we analyse the different cases depending on when $\mathcal{A}$ aborts. Assume that $\mathcal{A}$ aborts before the message m_t:

1. $t \leq \min(R_B, R_A)$: the real-world protocol achieves nothing and $\mathcal{S}$ does not interact with the fair exchange functionality. Thus, the honest party will output an "abort token" $\perp$ in both cases.

2. $t > \max(R_B, R_A)$: the real-world protocol successfully exchanges the secrets and $\mathcal{S}$ interacts with the fair exchange functionality. The honest party will output s'_A.

3. $R_A < t = R_B$: in the real-world Bob receives s'_A and so does in the ideal world where $\mathcal{S}$ interacts with the functionality.

4. $R_B < t = R_A$: in the ideal-world $\mathcal{S}$ cannot extract s'_A, hence Bob receives s_A. In the real-world, Bob receives nothing.

To prove that the last case happens with probability at most $\frac{1}{2N}$, we recall Lemma 5.2. Since $\mathcal{A}$ cannot distinguish between different protocol runs where all that differs is the choice of d_B, b_B, $\mathcal{A}$'s choice of aborting at time t is independent of these values. Hence, the probability of the last event is the same as $\Pr\left[s_A \in \boldsymbol{m^A}[t] \wedge s_b \in \boldsymbol{m^B}[t-1]\right] = \frac{1}{2N}$. □

5.3.3 Covert Adversaries

So far we have analysed the fairness of our protocol when the adversary is passive. We now look at the case where the adversary can alter the messages they send. Our aim is to show that any such adversary will be caught. That is, our protocol allows the honest party to demonstrate the misbehaviour of the other party. Since channels are authenticated a transcript of the protocol constitutes a proof of misbehaviour. It will then follow that covert adversary can only behave like passive adversaries.

Theorem 5.5. *If a party misbehaves causing unfairness, the other can prove it (unless both misbehaved).*

Proof. Firstly, note that inconsistencies between the delayed messages and the shuffling "sub-messages" $\boldsymbol{c^A}, \boldsymbol{c^B}, \boldsymbol{d}, \boldsymbol{e^B}$ will result in inconsistent $\mathbf{E_i}$'s. We assume

that Alice and Bob have agreed on predicates (i.e. boolean functions) $\mathsf{Exp}_A, \mathsf{Exp}_B$ to indicate what they expect to receive. Recall Equations 5.2 and 5.3 and assume that both parties have knowledge of the content of the delayed messages D_A and D_B, which they eventually will. Therefore, both parties have complete knowledge of the permutations used. Each exchange message $\mathbf{E_i}$ can be unblinded and reordered so as to reconstruct $\boldsymbol{k^A}$ and $\boldsymbol{k^B}$. Finally, the secrets can be retrieved and checked against Exp_A or Exp_B. Assume that $\boldsymbol{m^A}[i]$ is not correct, then Bob could be responsible only if they have wrongly computed $\boldsymbol{d}$. However, note that the function $\boldsymbol{c^A} \mapsto \boldsymbol{d}$ is entirely determined by (d_B, b_B, B_2^B). Therefore, one can verify the computation of $\boldsymbol{d}$ and, if this is correct, prove that Alice was responsible for the issue with $\boldsymbol{m^A}[i]$. Similarly, Alice's computation $\boldsymbol{c^B} \mapsto \boldsymbol{e^B}$ is determined by (d_A, b_A, B_2^A). It follows that anyone holding a transcript of the protocol (and $\mathsf{Exp}_A, \mathsf{Exp}_B$) can check if each $\mathbf{E_i}$ is correct and determine who introduced any eventual error. □

A caveat of the above theorem is the fact that a party could not follow the protocol yet appear to be passive. It should be clear that constructing the shuffle without respecting the values committed in the delay messages will result in an incoherent exchange phase. Thus, this would be detected and punished as per the above theorem. However, a covert adversary which deviates from the protocol by picking values not randomly but according to some other distribution, will not be detectable with certainty. Thus, we cannot enforce that a covert adversary will adhere to the correct guidelines on choosing (for instance) $\boldsymbol{k^A}, B_1^A, B_2^A, d_A, b_A$ and all the other random seeds used in the commutative blinding. This could cause some problem if such an adversary could obtain more information during the setup phase by cherry-picking the supposedly random values. Thankfully, the strong requirements on the commutative blinding are picked assuming this behaviour. In particular, the proof of Lemma 5.2 assumes that the adversary is free to pick its inputs however they wish.

5.3.4 Active Adversaries

Recall from Chapter 2 that an *active* adversary, in the context of MPC, is simply an adversary with unrestricted behaviour. Our protocol requires some modifications to achieve security against active adversaries. While we could rely on the GMW compiler from Chapter 2, we do not require the generation of a random tape or proof that random seeds are correctly sampled. This is because our construction is already secure against covert adversaries, who are allowed to sample random values as they please (they can never be blamed for cherry-picking a random value). Therefore, we only augment the protocol by accompanying each message with a non-interactive zero-knowledge proof of its correctness. Specifically, the set $\{d_A, b_A, B_1^A, \boldsymbol{k^A}, B_2^A\}$ will act as a witness to all messages sent by Alice (and similarly for Bob). Any message with an incorrect proof can be treated as an abort signal. We formalise this in the theorem below.

Theorem 5.6. *For any active adversary $\mathcal{A}$ to the modified protocol Π', there is a covert adversary $\mathcal{B}$ for the original protocol Π such that*

$$\{\text{REAL}_{\mathcal{A}}^{\Pi'}(x)\} \overset{c}{\approx} \{\text{REAL}_{\mathcal{B}}^{\Pi}(x)\}$$

Proof. Since this theorem is a simplified version of the GMW compiler, its proof is also a simplified version of the proof for the GMW compiler. Here, we follow the textbook by Goldreich [75], translating it into our notation.

Consider a protocol Π'' that is equivalent to Π, but instead of exchanging messages directly, the messages are sent to a trusted entity. This entity knows the secret values $\{d_A, b_A, B_1^A, \boldsymbol{k^A}, B_2^A\}$ (and the corresponding values for Bob) and will verify the validity of the messages it receives. If the message is valid, it is forwarded to the other party in the protocol; otherwise, a token indicating proof of misbehaviour is sent instead. Protocol Π'' represents the goal we aim to achieve with the augmented protocol Π'.[3] Classical results (e.g., Proposition 7.4.15 in [75]) state that the above

[3] In technical terms, Π'' is the hybrid model of Π' using a functionality often referred to as *authenticated computation*.

functionality can be securely computed using zero-knowledge proofs, as we did in the transformation of Π. Therefore, there exists an adversary $\mathcal{A}'$ such that

$$\{\text{REAL}^{\Pi'}_{\mathcal{A}}(x)\} \overset{c}{\approx} \{\text{REAL}^{\Pi''}_{\mathcal{A}'}(x)\}$$

Our theorem follows from showing that there is a covert adversary $\mathcal{B}$ such that

$$\{\text{REAL}^{\Pi''}_{\mathcal{A}'}(x)\} \overset{c}{\approx} \{\text{REAL}^{\Pi}_{\mathcal{B}}(x)\}$$

The adversary $\mathcal{B}$ is straightforward. As usual, $\mathcal{A}'$ is assumed to be deterministic, and $\mathcal{B}$ will provide a random tape to $\mathcal{A}'$. Upon receipt of a message from the honest party, $\mathcal{B}$ will forward it to $\mathcal{A}'$ as if it originated from the trusted entity. When $\mathcal{B}$ is expected to send a message, it acts as the trusted entity in its simulation with $\mathcal{A}'$. $\mathcal{A}'$ will send $\mathcal{B}$ a message m (or abort communication). $\mathcal{B}$ can verify whether the message is computed correctly since it knows the input of $\mathcal{A}'$ as well as its tape (and even the "program" of $\mathcal{A}'$ if required). If m does not pass the check or if $\mathcal{A}'$ aborts communication, $\mathcal{B}$ will also abort. Otherwise, $\mathcal{B}$ forwards m to the honest party. $\mathcal{B}$ will output whatever $\mathcal{A}'$ outputs.

We note that $\mathcal{B}$ is a covert adversary, as he would not send messages that could incriminate him. Moreover, the view of $\mathcal{A}'$ within $\mathcal{B}$ is perfectly indistinguishable from its execution in Π'', thus ensuring simulatability. □

We should also point out that there is a trivial optimisation which would allow the use of zero-knowledge proofs only during the setup phase. In particular, once Alice (respectively Bob) has computed $\boldsymbol{m^A}$ (resp. $\boldsymbol{m^B}$), they can exchange a message (and a zero-knowledge proof) committing to each individual item of $\boldsymbol{m^A}$ (resp. $\boldsymbol{m^B}$) in order. Therefore, the exchange phase can be carried out as in Figure 5.1, with the additional task that each party should check the received message against its committed value. That is, the zero-knowledge proofs for $\boldsymbol{m^A}$ (resp. $\boldsymbol{m^B}$) are combined into a single proof and exchanged after the shuffling.

Remark. In Chapter 4, we constructed an RLWE-based commutative blinding scheme, but we had to relax the security requirement of $\mathsf{Unblind}_1$. To accommodate this relaxed version in the presence of active adversaries, we need to further augment our protocol by asking all parties to prove (in zero-knowledge) that the noise used in the encryption scheme is correctly sampled. Using the GMW compiler solves this issue altogether, so we can rely on this without needing further proofs.

5.4 Optimality

In the protocol from Figure 5.1, we hide the secrets s_A, s_B among the messages $\mathbf{E_i}$'s and use timed-release encryption to guarantee that $T_A = i + 2 \iff s_B \in \mathbf{E_i}$ (and similarly for Bob). Note that the "+2" is needed to account for the setup phase. Hence, T_A and T_B are also hidden for the entire duration of the exchange phase.

Here, we prove that our construction is "optimal", meaning that no exchange phase of M messages can achieve fairness greater than $1 - \frac{1}{M-1}$. However, this leaves open the question of whether our setup phase can be shortened. In this regard, we point out that messages $\mathbf{S_3}$ and $\mathbf{E_1}$ can be sent as one. Therefore, the proposed protocol achieves $\frac{1}{M-1}$-partial fairness using $M + 2$ messages.

Lemma 5.7. *A receives a message at time T_A.*

Proof. Assume for a contradiction that m_{T_A} is a message from Alice to Bob. It follows that Alice can compute m_{T_A} from $\{m_1, \ldots, m_{T_A-1}\}$. As a result, everything that Alice can compute from $\{m_1, \ldots, m_{T_A}\}$ can be done from $\{m_1, \ldots, m_{T_A-1}\}$. Therefore T_A is not minimal. □

Lemma 5.8. *In the absence of third parties, $T_A \neq T_B$.*

Proof. This follows directly from Lemma 5.7. □

Lemma 5.9. *Assume there is no third party. If in Π messages do not alternate between Alice and Bob, then there is another protocol Π' which is as fair as Π in which the messages alternate.*

Proof. Let Π be a protocol where the messages between Alice and Bob do not alternate. Construct Π' from Π by collapsing consecutive messages from the same party into one. After this process, append "dummy" messages so that Π' and Π have the same length. In this process, the fairness of Π is left untouched. In particular, note that the optimal strategy of any adversary could have not been to stop between consecutive messages, and it is definitely not to stop after dummy messages. □

Theorem 5.10. *Let Π be a fair exchange protocol between Alice and Bob in absence of third parties. If Π consists of $M+1$ messages, then there is an unfair run with probability $\frac{1}{M}$*

Proof. By Lemma 5.9 we can assume that $\Pi = \langle m_0, \ldots, m_M \rangle$ where Alice sends the messages with even index, and Bob those with odd index. There are M possible attacks on the protocol by early abortion: stop the protocol after i messages where $0 < i \leq M$. By Lemma 5.8 we know that at least one of these attacks would be successful. So, the probability of an unfair run is at least $\frac{1}{M}$. □

The above proves that the protocol from Figure 5.1 is optimal in the sense that no protocol can exist which guarantees a lower chance of failure while having the same or fewer exchange messages.

5.5 Multiple Attempts

So far, we have designed a fair exchange protocol that achieves optimal partial fairness in a trust-less environment where true fairness cannot be achieved. Similar to other partial fairness protocols [48, 78], our designed is based on the idea of hiding when the secrets will be exchanged among other dummy messages. While this approach proves to be optimal, it carries a practical flaw: fairness does not guarantee that the secrets will actually be exchanged. Moreover, recall that we are working in a context where the network is unreliable, meaning that two honest parties could run the protocol and fail to exchange the secrets. In a real-world

application, if this scenario were to occur, the most common solution would be to repeat the protocol, attempting once more to exchange the secrets. However, if this were to happen, the optimality analysis we carried out would fail. A malicious party, knowing that the protocol would be attempted multiple times can design a better strategy than aborting at arbitrarily points each time. More specifically, they would be better off limiting the number of exchange messages they send during all but the last attempt. This essentially limits the chance of their secret to be revealed, while still slightly increasing their chance of causing unfairness. In this section, we analyse this new context where Alice and Bob will attempt at exchanging secrets again if they have previously failed.

5.5.1 New Context and Abstract Model

Let Alice and Bob hold secrets s_A and s_B which they wish to exchange across an unreliable network. They do not trust each other and no external parties are involved. In the previous analysis of a single protocol, we assumed that both parties have agreed on the length of the fair exchange protocol. Therefore, here we assume that Alice and Bob have agreed on a maximal number of attempts to carry out as well as on the length of each of these attempts.

To reason about the issue of multiple attempts independently of the concrete protocol instantiations, we construct a very general model of a partial-fairness protocols akin to our discussion in Section 5.3. In particular, we view a protocol Π simply as a sequence of messages $\langle m_1, \ldots, m_N \rangle$ which alternate between the two different parties. To distinguish the parties of a protocol from Alice and Bob which are actors across multiple protocols, we call $\mathcal{A}$ and $\mathcal{B}$ the parties involved each abstract protocol Π. Furthermore, in any given protocol Π, $\mathcal{A}$ represents the party sending the first message. Given a protocol $\Pi = \langle m_1, \ldots, m_N \rangle$ and assuming Π is not trivial, knowledge of the messages $\langle m_1, \ldots, m_N \rangle$ gives both $\mathcal{A}$ and $\mathcal{B}$ knowledge of the secrets. However, not all messages might be required, meaning that we can identify "crucial" messages $m_{T_\mathcal{A}}$ and $m_{T_\mathcal{B}}$ so that $\langle m_1, \ldots, m_{T_\mathcal{A}} \rangle$ (respectively $\langle m_1, \ldots, m_{T_\mathcal{B}} \rangle$) represent the shortest prefix of Π that allows $\mathcal{B}$ (resp. $\mathcal{A}$) to obtain

$s_{\mathcal{A}}$ (resp. $s_{\mathcal{B}}$). In Section 5.3, we would have written $s_{\mathcal{A}} \in m_i$. Here, we simplify the notation one step further and simply write $s_{\mathcal{A}} = i$. Thus, fairness of Π can be represented by a matrix M with:

$$[M]_{i,j} = \Pr[s_{\mathcal{A}} = 2i - 1 \wedge s_{\mathcal{B}} = 2j] \tag{5.4}$$

Therefore, $[M]_{i,j}$ is the probability that $\mathcal{B}$ (resp. $\mathcal{A}$) can obtain the secret $s_{\mathcal{A}}$ (resp. $s_{\mathcal{B}}$) after receiving i (resp. j) messages. Crucially, we assume that, at the end of a protocol, both parties know which secrets, if any, were exchanged. This allows them to determine whether the exchange should be attempted again or not. We model multiple attempts as sequences of protocols by introducing the concept of *schemes*.

Definition 5.2 (Scheme)**.** A *scheme* $\mathcal{S}(k, \boldsymbol{N})$ is a sequence of k protocols Π_i involving $\boldsymbol{N}[i]$ messages.

For ease of notation, we write $\mathcal{E}$ for the malicious entity and $\mathcal{H}$ for the honest party. Whether $\mathcal{E}$ is $\mathcal{A}$ or $\mathcal{B}$ is outside our control. Hence, we assume that $\mathcal{E}$ will be able to pick which role to play. We say that $\mathcal{E}$ *succeeds in* (or *wins*) a protocol Π if they manage to obtain $s_{\mathcal{H}}$ without revealing $s_{\mathcal{E}}$. Similarly, we say that a protocol Π *fails* with probability ρ if $\mathcal{E}$ can succeed in Π with probability ρ. When analysing a single protocol, we defined optimal fairness in respect of the number of messages. Here, we can extend this to the concept of schemes.

Definition 5.3. We say that a scheme $\mathcal{S}(k, \boldsymbol{N})$ is *optimally fair* if there is no other scheme $\mathcal{S}'(k, \boldsymbol{N})$ such that $\forall \mathcal{E}$ $\Pr[\mathcal{E}$ succeeds in $\mathcal{S}'] < \Pr[\mathcal{E}$ succeeds in $\mathcal{S}]$.

5.5.2 Optimally-Fair Scheme

In this section, we use our abstract model to characterise optimally-fair schemes. We start by showing how to tackle the problem recursively.

Theorem 5.11. *If a scheme* $\mathcal{S}(k', \boldsymbol{N}) = \langle \Pi_1, \ldots, \Pi_{k'} \rangle$ *is optimally fair, then for any* $k < k'$ *the scheme given by* $\langle \Pi_{k'-k+1}, \ldots, \Pi_{k'} \rangle$ *is also optimally fair.*

Proof. Let $\mathcal{S}(k', \boldsymbol{N}) = \langle \Pi_1, \ldots, \Pi_{k'} \rangle$ be optimally fair with probability of failure ρ. Define

$$p_i = \Pr[\mathcal{E} \text{ succeeds in } \Pi_i] \qquad f_i = \Pr[\text{no secrets are exchanged in } \Pi_i]$$

Notice that $\mathcal{E}$ wins in a scheme $\mathcal{S}$ if it wins in a protocol Π_L for some L and in all the previous $L-1$ protocols no secrets were exchanged (so that $s_{\mathcal{E}}$ and $s_{\mathcal{H}}$ are still unknown in Π_L). Therefore:

$$\Pr[\mathcal{E} \text{ succeeds}] = \sum_{i=1}^{k'-k} \left(p_i \prod_{j=1}^{i-1} f_j \right) + \left(\prod_{j=1}^{k'-k+1} f_j \right) \sum_{i=k'-k+1}^{k'} \left(p_i \prod_{j=k'-k+1}^{i-1} f_j \right)$$

Since $\mathcal{S}(k', \boldsymbol{N})$ is optimal, then it minimises the above equation over the p_i's and f_i's. Note that $\sum_{i=k'-k+1}^{k'} \left(p_i \prod_{j=k'-k+1}^{i-1} f_j \right)$ is the probability of failure of the last k attempts. In particular, it does not depend on any of the previous protocols $\Pi_1, \ldots, \Pi_{k'-k}$. Hence, the overall equation is minimal when $\sum_{i=k'-k+1}^{k'} \left(p_i \prod_{j=k'-k+1}^{i-1} f_j \right)$ is minimal as well. It follows that the scheme $\langle \Pi_{k'-k+1}, \ldots, \Pi_{k'} \rangle$ is optimally fair as well. □

The above theorem shows us a way to compute an optimally-fair scheme by recursion. Assume we are given an optimally-fair scheme $\mathcal{S}(k, \boldsymbol{N})$ with failure chance ρ, we can compute an optimally-fair scheme $\mathcal{S}'(k+1, \langle N_0 \rangle^\frown \boldsymbol{N})$ by picking a protocol that minimises the following equation:

$$\Pr[\mathcal{E} \text{ succeeds}] = p + \rho f$$

where p and f are as in the proof of Theorem 5.11. Let M be the matrix related to the protocol we aim to construct. To minimise the above over all adversaries, we need to minimise:

$$\max_l \left(\sum_{\substack{i>l \\ j \leq l}} [M]_{i,j} + \rho \sum_{\substack{i>l \\ j>l}} [M]_{i,j}, \sum_{\substack{i \leq l \\ j \geq l}} [M]_{i,j} + \rho \sum_{\substack{i>l \\ j \geq l}} [M]_{i,j} \right) \tag{5.5}$$

We first prove the intuitive fact that the optimal fairness is achieved when the secrets are kept close to each other.

