A Good Use of Time: Techniques and Applications of Delay-Based Cryptography

Liam Medley

Information Security Group

Royal Holloway, University of London

This dissertation is submitted for the degree of

Doctor of Philosophy

October 2023

Declaration

These doctoral studies were conducted under the supervision of Dr. Elizabeth A. Quaglia.

The work presented in this thesis is the result of original research I conducted, in collaboration with others, whilst enrolled in the School of Electrical Engineering, Physical and Mathematical Sciences as a candidate for the degree of Doctor of Philosophy. This work has not been submitted for any other degree or award in any other university or educational establishment.

Liam Medley

October 2023

Acknowledgements

I would first like thank my supervisor Liz, whose support, guidance and positivity made my PhD highly enjoyable, and led to me to pursue further roles in academia. I have been very lucky to have you as my supervisor.

I would like to thank my principal co-author Angelique: her endless enthusiasm and drive made all of our work together rewarding and enjoyable.

I am grateful to Myrto, for facilitating my visit to Edinburgh, and to Thomas for patiently teaching me the joys of Universal Composability.

I would like to thank Carlos for the early supervision, and all of the entertaining discussions.

I would like to acknowledge those who supported me throughout my PhD journey.

To the friends who made Egham an enjoyable place to live: Erin, Simon, Lenka, Keele, Joe, Jeroen, Balázs, Eamonn, Pesh, Nick, Rob, Amy, and of course Jodie.

To those who were supportive outside of Royal Holloway: Michael, Sophie, Tom, Dan, Harry, and my family.

Abstract

The field of delay-based cryptography arose from the notion of encrypting a message "to the future", as first proposed by May in 1993, in a post on the Cypherpunks mailing list [120]. In this post, May discussed applications including sending money into the future to avoid taxation, or sending a message 'in the event of death'. However, May's idea of how to achieve this was based on trusted agents, which is now considered impractical. Fortunately, since the publication of this post, the cited applications and the methods suggested to achieve this have been extensively augmented and modernised. Additionally, the concept of encrypting a message for a set length of time has become relevant across many areas of cryptography and often acts as a tool to enable stronger notions of security and privacy.

The first formal treatment of this subject, which utilises a computational delay, was provided in 1996 by Rivest, Shamir and Wagner in their seminal work "Time lock puzzles and timed-release Crypto" [136]. In this work, the authors suggested encrypting a message in such a way that decrypting the message requires computing an iterated sequential function. The underlying cryptographic concept is that this computation must take at least a certain amount of real-world clock time to compute. This assumption is based upon the fact that each iteration of the function requires the input of the previous step, and hence one cannot run all of the steps in parallel, arbitrarily speeding up the computation. This seemingly simple idea birthed the rich subject of delay-based cryptography, which has found use in many different areas of cryptography. In the modern day, a cryptographic delay is regularly applied to messages, symmetric keys, asymmetric keys, signatures, commitments, and simply

a random string.

In this thesis, we first study delay-based cryptography at a high level, providing an exposition of the definitions and applications of various cryptographic primitives, and their applications.

We then delve into three topics and their associated applications, in each case dedicating a chapter to a primitive and a construction, for which we provide game-based security definitions and security proofs.

The first subject of our attention is that of sealed-bid auctions, where we use the primitive of *timed-release encryption*. In this chapter we use classical, well-known cryptographic primitives as building blocks to produce a novel TRE scheme. Our approach utilises number theoretic properties to ensure that the outcome of computing the delay allows a solver to factor an RSA modulus, and hence decrypt all messages which have been encrypted to it.

We next turn our attention to further applications of TRE, and in particular the application of whistleblowing. We argue that some the properties inherent to timed-release encryption could be valuable in the hands of a vulnerable party such as a whistleblower, but in its current form it is not suitable for such a sensitive use case. For this reason we introduce *timed-release encryption with implicit authentication*, which offers a novel approach to using public-key encryption, namely that the encryption key which is traditionally made public is instead kept private, in order to be able to achieve practical, implementable delay-based encryption, whilst offering additional privacy properties that may aid at-risk individuals such as whistleblowers.

Finally, we discuss cryptographic randomness, and its use and applications within cryptography. In this section we utilise a primitive known as verifiable delay function, and in particular an extension known as a *continuous* verifiable delay function. One of the main themes of this chapter is the trade-off between trust and efficiency in protocols which of-

ten find application in the decentralised setting. We leverage a stronger than usual trust assumption in order to maximise the efficiency of our construction.

We conclude this thesis by looking forward to potential future research in this field, discussing ways that the work in this thesis can be built upon, highlighting gaps in the knowledge of current literature, and posing relevant open questions.

Table of contents

Chapter 1

Introduction

Contents

In this chapter we introduce the concept of delay-based cryptography, which helps to provide the motivation and context for this thesis. We then outline the structure of this thesis.

1.1 An Introduction to Computational Delay

In this thesis, we discuss the design and applications of a branch of cryptography referred to as delay-based cryptography. At its core, delay-based cryptography broadly refers to the practice of using a computational delay to mimic real world time. Our goal is to associate 'clock-time', which is to say time as we experience it in the real world, with a computational

delay. In other words, we want to be able to run a program, and state with confidence that this computation will take a certain length of real world time to execute. Generally speaking, this is a difficult thing to do. This is because in computing, one can generally add more computational power to a program, with the result that it will run more quickly. If one is to think about this statement a little more deeply, one can observe that applying more computational power can be done in one of two ways: either *faster* computation, or *more* computation.

The former involves using faster, more modern hardware, and is something that we as computer scientists can do nothing about: we cannot change the fact that the underlying hardware that various machines use in practice have different levels of power, and as such will compute identical computations at different speed. However, we can assume that there is a limit to how fast such computations can be.

Therefore, what we aim for is something a little more nuanced, but that is more realistic: We wish to state that a computational task will take no less than a given length of time to compute.

By choosing our computation carefully, we can limit the effect of increasing the quantity of computational power, by using a computation that cannot be parallelised. The key to negating the effect of additional computation is therefore to limit the number of useful things a program can do at once. In an ideal world, we would like to run a program which cannot be parallelised at all. The way we achieve this is by utilising an *iterated sequential computation*, that is to say a computation that comprises of multiple sequential steps, where each step takes as input the output of the previous step.

By requiring a recipient to compute an iterated sequential computation, a sender can ensure that such a computation finishes at some point in the future, a concept first introduced by Rivest, Shamir and Wagner in their seminal paper in 1996 [136]. In modern times, this idea of iterated sequential computation is used across multiple areas of cryptography,

encompassing primitives including time-lock puzzles [136], timed-release encryption [53], timed commitments [35] and verifiable delay functions [32]. These primitives have suggested the output of the delay be a message (such as a vote/bid) [118], an encryption key [53], a signature [150], or simply a proof that a delay has occurred [32].

1.2 Thesis Structure

In this thesis, we begin in Chapter 2 by presenting a systematisation of knowledge of delay-based primitives [121], discussing the existing literature for delay-based cryptography. We discuss the proposed approaches to constructing a delay, the primitives that use these delays, and the applications that such primitives are suited for. We discuss the advantages and disadvantages of various techniques, including topics such as efficiency, trust and practicality. After introducing some preliminary material in Chapter 3, we present three constructions, each of which is centred around a different delay-based primitive.

In Chapter 4 we present TIDE (TIme-Delayed Encryption) [110], a timed-release encryption scheme. TIDE combines the traditional public-key encryption techniques of RSA with the sequential properties of the Blum Blum Shub random number generator. This allows parties to encrypt messages to the RSA public-key, and spend a predictable length of time decrypting the private key, which in turn allows all messages to be decrypted. Such a scheme is particularly useful in the context of sealed-bid auctions.

In Chapter 5, we present "Applications of Timed-release Encryption with Implicit Authentication" [112]. In this work, we alter the standard framework of timed-release encryption; making the encryption key private rather than public, and introducing a new security property known as implicit authentication. We introduce this notion in order to apply the concepts

of timed-release encryption scheme to sensitive topics such as whistleblowing, where the control of information is critical.

In Chapter 6, we analyse the *continuous verifiable delay function* (cVDF) primitive, discussing its definitions, and its key application of randomness. We propose new, generic definitions, which are in keeping with the literature, and we show that under our definition any such cVDF yields a randomness beacon. We then present a cVDF construction, where we introduce a precomputation phase, which computes various states at which verification can occur. These states can then be computed by means of an iterated sequential computation, which we show can be publicly verified extremely efficiently.

1.3 Publications

The contents of this thesis contains adaptations of the following three publications:

1. Chapter 2 is based upon the paper *SoK:Delay-Based Cryptography*, which is a joint work with Angelique Loe and Elizabeth A. Quaglia. This paper was presented at the *36th IEEE Computer Security Foundations Symposium* in 2023.

2. Chapter 4 is based upon the paper *TIDE: A novel approach to constructing timed-release encryption*, which is a joint work with Angelique Loe, Christian O'Connell and Elizabeth A. Quaglia. This paper was presented at the *27th Australasian Conference on Information Security and Privacy* in 2022.

3. Chapter 5 is based upon the paper *Timed-Release Encryption with Implicit Authentication*, which is a joint work with Angelique Loe, Christian O'Connell and Elizabeth A. Quaglia. This paper was presented at *AFRICACRYPT 2023: 14th International Conference on Cryptology* in 2023.

1.3 Publications

My research was supported by the EPSRC and the UK government as part of the Centre for Doctoral Training in Cyber Security at Royal Holloway, University of London (EP/P009301/1).

Chapter 2

Foundations of Delay-Based Cryptography

Contents

The work in this chapter appears in [121], for which I was the lead author. In this chapter, we provide a systematisation of knowledge of delay-based cryptography, in order to provide the reader with an understanding of the modern and historical context of delay-based cryptography. We start by considering the role of time within cryptography, explaining broadly what a delay aimed to achieve at its inception and now, in the modern age. We then move on to describing the underlying assumptions used to achieve these goals, analysing topics including trust and decentralisation, along with concrete methods to implement a delay. We then survey existing primitives, discussing their security properties, instantiations and applications. We finish this section by discussing which of these primitives will feature in the later chapters of this thesis, and highlighting their applications.