Theorem 5.12. *Let M be a $N \times N$ matrix of a protocol and $\rho \in (0, 1]$. If M minimises Equation 5.5, then $[M]_{i,j} \neq 0$ only if $i = j$ or $i = j + 1$.*

Proof. Assume for a contradiction that M minimises the required equation, but has non-zero entries in other locations. Consider the following matrix Q.

$$[Q]_{i,j} = \begin{cases} \sum_{l=i}^{N} [M]_{i,l} & \text{if } i = j \\ \sum_{l=i}^{N} [M]_{l,j} & \text{if } i = j + 1 \\ 0 & \text{otherwise} \end{cases}$$

Through simple algebra, one can easily show that Equation 5.5 for Q is lower than for M. Hence, the theorem follows. □

By the theorem above, we know that any protocol in an optimally-fair scheme will only have at most $N - 1$ non-zero values and these are on the diagonal and sub-diagonal. Thus, any matrix in this form can be written as:

$$[M]_{i,j} = \begin{cases} \alpha_{2i-1} & \text{if } i = j \\ \alpha_{2j} & \text{if } i = j + 1 \\ 0 & \text{otherwise} \end{cases}$$

for some values α_i. Note that α_i is the probability that the secrets are in the messages m_i, m_{i+1}.

As a result, our target Equation 5.5 can be simplified to

$$\max_{\mathcal{E}} \Pr[\mathcal{E} \text{ succeeds}] = \max_{l} \left(\alpha_l + \rho \sum_{i>l} \alpha_i \right) \tag{5.6}$$

Consider the protocol $\Pi_g(N, \rho)$ of N messages where the probability that the secrets are in messages m_i, m_{i+1} is $\alpha_i = \frac{\rho(1-\rho)^{N-1-i}}{1-(1-\rho)^{N-1}}$.

We will prove that $\Pi_g(N, \rho)$ is the protocol to use to extend optimally-fair schemes into other optimally-fair schemes.

Lemma 5.13. *$\Pi_g(N, \rho)$ is well-defined.*

Proof. All we need to check is that $\sum_i \alpha_i = 1$ and that $\alpha_i \geq 0$ for all i. The latter follows immediately by the definition of α_i noting that $0 < \rho \leq 1$. The former can be proved through basic algebra. □

Theorem 5.14. *Let $\mathcal{S}(k, \boldsymbol{N}) = \langle \Pi_1, \ldots, \Pi_k \rangle$ be an optimally-fair scheme with failure chance ρ. Fix an N_0, then the scheme $\mathcal{S}(k+1, \langle N_0 \rangle^\frown \boldsymbol{N}) = \langle \Pi_g(N_0, \rho), \Pi_1, \ldots, \Pi_k \rangle$ is optimally fair and unique.*

Proof. Since we know that $\Pi_g(N_0, \rho)$ is well-defined, we only need to prove that no other protocol Π_b would lead to a scheme of better or same fairness. Assume for a contradiction that such Π_b with matrix M_b exists. By Theorem 5.12, we know that we can represent M_b with $N_0 - 1$ entries $\beta_1, \ldots, \beta_{N_0-1}$. Therefore, using Equation 5.6, $\Pr[\mathcal{E} \text{ succeeds in } \Pi_b] \leq \Pr[\mathcal{E} \text{ succeeds in } \Pi_g(N_0, \rho)]$ implies

$$\max_l \left(\beta_l + \rho \sum_{i>l} \beta_i \right) \leq \max_l \left(\alpha_l + \rho \sum_{i>l} \alpha_i \right)$$

Note that $\alpha_l + \rho \sum_{i>l} \alpha_i = (1-\rho)\alpha_{l+1} + \rho \sum_{i>l} \alpha_i = \alpha_{l+1} + \rho \sum_{i>l+1} \alpha_i$. Hence, for any L, $\max_l \left(\alpha_l + \rho \sum_{i>l} \alpha_i \right) = \alpha_L + \rho \sum_{i>L} \alpha_i$. Moreover, we can rewrite $\sum_{i>L} \gamma_i = 1 - \sum_{i \leq L} \gamma_i$ for any distribution γ_i. This leads to $\alpha_L + \rho \sum_{i>L} \alpha_i = \rho + (1-\rho)\alpha_L - \rho \sum_{i<L} \alpha_i$. Thus, for any K, L

$$\rho + (1-\rho)\beta_K - \rho \sum_{i<K} \beta_i \leq \rho + (1-\rho)\alpha_L - \rho \sum_{i<L} \alpha_i \tag{5.7}$$

We prove by induction that for all i $\beta_i \leq \alpha_i$. Since $\sum_i \beta_i = \sum_i \alpha_i = 1$, this leads to $\beta_i = \alpha_i$ which is a contradiction. First, use Equation 5.7 with $K = L = 1$ to obtain $\beta_1 \leq \alpha_1$. Fix H and assume that for all $i < H$, $\beta_i \leq \alpha_i$. Use Equation 5.7 with $K = L = H$ to obtain $(1-\rho)(\beta_H - \alpha_H) \leq \rho \sum_{i<H} (\beta_i - \alpha_i) \leq 0$. Hence $\beta_H \leq \alpha_H$, and this concludes our proof. □

The above theorem gives us a way to construct optimally-fair schemes given the number of maximal attempts and the number of rounds per attempt. However, this leaves unanswered the question of how to pick those values of k and $\boldsymbol{N}$. Rather unsurprisingly, we can easily prove that allowing more attempts increases the overall chance of failure.

Theorem 5.15. *The failure chance of a scheme $\mathcal{S}(k, \boldsymbol{N})$ is always greater than the failure chance of the optimally-fair $\mathcal{S}(k', \boldsymbol{N'})$ where $k' < k$ even if $\sum_i \boldsymbol{N'}[i] = \sum_i \boldsymbol{N}[i]$.*

Proof. Using only one attempt, the probability of unfairness is $\frac{1}{2N}$. Now assume $n_1 + n_2 = N$, then doing two rounds with respectively, n_1 and n_2 rounds results in unfairness of $\frac{1-\rho}{2}\alpha_1 + \rho$ where ρ is $\frac{1}{2n_2}$ and $\alpha_1 = \frac{2\rho(1-\rho)^{N-1}}{(1+\rho)^N-(1-\rho)^N}$. It follows that the above is strictly greater than $\frac{1}{2n_2}$ which is greater than $\frac{1}{2N}$. So doing one round of n_2 is better than doing two rounds of n_1 and n_2. □

The above confirms the intuition that multiple attempts, while increasing the probability of a successful exchange, they also increases the chance of unfairness.

5.5.3 On Deciding The Length Of Each Attempt

Recall that the only purpose of allowing multiple attempts when exchanging secrets was to increase the chance of one of these exchanges to succeed. We have also proved that that multiple attempts increase the chances of malicious adversaries succeeding. Therefore, when picking the best scheme, one should consider both the probability of a malicious adversary succeeding and the probability of the exchange to succeed. The problem as defined here is not trivial as we believe there are multiple sensible ways of tackling it, which lead to different solutions. More specifically, we can characterise the setting of Alice and Bob exchanging secrets using four different parameters:

1. q: chance of non-delivery of message due to network failure.
2. τ: maximal probability of unfairness accepted by Alice and Bob.
3. ξ: minimal probability of successful exchange between honest parties.
4. η: maximal number of messages the parties are willing to exchange.

The above parametrisation is not unique as one could decide to evaluate the expected number of messages the party exchange rather than η. Moreover, the definition of ξ does not take into account malicious adversaries. In particular, note that a malicious adversary can always decide to abort an exchange protocol before

sending any message, meaning that we cannot place any lower-bound of the probability of successful exchange. Within the given characterisation, three different approaches can be considered:

1. Minimise τ subject to the constraints given by ξ and η.
2. Maximise ξ subject to the constraints given by τ and η.
3. Minimise η subject to the constraints given by ξ and τ.

Finding the solution to any of the above is independent of the study done in the previous section on optimally-fair schemes since it becomes a question of picking the correct distributions in each protocol as well as finding the best values for k and $\boldsymbol{N}$. While we have some preliminary results that proves the use of optimally-fair schemes in both maximising ξ and minimising η, the full description of these fall outside the scope of this thesis.

5.6 Using Oblivious Transfer

In the protocol described so far, the use of commutative blinding is a feature that sets it apart from most other 2-party computation protocols. In particular, we recall that any functionality can be computed securely with abort without relying on commutative blinding (refer to Chapter 2). Thus, the setup phase of our exchange protocol could be computed without the use of the commutative blinding. However, the standard techniques that rely on garbled circuit and oblivious transfer [163, 75] are vastly more inefficient. Specifically, one party must construct a *deterministic* circuit to compute the setup functionality and then engage in an oblivious transfer protocol where the other party selects random strings representing their inputs.

To completely hide a 256-bit key, at least 2^{256} inputs would be required in the aforementioned OT protocol. This can be optimised to running 256 OT protocols with binary inputs, but note that our setup functionality requires N such strings per party. Consequently, the communication complexity of this simple aspect (which

excludes the shuffling) would already necessitate $256N$ independent parallel OT protocols.

As we will discuss in this section, even with the assumption of a common reference string, these OT protocols would require exchanging approximately $768N$ individual "items" (i.e., in the protocol by Peikert [129], each item is a vector for the underlying RLEW problem), whereas our complete setup phase requires only $4N + 2$ similar items (i.e., ciphertexts from the blinding scheme). Thus, our setup phase is significantly more efficient, even without considering the construction and evaluation of the garbled circuit. Therefore, the existence of an efficient oblivious transfer-based protocol is an interesting research question.

Couteau *et al.* [48] have already shown that a similar fair exchange protocol can be constructed, yet they fail to achieve optimal partial fairness. Moreover, their use of OT can be seen as performing 2 independent shuffles. To achieve optimality, we require the shuffles to depend on each other as we wish for the final positions of the secrets to be linked. In this section, we outline a method to maintain both optimality and efficiency by relying on a slightly stronger definition of oblivious transfer.

This research is still ongoing, so we do not provide any security proofs at this stage. Instead, we offer intuitions on why the protocol should meet the required guarantees.

5.6.1 A New Oblivious Transfer

As already described in Chapter 2, an OT between a sender and a receiver is a protocol where the sender inputs k items and the receiver will receive only the one whose index they picked. Moreover, the sender will not learn which item the receiver picked, and the receiver will not learn anything about the other $k - 1$ items. In this section, we define OT more strictly as a cryptographic primitive. This definition, which was already used in [48], will rule out some OT constructions, but

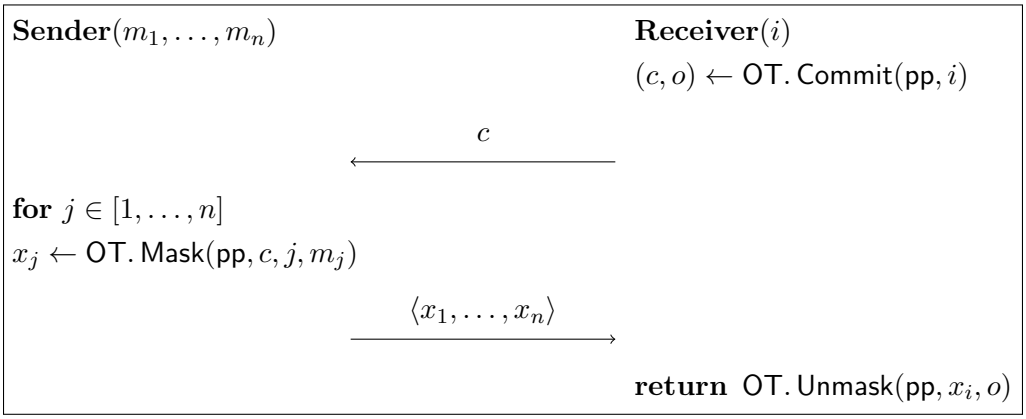

Figure 5.2: Oblivious Transfer Protocol.

will give us a way to design an efficient protocol. In particular, we define a 2-round 1-out-of-n OT as a tuple $(\mathsf{OT.Setup}, \mathsf{OT.Commit}, \mathsf{OT.Mask}, \mathsf{OT.Unmask})$ where

- $\mathsf{OT.Setup}(1^\lambda) \to \mathsf{pp}$ generates some public parameters.
- $\mathsf{OT.Commit}(\mathsf{pp}, i) \to (c, o)$ is run by the receiver with its input $i \in [1, \ldots, n]$ to generate a commitment c to i which is used by the sender and an "opening information" o which the receiver will use to retrieve its message.
- $\mathsf{OT.Mask}(\mathsf{pp}, c, i, m) \to d$ is used by the sender to combine the commitment c with a message m at index i.
- $\mathsf{OT.Unmask}(\mathsf{pp}, d, o) \to m/\perp$ is run by the receiver to retrieve the message m if it were masked with the same index used for the commitment c.

In particular, the OT is *correct* if for all $i \in [1, \ldots, n]$

$$\Pr\left[m \leftarrow \mathsf{OT.Unmask}(\mathsf{pp}, d, o) \;\middle|\; \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{OT.Setup}(1^\lambda) \\ (c, o) \leftarrow \mathsf{OT.Commit}(\mathsf{pp}, i) \\ d \leftarrow \mathsf{OT.Mask}(\mathsf{pp}, c, i, m) \end{array} \right] = 1 \quad (5.8)$$

Using the above OT primitive, one can easily construct an OT protocol as seen in Figure 5.2. We omit the use of the $\mathsf{OT.Setup}$, as different implementation will require different uses. Indeed, this might need to be run in a trusted environment [129], or perhaps it can be run once for many instances. Regarding security, we do not delve deeply as we only provide a sketch proof of our protocol. However,

1-out-of-n f-double oblivious transfer

Sender: $\mathcal{S}(x_1, \ldots, x_n, a)$
Receiver: $\mathcal{R}(b)$

1. $\mathcal{S}$ sends its inputs $(x_1, \ldots, x_n), a$ to the functionality $\mathcal{F}$.
2. $\mathcal{R}$ sends its input b to $\mathcal{F}$.
3. $\mathcal{F}$ sends $x_{f(a,b)}$ to $\mathcal{R}$.

Figure 5.3: Ideal functionality for the double oblivious transfer.

it is evident from Figure 5.2 and our informal discussion that OT.Commit must completely hide i from the sender, and that OT.Mask perfectly hides all messages but the one corresponding to the committed index. More precise formulations change depending on the use of OT. In particular, one often requires the OT to be simulatable in the ideal-real world paradigm where we defined partial fairness [48] or even require the protocol to be universally composable [129].

We can construct a new type of OT, that we name double oblivious transfer, where both parties have a say on which message will be transferred to the receiver. The idea is to allow the sender to influence which message the receiver will obtain. Figure 5.3 shows this double OT as an ideal functionality. In this new protocol, the sender has an extra input a which is combined with the receiver's input b to generate the index $f(a, b)$ of the message to transfer. In our description, we let $f : \mathcal{D}^2 \to [1, \ldots, n]$[4] be a parameter of the protocol, yet one could design the functionality to have f as one the receiver's inputs. The transition from Figure 5.3 to our setup phase is immediate: run two instances of this OT in parallel with the appropriate functions that would result in the secrets to have indices close to each other. Section 5.6.2 is dedicated to explaining more accurately how to the new setup phase works.

The idea behind our implementation of a double oblivious transfer is to nest two OT inside each other. More precisely, we use an OT protocol so that the sender can

[4]It is important that the domain of the function f is polynomial in n, so that our construction using nested OTs works. However, we can imagine a world where this function has much larger domains.

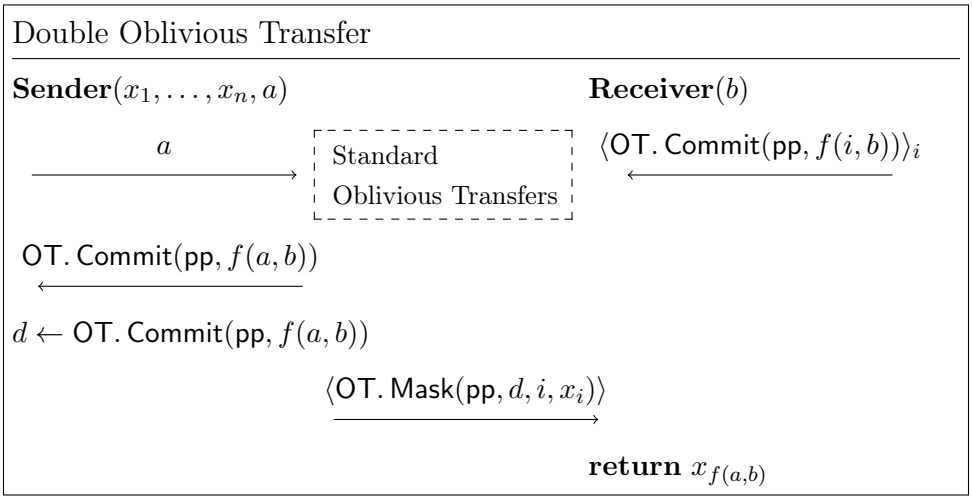

Figure 5.4: Implementing the new OT using "standard" OTs.

obtain $\mathsf{OT.Commit}(\mathsf{pp}, f(a,b))$ without revealing a. Once, the sender holds that value, they can proceed to transfer over the $f(a,b)^{\text{th}}$ message they hold, without learning $f(a,b)$. Figure 5.4 shows this high-level idea of nesting OTs.

However, as described there, the protocol would not be secure since the sender obtains the opening information used by the receiver to get $x_{f(a,b)}$. Knowing this information means that the sender could try to unmask the masked messages to find out the value of $f(a,b)$. As a result, we ask the receiver to only transfer the committing part of $\mathsf{OT.Commit}(\mathsf{pp}, f(\_,b))$. This leads to another issue: for the receiver to retrieve the message they would have to try to unmask each message using all the opening information. While this can be done in polynomial time, it would lead the receiver to obtain a. Hence, we add a new constraint on the OT protocol we use: we require the opening information to be independent of the committed index. Figure 5.5 depicts our proposed construction with all details, where $\mathcal{D}$ represent the domain from where a is sampled. An example of a OT protocol that can be used to instantiate our new primitive is the construction by Peikert *et al.* [129, 130] or Döttling *et al.* [52]. Here, we just give a sketch on how the former technique could be used. The scheme we analyse is based on the LWE problem. While alternatives based on the Computational DH problem exist, relying on the LWE gives our construction more robustness against quantum

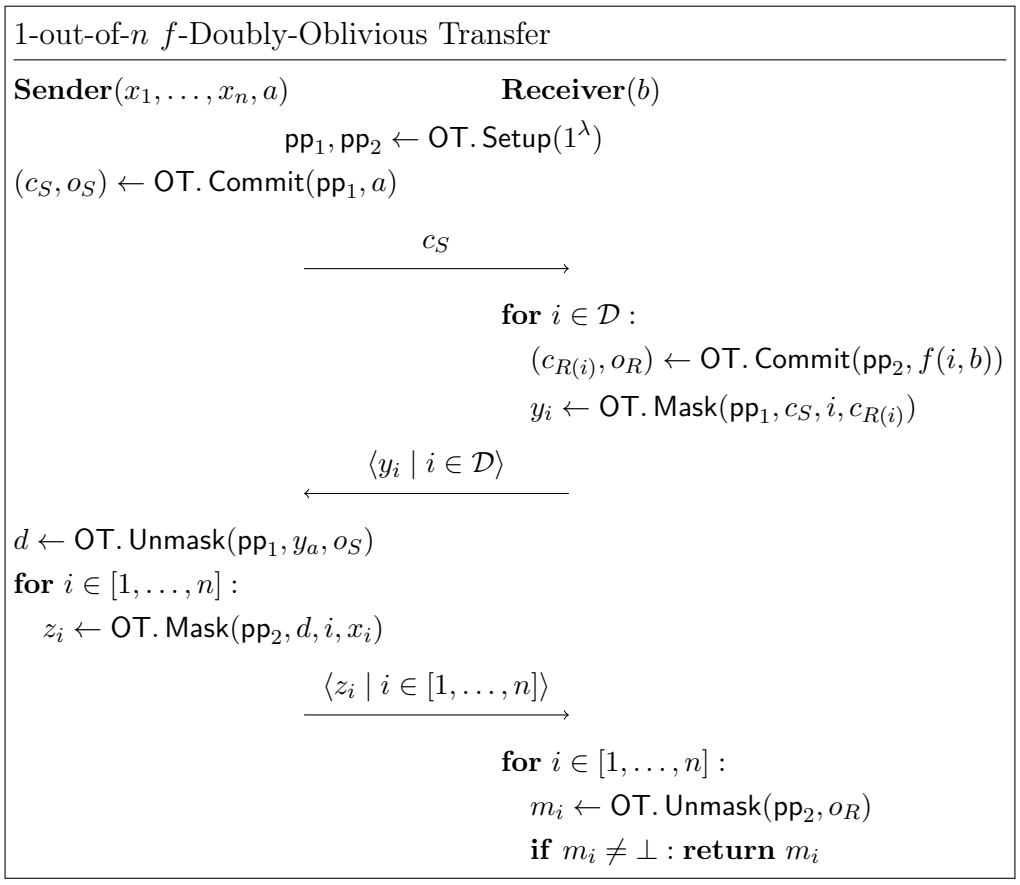

Figure 5.5: Double Oblivious Transfer Construction.

computation. The $\mathsf{OT}.\mathsf{Setup}$ will generate a matrix A and n vectors $v_1, \ldots, v_n$. The receiver commits to an index b by constructing keys for a LWE-based encryption: $\mathsf{pk} = s^T A + e - v_b$ and $\mathsf{sk} = s$, where s and e are random. Crucially, note that the opening information is sk which does not depend on the committed index. The sender can mask a message m of index i by simply encrypting it using A and $\mathsf{pk}+v_i$. That is, they compute $(c_1, c_2) = (Ae, (\mathsf{pk} + v_i)e + m)$ for some random e. This can be opened by decrypting using the secret key, i.e. $m \leftarrow \lceil c_2 - \mathsf{sk}^T c_1 \rfloor$. Therefore, the receiver only needs to decrypt n messages since the opening information does not depend on a. It is also crucial that both the schemes cited earlier [130, 52] are universally composable, meaning that they maintain their security properties even when nested together. Therefore, the sender does not learn $f(a, b)$ and the receiver does not learn any of the $n - 1$ values they are not meant to receive.

5.6.2 New shuffling phase

Given this new oblivious transfer, the exchange protocol can be instantiated without using the commutative blinding. In particular, Alice and Bob will run in parallel the protocol of Figure 5.5 but withhold the last message. Then, they will proceed to exchange the sequence constituting this message one item at a time. This last step is the new exchange phase. The full protocol is given in Figure 5.6, where we use the following functions:

$$f_A : \mathbb{Z}_{2N} \times \mathbb{Z}_{2N} \to \mathbb{Z}_{N+1} \qquad f_B : \mathbb{Z}_{2N} \times \mathbb{Z}_{2N} \to \mathbb{Z}_N$$
$$f_A(a, b) = \left\lfloor \frac{a + b + 1}{2} \right\rfloor \bmod N \qquad f_B(a, b) = \left\lfloor \frac{a + b}{2} \right\rfloor \bmod N$$

First note that the protocol depicted is nothing more than two parallel runs of the double OT using functions f_A, f_B, and by sending the last message during the exchange phase. The function f_A (respectively f_B) indicates the position of Alice's (respectively Bob's) secret. To see how these functions relate to each other, note that when the protocol uses $2N + 1$ exchange messages, there are exactly $2N$ possible (joint) locations for the secrets. Therefore, the function $f(a, b) = a + b \bmod 2N$ uniformly picks one of them (where indices start at 0). We order these

Fair Exchange using OTs

Alice	**Bob**
$k_A \xleftarrow{\$} \mathcal{K}$	$k_B \xleftarrow{\$} \mathcal{K}$
$r_A \xleftarrow{\$} \mathbb{Z}_{2N}$	$r_B \xleftarrow{\$} \mathbb{Z}_{2N}$
$D_A \leftarrow \mathsf{Delay}(\mathsf{Enc}_{k_A}(s_A), r_A)$	$D_B \leftarrow \mathsf{Delay}(\mathsf{Enc}_{k_B}(s_B), r_B)$
$(c_A, o_A) \leftarrow \mathsf{OT.Commit}(\mathsf{pp}_1, r_A)$	$(c_B, o_B) \leftarrow \mathsf{OT.Commit}(\mathsf{pp}_3, r_B)$
$\xrightarrow{c_A, D_A}$	
$\xleftarrow{c_B, D_B}$	
for $i \in \mathbb{Z}_{2N}$:	
$(c'_{A(i)}, o'_A) \leftarrow \mathsf{OT.Commit}(\mathsf{pp}_2, f_B(r_A, i))$	
${y_i}^{(A)} \leftarrow \mathsf{OT.Mask}(\mathsf{pp}_1, c_B, i, c'_{A(i)})$	
	for $i \in \mathbb{Z}_{2N}$:
	$(c'_{B(i)}, o'_B) \leftarrow \mathsf{OT.Commit}(\mathsf{pp}_4, f_A(i, r_B))$
	${y_i}^{(B)} \leftarrow \mathsf{OT.Mask}(\mathsf{pp}_3, c_A, i, c'_{B(i)})$
$\xleftarrow{\langle {y_i}^{(B)} \rangle_{i \in \mathbb{Z}_{2N}}}$	
$\xrightarrow{\langle {y_i}^{(A)} \rangle_{i \in \mathbb{Z}_{2N}}}$	
$d_A \leftarrow \mathsf{OT.Unmask}(\mathsf{pp}_1, {y_{r_A}}^{(B)}, o_A)$	$d_B \leftarrow \mathsf{OT.Unmask}(\mathsf{pp}_3, {y_{r_B}}^{(A)}, o_B)$
for $i \in \mathbb{Z}_{N+1}$:	**for** $i \in \mathbb{Z}_N$:
${m_i}^{(A)} \leftarrow \mathsf{OT.Mask}(\mathsf{pp}_2, d_A, i, k_A)$	${m_i}^{(B)} \leftarrow \mathsf{OT.Mask}(\mathsf{pp}_4, d_B, i, k_B)$
..... Exchange phase	
$\xrightarrow{{m_i}^{(A)}}$	
$\xleftarrow{{m_i}^{(B)}}$	

Figure 5.6: New fair exchange protocol using OT.

$2N$ options by stating that the i^{th} location is the one where the first secret is in the i^{th} message where $0 \leq i < 2N + 1$. Therefore, Bob's secret would be in the $\lfloor \frac{i}{2} \rfloor^{\text{th}}$ message he sends, and Alice's one precedes Bob's if and only if i is even. Thus, $f_A(a, b)$ is simply the locations of Alice's secret among her own messages when the joint secret location is $f(a, b)$. Similarly, $f_B(a, b)$ is the position of Bob's secret when the secrets' location picked is $f(a, b)$. Due to the properties of the OTs used, we know that both $f_A(r_A, r_B)$ and $f_B(r_A, r_B)$ are kept secret during the setup phase. Moreover, the secrets are also kept private as they are only exchange encapsulated in the TLP, and the keys required to decrypt the secrets (i.e. k_A, k_B) are completely independent of the messages sent during the setup phase. Therefore, this new setup phase is secure. However, the proof of Theorem 5.4 does not follow immediately from having a secure setup phase. In particular, we need to show that no PPT adversary can learn the position of the secrets during the exchange phase. In the context of Figure 5.6, this means that neither Alice nor Bob should be able to deduce $f_B(r_A, r_B)$ (respectively $f_A(r_A, r_B)$). Therefore, we set another requirement on the OTs used in the protocol: the "unmasking" of "masked" items with the wrong opening information should return random elements. In particular, $\{\mathsf{OT.Unmask}(\mathsf{pp}, \mathsf{OT.Mask}(\mathsf{pp}, c, j, m), o) \mid (c, o) \leftarrow \mathsf{OT.Commit}(\mathsf{pp}, i), i \neq j\}$ should be indistinguishable from a random element from the same domain as m. Under this condition, one would not be able to distinguish a wrongly unmasked item from a correct one, provided this latter is also a random item. In Figure 5.6, the message Alice and Bob are trying to exchange is indeed a random bit-string. Therefore, they would not be able to tell which of the exchange messages is hiding the encryption key until the delayed messages open. Finally, we remark that the OT instantiations mentioned earlier in this section satisfy this extra condition. Looking at the previous LWE-based example, a "wrongly" decrypted item would return a random vector which is then mapped to a random bit-string.

5.7 Conclusions and Future Research

In this chapter, we designed a 2-party exchange protocol that achieves optimal partial fairness, meaning that no shorter protocol can have a higher probability of a fair exchange. This result is achieved through the careful use of Time-Lock Puzzles (TLPs). Specifically, we demonstrated how using only one TLP per party results in this optimality. This drastically improves upon previous protocols [78] in terms of efficiency and circumvents their impossibility result. Moreover, we proved that our proposed protocol is secure against covert adversaries without the need for any zero-knowledge proofs or other potentially costly cryptographic primitives.

Later in the chapter, we briefly investigated the scenario where an exchange fails, and both parties are willing to attempt it again. We showed how repeating our optimal protocol in this scenario results in a sub-optimal interaction. While a full classification and deeper analysis are left for future research, our initial investigation revealed a method to establish the optimal interaction given certain parameters.

Finally, we sketched a modified version of our 2-party protocol where commutative blinding is replaced with oblivious transfer. This reinforced our argument that TLPs are the key to our ability to surpass previous results. Furthermore, this newly designed protocol offers the advantage of allowing for more flexibility in the distribution of secrets, potentially facilitating easier implementations of our abstract study on multiple attempts.

6

Multi-Party Fair Exchange

Contents

In this chapter, we extend our investigation of how TRE enhances fair exchange protocols to the multi-party setting. Building upon the 2-party protocol with optimal partial fairness developed in the previous chapter, we present a new protocol designed for multiple participants. However, this extension is not a straightforward generalisation; achieving optimality in the multi-party context requires careful consideration of the number of participants controlled by the adversary and the corresponding probability distribution arising from the location of the secrets. Nevertheless, we provide a versatile protocol adaptable to various adversarial scenarios and offer an in-depth analysis of the abstract model underlying partially fair exchange protocols.

The chapter opens with a brief introduction to the multi-party settings and the notation we use. Following this, we present our protocol in its more general form,

which depends on a user-selected parameter. Section 6.3 demonstrates the partial fairness of our construction, while the subsequent section illustrates its optimality. Specifically, we provide an upper bound on the fairness for any protocol with a fixed number of exchange rounds and then show how our protocol achieves this bound.

6.1 Introduction

Consider a scenario where K parties, $\mathbf{P_1}, \ldots, \mathbf{P_K}$, wish to exchange items but lack mutual trust and cannot agree on a neutral intermediary. If a majority colludes, no method can guarantee fairness, leaving some honest parties vulnerable to being cheated. This impossibility of guaranteeing fairness is well-documented in the literature for both the two-party case [128, 46] and the general multi-party setting [70].

A critical consequence of this impossibility is the inability to guarantee fairness in multi-party computation (MPC) protocols. For instance, if these K parties want to compute statistics on sensitive medical data, general techniques may ensure correct computation but cannot guarantee that all honest parties will receive the output.

This thesis focuses on the concept of partial fairness as a compromise. Introduced by Gordon and Katz [78], partial fairness relaxes the strict fairness requirement by allowing a small but non-negligible probability of failure. Beimel *et al.* [18] extended this concept to the multi-party setting, where an ε-partially fair protocol has a probability ε of being unfair. Chapter 2 provides a formal definition and further background information.

In this chapter, we propose a protocol achieving roughly $\frac{L}{N+1-K}$-partial fairness against coalitions of size L using only N exchange messages. Furthermore, we conduct a thorough analysis of partially-fair exchange protocols, ultimately proving the optimality of our construction.

6.1.1 Preliminaries

The exchange protocol we construct utilises a few cryptographic primitives similar to those used in the protocol from Chapter 5.

We leverage a symmetric encryption scheme $(\mathsf{Enc}, \mathsf{Dec})$, which is required to be non-committing [49]. Recall that this means that for any message-message-key triple (m_1, m_2, k_1), there exists another key k_2 such that:

$$\mathsf{Dec}_{k_2}(\mathsf{Enc}_{k_1}(m_1)) = m_2$$

For simplicity, the reader can assume the use of the one-time pad.

Additionally, we employ a timed-release encryption scheme with encryption/decryption functions $\mathsf{Delay}/\mathsf{Open}$. Chapter 2 provides a detailed overview of this scheme, and Chapter 3 details its construction. Although the protocol could function with a scheme implemented using trusted parties, we assume this is not the case to avoid introducing any trust assumptions.

As in the 2-party protocol of Chapter 5, a key feature of our protocol is the use of commutative blinding (see Chapter 4) to efficiently shuffle secrets. We require a multi-party version of the definition given in Chapter 4. For ease of presentation, we present a stronger definition below to simplify notation and avoid excessive subscripts.

Definition 6.1. A pair of PPT algorithms $(\mathsf{KGen}, \mathsf{Blind}, \mathsf{Unblind})$ is a *commutative blinding scheme* if:

$$\forall m \in \mathcal{M}\ \forall k \xleftarrow{\$} \mathsf{KGen}(1^\lambda)\ \ \mathsf{Unblind}(\mathsf{Blind}(m, k), k) = m$$

Moreover, the commutativity property is given by:

$$\forall m \in \mathcal{M}\ \forall k_1, k_2 \xleftarrow{\$} \left\{\mathsf{KGen}(1^\lambda)\right\}^2$$
$$\mathsf{Blind}(\mathsf{Blind}(m, k_1), k_2) = \mathsf{Blind}(\mathsf{Blind}(m, k_2), k_1)$$

For completeness, we state the weaker commutativity requirement needed to run the protocol with K parties. We assume the existence of $K+1$ pairs $(\mathsf{Blind}_0, \mathsf{Unblind}_0), \ldots, (\mathsf{Blind}_K, \mathsf{Unblind}_K)$, such that:

$$\mathsf{Unblind}_0(\mathsf{Blind}_K(\ldots \mathsf{Blind}_0(m, k_0) \ldots, k_K), k_0) = \mathsf{Blind}_K(\ldots \mathsf{Blind}_1(m, k_1), \ldots, k_K)$$

Using the above notation for the protocol and security requirements would result in a less readable presentation. Therefore, we utilise the stronger commutativity property to simplify the exposition. Regarding security, we require the blinding to satisfy the standard IND-CPA definition after each application. Please refer to Chapter 4 for a more detailed discussion of this primitive.

6.2 The Protocol

We consider a scenario with K parties, $\mathbf{P_1}, \ldots, \mathbf{P_K}$, and define a matrix Σ to represent an arbitrary exchange. Specifically, $\Sigma_{i,j}$ denotes the secret that $\mathbf{P_i}$ intends to send to $\mathbf{P_j}$. While this model places no restrictions on the type of exchange, we assume a complete point-to-point network, meaning any party can communicate directly with any other party.

Intuitively, we define a fair exchange as one where either all parties receive all their expected items or no one receives anything. Even if the exchange Σ comprises multiple sub-exchanges, our fairness requirement mandates that the entire exchange Σ must be fair. Thus, the completion of one sub-exchange is contingent upon the completion of all other sub-exchanges. While this constraint might seem restrictive in the abstract model, it often reflects real-world scenarios, such as buying a new house and selling an old one, where the two transactions are typically interdependent. For a more in-depth discussion of the fair exchange problem and its formal definition, please refer to Chapter 2.

In the following section, we demonstrate that this complex exchange can be reduced to a simpler one. In this simplified model, each of the K parties possesses a secret s_i,

and the objective is for all parties to learn all secrets. We assume that knowledge of all but one secret is of no value. This assumption is justified by considering these secrets as shares of a larger secret (as is frequently the case in MPCs). In this simpler model, fairness is achieved if and only if, whenever a party discloses their secret, they receive all the other secrets in return.

6.2.1 Protocol Overview

We begin by showing a reduction from the general fair exchange problem to a simpler context that we will focus on for the remainder of this thesis. Recall that we have K parties, $\{\mathbf{P_1}, \ldots, \mathbf{P_K}\}$, and a matrix Σ, where $\Sigma_{i,j}$ represents the item that $\mathbf{P_i}$ intends to send to $\mathbf{P_j}$.

To simplify this context, we assume either the existence of private channels for confidential communication or a public key infrastructure (PKI) where parties agree on each other's public keys.[1]

With private channels, $\mathbf{P_i}$ can split $\Sigma_{i,j}$ into two shares, sending one privately to $\mathbf{P_j}$. The remaining shares held by $\mathbf{P_i}$ constitute their secret s_i. In the PKI scenario, $\mathbf{P_i}$'s secret is defined as $s_i = \langle \mathsf{Enc}_{\mathsf{pk}_j}(\Sigma_{i,j}) \mid j \neq i \rangle$, where pk_j is $\mathbf{P_j}$'s public key. In either case, knowing s_i only reveals $\Sigma_{i,j}$ to $\mathbf{P_j}$. Thus, we can frame the exchange as one where all parties share their secrets to obtain the exchange specified by Σ.

In line with other partially-fair protocols, our protocol conceals the moment secrets are exchanged from all parties. It consists of two phases: a setup phase for shuffling the secrets and an exchange phase for exchanging secrets and dummy values.

Parties agree on the number of exchange messages N and the malicious coalition size L for which the protocol is optimised. Typically, $L = K - 1$, but contexts where external trust assumptions are present might wish to lower this value. Figure 6.1 illustrates the protocol, using S_n for set of permutations over $\{1, \ldots, n\}$, R_n for

[1] This is not unrealistic since security against active adversaries necessitates a broadcast channel (see Chapter 2), usually implemented in one of these ways.

rotations, and for any vector of permutations $\boldsymbol{\sigma}$, we write $\boldsymbol{\sigma}(M)$ for applying the permutation $\boldsymbol{\sigma}[i]$ to the i^{th} row of M.

As detailed in Section 6.4, the secrets might not be uniformly distributed, but rather appear in one of a few "configurations" or "states" chosen uniformly. Given K, N, and L, we can compute the δ possible configurations. Let $\langle \pi_1, \ldots, \pi_\delta \rangle$ be the permutations mapping an initial state where all secrets are sent first to these δ valid configurations. The shuffling phase involves selecting a random permutation.[2] For simplicity, we assume $N = nK$, meaning each party sends n exchange messages.

Crucially, parties will encrypt their secret and exchange encryption keys instead of secrets directly, as in [18, 78]. Using TLPs to delay the ciphertexts after the protocol termination renders auxiliary information useless. Parties can only test exchanged keys when the delayed message opens *after* the protocol ends. This circumvents the impossibility result of by Gordon and Katz [78] and improves efficiency compared to Beimel *et al.* [18].

In the exchange phase, each party sends n keys, only one of which decrypts their secret. Therefore, they create a list of random keys, encrypting their secret with the first. These lists are combined into a matrix M by $\mathbf{P_1}$[3], where the x^{th} row is $\mathbf{P_x}$'s list after blinding each element. We can use this matrix to describe the exchange phase: $M_{i,j}$ is the item $\mathbf{P_i}$ sends during round j. At the moment, the first column of M contains the keys used to encrypt the secrets, so $\mathbf{P_1}$ can compute δ functions π_i mapping M to all the valid configurations. To represent the configurations, $\mathbf{P_1}$ creates pairs $\langle (M, \pi_1), \ldots, (M, \pi_\delta) \rangle$. The aim of the protocol now becomes to pick one of these pairs uniformly.

To achieve this, each party will blind the list, rotate it and pass it to the next party. To mask the list, each (M, f) is transformed into $(g(\mathsf{Blind}(M, b)), f \circ g^{-1})$ for random g and blinding key b. We note that the transformed pair (M', f') satisfies the property that $f'(M') = f(M)$, ignoring blinding. Figure 6.1 represents this

[2] The permutations and δ are described in Section 6.4.
[3] We stress that $\mathbf{P_1}$ is not a trusted entity.

Multi-party Fair Exchange

Define $F(M, f, b, g) = (g(\mathsf{Blind}(M, b)), f \circ g^{-1})$

$\mathbf{P_i}$

$\boldsymbol{k_i} \xleftarrow{\$} \mathcal{K}^n;\ \boldsymbol{b_i} \xleftarrow{\$} \left\{\mathsf{KGen}(1^\lambda)\right\}^2;\ \boldsymbol{\sigma_i} \xleftarrow{\$} (S_n^K)^\delta; \tau_i \xleftarrow{\$} R_\delta$

$\boldsymbol{L_i} \leftarrow \langle \mathsf{Blind}(\boldsymbol{k_i}[i], \boldsymbol{b_i}[1]) \mid 1 \le i \le n \rangle$

$D_i \leftarrow \mathsf{Delay}(\mathsf{Enc}_{\boldsymbol{k_i}[1]}(s_i) \| \tau_i \| \boldsymbol{b_i}[2])$

$\boldsymbol{L_i}, D_i \longrightarrow \mathbf{P_1}$

$\mathbf{P_1}$

$\boldsymbol{E} \leftarrow \langle \boldsymbol{L_1}, \ldots, \boldsymbol{L_K} \rangle$

$\boldsymbol{S^1} \leftarrow \tau_1(\langle F(\boldsymbol{E}, \pi_i, \boldsymbol{b_1}[2], \boldsymbol{\sigma_1}[i]) \mid 1 \le i \le \delta \rangle)$

$\boldsymbol{S^1} \longrightarrow \mathbf{P_2}$

$\mathbf{P_j}$ after receiving $\boldsymbol{S^{j-1}}$

let $\boldsymbol{S^{j-1}} = \langle (M_1, o_1), \ldots, (M_\delta, o_\delta) \rangle$

$\boldsymbol{S^j} \leftarrow \tau_j(\langle F(M_i, o_i, \boldsymbol{b_j}[2], \boldsymbol{\sigma_j}[i]) \mid 1 \le i \le \delta \rangle)$

$\boldsymbol{S^j} \longrightarrow \mathbf{P_{j+1}}$

$\mathbf{P_K}$ after receiving $\boldsymbol{S^{K-1}}$

let $\boldsymbol{S^{K-1}} = \langle (M_1, o_1), \ldots, (M_\delta, o_\delta) \rangle$

$\boldsymbol{S^K} \leftarrow \tau_j(\langle F(M_i, o_i, \boldsymbol{b_K}[2], \boldsymbol{\sigma_K}[i]) \mid 1 \le i \le \delta \rangle)$

$\boldsymbol{E^K} \leftarrow o_f(M_f)$ where $(M_f, o_f) \leftarrow \boldsymbol{S^K}[1]$

$\boldsymbol{E^K}[i] \longrightarrow \mathbf{P_i}$

$\mathbf{P_i}$ after receiving $\boldsymbol{E^K}[i]$

$\boldsymbol{M^i} \leftarrow \mathsf{Unblind}(\boldsymbol{E^K}[i], \boldsymbol{b_i}[1])$

Figure 6.1: Protocol setup phase.

masking as F, with random permutations g as $\boldsymbol{\sigma_j}[i]$. The messages $\boldsymbol{S^i}$ contain the list after being shuffled and blinded i times.

$\mathbf{P_K}$ performs a final rotation and blinding, and selects the first element (M, f). Computing $f(M)$ yields a random valid configuration $\boldsymbol{E^K}$, representing the exchange phase that is about to commence. All elements are blinded by all parties, with each row blinded twice by the corresponding party.[4] $\mathbf{P_K}$ sends each party their row of $\boldsymbol{E^K}$, and they unblind once with keys $\boldsymbol{b_i}[1]$ before exchanging items.

In order to retrieve the secrets, one needs to identify the relevant messages and remove all the blinding. Therefore, delayed messages contain each party's rotation and blinding keys $\boldsymbol{b_i}[2]$. The setup messages are blinded with never-revealed keys $\boldsymbol{b_i}[1]$ to prevent any information leakage before the exchange phase. After the setup, parties exchange the items of their list sequentially.

6.3 Security Analysis

Before proving the partial fairness of our protocol, we need to understand the setup phase. Ignoring blinding, the final shuffled matrix $\boldsymbol{E^K}$ satisfies the following equation:

$$\boldsymbol{E^K} = \pi_{(\tau_1^{-1} \circ \cdots \circ \tau_K^{-1})(1)}(\boldsymbol{E})$$

If at least one party is honest, there is at least one truly random τ_i, implying the uniform distribution of the chosen permutation π_i. If no information is leaked during the exchange phase[5], then the corrupted coalition remains unaware of when the $\boldsymbol{k_i}[1]$ keys are exchanged. Thus, their only attack strategy is to prematurely abort the exchange phase at a random point.

This strategy succeeds only if the coalition halts after receiving all honest parties' secret keys without revealing all their own. In Section 6.4, we prove that the permutations π_i can be chosen to minimise the success rate of this strategy.

[4] This is why we need $K + 1$ blinding functions in Section 6.1.1.
[5] Refer to Section 6.3.2 for proof.

Notably, the setup phase has a round complexity of $K + 2$ with $3K$ messages sent. The exchange phase takes n rounds with $N = nK$ messages. These numbers significantly improve upon the protocol by Beimel *et al.* [18].

6.3.1 Adversarial Model

We assume a complete network topology where communication channels are authenticated and provide confidentiality but need not be secure, i.e., they might not provide integrity, replay protection or out-of-order delivery. As noted earlier, a public key infrastructure (PKI) may be necessary to simplify the exchange topology and establish a broadcast channel.

Message delivery is not guaranteed, and parties use agreed-upon timeouts to determine protocol abortion universally. We focus on scenarios with at least one honest party and assume static corruptions (non-adaptive adversary), meaning malicious coalitions remain fixed throughout the protocol.

We consider a group of malicious parties as a coalition that coordinates and shares knowledge, akin to a group of entities controlled by a single adversary. Importantly, we allow a dishonest majority, but adversaries are passive, meaning they follow the protocol but may abort communication undetected at any time. We address active adversaries by relying on general compilers, such as the one described in Chapter 2.

6.3.2 Partial Fairness

For ease of presentation, we focus on the scenario where the adversary controls all but one honest party, $\mathbf{P}_{\mathcal{H}}$. We also assume that $1 \neq \mathcal{H} \neq K$.[6]

The main result of this section is Theorem 6.3. However, we first establish a few lemmas to gain a better understanding of the setup phase of our protocol.

Lemma 6.1. *No non-uniform PPT adversary $\mathcal{A}$ can obtain $s_{\mathcal{H}}$ unless $D_{\mathcal{H}}$ is opened and they have received the message containing $\boldsymbol{k}_{\boldsymbol{\mathcal{H}}}[1]$ during the exchange phase.*

[6] The other cases are simplifications of this scenario and are omitted for brevity.

Proof. This should be fairly intuitive. To obtain $s_{\mathcal{H}}$, one must first open $D_{\mathcal{H}}$ to retrieve $\mathsf{Enc}_{\boldsymbol{k}_{\mathcal{H}}[1]}(s_{\mathcal{H}})$. Assuming the security of the encryption scheme, retrieving the secret requires knowledge of $\boldsymbol{k}_{\mathcal{H}}[1]$.

This key is blinded with $\boldsymbol{b}_{\mathcal{H}}[1]$ in $\boldsymbol{L}_{\mathcal{H}}$ and by other parties in the lists $\boldsymbol{S}^{i}$. Since this blinding key remains present in all these lists, the key cannot be recovered by knowing the lists alone.

The blinding key $\boldsymbol{b}_{\mathcal{H}}[1]$ is removed when computing $\boldsymbol{M}^{\mathcal{H}}$. All other blinding keys are included in the delayed messages, so knowledge of $\boldsymbol{M}^{\mathcal{H}}$ would reveal $\boldsymbol{k}_{\mathcal{H}}[1]$ and, consequently, $s_{\mathcal{H}}$. However, since each item in $\boldsymbol{M}^{\mathcal{H}}$ is independent, only the one corresponding to the blinding of $\boldsymbol{k}_{\mathcal{H}}[1]$ is needed. □

Lemma 6.1 demonstrates that we achieve partial fairness by hiding $\boldsymbol{k}_{\mathcal{H}}[1]$. Thus, the remainder of the setup phase focuses on concealing this key. Lemma 6.2 proves that the setup phase effectively hides the key. Specifically, the adversary remains unaware of the key's location in the final matrix, even during the exchange phase, as long as the delayed message is unread. This requires the function F to be an effective masking tool, but we defer the proof of F's security to Lemma 6.4 after the main theorem to maintain the focus on the high-level ideas.

Lemma 6.2. *Let $\mathcal{A}$ be a non-uniform PPT adversary controlling all but one honest party $\mathcal{H}$. $\mathcal{A}$'s view during the protocol, while $D_{\mathcal{H}}$ is "closed", is independent of $\tau_{\mathcal{H}}$, $s_{\mathcal{H}}$, and $\boldsymbol{k}_{\mathcal{H}}[1]$.*

Proof. Consider Figure 6.2, which represents a function describing $\mathcal{A}$'s view during the execution of the exchange protocol. For brevity, only the messages involving the honest party are shown.

First, due to the assumed security of the time-release encryption, we disregard the content of the delayed message. Specifically, the view is indistinguishable from that of another protocol where the delayed message contains a random bit-string. This already demonstrates that $\mathcal{A}$'s view is independent of $s_{\mathcal{H}}$.

Max coalition view

$\mathsf{ColView}(\boldsymbol{k_i}, \boldsymbol{b_i}, \boldsymbol{\sigma_i}, \tau_i, \quad \textbf{for } i \neq \mathcal{H}) :=$

$\boldsymbol{k_\mathcal{H}}, \boldsymbol{b_\mathcal{H}}, \boldsymbol{\sigma_\mathcal{H}}, \tau_\mathcal{H} \xleftarrow{\$} \mathbf{P}_\mathcal{H}$'s input

$D_\mathcal{H} \leftarrow \mathsf{Delay}(\mathsf{Enc}_{\boldsymbol{k_\mathcal{H}}[1]}(s_\mathcal{H}) \| \tau_\mathcal{H} \| \boldsymbol{b_\mathcal{H}}[2])$

$\boldsymbol{L_\mathcal{H}} \leftarrow \langle \mathsf{Blind}(\boldsymbol{k_\mathcal{H}}[i], \boldsymbol{b_\mathcal{H}}[1]) \mid 1 \leq i \leq n \rangle$

let $\boldsymbol{S^{\mathcal{H}-1}} = \langle (M_1, o_1), \ldots, (M_\delta, o_\delta) \rangle$

$\boldsymbol{S^\mathcal{H}} \leftarrow \tau_\mathcal{H}(\langle F(M_i, o_i, \boldsymbol{b_\mathcal{H}}[2], \boldsymbol{\sigma_\mathcal{H}}[i]) \mid 1 \leq i \leq \delta \rangle)$

$\boldsymbol{M^\mathcal{H}} \leftarrow \mathsf{Unblind}(\boldsymbol{E^K}[i], \boldsymbol{b_\mathcal{H}}[1])$

return $D_i, \boldsymbol{L_i}, \boldsymbol{S^i}, \boldsymbol{E^K}, \boldsymbol{M^i} \quad \textbf{for } \forall i$

Figure 6.2: View of a maximal malicious coalition.

Since Blind is IND-CPA, $\boldsymbol{L_\mathcal{H}}$ is indistinguishable from a list of the same size containing blinded strings of zeros. We define $\mathsf{ColView}_2$ as a function identical to ColView, except that the delayed message hides random strings and each $\boldsymbol{k_\mathcal{H}}[i]$ is replaced with a string of zeros. Consequently, $\mathcal{A}$ cannot distinguish between ColView and $\mathsf{ColView}_2$.

Given that F is IND-CPA, we observe that $\mathsf{ColView}_2$ is computationally indistinguishable from $\mathsf{ColView}_3$, which behaves the same way but with $S^\mathcal{H}$ replaced by $\langle F(M_\varepsilon, o_\varepsilon, \boldsymbol{b_\mathcal{H}}[2], \boldsymbol{\sigma'_\mathcal{H}}[i]) \mid 1 \leq i \leq \delta \rangle$ for some fixed (and known to $\mathcal{A}$) M_ε, o_ε, and fresh $\sigma'_\mathcal{H}[i]$.

Therefore, $\mathcal{A}$'s view can be considered equivalent to $\mathsf{ColView}_3$, where $\tau_\mathcal{H}$, $s_\mathcal{H}$, and $\boldsymbol{k_\mathcal{H}}[1]$ do not appear at all. This implies that $\mathcal{A}$'s view is computationally independent of these parameters. □

We have now established that the setup phase achieves its goals: it keeps the secret hidden and shuffles the keys without leaking information. Consequently, an adversary can only abort the exchange phase at an arbitrary point independent of when the secrets are revealed. Therefore, the probability of $\mathcal{A}$ forcing unfairness is bounded above by $\frac{1}{\delta}$, which is explained in detail in Section 6.4. Theorem 6.3 formalises this reasoning and proves the partial fairness property of our protocol in the worst-case scenario of only one honest party.

Theorem 6.3. *For every non-uniform PPT adversary $\mathcal{A}$ for the exchange protocol of Figure 6.1, there exists a non-uniform PPT adversary $\mathcal{S}$ for the ideal fair exchange functionality such that*

$$\left\{\text{IDEAL}_{\mathcal{S}}^{\mathcal{F}_{ex}}(\Sigma)\right\} \overset{\frac{K-1}{N+1-K}}{\approx} \left\{\text{REAL}_{\mathcal{A}}^{\Pi}(\Sigma)\right\}$$

Proof. Fix an adversary $\mathcal{A}$, and construct a simulator $\mathcal{S}$ that behaves as follows:

1. Choose a random secret $s'_{\mathcal{H}}$ for the honest party.
2. Run a local copy of $\mathcal{A}$, impersonating the honest party with input $s'_{\mathcal{H}}$.
3. After the first round when all the delay messages are sent, open them and rewind $\mathcal{A}$.
4. Proceed with the protocol, simulating $\mathcal{H}$, until the point where $\mathcal{H}$ would send the key that discloses $s'_{\mathcal{H}}$.
5. Engage with the TNP in the ideal world to perform the exchange and obtain the real secret $s_{\mathcal{H}}$.
6. Utilise the equivocability of the symmetric encryption scheme to construct the message that would allow $\mathcal{A}$ to obtain $s_{\mathcal{H}}$, and send it.
7. Continue until the protocol terminates, and then return whatever $\mathcal{A}$ returns.

We remark that $\mathcal{S}$ must always return $\mathcal{A}$'s output, even if $\mathcal{A}$ aborts early.

Essentially, the simulator mimics an honest party but lacks knowledge of $\mathcal{H}$'s actual input. To address this, $\mathcal{S}$ leverages the symmetric encryption scheme's equivocability. However, this requires knowing when the secrets are exchanged, which is achieved by opening the delayed messages after the first round to reveal the random permutations τ_i used by $\mathcal{A}$ during shuffling. Rewinding $\mathcal{A}$ is necessary due to the assumed protocol termination before opening delayed messages is possible.

To prove partial fairness, we first identify differences between the ideal and real worlds. In the ideal world, the exchange happens when $\mathcal{H}$ is expected to send its secret in the real world. Thus, the ideal-world honest party returns $\mathcal{A}$'s controlled

secrets whenever $\mathcal{S}$ reaches this point. In the real world, the honest party only outputs these secrets if $\mathcal{A}$ sends all of them.

These scenarios diverge when $s_{\mathcal{H}}$ is sent before some of the other secrets, and $\mathcal{A}$ aborts during this interval.[7] $\mathcal{A}$ has at most $K-1$ such opportunities out of $\delta = N+1-K$ total opportunities.[8]

Due to equivocability, $\mathcal{A}$ cannot distinguish between the simulation with $\mathcal{S}$ and interaction with the honest party. Furthermore, Lemma 6.2 establishes that $\mathcal{A}$ has no information about the timing of $\mathcal{H}$'s key message, making their abortion choice independent of this message's position.

Therefore, the probability of $\mathcal{A}$ aborting in a favourable position is at most $\frac{K-1}{N+1-K}$. This concludes our proof. Note that this probability decreases linearly if $\mathcal{A}$ controls fewer parties, and the distribution changes if the setup is run with $L \neq K-1$. Section 6.4 presents further details on the different achievable distributions.

□

While we have concluded the proof of partial fairness, we still need to demonstrate the security of our masking function F. Lemma 6.4 shows that the IND-CPA properties of the blinding scheme are sufficient to prove that F is also IND-CPA.

Lemma 6.4. *If* Blind *is* IND-CPA*, then the function $F(M, f, b, g)$ defined in Figure 6.1 is an* IND-CPA *encryption scheme of (M, f), provided that g is used once.*[9]

Proof. Assuming Blind is IND-CPA-secure, we will prove that an adversary $\mathcal{A}$ with a non-negligible advantage in the IND-CPA game for F leads to an adversary $\mathcal{S}$ obtaining a non-negligible advantage in an impossible game. Consider Figure 6.3, which illustrates the construction of $\mathcal{S}$ exploiting $\mathcal{A}$.

The impossible game that $\mathcal{S}$ aims to win is as follows: Fix integers K, n, and a set $\mathcal{X}$. $\mathcal{S}$ chooses two permutations, τ and σ, that permute the rows of a $K \times n$

[7] This is due to Lemma 6.1.

[8] This assumes the protocol is run with $L = K-1$. Section 6.4 provides a deeper analysis of other distributions.

[9] See Figure 6.3 for a precise definition of the IND-CPA game for F.

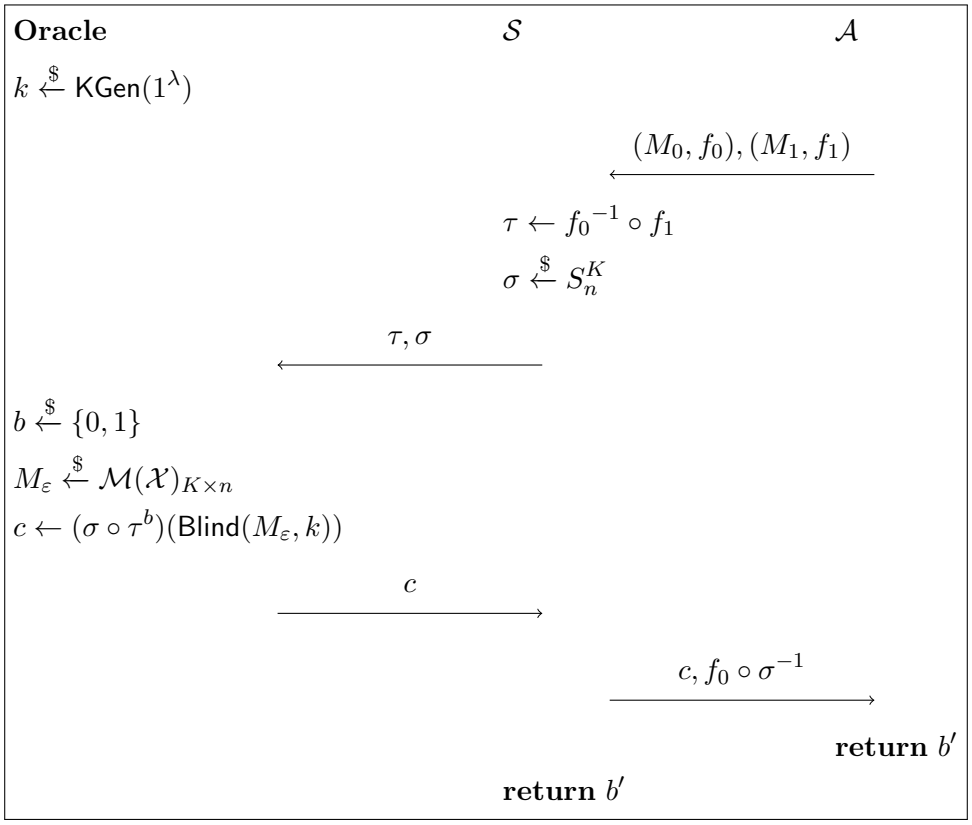

Figure 6.3: Proof that F is IND-CPA.

matrix with entries from $\mathcal{X}$. The oracle then samples a random matrix, blinds it, and permutes the result with either $\sigma \circ \tau$ or σ. These permutations are chosen of this form specifically for the security reduction that follows, but their structure is irrelevant within the context of the impossible game.

To understand the impossibility, note that $\mathcal{S}$ receives the encryption of a matrix with entries it has never seen. For each matrix M_ε sampled by the oracle, there exists a "sibling" matrix $M'_\varepsilon = \tau^{2b-1}(M_\varepsilon)$ such that $(\sigma \circ \tau^b)(M_\varepsilon) = (\sigma \circ \tau^{1-b})(M'_\varepsilon)$. Therefore, given an oracle response c, it could have been obtained from either M_ε or M'_ε depending on the value of b. Since the matrix is chosen uniformly, this provides no information to $\mathcal{S}$, resulting in no advantage (not even a negligible one).

Now, consider Figure 6.3 again. If we prove that $\mathcal{A}$'s view is indistinguishable from its view in the IND-CPA game for F, we establish that $\mathcal{A}$'s advantage is at most negligibly larger than $\mathcal{S}$'s advantage (which is exactly $\frac{1}{2}$).

Assume b=0. Then the message $\mathcal{A}$ receives is $\sigma(\mathsf{Blind}(M_\varepsilon, k)), f_0 \circ \sigma^{-1}$. Since Blind is IND-CPA, this is computationally indistinguishable from $\sigma(\mathsf{Blind}(M_0, k)), f_0 \circ \sigma^{-1}$, which is exactly what $\mathcal{A}$ would expect.

If $b = 1$, then $\mathcal{A}$ receives $(\sigma \circ \tau)(\mathsf{Blind}(M_\varepsilon, k)), f_0 \circ \sigma^{-1}$. Again, this is computationally indistinguishable from $(\sigma \circ \tau)(\mathsf{Blind}(M_1, k)), f_0 \circ \sigma^{-1}$, which is equivalent to $(\sigma \circ f_0^{-1} \circ f_1)(\mathsf{Blind}(M_1, k)), f_1 \circ (\sigma \circ f_0^{-1} \circ f_1)^{-1}$. Since σ is uniformly distributed, so is $\sigma \circ {f_0}^{-1} \circ f_1$.

Thus, $\mathcal{A}$'s view in Figure 6.3 is indistinguishable from its view in the IND-CPA game for F, concluding our proof. □

6.4 Fairness Optimality

We represent a protocol with N messages and K parties $\{\mathbf{P_1}, \ldots, \mathbf{P_K}\}$ as a pair $\Pi = (\mathcal{M}, \mathcal{P})$, where:

- $\mathcal{M} : \mathbb{N} \to \{1, \ldots, K\}$ indicates the order in which parties send messages. $\mathcal{M}(i) = j$ means party $\mathbf{P_j}$ sends message i.
- A "configuration" $t : \{1, \ldots, K\} \to \mathbb{N}$ indicates secret locations. $t(i) = j$ means $\mathbf{P_i}$'s secret is in the j^{th} message.
- $\mathcal{P}$ assigns probabilities to each configuration.

We impose the constraint $t(i) = j \implies \mathcal{M}(j) = i$. This means if $\mathbf{P_i}$'s secret is disclosed in the j^{th} message, $\mathbf{P_i}$ must send it. This reflects the principle that a secret, once disclosed, is disclosed to everyone. "Disclosed" doesn't mean immediate knowledge but guaranteed knowledge in a nearby future. In the protocol of Section 6.2, s_i is "disclosed" when the blinded $\boldsymbol{k_i}[1]$ is sent by $\mathbf{P_i}$, but others learn s_i only after opening D_i.

For a protocol $\Pi = (\mathcal{M}, \mathcal{P})$, the probability of a coalition I "winning" by stopping at $S \in \mathbb{N}$ is:

$$\Pr_\Pi[I,S] = \sum_{t \in \mathcal{T}(\Pi,I,S)} \mathcal{P}(t)$$
$$\text{where } \mathcal{T}(\Pi,I,S) = \{t \mid \exists i \in I \;\; t(i) > S \wedge \forall j \notin I \;\; t(j) \leq S\}$$

I wins by stopping at S if someone in I hasn't released their secret after the S^{th} message, but all outside I have. We write $t < S$ if $\forall i \; 1 \leq i \leq K, t(i) < S$, and similarly for $t > S$.

To find an optimal protocol, we assume no consecutive messages from one party. Unfairness against size-L coalitions is measured by:

$$\max_{\substack{I,S \\ |I|=L}} \Pr_\Pi[I,S]$$

Lemma 6.5. *Let $\Pi_b = (\mathcal{M}_b, \mathcal{P}_b)$ be a protocol with configuration t_b such that $\mathcal{P}_b(t_b) > 0$. Let t_g be a configuration where $\max t_g = \max t_b$ and $\forall i \; t_g(i) \geq t_b(i)$. Define $\Pi_g = (\mathcal{M}_b, \mathcal{P}_g)$ with $\mathcal{P}_g(t_b) = 0$, $\mathcal{P}_g(t_g) = \mathcal{P}_b(t_g) + \mathcal{P}_b(t_b)$, and $\mathcal{P}_g(t) = \mathcal{P}_b(t)$ for other t. Then for any L:*

$$\max_{\substack{I,S \\ |I|=L}} \Pr_{\Pi_b}[I,S] \geq \max_{\substack{I,S \\ |I|=L}} \Pr_{\Pi_g}[I,S]$$

Proof. We prove the statement by showing that for all I, S we have:

$$\Pr_{\Pi_b}[I,S] \geq \Pr_{\Pi_g}[I,S] \tag{6.1}$$

Note that $\mathcal{T}(\Pi_g, I, S) = \mathcal{T}(\Pi_b, I, S)$. Hence, the inequality above becomes:

$$\sum_{t \in \mathcal{T}(\Pi_b,I,S)} \mathcal{P}_b(t) \geq \sum_{t \in \mathcal{T}(\Pi_b,I,S)} \mathcal{P}_g(t)$$

Therefore, it suffices to show that if $t_g \in \mathcal{T}(\Pi_b, I, S)$, then $t_b \in \mathcal{T}(\Pi_b, I, S)$. In other words, if coalition I wins by stopping at S when configuration t_g occurs, then I also wins by stopping at S when configuration t_b occurs.

Assume $t_g \in \mathcal{T}(\Pi_b, I, S)$. This implies that for all $i \notin I$, we have $t_g(i) \leq S$. Since $t_b(i) \leq t_g(i)$ for all i, it follows that for all $i \notin I$, $t_b(i) \leq S$.

Furthermore, let $j = \arg\max t_g$. By the definition of $\mathcal{T}(\Pi_b, I, S)$, we know that $j \in I$ and $t_g(j) > S$. Since $\max t_b = \max t_g$, we have $t_b(j) > S$.

Hence, we have shown that $t_b \in \mathcal{T}(\Pi_b, I, S)$, completing the proof. □

For each x where a configuration t exists with $\max t = x$, there's a unique "shortest" configuration t_x where $\forall i, t_x(i)$ is the largest valid integer up to x. The lemma shows we only need to consider these, as longer ones are worse.

Let $\Pi_g^L = (\mathcal{M}_g, \mathcal{P}_g)$ be a protocol where:

- $\mathcal{M}_g(i) = (i - 1 \bmod K) + 1$
- $\mathcal{P}_g(t_i) = \alpha_i$
- $\mathcal{P}_g(t) = 0$ for other configurations
- α_i's are chosen to minimise: $\max\limits_{\substack{I,S \\ |I|=L}} \Pr_{\Pi_g^L}[I, S]$

Lemma 6.6. *For $L < K$ and a protocol Π_g^x as above:*

$$\max_{\substack{I,S \\ |I|=L}} \Pr_{\Pi_g^x}[I, S] = \max\{\alpha_i + \cdots + \alpha_{i+L-1} \mid K \leq i \leq N + 1 - L\}$$

Proof. The short configurations t_i can be described as the map $y \mapsto i - ((\mathcal{M}(i) - y) \bmod K)$. That is, the secrets are sent in messages $\{i - K + 1, i - K + 2, \ldots, i\}$. All short configurations of the protocol Π_g^x are precisely the t_i configurations described above, where $K \leq i \leq N$.

Note that:

$$\max_{\substack{I,S \\ |I|=L}} \Pr_{\Pi_g^x}[I, S] = \max\left\{ \max_{I;|I|=L} \Pr_{\Pi_g^x}[I, S] \mid 1 \leq S \leq N \right\}$$

Fix an S. Consider the configurations $T = \{t_{S+1}, t_{S+2}, \ldots, t_{S+L}\}$.[10]

For any other configuration t_i, one of two scenarios can occur:

1. $i \leq S$, meaning all secrets are exchanged up to (and including) the S^{th} message.

[10] If $S + L > N$, some of these configurations may not exist and can be removed from the set.

2. $i > S + L$, meaning more than L secrets are exchanged after the S^{th} message.

In both scenarios, no coalition of size L can win by stopping at S in those configurations. However, for any configuration $t \in T$, any coalition containing $\{\mathbf{P_j} \mid t(j) > S\}$ can win in t by stopping at S. In particular, only the coalition $J = \{\mathbf{P_j} \mid j = \mathcal{M}(i), S+1 \leq i \leq S+L\}$ wins in all such configurations. Therefore, $\max_{I;|I|=L} \Pr_{\Pi_g^x}[I, S] = \Pr_{\Pi_g^x}[J, S] = \alpha_{S+1} + \cdots + \alpha_{S+L}$.[11]

By iterating over all possible values of S, we obtain the following set of sums:

$$\begin{aligned}
&\{\alpha_K, \alpha_K + \alpha_{K+1}, \alpha_K + \alpha_{K+1} + \alpha_{K+2}, \ldots, \alpha_K + \cdots + \alpha_{K+L-1}\} \\
&\cup \{\alpha_i + \cdots + \alpha_{i+L-1} \mid K \leq i \leq N+1-L\} \\
&\cup \{\alpha_{N+1-L} + \cdots + \alpha_N, \alpha_{N+2-L} + \cdots + \alpha_N, \ldots, \alpha_N\}
\end{aligned}$$

Note that the sums in the first and third sets are all "sub-sums" of sums in the second set. Therefore, the maximum over these three sets is the same as the maximum over the second set, which proves the lemma. □

Theorem 6.7. *For any protocol* Π*:*

$$\max_{\substack{I,S \\ |I|=L}} \Pr_{\Pi}[I, S] \geq \max_{\substack{I,S \\ |I|=L}} \Pr_{\Pi_g^L}[I, S]$$

Proof. First, we construct an intermediate protocol $\Pi_b = (\mathcal{M}_b, \mathcal{P}_b)$ such that:

- $\mathcal{M}_b = \mathcal{M}$
- $\mathcal{P}_b(t_i) = \beta_i$, where t_i are the "shortest" configurations in $\mathcal{M}_b$
- $\mathcal{P}_b(t) = 0$ for other configurations t
- The β_i's minimise: $\max_{\substack{I,S \\ |I|=L}} \Pr_{\Pi_b}[I, S]$

Lemma 6.5 and the definition of $\mathcal{P}_b$ imply that $\max_{\substack{I,S \\ |I|=L}} \Pr_{\Pi}[I, S] \geq \max_{\substack{I,S \\ |I|=L}} \Pr_{\Pi_b}[I, S]$. Therefore, we can prove the theorem by showing the same inequality between Π_b and Π_g^L.

[11] If some of these α's refer to non-existent configurations, those values can be considered zero.

By previous lemmas, the α_i's minimise:

$$\max\{\alpha_i + \cdots + \alpha_{i+L-1} \mid K \leq i \leq N+1-L\} \left(= \max_{\substack{I,S \\ |I|=L}} \Pr_{\Pi_g^L}[I,S] \right)$$

Recall that β_i is the probability of the configuration t_i, which is the unique shortest configuration with $\max t_i = i$.[12] We have now constructed the set $\{\beta_K, \ldots, \beta_N\}$.

We prove the result by showing that:

$$\begin{aligned} \max_{\substack{I,S \\ |I|=L}} \Pr_{\Pi_b}[I,S] &\geq \max\{\beta_i + \cdots + \beta_{i+L-1} \mid K \leq i \leq N+1-L\} \\ &\geq \max\{\alpha_i + \cdots + \alpha_{i+L-1} \mid K \leq i \leq N+1-L\} \end{aligned}$$

The second inequality follows directly from the definition of the α_i's and Lemma 6.6.

Fix any i and consider the configurations $T = \{t_i, t_{i+1}, \ldots, t_{i+L-1}\}$ and the coalition $I = \{\mathbf{P_j} \mid \mathcal{M}_b(\max t) = j \text{ for } t \in T\}$ of parties sending messages from i to $i+L-1$.[13] Set $S = i-1$. By the definition of I, all messages sent after S up to (and including) $i+L-1$ are sent by parties in I. Therefore, I wins in all the coalitions in T by stopping at S. Hence, $\Pr_{\Pi_b}[I,S] \geq \beta_i + \cdots + \beta_{i+L-1}$.

Since i was arbitrary, we have:

$$\max_{\substack{I,S \\ |I|=L}} \Pr_{\Pi_b}[I,S] \geq \max\{\beta_i + \cdots + \beta_{i+L-1} \mid K \leq i \leq N+1-L\}$$

And the theorem follows. □

This shows Π_g^L is optimal against size-L coalitions. To construct such protocol, we analyse Π_g^L to reveal the values of α's.

Lemma 6.8. *Given non-negative $\alpha_1, \ldots, \alpha_n$ summing to 1, and $l \leq n$, with $\lambda = \lceil \frac{n}{l} \rceil$:*

$$\max\{\alpha_i + \cdots + \alpha_{i+l-1} \mid 1 \leq i \leq n+1-l\} \geq \frac{1}{\lambda} \tag{6.2}$$

[12] It's possible that for some small i up to a value x, t_i does not exist. For example, if the first K messages are sent by only two parties, the "earliest" configuration t will have $\max t \geq 2K-2$, so $t = t_K$ doesn't exist. If t_i doesn't exist, we let it be any configuration where $\max t_i$ is minimal, even if it's not short.

[13] If T contains configurations t_i, t_j with $\mathcal{M}_b(i) = \mathcal{M}_b(j)$, then I might not have size L. In this case, we can arbitrarily extend I to a coalition of size L, so we assume $|I| = L$.

Proof. Assume, for the sake of contradiction, that equation 6.2 does not hold.

Define $v_i = \{\alpha_{l(i-1)+1}, \alpha_{l(i-1)+2}, \ldots, \alpha_{li}\}$ for $1 \leq i \leq \lfloor \frac{n}{l} \rfloor$. If $n \bmod l \neq 0$, then set $v_\lambda = \{\alpha_{n+1-l}, \ldots, \alpha_n\}$. Since equation 6.2 is assumed not to hold, we have that for all $1 \leq i \leq \lambda$, the inequality $\sum_{\alpha \in v_i} \alpha < \frac{1}{\lambda}$ holds. However, this leads to a contradiction:

$$1 = \sum_{i=1}^{n} \alpha_i \leq \sum_{i=1}^{\lambda} \sum_{\alpha \in v_i} \alpha < \sum_{i=1}^{\lambda} \frac{1}{\lambda} = 1$$

Therefore, our initial assumption must be false, and equation 6.2 holds. □

Theorem 6.9. *With the same setup as Lemma 6.8:*

$$\min_{\alpha} \max\{\alpha_i + \cdots + \alpha_{i+l-1} \mid 1 \leq i \leq n+1-l\} = \frac{1}{\lambda} \tag{6.3}$$

Proof. By Lemma 6.8, it suffices to exhibit a set of α's such that:

$$\max\{\alpha_i + \cdots + \alpha_{i+l-1} \mid 1 \leq i \leq n+1-l\} = \frac{1}{\lambda}$$

For $1 \leq i < \lambda$, set $\alpha_{l(i-1)+1} = \frac{1}{\lambda}$, and let $\alpha_n = \frac{1}{\lambda}$. Set all other α's to zero. Note that these values are chosen so that non-zero values have at least $l-1$ zero values between them. Therefore, any sum $\alpha_i + \cdots + \alpha_{i+l-1}$ will contain at most one non-zero value, and all non-zero sums will be equal to $\frac{1}{\lambda}$. The theorem follows immediately. □

This theorem reveals the optimal fairness is $\frac{1}{\lambda}$ and how to achieve it. Other α values exist for this fairness, so this isn't a complete classification, but it's enough for an optimally fair protocol.

Corollary 6.10. *If $N+1-K$ is a multiple of $\operatorname{lcm}\{2, 3, \ldots, K-1\}$, then Π_g^1 is optimally fair against coalitions of any size.*

Proof. By our definition of $\Pi_g^1 = (\mathcal{M}_1, \mathcal{P}_1)$ and Lemma 6.6, the α_i's minimise:

$$\max\{\alpha_i \mid K \leq i \leq N\}$$

Hence, $\alpha_i = \frac{1}{N+1-K}$ for all i. This means all the $N+1-K$ shortest configurations are equally likely.

Pick any $L < K$. From Theorem 6.7, we know that $\Pi_g^L = (\mathcal{M}_L, \mathcal{P}_L)$ is optimally fair against coalitions of size L. Let $\mathcal{P}_L(t_i) = \beta_i$. Since the shortest configurations in Π_g^1 and Π_g^L are the same, β_i and α_i refer to the same configuration t_i. By Lemma 6.6, we know that these β_i's minimise:

$$\max_{\substack{I,S \\ |I|=L}} \Pr_{\Pi_g^L}[I, S] = \max\{\beta_i + \cdots + \beta_{i+L-1} \mid K \leq i \leq N+1-L\}$$

Theorem 6.9 states that this maximum value is $\frac{1}{\left\lceil \frac{N+1-K}{L} \right\rceil} = \frac{L}{N+1-K}$, since L divides $N+1-K$. Finally, by Lemma 6.6, the theorem follows:

$$\max_{\substack{I,S \\ |I|=L}} \Pr_{\Pi_g^1}[I, S] = \max\{\alpha_i + \cdots + \alpha_{i+L-1} \mid K \leq i \leq N+1-L\} = \frac{L}{N+1-K}$$

□

This corollary guides the construction of a protocol optimally fair against any coalition size, achievable with the protocol of Figure 6.1. If Corollary 6.10 holds, $\mathbf{P_1}$ constructs a permutation π_i for each of the $N+1-K$ "shortest" configurations, where $\pi_i[x] = i-1+((x-i) \bmod K)$. Note that our protocol of Figure 6.1 can also instantiate Π_g^L's distribution for any L. Thus, for large K, where using N suitable for Corollary 6.10 is impractical, the protocol can be tuned for optimality against specific L values.

6.5 Conclusions and Future Research

In this chapter, we designed a multi-party exchange protocol achieving optimal partial fairness, meaning that no protocol with a shorter exchange phase can achieve a higher probability of a fair exchange. While the underlying ideas are the same as in other partially fair protocols, such as the one in Chapter 5, the protocol described here differs substantially in the setup phase. As revealed in the abstract study conducted to prove the optimality of our protocol, the exchange becomes much more complex when multiple parties are involved. Furthermore, the natural notion of optimality implies that it might not be possible to construct a single protocol

that is optimal against all adversaries. Therefore, the protocol we designed must accommodate different shuffling strategies, allowing the involved parties to decide on the distributions their secrets should follow.

Building upon this work, future research could focus on developing a more comprehensive understanding of partial fairness through a classification of optimal protocols. Of particular interest is the design of practical protocols that guarantee full fairness when possible, but can still achieve partial fairness in adverse conditions. Exploring the use of oblivious transfer as an alternative to commutative blinding could result in a more efficient setup phase. Similar to the two-party setting, an OT-based protocol could provide provable security within the Universally Composable framework.

7 Conclusions

This thesis has explored the fascinating field of Time-Lock Puzzles (TLPs) and their applications to fair exchange. Over five chapters, we have delved into different branches of cryptography, answering two important research questions: how can we construct an efficient and quantum-safe TLP? Can we use TLPs to solve previously impossible Multi-Party Computation (MPC) problems? Our research provides positive answers to both, designing a novel TLP and partially-fair exchange protocols.

7.1 Contributions

This thesis commenced with a comprehensive review and classification of existing TLPs in Chapter 2. This analysis serves as a valuable reference, consolidating the knowledge base of TLPs based on their underlying assumptions and security properties.

Chapter 3 introduced a novel TLP design based on extracting cube roots, offering the first detailed analysis of a TLP based on exponentiation in a group of known order. Notably, we introduced the use of a proper pseudo-random permutations for chaining TLPs and achieved quantum safety akin to hash-chain-based puzzles without completely sacrificing the efficiency of repeated squaring-based puzzles like RSW. The chapter includes a proof of the sequentiality of cubing in the strongly

algebraic group model, and it establishes a link between repeated squaring (with known or unknown order) and the discrete logarithm problem. It concludes with a first-of-its-kind extensive cryptanalysis, revealing the theoretical limits of repeated squaring, surveying practical attack vectors, and analysing the quantum safety of standard assumptions.

In Chapter 4, a detour was taken to present a commutative encryption scheme with enhanced privacy requirements, a key primitive for the protocols introduced in later chapters. This chapter includes three constructions: one based on ElGamal, a generic construction, and a post-quantum secure design based on LWE, addressing the need for secure commutative blinding in various cryptographic settings.

Leveraging previous constructions, Chapter 5 showcases a 2-party partially-fair exchange protocol that improves upon previous bounds and achieves optimality, providing security against covert adversaries without expensive zero-knowledge proofs. After a thorough security proof, the real-world scenario of repeated attempts at exchanging the same items is investigated, and an approach using nested oblivious transfers is outlined to achieve more complex shufflings and eliminate the need for commutative blinding. This emphasises that timed-release encryption is the key primitive enabling the protocol's optimality and partial fairness for all types of secrets.

Building upon the 2-party case, Chapter 6 extends the concept to a multi-party context. The introduction of more parties renders the protocol analysis more complex, and the natural definition of optimality leads to less-than-obvious distributions of secrets requiring non-trivial shuffling techniques. As a result, a protocol far more flexible than its 2-party counterpart is obtained. The chapter concludes with a high-level analysis of all exchange protocols based on similar underlying ideas, showing a way to achieve optimality against all adversaries using the proposed protocol.

In conclusion, this thesis has advanced our understanding of TLPs and their applications in cryptography. The theoretical results span various branches of cryptography, from TLP construction and cryptanalysis to applications in fair exchange, with new commutative blinding designs as a bonus.

7.2 Future Research

Despite fully answering important research questions, the research paths travelled in this thesis are yet to be fully explored. While the theoretical foundations of TLPs have been researched, practical implementation and hardware optimisation remain almost completely unexplored. Future research should focus on developing concrete benchmarks and hardware designs to bridge this gap, particularly for synthesising theoretical and simulated circuits.

Another research avenue involves studying the advantages and drawbacks of using different underlying groups instead of prime fields. Our proofs of sequentiality remain independent of the underlying group used for cubing. However, different group operations could offer varying guarantees in terms of sequentiality. For example, matrix groups are not ideal since matrix multiplication can be easily parallelised. Nonetheless, other groups might provide operations that are less parallelisable than integer multiplication.

A related research direction could investigate sequentiality results similar to ours but in the context of group actions. This approach would enable basing the puzzle on the action of isogenies on elliptic curves. CSIDH [39] is a prime example of the use of such group actions. However, exploring other groups will also be beneficial, as current isogeny-based TLPs are not practical (see Section 2.1.7).

While we have proven the security of our construction, it can be further strengthened by adopting more tailored security models instead of abstracting the overall construction as consecutive calls to a random oracle. Alternatively, the puzzle could be adapted to use different underlying groups for each cubing operation, which would further limit the effectiveness of any type of precomputation attack.

However, this approach requires generating many groups for a single puzzle, necessitating a deep study on efficient generation of the underlying groups.

Another area that requires more work to further push towards adopting the cubing approach instead of the standard RSW is the design of all the "extra" properties that RSW gained throughout years of research. Specifically, can we design the cubing to obtain homomorphic TLPs? What about simple proofs of unlocking time? Additionally, can we efficiently prove properties of the plaintext in zero-knowledge?

Our results on fair exchange point to exciting new prospects. A complete classification of optimally-fair multi-party exchanges is within reach, following our abstract analysis in Chapter 6, and could unlock simpler protocols. Future research should focus on studying the multiple-attempts scenario presented in Chapter 5, which is likely to highlight the need for more flexible protocols for practical applications. While our sketched construction using oblivious transfer partially addresses this need, more work is required to produce sound security proofs and adapt the design to a multi-party context. Specifically, our multi-party setup phase has round complexity depending not only on the desired fairness but also on the number of parties. Ideally, this dependency would be removed by sending more messages concurrently, without significantly increasing the overall message complexity. Moreover, an OT-based design is more likely to ensure our protocols are secure within the UC framework.

Bibliography

[1] Aydin Abadi and Aggelos Kiayias. Multi-instance publicly verifiable time-lock puzzle and its applications. In *Financial Cryptography and Data Security: 25th International Conference, FC 2021, Virtual Event, March 1–5, 2021, Revised Selected Papers, Part II 25*, pages 541–559. Springer, 2021. 8, 54

[2] Aydin Abadi, Dan Ristea, and Steven J. Murdoch. Delegated time-lock puzzle, 2023. https://arxiv.org/abs/2308.01280. 23

[3] Martin Abadi, Mike Burrows, Mark Manasse, and Ted Wobber. Moderately Hard, Memory-bound Functions. *ACM Transactions on Internet Technology*, 5:299–327, May 2005. 8, 13

[4] David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J. Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, Benjamin VanderSloot, Eric Wustrow, Santiago Zanella-Béguelin, and Paul Zimmermann. Imperfect forward secrecy: How Diffie-Hellman fails in practice. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 5–17, New York, NY, USA, 2015. Association for Computing Machinery. 71, 77

[5] Abtin Afshar, Kai-Min Chung, Yao-Ching Hsieh, Yao-Ting Lin, and Mohammad Mahmoody. On the (im)possibility of time-lock puzzles in the quantum random oracle model. In Jian Guo and Ron Steinfeld, editors, *Advances in Cryptology – ASIACRYPT 2023*, pages 339–368, Singapore, 2023. Springer Nature Singapore. 3, 12, 17, 31, 55

[6] VDF Alliance. Vdf alliance fpga contest round 2 results, 2020. https://github.com/supranational/vdf-fpga-round2-results, Last accessed: 2023-11-28. 68

[7] Mohammad Hassan Ameri, Alexander R. Block, and Jeremiah Blocki. Memory-hard puzzles in the standard model with applications to memory-hard functions and resource-bounded locally decodable codes. In Clemente Galdi and Stanislaw Jarecki, editors, *Security and Cryptography for Networks*, pages 45–68, Cham, 2022. Springer International Publishing. 14

[8] 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*, pages 398–426, Cham, 2021. Springer International Publishing. 8

[9] N. Asokan, Matthias Schunter, and Michael Waidner. Optimistic protocols for multi-party fair exchange. Journal article, IBM Research Division, 1996. 19, 99

[10] Gildas Avoine, Felix Gärtner, Rachid Guerraoui, and Marko Vukolić. Gracefully Degrading Fair Exchange with Security Modules. In Mario Dal Cin, Mohamed Kaâniche, and András Pataricza, editors, *Dependable Computing - EDCC 5*, pages 55–71, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. 19

[11] Gildas Avoine and Serge Vaudenay. Fair exchange with guardian angels. In Ki-Joon Chae and Moti Yung, editors, *Information Security Applications*, pages 188–202, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. 19

[12] Ahmad Al Badawi, Andreea Alexandru, Jack Bates, Flavio Bergamaschi, David Bruce Cousins, Saroja Erabelli, Nicholas Genise, Shai Halevi, Hamish Hunt, Andrey Kim, Yongwoo Lee, Zeyu Liu, Daniele Micciancio, Carlo Pascoe, Yuriy Polyakov, Ian Quah, Saraswathy R.V., Kurt Rohloff, Jonathan Saylor, Dmitriy Suponitsky, Matthew Triplett, Vinod Vaikuntanathan, and

Vincent Zucca. Openfhe: Open-source fully homomorphic encryption library. Cryptology ePrint Archive, Paper 2022/915, 2022. https://eprint.iacr.org/2022/915,https://www.openfhe.org/. 96

[13] Fadi Barbàra, Nadir Murru, and Claudio Schifanella. Towards a broadcast time-lock based token exchange protocol. In Ricardo Chaves, Dora B. Heras, Aleksandar Ilic, Didem Unat, Rosa M. Badia, Andrea Bracciali, Patrick Diehl, Anshu Dubey, Oh Sangyoon, Stephen L. Scott, and Laura Ricci, editors, *Euro-Par 2021: Parallel Processing Workshops*, pages 243–254, Cham, 2022. Springer International Publishing. 8, 10

[14] Elaine Barker, Lily Chen, Allen Roginsky, Apostol Vassilev, and Richard Davis. Recommendation for pair-wise key-establishment schemes using discrete logarithm cryptography. Technical Report NIST SP 800-56Ar3, National Institute of Standards and Technology, Gaithersburg, MD, April 2018. 11

[15] Paul Barrett. Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor. In Andrew M. Odlyzko, editor, *Advances in Cryptology — CRYPTO' 86*, pages 311–323, Berlin, Heidelberg, 1987. Springer Berlin Heidelberg. 60

[16] Carsten Baum, Bernardo David, Rafael Dowsley, Jesper Buus Nielsen, and Sabine Oechsner. Tardis: A foundation of time-lock puzzles in uc. Advances in Cryptology – EUROCRYPT 2021, pages 429–459. Springer International Publishing, 2021. 8

[17] P.W. Beame, S.A. Cook, and H.J. Hoover. Log depth circuits for division and related problems. In *25th Annual Symposium onFoundations of Computer Science, 1984.*, pages 1–6, 1984. 64, 66

[18] Amos Beimel, Yehuda Lindell, Eran Omri, and Ilan Orlov. $\frac{1}{p}$-secure multiparty computation without an honest majority and the best of both worlds. *Journal of Cryptology*, 33(4):1659–1731, 2020. 4, 20, 21, 137, 141, 144

[19] Mihir Bellare, Anand Desai, David Pointcheval, and Phillip Rogaway. Relations among notions of security for public-key encryption schemes. Cryptology ePrint Archive, Paper 1998/021, 1998. 82

[20] M. Ben-Or, O. Goldreich, S. Micali, and R. L. Rivest. A fair protocol for signing contracts. *IEEE Transactions on Information Theory*, 36(1):40–46, 1990. 20

[21] Michael A. Bennett, Greg Martin, Kevin O'Bryant, and Andrew Rechnitzer. Explicit bounds for primes in arithmetic progressions. *Illinois Journal of Mathematics*, 62(1-4):427 – 532, 2018. 37

[22] Alex Biryukov, Ben Fisch, Gottfried Herold, Dmitry Khovratovich, Gaëtan Leurent, María Naya-Plasencia, and Benjamin Wesolowski. Cryptanalysis of Algebraic Verifiable Delay Functions. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024*, pages 457–490, Cham, 2024. Springer Nature Switzerland. 11

[23] Nir Bitansky, Shafi Goldwasser, Abhishek Jain, Omer Paneth, Vinod Vaikuntanathan, and Brent Waters. Time-lock puzzles from randomized encodings. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, ITCS '16, pages 345–356, New York, NY, USA, 2016. Association for Computing Machinery. 8, 14, 23, 31

[24] John Black and Phillip Rogaway. Ciphers with arbitrary finite domains. In Bart Preneel, editor, *Topics in Cryptology — CT-RSA 2002*, pages 114–130, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. 45

[25] B. Blanchet. A computationally sound mechanized prover for security protocols. In *2006 IEEE Symposium on Security and Privacy (S&P'06)*, pages 15 pp.–154, 2006. CryptoVerif is available online at https://bblanche.gitlabpages.inria.fr/CryptoVerif/. 110

[26] Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible protocols and atomic proxy cryptography. In Kaisa Nyberg, editor, *Advances in Cryptology — EUROCRYPT'98*, pages 127–144, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg. 84

[27] Lenore Blum, Manuel Blum, and Mike Shub. A simple unpredictable pseudo-random number generator. *SIAM Journal on computing*, 15(2):364–383, 1986. 53

[28] Marco Bodrato. Towards optimal Toom-Cook multiplication for univariate and multivariate polynomials in characteristic 2 and 0. In Claude Carlet and Berk Sunar, editors, *WAIFI'07 proceedings*, volume 4547 of *LNCS*, pages 116–133. Springer, June 2007. http://bodrato.it/papers/#WAIFI2007. 62

[29] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable Delay Functions. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018*, pages 757–788. Springer International Publishing, 2018. 8, 10, 11, 42, 45

[30] Dan Boneh and Moni Naor. Timed commitments. In Mihir Bellare, editor, *Advances in Cryptology — CRYPTO 2000*, pages 236–254. Springer Berlin Heidelberg, 2000. 8, 20, 53

[31] 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*, pages 407–437, Cham, 2019. Springer International Publishing. 8

[32] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Trans. Comput. Theory*, 6(3), jul 2014. 83, 92

[33] Jeffrey Burdges and Luca De Feo. Delay encryption. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT*

2021, pages 302–326, Cham, 2021. Springer International Publishing. 2, 8, 16, 17, 31

[34] Ran Canetti. Universally composable security. *J. ACM*, 67(5), sep 2020. 25, 112

[35] Ran Canetti, Uri Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, page 639–648, New York, NY, USA, 1996. Association for Computing Machinery. 100

[36] Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *Advances in Cryptology — CRYPTO 2001*, pages 19–40, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. 112

[37] Zhengjun Cao, Ruizhong Wei, and Xiaodong Lin. A fast modular reduction method. Cryptology ePrint Archive, Report 2014/040, 2014. https://ia.cr/2014/040. 61

[38] Wouter Castryck and Thomas Decru. An efficient key recovery attack on sidh. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023*, pages 423–447, Cham, 2023. Springer Nature Switzerland. 17

[39] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. CSIDH: An efficient post-quantum commutative group action. In Thomas Peyrin and Steven Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018*, pages 395–427, Cham, 2018. Springer International Publishing. 160

[40] Boon-Chiao Chang, Bok-Min Goi, Raphael C.-W. Phan, and Wai-Kong Lee. Multiplying very large integer in GPU with Pascal architecture. In *2018 IEEE Symposium on Computer Applications Industrial Electronics (ISCAIE)*, pages 401–405, 2018. 70

[41] Chao Chen and Fangguo Zhang. Verifiable delay functions and delay encryptions from hyperelliptic curves. *Cybersecurity*, 6(1):54, 2023. 8, 17

[42] H. Chen and R. Deviani. A Secure E-Voting System Based on RSA Time-Lock Puzzle Mechanism. In *2012 Seventh International Conference on Broadband, Wireless Computing, Communication and Applications*, pages 596–601, 2012. 8

[43] Hao Chen, Ilaria Chillotti, and Yongsoo Song. Multi-key homomorphic encryption from tfhe. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019*, pages 446–472, Cham, 2019. Springer International Publishing. 83

[44] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017*, pages 409–437, Cham, 2017. Springer International Publishing. 92

[45] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Tfhe: Fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, 2020. 83, 92

[46] R Cleve. Limits on the security of coin flips when half the processors are faulty. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, STOC '86, pages 364–369, New York, NY, USA, 1986. Association for Computing Machinery. 19, 137

[47] Bram Cohen and Krzysztof Pietrzak. Simple proofs of sequential work. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018*, pages 451–467, Cham, 2018. Springer International Publishing. 8, 13

[48] Geoffroy Couteau, A. W. Roscoe, and Peter Y. A. Ryan. Partially-fair computation from timed-release encryption and oblivious transfer. In Joonsang Baek and Sushmita Ruj, editors, *Information Security and Privacy*, pages

330–349, Cham, 2021. Springer International Publishing. 20, 21, 106, 119, 127, 129

[49] Ivan Damgård and Jesper Buus Nielsen. Improved non-committing encryption schemes based on a general complexity assumption. In Mihir Bellare, editor, *Advances in Cryptology — CRYPTO 2000*, pages 432–450, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg. 138

[50] Ivan Bjerre Damgård. Practical and provably secure release of a secret and exchange of signatures. *Journal of Cryptology*, 8(4):201–222, 1995. 20

[51] Moldovyan Dmitriy. Non-commutative finite groups as primitive of public key cryptosystems. *Quasigroups and Related Systems*, 18(2):165–176, 2010. 83

[52] 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*, pages 768–797, Cham, 2020. Springer International Publishing. 130, 132

[53] Cynthia Dwork, Andrew Goldberg, and Moni Naor. On Memory-Bound Functions for Fighting Spam. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, pages 426–444, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. 8, 13

[54] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In Ernest F. Brickell, editor, *Advances in Cryptology — CRYPTO' 92*, pages 139–147, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg. 8

[55] Cynthia Dwork, Moni Naor, and Hoeteck Wee. Pebbling and proofs of work. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, pages 37–54, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. 8, 10

[56] John G Earle. Latched carry save adder circuit for multipliers, 07 1965. U.S. Patent 3340388A. 63

[57] Edward Eaton and Douglas Stebila. The "quantum annoying" property of password-authenticated key exchange protocols. In Jung Hee Cheon and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography*, pages 154–173, Cham, 2021. Springer International Publishing. 52

[58] Martin Ekerå. Quantum algorithms for computing general discrete logarithms and orders with tradeoffs. *Journal of Mathematical Cryptology*, 15(1):359–407, 2021. 72, 73

[59] T. Elgamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985. 84

[60] Pavel Emeliyanenko. Efficient multiplication of polynomials on graphics hardware. In Yong Dou, Ralf Gruber, and Josef M. Joller, editors, *Advanced Parallel Processing Technologies*, pages 134–149, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. 70

[61] 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*, pages 125–154, Cham, 2020. Springer International Publishing. 8

[62] Shimon Even and Yacov Yacobi. Relations among public key signature systems. Report, TECHNION - Israel Istitute of Technology, 1980. 19

[63] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Paper 2012/144, 2012. https://eprint.iacr.org/2012/144. 91, 92

[64] Bao Feng, R. Deng, K. Q. Nguyen, and V. Varadharajan. Multi-party fair exchange with an off-line trusted neutral party. In *Proceedings. Tenth International Workshop on Database and Expert Systems Applications. DEXA 99*, pages 858–862, 1999. 19

[65] Matt Franklin and Gene Tsudik. Secure group barter: Multi-party fair exchange with semi-trusted neutral parties. In *Financial Cryptography*, pages 90–102. Springer Berlin Heidelberg, 1998. 18, 99

[66] Matthew K. Franklin and Michael K. Reiter. Fair Exchange with a Semi-Trusted Third Party (Extended Abstract). In *Proceedings of the 4th ACM Conference on Computer and Communications Security*, CCS '97, pages 1–5, New York, NY, USA, 1997. Association for Computing Machinery. 18

[67] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018*, pages 33–62, Cham, 2018. Springer International Publishing. 38

[68] Juan A. Garay and Markus Jakobsson. Timed Release of Standard Digital Signatures. In *Financial Cryptography*, pages 168–182. Springer Berlin Heidelberg, 2003. 8, 20

[69] Juan A. Garay and Carl Pomerance. Timed Fair Exchange of Standard Signatures. In *Financial Cryptography*, pages 190–207. Springer Berlin Heidelberg, 2003. 20

[70] Benoît Garbinato and Ian Rickebusch. Impossibility results on fair exchange. In Gerald Eichler, Peter Kropf, Ulrike Lechner, Phayung Meesad, and Herwig Unger, editors, *10th International Conferenceon Innovative Internet Community Systems (I2CS) – Jubilee Edition 2010 –*, pages 507–518, Bonn, 2010. Gesellschaft für Informatik e.V. 19, 99, 137

[71] Pierrick Gaudry, Alexander Kruppa, and Paul Zimmermann. A GMP-based implementation of Schönhage-Strassen's large integer multiplication algorithm. In *Proceedings of the 2007 International Symposium on Symbolic and Algebraic Computation*, ISSAC '07, pages 167–174, New York, NY, USA, 2007. Association for Computing Machinery. 63

[72] Craig Gidney and Martin Ekerå. How to factor 2048 bit rsa integers in 8 hours using 20 million noisy qubits. *Quantum*, 5:433, April 2021. 73

[73] GMP contributors. GMP assembly chart. https://gmplib.org/devel/asm Accessed: 2022-02-09. 69

[74] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, pages 218–229, New York, NY, USA, 1987. Association for Computing Machinery. 27, 106

[75] Oded Goldreich. *Foundations of Cryptography*. Cambridge University Press, 2004. 19, 27, 106, 112, 116, 126

[76] Shafi Goldwasser and Silvio Micali. Probabilistic encryption & how to play mental poker keeping secret all partial information. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, STOC '82, pages 365–377, New York, NY, USA, 1982. Association for Computing Machinery. 106

[77] P. Golle. Dealing cards in poker games. In *International Conference on Information Technology: Coding and Computing (ITCC'05) - Volume II*, volume 1, pages 506–511 Vol. 1, 2005. 106

[78] S. Dov Gordon and Jonathan Katz. Partial fairness in secure two-party computation. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, pages 157–176, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. 4, 20, 99, 119, 135, 137, 141

[79] T. Hamano, N. Takagi, S. Yajima, and F.P. Preparata. O(n)-depth circuit algorithm for modular exponentiation. In *Proceedings of the 12th Symposium on Computer Arithmetic*, pages 188–192, 1995. 65, 66

[80] David Harvey and Joris van der Hoeven. Integer multiplication in time O(n log n). *Annals of Mathematics*, 193(2):563–617, 2021. 63, 64

[81] Viet Tung Hoang, Ben Morris, and Phillip Rogaway. An enciphering scheme based on a card shuffle. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, pages 1–13, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. 48, 49

[82] Charlotte Hoffmann, Pavel Hubáček, Chethan Kamath, and Tomáš Krňák. (verifiable) delay functions from lucas sequences. In Guy Rothblum and Hoeteck Wee, editors, *Theory of Cryptography*, pages 336–362, Cham, 2023. Springer Nature Switzerland. 8, 10

[83] Kaibin Huang and Raylin Tso. A commutative encryption scheme based on elgamal encryption. In *2012 International Conference on Information Security and Intelligent Control*, pages 156–159, 2012. 28, 83, 84, 85, 88

[84] Kaibin Huang, Raylin Tso, and Yu-Chi Chen. One-time-commutative public key encryption. In *2017 Computing Conference*, pages 814–818, 2017. 28

[85] Markus Jakobsson. Flash mixing. In *Proceedings of the Eighteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '99, page 83–89, New York, NY, USA, 1999. Association for Computing Machinery. 84

[86] Y. I. Jerschow and M. Mauve. Offline Submission with RSA Time-Lock Puzzles. In *2010 10th IEEE International Conference on Computer and Information Technology*, pages 1058–1064, 2010. 8, 10

[87] Yves Igor Jerschow and Martin Mauve. Modular square root puzzles: Design of non-parallelizable and non-interactive client puzzles. *Computers & Security*, 35:25–36, 2013. 8, 10

[88] Katz Jonathan and Lindell Yehuda. *Introduction to Modern Cryptography.*, volume Second edition of *Chapman & Hall/CRC Cryptography and Network Security*. Chapman and Hall/CRC, 2014. 85

[89] Ahmet Kakacak, Aydin Emre Guzel, Ozan Cihangir, Sezer Gören, and H. Fatih Ugurdag. Fast multiplier generator for FPGAs with LUT based

partial product generation and column/row compression. *Integration*, 57:147–157, 2017. 69

[90] 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*, pages 390–413, Cham, 2020. Springer International Publishing. 3, 8, 9, 38, 39, 58

[91] Insoo Khill, Jiseon Kim, Ingoo Han, and Jaecheol Ryou. Multi-party Fair Exchange Protocol Using Ring Architecture Model. *Computers & Security*, 20(5):422–439, 2001. 19

[92] Dmitry Khovratovich, Mary Maller, and Pratyush Ranjan Tiwari. MinRoot: Candidate sequential function for ethereum VDF. Cryptology ePrint Archive, Paper 2022/1626, 2022. https://eprint.iacr.org/2022/1626. 8, 11, 68, 75

[93] Handan Kılınç and Alptekin Küpçü. Optimally Efficient Multi-Party Fair Exchange and Fair Secure Multi-Party Computation. In *Topics in Cryptology — CT-RSA 2015*, pages 330–349. Springer International Publishing, 2015. 19

[94] Russell W. F. Lai and Giulio Malavolta. Lattice-based timed cryptography. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023*, pages 782–804, Cham, 2023. Springer Nature Switzerland. 8, 13

[95] Arjen K. Lenstra and Benjamin Wesolowski. A random zoo: sloth, unicorn, and trx. Cryptology ePrint Archive, Paper 2015/366, 2015. https://eprint.iacr.org/2015/366. 8, 11, 42, 43, 44

[96] Gaëtan Leurent, Bart Mennink, Krzysztof Pietrzak, Vincent Rijmen, Alex Biryukov, and et al. Annalysis of minroot: Public report. 2023. https://inria.hal.science/hal-04320126. 76

[97] Y. Liu and H. Hu. An improved protocol for optimistic multi-party fair exchange. In *Proceedings of 2011 International Conference on Electronic & Mechanical Engineering and Information Technology*, volume 9, pages 4864–4867, 2011. 19

[98] Yi Liu, Qi Wang, and Siu-Ming Yiu. Towards practical homomorphic time-lock puzzles: Applicability and verifiability. In Vijayalakshmi Atluri, Roberto Di Pietro, Christian D. Jensen, and Weizhi Meng, editors, *Computer Security – ESORICS 2022*, pages 424–443, Cham, 2022. Springer International Publishing. 8

[99] Angelique Loe, Liam Medley, Christian O'Connell, and Elizabeth A Quaglia. Applications of timed-release encryption with implicit authentication. In *International Conference on Cryptology in Africa*, pages 490–515. Springer, 2023. 8, 9

[100] 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*, pages 244–264, Cham, 2022. Springer International Publishing. 8, 15, 53

[101] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *Proceedings of the Forty-Fourth Annual ACM Symposium on Theory of Computing*, STOC '12, page 1219–1234, New York, NY, USA, 2012. Association for Computing Machinery. 83

[102] Anna Lysyanskaya and Aaron Segal. Rational secret sharing with side information in point-to-point networks via time-delayed encryption. Cryptology ePrint Archive, Paper 2010/540, 2010. https://eprint.iacr.org/2010/540. 8, 13

[103] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, pages 1–23, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. 91

[104] Ivo Maffei and A. W. Roscoe. Delay encryption by cubing, 2022. https://arxiv.org/abs/2205.05594. 5

[105] Ivo Maffei and A. W. Roscoe. Optimally-fair multi-party exchange without trusted parties. In Gene Tsudik, Mauro Conti, Kaitai Liang, and Georgios Smaragdakis, editors, *Computer Security – ESORICS 2023*, pages 313–333, Cham, 2024. Springer Nature Switzerland. 5

[106] Ivo Maffei and Andrew W. Roscoe. Optimally-fair exchange of secrets via delay encryption and commutative blinding. In Foteini Baldimtsi and Christian Cachin, editors, *Financial Cryptography and Data Security*, pages 94–111, Cham, 2024. Springer Nature Switzerland. 5

[107] Mohammad Mahmoody, Tal Moran, and Salil Vadhan. Time-Lock Puzzles in the Random Oracle Model. In *Advances in Cryptology – CRYPTO 2011*, pages 39–50. Springer Berlin Heidelberg, 2011. 2, 8, 12

[108] Mohammad Mahmoody, Tal Moran, and Salil Vadhan. Publicly verifiable proofs of sequential work. In *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science*, ITCS '13, pages 373–388, New York, NY, USA, 2013. Association for Computing Machinery. 8, 12

[109] Giulio Malavolta and Sri Aravinda Krishnan Thyagarajan. Homomorphic Time-Lock Puzzles and Applications. In *Advances in Cryptology – CRYPTO 2019*, pages 620–649. Springer International Publishing, 2019. 8, 9, 23

[110] Wenbo Mao. Send Message into a Definite Future. In *Information and Communication Security*, pages 244–251. Springer Berlin Heidelberg, 1999. 8, 15

[111] Wenbo Mao. Time-Lock Puzzle with Examinable Evidence of Unlocking Time. In *Security Protocols*, pages 98–102. Springer Berlin Heidelberg, 2000. 8, 15

[112] Wenbo Mao. Timed-Release Cryptography. In *Selected Areas in Cryptography*, pages 342–357. Springer Berlin Heidelberg, 2001. 8, 9

[113] George Marsaglia. Xorshift rngs. *Journal of Statistical Software*, 8(14):1–6, 2003. 56

[114] Timothy May. Timed-Release Crypto, 1993. http://cypherpunks.venona.com/date/1993/02/msg00129.html. 1, 7

[115] Ahmet Can Mert, Erdinç Öztürk, and Erkay Savaş. Low-latency asic algorithms of modular squaring of large integers for vdf evaluation. *IEEE Transactions on Computers*, 71(1):107–120, 2022. 60, 68

[116] Alexander Andreevich Moldovyan, Dmitriy Nikolaevich Moldovyan, and Nikolay Andreevich Moldovyan. Post-quantum commutative encryption algorithm. *Comput. Sci. J. Moldova*, 27:299–317, 2019. 83

[117] Peter L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, 1985. 60

[118] Ben Morris and Phillip Rogaway. Sometimes-recurse shuffle. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, pages 311–326, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg. 48

[119] Ben Morris, Phillip Rogaway, and Till Stegers. How to encipher messages on a small domain. In Shai Halevi, editor, *Advances in Cryptology - CRYPTO 2009*, pages 286–302, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. 48

[120] Aybek Mukhamedov, Steve Kremer, and Eike Ritter. Analysis of a Multi-party Fair Exchange Protocol and Formal Proof of Correctness in the Strand

Space Model. In *Financial Cryptography and Data Security*, pages 255–269. Springer Berlin Heidelberg, 2005. 18

[121] A. D. Myasnikov and A. Ushakov. Quantum algorithm for the discrete logarithm problem for matrices over finite group rings. Cryptology ePrint Archive, Paper 2012/574, 2012. https://eprint.iacr.org/2012/574. 83

[122] Phong Q. Nguyen and Damien Stehlé. An LLL algorithm with quadratic complexity. *SIAM Journal on Computing*, 39(3):874–903, 2009. 73

[123] NIST. Recommendation for block cipher modes of operation: Methods for format-preserving encryption, 2019. https://csrc.nist.gov/publications/detail/sp/800-38g/rev-1/draft, Last accessed: 2023-11-28. 45

[124] Eduardo Ochoa-Jiménez, Luis Rivera-Zamarripa, Nareli Cruz-Cortés, and Francisco Rodríguez-Henríquez. Implementation of RSA signatures on GPU and CPU architectures. *IEEE Access*, 8:9928–9941, 2020. 70

[125] Yasuo Okabe, Naofumi Takagi, and Shuzo Yajima. Log-depth circuits for elementary functions using residue number system. *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, 74(8):31–38, 1991. 66

[126] OpenMP Architecture Review Board. OpenMP application program interface version 5.2, November 2021. https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5-2.pdf. 62

[127] OpenSSL. The Open Source Toolkit for SSL/TLS v. 3.1.4, October 2023. http://openssl.org/. 55

[128] Henning Pagnia and Felix C. Gärtner. On the impossibility of fair exchange without a trusted third party. Technical report, Darmstadt University of Technology, 1999. 2, 19, 99, 137

[129] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In David Wagner, editor, *Advances in Cryptology – CRYPTO 2008*, pages 554–571, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. 127, 128, 129, 130

[130] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. Cryptology ePrint Archive, Paper 2007/348, 2019. https://eprint.iacr.org/2007/348. 130, 132

[131] Colin Percival. Stronger key derivation via sequential memory-hard functions. 2009. https://sites.cs.ucsb.edu/~rich/class/old.cs290/papers/scrypt.pdf. 8, 54, 76

[132] Josef Pieprzyk and Eiji Okamoto. Verifiable secret sharing and time capsules. In *Information Security and Cryptology - ICISC'99*, pages 169–183. Springer Berlin Heidelberg, 1999. 8, 11

[133] Krzysztof Pietrzak. Simple verifiable delay functions. In *10th innovations in theoretical computer science conference (itcs 2019)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2019. 8, 10

[134] S. Pohlig and M. Hellman. An improved algorithm for computing logarithms overgf(p)and its cryptographic significance (corresp.). *IEEE Transactions on Information Theory*, 24(1):106–110, 1978. 28

[135] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '05, pages 84–93, New York, NY, USA, 2005. Association for Computing Machinery. 91

[136] Thomas Ristenpart and Scott Yilek. The mix-and-cut shuffle: Small-domain encryption secure against n queries. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013*, pages 392–409, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. 48

[137] R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock Puzzles and Timed-release Crypto. Report, Massachusetts Institute of Technology, 1996. 1, 7, 8, 9, 52, 75

[138] A. W. Roscoe. Detecting failed attacks on human-interactive security protocols. In Jonathan Anderson, Vashek Matyáš, Bruce Christianson, and Frank Stajano, editors, *Security Protocols XXIV*, pages 181–197, Cham, 2017. Springer International Publishing. 8, 11

[139] A. W. Roscoe and Peter Y. A. Ryan. Auditable PAKEs: Approaching fair exchange without a TTP. In Frank Stajano, Jonathan Anderson, Bruce Christianson, and Vashek Matyáš, editors, *Security Protocols XXV*, pages 278–297, Cham, 2017. Springer International Publishing. 2, 20, 21, 99, 101

[140] L. Rotem and G. Segev. Generically speeding-up repeated squaring is equivalent to factoring: Sharp thresholds for all generic-ring delay functions. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 12172 LNCS, pages 481–509. 2020. 3, 9, 58

[141] 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*, pages 155–180, Cham, 2020. Springer International Publishing. 9

[142] Adi Shamir, Ronald L Rivest, and Leonard M Adleman. Mental poker. In *The mathematical gardner*, pages 37–43. Springer, 1981. 28, 78, 106

[143] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997. 54, 72

[144] Victor Shoup. *A computational introduction to number theory and algebra*. Cambridge university press, 2009. Version 2. 36

[145] Cambridge Philosophical Society. and David Bohr, Niels Henrik. *Proceedings - Cambridge Philosophical Society Mathematical and physical sciences*, volume v.17-18 (1913-1916). Cambridge [etc.], Cambridge Philosophical Society [etc.], 1913. https://www.biodiversitylibrary.org/bibliography/39008. 37

[146] Shravan Srinivasan, Julian Loss, Giulio Malavolta, Kartik Nayak, Charalampos Papamanthou, and Sri AravindaKrishnan Thyagarajan. Transparent batchable time-lock puzzles and applications to byzantine consensus. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *Public-Key Cryptography – PKC 2023*, pages 554–584, Cham, 2023. Springer Nature Switzerland. 8

[147] Stefan Steinerberger. Some remarks on the Erdős distinct subset sums problem, 2023. https://arxiv.org/abs/2208.12182. 15

[148] Erich Strohmaier, Jack Dongarra, Horst Simon, Martin Meuer, and Hans Meuer. November 2023 | top500, 2023. https://www.top500.org/lists/top500/2023/11/, Last accessed: 2023-11-28. 67

[149] Souvik Sur. Single squaring verifiable delay function from time-lock puzzle in the group of known order, 2023. https://arxiv.org/abs/2211.08162. 8, 11

[150] Sri Aravinda Krishnan Thyagarajan, Guilhem Castagnos, Fabian Laguillaumie, and Giulio Malavolta. Efficient cca timed commitments in class groups. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, page 2663–2684, New York, NY, USA, 2021. Association for Computing Machinery. 8, 10

[151] Torbjörn Granlund *et al.* The GNU Multiple Precision Arithmetic Library 6.3.0, July 2023. https://gmplib.org. 55

[152] Suratose Tritilanunt, Colin Boyd, Ernest Foo, and Juan Manuel González Nieto. Toward Non-parallelizable Client Puzzles. In Feng Bao, San Ling, Tatsuaki Okamoto, Huaxiong Wang, and Chaoping Xing, editors, *Cryptology and*

Network Security, pages 247–264, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. 8, 14

[153] Yiannis Tsiounis and Moti Yung. On the security of elgamal based encryption. In Hideki Imai and Yuliang Zheng, editors, *Public Key Cryptography*, pages 117–134, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg. 85

[154] Dominique Unruh. Revocable Quantum Timed-Release Encryption. Advances in Cryptology – EUROCRYPT 2014, pages 129–146. Springer Berlin Heidelberg, 2014. 8, 43

[155] Joachim von zur Gathen and Igor E. Shparlinski. Generating safe primes. *Journal of Mathematical Cryptology*, 7(4):333–365, 2013. 36

[156] Ingo Wegener. *The complexity of Boolean functions*. John Wiley & Sons, Inc., 1987. 63

[157] Benjamin Wesolowski. Efficient verifiable delay functions. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019*, pages 379–407, Cham, 2019. Springer International Publishing. 8, 10

[158] Benjamin Wesolowski and Ryan Williams. Lower bounds for the depth of modular squaring. Cryptology ePrint Archive, Paper 2020/1461, 2020. https://eprint.iacr.org/2020/1461. 64, 68

[159] Wikipedia contributors. Transistor count — Wikipedia, the free encyclopedia, 2023. https://en.wikipedia.org/wiki/Transistor_count, Last accessed: 2023-11-28. 67

[160] Mark A. Will and Ryan K. L. Ko. Computing mod with a variable lookup table. In Peter Mueller, Sabu M. Thampi, Md Zakirul Alam Bhuiyan, Ryan Ko, Robin Doss, and Jose M. Alcaraz Calero, editors, *Security in Computing and Communications*, pages 3–17, Singapore, 2016. Springer Singapore. 61

[161] Kai Zhi Woo, Poh Kit Chong, and Lee Kong Chian. Implementation of parallel multiplications on FPGA. In *2015 IEEE 6th Control and System Graduate Research Colloquium (ICSGRC)*, pages 32–37, 2015. 69

[162] Ma Yanlong. Cryptanalysis of the cryptosystems based on the generalized hidden discrete logarithm problem. Cryptology ePrint Archive, Paper 2021/1701, 2021. https://eprint.iacr.org/2021/1701. 83

[163] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pages 162–167, 1986. 14, 126

[164] Ahmed Zawia and M. Anwar Hasan. A new class of trapdoor verifiable delay functions. In Guy-Vincent Jourdan, Laurent Mounier, Carlisle Adams, Florence Sèdes, and Joaquin Garcia-Alfaro, editors, *Foundations and Practice of Security*, pages 71–87, Cham, 2023. Springer Nature Switzerland. 8

[165] N. Zhang, Q. Shi, and M. Merabti. A Flexible Approach to Secure and Fair Document Exchange. *The Computer Journal*, 42(7):569–581, 1999. 19

[166] Chuan Zhao, Shengnan Zhao, Minghao Zhao, Zhenxiang Chen, Chong-Zhi Gao, Hongwei Li, and Yu an Tan. Secure multi-party computation: Theory, practice and applications. *Information Sciences*, 476:357–372, 2019. 106

[167] Erdinç Öztürk. Modular multiplication algorithm suitable for low-latency circuit implementations. Cryptology ePrint Archive, Paper 2019/826, 2019. https://eprint.iacr.org/2019/826. 60, 61, 67, 68

A

OpenFHE benchmark

```
#include "openfhe.h"
#include <cstdlib> // rand
#include <string> // stoi
#include <chrono>
#include <cmath> // sqrt

using namespace lbcrypto;

int main(int argc, char* argv[]) {

    // SETUP the OpenFHE context and related variables
    const unsigned long modulusSize = (1L<<16) +1;
    const unsigned int nblocks = 16; // 256 / 16
    Plaintext plaintext, rerand, result;
    Ciphertext<DCRTPoly> blind1, cipherrand, randblind1;

    CCParams<CryptoContextBFVRNS> parameters;
    parameters.SetPlaintextModulus(modulusSize); // each plaintext
       ↪  has 16 bits; so we need 16 of theese

    CryptoContext<DCRTPoly> context=GenCryptoContext(parameters);
    // Enable encryption and addidtion
    context->Enable(PKE);
    context->Enable(LEVELEDSHE);

    // Generate a public/private key pair
    auto keyPair = context->KeyGen();

    // to re-randomise we add a zero message
    std::vector<int64_t> message (nblocks, 0);
    rerand = context->MakePackedPlaintext(message); // re-rand is
       ↪ adding a zero message

    // PARSE INPUTS: i.e. number of iterations for our timer
    unsigned int niters = 100;
    if (argc > 1) { // the first argument is the program path,
```

```
    ↪ then user-defined arguments
    niters = std::stoi(argv[1]);
}

// NOW THE TESTING
// we want mean and std, so we keep track of sum and sum of
    ↪ squares
uint64_t enctime = 0, randtime = 0, dectime = 0;
uint64_t enctimesq = 0, randtimesq = 0, dectimesq = 0;
uint64_t elapsed = 0;

for (unsigned int iter=0; iter < niters; ++iter) {
    std::chrono::time_point<std::chrono::high_resolution_clock
        ↪ > start, end;

    // now create a random message
    for (unsigned int i=0; i<nblocks; ++i) {
        message[i] = rand() >> 16; // rand generates a 32 bit
            ↪ integer; so we remove the lower 16 bits
    }

    // ENCRYPTION
    start = std::chrono::high_resolution_clock::now();

    plaintext = context->MakePackedPlaintext(message);
    blind1 = context->Encrypt(keyPair.publicKey, plaintext);

    end = std::chrono::high_resolution_clock::now();
    elapsed = std::chrono::duration_cast<std::chrono::
        ↪ microseconds>(end - start).count();
    enctime += elapsed;
    enctimesq += (elapsed * elapsed);

    // RE-RANDOMISATION
    start = std::chrono::high_resolution_clock::now();

    cipherrand = context->Encrypt(keyPair.publicKey, rerand);
    randblind1 = context->EvalAdd(blind1, cipherrand);

    end = std::chrono::high_resolution_clock::now();
    elapsed = std::chrono::duration_cast<std::chrono::
        ↪ microseconds>(end - start).count();
    randtime += elapsed;
    randtimesq += (elapsed*elapsed);

    // Decrypt the re-randomised cipher
    start = std::chrono::high_resolution_clock::now();
    context->Decrypt(keyPair.secretKey, randblind1, &result);

    end = std::chrono::high_resolution_clock::now();
    elapsed = std::chrono::duration_cast<std::chrono::
        ↪ microseconds>(end - start).count();
    dectime += elapsed;
    dectimesq += (elapsed*elapsed);
```

```
        // CHECK that enc-dec worked correctly
        auto rawres = result->GetPackedValue();
        int ok = 1;
        for (int i=0; i<nblocks; ++i) {
            ok = rawres[i] == message[i];
            if (!ok) break;
        }
        if (!ok){
            std::cout << "Error!!!!!!!" << std::endl;
            std::cout << "Message:           " << plaintext << std
               ↪ ::endl;
            std::cout << "Decrypted message: " << result << std::
               ↪ endl;
        }

    }

    std::cout << "Measured times:\n";
    std::cout << "encryption: " << enctime / (double) niters << "
       ↪ micros (" << sqrt( (enctimesq / (double)niters) - (
       ↪ enctime*enctime / (double)(niters*niters)) ) <<")micros\
       ↪ n";
    std::cout << "re-rand: " << randtime / (double) niters << "
       ↪ micros (" << sqrt( (randtimesq / (double)niters) - (
       ↪ randtime*randtime / (double)(niters*niters)) ) <<")
       ↪ micros\n";
    std::cout << "decryption: " << dectime / (double) niters << "
       ↪ micros (" << sqrt( (dectimesq / (double)niters) - (
       ↪ dectime*dectime / (double)(niters*niters)) ) <<")micros"
       ↪  << std::endl;

    return 0;
}
```

Sample output of the program.

```
Measured times:
encryption: 3793.28micros (209.058)micros
re-rand: 3624.74micros (260.985)micros
decryption: 516.71micros (65.026)micros
```

B

CryptoVerif proof of the 2-party protocol

To model our protocol, we represented each party's list of secrets as a single entity. Consequently, we defined commutative encryption as a function operating on lists. Since Alice's list is longer than Bob's, we had to define two distinct encryption schemes. We also needed to specify all the shuffling functions used in the protocol. To matain accuracy, we defined equivalent shuffling functions for the random nonces' seeds, mirroring the shuffling of the nonces themselves.

In our model, we disregarded the delayed message, assuming it remains unopened. Therefore, its content should be indistinguishable from random data.

We tasked CryptoVerif with verifying whether Alice's (respectively Bob's) view of the entire protocol (including the exchange phase) is independent of Bob's (respectively Alice's) choice of values b_B and d_B (respectively b_A and d_A). In other words, we asked CryptoVerif to determine whether Alice's view would be influenced by Bob's choice of permutations (or vice-versa).

The script below contains the description of the model and the instructions CryptoVerif needs to verify these statements. As the script can only verify one statement at a time, the code below is specifically for checking Bob's view. The commented section contains the code to instruct CryptoVerif to check Alice's view.

```
(* fair exchange setup where we used a list of keys as a single
   ↪ variable *)

type key [fixed].
type blindkey [fixed].
type keylistA [fixed].
type keylistB [fixed].
type smallNum [fixed, size16]. (*small number for list shift; must
   ↪  be in 16 bits *)
type enc_seed [fixed].

proba PencA.
proba PencB.

(* Shared-key encryption (CPA Stream cipher) *)
expand IND_CPA_sym_enc_all_args(blindkey, keylistA, keylistA,
   ↪ enc_seed, mapA_enc, mapA_enc_r, mapA_enc_r', mapA_dec,
   ↪ injbotA, mapA_Z, PencA).
expand IND_CPA_sym_enc_all_args(blindkey, keylistB, keylistB,
   ↪ enc_seed, mapB_enc, mapB_enc_r, mapB_enc_r', mapB_dec,
   ↪ injbotB, mapB_Z, PencB).

(* modular addition *)
fun modAdd(smallNum, smallNum) : smallNum.
fun modNegative(smallNum) : smallNum. (* negate a number; i.e. x
   ↪ -> -x *)

(*  sigma^E for rotations on keylistA*)
fun sigmaE(keylistA, smallNum) : keylistA.
fun sigmaE_r(enc_seed , smallNum) : enc_seed.

(*  sigma for rotation on keylistB*)
fun sigma(keylistB, smallNum) : keylistB.
fun sigma_r(enc_seed, smallNum) : enc_seed.

(* tau on keylistB to reflect the whole list*)
fun tauB(keylistB, bool) : keylistB.
fun tauB_r(enc_seed, bool) : enc_seed.

(* Taua on keylistA to reflect the whole list*)
fun tauA(keylistA, bool) : keylistA.
fun tauA_r(enc_seed, bool) : enc_seed.

(* pi on keylistA to move the last element to the front*)
fun pi(keylistA, bool) : keylistA.
fun pi_r(enc_seed, bool) : enc_seed.

(* EQUATIONS AND OTHER PROPERTIES *)

equation forall kl: keylistA, bk: blindkey, r: enc_seed; mapA_Z(
   ↪ mapA_enc_r(kl, bk, r)) = mapA_Z(kl).
equation forall kl: keylistB, bk: blindkey, r: enc_seed; mapB_Z(
   ↪ mapB_enc_r(kl, bk, r)) = mapB_Z(kl).

equiv(ind_cpa(mapA_dec))
```

```
        O(kl: keylistA, bk1: blindkey, bk2: blindkey, r1:enc_seed,
          ↪  r2: enc_seed) := return(mapA_dec(mapA_enc_r(
          ↪ mapA_enc_r(kl, bk1, r1), bk2, r2), bk1)) [all]
        <=(0)=>
        O(kl: keylistA, bk1: blindkey, bk2: blindkey, r1:enc_seed,
          ↪  r2: enc_seed) := r <-R enc_seed; return(injbotA(
          ↪ mapA_enc_r(kl, bk2, r))).

equiv(ind_cpa(mapB_dec))
        O(kl: keylistB, bk1: blindkey, bk2: blindkey, r1:enc_seed,
          ↪  r2: enc_seed) := return(mapB_dec(mapB_enc_r(
          ↪ mapB_enc_r(kl, bk1, r1), bk2, r2), bk1)) [all]
        <=(0)=>
        O(kl: keylistB, bk1: blindkey, bk2: blindkey, r1:enc_seed,
          ↪  r2: enc_seed) := r <-R enc_seed; return(injbotB(
          ↪ mapB_enc_r(kl, bk2, r))).

equation forall kl: keylistA, bk: blindkey, r: enc_seed, s:
   ↪ smallNum; sigmaE(mapA_enc_r(kl, bk, r), s) = mapA_enc_r(
   ↪ sigmaE(kl, s), bk, sigmaE_r(r, s)).
equation forall kl: keylistB, bk: blindkey, r: enc_seed, s:
   ↪ smallNum; sigma(mapB_enc_r(kl, bk, r), s) = mapB_enc_r(sigma
   ↪ (kl, s), bk, sigma_r(r, s)).
equation forall kl: keylistB, bk: blindkey, r: enc_seed, b: bool;
   ↪ tauB(mapB_enc_r(kl, bk, r), b) = mapB_enc_r(tauB(kl, b), bk,
   ↪  tauB_r(r, b)).
equation forall kl: keylistA, bk: blindkey, r: enc_seed, b: bool;
   ↪ tauA(mapA_enc_r(kl, bk, r), b) = mapA_enc_r(tauA(kl, b), bk,
   ↪  tauA_r(r, b)).
equation forall kl: keylistA, bk: blindkey, r: enc_seed, b: bool;
   ↪ pi(mapA_enc_r(kl, bk, r), b) = mapA_enc_r(pi(kl, b), bk,
   ↪ pi_r(r, b)).

equation forall kl: keylistA, bk: blindkey, r: enc_seed, s:
   ↪ smallNum; sigmaE(mapA_enc_r'(kl, bk, r), s) = mapA_enc_r'(
   ↪ sigmaE(kl, s), bk,  sigmaE_r(r, s)).
equation forall kl: keylistB, bk: blindkey, r: enc_seed, s:
   ↪ smallNum; sigma(mapB_enc_r'(kl, bk, r), s) = mapB_enc_r'(
   ↪ sigma(kl, s), bk, sigma_r(r, s)).
equation forall kl: keylistB, bk: blindkey, r: enc_seed, b: bool;
   ↪ tauB(mapB_enc_r'(kl, bk, r), b) = mapB_enc_r'(tauB(kl, b),
   ↪ bk, tauB_r(r, b)).
equation forall kl: keylistA, bk: blindkey, r: enc_seed, b: bool;
   ↪ tauA(mapA_enc_r'(kl, bk, r), b) = mapA_enc_r'(tauA(kl, b),
   ↪ bk, tauA_r(r, b)).
equation forall kl: keylistA, bk: blindkey, r: enc_seed, b: bool;
   ↪ pi(mapA_enc_r'(kl, bk, r), b) = mapA_enc_r'(pi(kl, b), bk,
   ↪ pi_r(r, b)).

equiv(ind_perm(sigmaE))
        O(s: smallNum) := kl <-R keylistA; return(sigmaE(kl, s))
        <=(0)=>
        O(s: smallNum) := kl <-R keylistA; return(kl).

equiv(ind_perm(sigma))
```

```
        O(s: smallNum) := kl <-R keylistB; return(sigma(kl, s))
        <=(0)=>
        O(s: smallNum) := kl <-R keylistB; return(kl).

equiv(ind_perm(tauA))
        O(b: bool) := kl <-R keylistA; return(tauA(kl, b))
        <=(0)=>
        O(b: bool) := kl <-R keylistA; return(kl).

equiv(ind_perm(tauB))
        O(b: bool) := kl <-R keylistB; return(tauB(kl, b))
        <=(0)=>
        O(b: bool) := kl <-R keylistB; return(kl).

equiv(ind_perm(pi))
        O(b: bool) := kl <-R keylistA; return(pi(kl, b))
        <=(0)=>
        O(b: bool) := kl <-R keylistA; return(kl).

equiv(ind_perm(sigmaE_r))
        O(s: smallNum) := r <-R enc_seed; return(sigmaE_r(r, s))
        <=(0)=>
        O(s: smallNum) := r <-R enc_seed; return(r).

equiv(ind_perm(sigma_r))
        O(s: smallNum) := r <-R enc_seed; return(sigma_r(r, s))
        <=(0)=>
        O(s: smallNum) := r <-R enc_seed; return(r).

equiv(ind_perm(tauA_r))
        O(b: bool) := r <-R enc_seed; return(tauA_r(r, b))
        <=(0)=>
        O(b: bool) := r <-R enc_seed; return(r).

equiv(ind_perm(tauB_r))
        O(b: bool) := r <-R enc_seed; return(tauB_r(r, b))
        <=(0)=>
        O(b: bool) := r <-R enc_seed; return(r).

equiv(ind_perm(pi_r))
        O(b: bool) := r <-R enc_seed; return(pi_r(r, b))
        <=(0)=>
        O(b: bool) := r <-R enc_seed; return(r).

const zero: smallNum.
equation forall x: smallNum; modNegative(modNegative(x)) = x.
equation builtin commut_group(modAdd, modNegative, zero).

(* cannot distinguish between random number and adding a random
   ↪ number to your choice *)
equiv(modAddDist)
        y <-R smallNum; Oadd(x: smallNum) := return(modAdd(x,y))
        <=(0)=>
        y <-R smallNum; Oadd(x: smallNum) := return(y).
```

```
equation forall kl: keylistA; sigmaE(kl, zero) = kl.
equation forall kl: keylistA, s1: smallNum, s2: smallNum; sigmaE(
   ↪ sigmaE(kl, s1), s2) = sigmaE(kl, modAdd(s1, s2)).
equation forall kl: keylistB; sigma(kl, zero) = kl.
equation forall kl: keylistB, s1: smallNum, s2: smallNum; sigma(
   ↪ sigma(kl, s1), s2) = sigma(kl, modAdd(s1, s2)).
equation forall kl: keylistB; tauB(kl, false) = kl.
equation forall kl: keylistB; tauB(tauB(kl, true), true) = kl.
equation forall kl: keylistA; tauA(kl, false) = kl.
equation forall kl: keylistA; tauA(tauA(kl, true), true) = kl.
equation forall kl: keylistA; pi(kl, false) = kl.

const zerolistA: keylistA. (* zero list *)
const zerolistB: keylistB. (* zero list *)
equation forall kl: keylistA, s: smallNum; mapA_Z(sigmaE(kl, s)) =
   ↪  mapA_Z(kl).
equation forall kl: keylistB, s: smallNum; mapB_Z(sigma(kl, s)) =
   ↪ mapB_Z(kl).
equation forall kl: keylistB, b: bool; mapB_Z(tauB(kl, b)) =
   ↪ mapB_Z(kl).
equation forall kl: keylistA, b: bool; mapA_Z(tauA(kl, b)) =
   ↪ mapA_Z(kl).
equation forall kl: keylistA, b: bool; mapA_Z(pi(kl, b)) = mapA_Z(
   ↪ kl).
equation forall kl: keylistA; mapA_Z(kl) = zerolistA.
equation forall kl: keylistB; mapB_Z(kl) = zerolistB.
equation forall s: smallNum; sigmaE(zerolistA, s) = zerolistA.
equation forall s: smallNum; sigma(zerolistB, s) = zerolistB.
equation forall b: bool; tauA(zerolistA, b) = zerolistA.
equation forall b: bool; tauB(zerolistB, b) = zerolistB.
equation forall b: bool; pi(zerolistA, b) = zerolistA.

query secret bA.
query secret dA.

proof {
      auto;
      replace 26 "pi(sigmaE(cA, dB), bB)";
      auto}

process
        Bview(klB :keylistB, bkB1 :blindkey, bkB2 :blindkey, bB :
           ↪ bool, dB :smallNum, r1: enc_seed, r2: enc_seed) :=

        (* A's parameters *)
        klA <-R keylistA;
        bkA1 <-R blindkey;
        bkA2 <-R blindkey;
        bA <-R bool;
        dA <-R smallNum;

        (* protocol's messages*)
        cA <- sigmaE(mapA_enc(klA, bkA1), dA);
        cB <- sigma(mapB_enc_r(klB, bkB1, r1), dB);
        d <- pi( sigmaE(mapA_enc_r(cA, bkB2, r2), dB), bB );
```

```
        eB <- tauB( sigma( mapB_enc(cB, bkA2), dA), bA );
        eA <- tauA(d, bA);
        mA <- mapA_dec(eA, bkA1);
        return(cA, cB, d, eB, mA)
(* Above we show that B cannot distinguish between different
   ↪ permutations chosen by A*)

(*
query secret bB.
query secret dB.

proof {
      auto;
      replace 32 "tauB(sigma(cB, dA), bA)";
      auto}

process
        Aview(klA :keylistA, bkA1 :blindkey, bkA2 :blindkey, bA :
           ↪ bool, dA :smallNum, r1: enc_seed, r2: enc_seed) :=

        (* B's parameters *)
        klB <-R keylistB;
        bkB1 <-R blindkey;
        bkB2 <-R blindkey;
        bB <-R bool;
        dB <-R smallNum;

        (* protocol's messages*)
        cA <- sigmaE(mapA_enc_r(klA, bkA1, r1), dA);
        cB <- sigma(mapB_enc(klB, bkB1), dB);
        d <- pi( sigmaE(mapA_enc(cA, bkB2), dB), bB );
        eB <- tauB( sigma( mapB_enc_r(cB, bkA2, r2), dA), bA );
        mB <- mapB_dec(eB, bkB1);
        return(cA, cB, d, eB, mB)
*)
```