2.1 Fundamentals of a Cryptographic Delay

In this section, we introduce the concept of a cryptographic delay, explaining why such a delay is useful, and highlighting some of the challenges associated with implementing such a delay.

2.1.1 Why a Delay is Useful

Increasing Fairness and Reducing trust Fairness in multi-party protocols is an important concept. In auctions, it is important that no party sees the bid of another party before they bid. Similarly, in voting it is important that parties do not learn the current tally during the vote phase. Knowing a vote tally during the voting phase can allow someone to alter their vote to 'vote tactically' in an election [88]. In a coin-flipping protocol where multiple parties each contribute some input it is vital that no party learns the other outputs before they commit to their own.

In each of the above examples the use of a trusted third party would provide a simple solution to the fairness problem. The use of a trusted third party is common in cryptography, and is often used as a security standard to compare the security of a protocol against. If all parties give the trusted party their inputs and the trusted party then computes the output and sends it to all parties then a fair solution is trivially achieved. However, giving a trusted party all inputs is a strong assumption, which leads to the natural question of who should be trusted to perform this role, and how to ensure they remain trustworthy of handling sensitive data.

By using a delay, it becomes possible to reduce the role of the trusted party (typically to generate the public parameters) or remove it altogether from such scenarios. Intuitively, this can be seen as all parties submitting their input before some deadline, with the consequence that parties can only recompute other parties' inputs after the deadline, hence achieving fairness. Indeed, a theme that we shall see throughout this thesis is the desire to the remove trust from delay-based primitives, and the repercussions this has on the efficiency of the resulting schemes.

Enabling new applications The use of delay-based primitives has allowed the order of actions in cryptographic protocols to be altered. These ordering alterations have lead to novel approaches to solving problems. This has resulted in new cryptographic security

properties and furthermore, various impossibility results have been overcome. We provide two illustrative examples of this next.

- *Overcoming an impossibility result.* In 1986, Cleve [54] proved that fair coin flipping was impossible in the standard model. It was shown that for any k round protocol one of the parties can achieve a bias of at least $1/k$. The fundamental problem with designing a fair coin flipping protocol between two parties is that one party would always learn the output first and could therefore abort if they did not favour the outcome. In 2000, Boneh and Naor [35] showed that this impossibility result could be circumvented using timed commitments, a primitive we will discuss in Section 2.2.5. Both parties use a timed-commitment to commit themselves to a hidden value, under the understanding that the combination of the two hidden values will be the randomness. In contrast to a standard commitment, using a timed-commitment means that if one party aborts rather than revealing their hidden value, then the remaining party could compute a sequential computation in order to obtain the output, and learn the coin-flip anyway.

- *A modern approach to building randomness beacons.* A randomness beacon was first proposed by Rabin in 1983 [132] to remove the need for trusted intermediary parties in protocols such as contract signing. In modern times, interest in randomness beacons has recently seen a sharp rise in interest, largely due to the advent of blockchain technology. In recent years there have been many novel approaches to constructing such a beacon to be used in applications like the generation of cryptographic parameters and designing consensus protocols in blockchain technology. In Chapter 6 we will provide an exposition into such approaches, with a focus on the approach pertinent to this work: randomness beacons based upon a cryptographic delay. These rely upon a primitive known as a verifiable delay function (VDF), which we shall see in Section 2.2.7. By utilising an iterated sequential function, VDF-based randomness beacons dispense of the need for a synchronous network, or for multiple parties to interact to compute the beacon output, which are common downsides

of alternative randomness beacons [133] Additionally, a VDF-based randomness beacon ensures that the property of *timeliness* is satisfied, i.e., each pulse of randomness will come at a regular interval. Both of these properties are desired in an ideal randomness beacon, but not achieved by the majority of current techniques [134]. We will discuss randomness beacons built from VDFs further in Section 2.4, and propose a novel VDF-based randomness beacon in Chapter 6.

Computationally efficient proof of work A topical example of the use of delay is in the decentralised space. It is well-documented that the rise in popularity of blockchain technology has come with a huge computational expense. This computational burden has a negative environmental impact [74, 151], and is viewed as unsustainable. By using techniques from delay cryptography, a proof of *sequential* work is possible as an alternative to 'classical' proof of work in the style of bitcoin [126]. When using a classical proof of work, the more computational power that is used, the faster the computation will be finished. In contrast, by using an iterated sequential function, each step requires the output of the previous step, and hence the benefit of parallelisation is limited to its application to each individual step. In modern delay functions, these steps are as small as a single hash, or a single square-and-reduction, meaning that parallelisation has little effect. This removes the incentive to spend vast amounts of computing power on a proof of work, and lays the foundation for a sustainable alternative. An example of this is in computational timestamping [6, 101, 115].

2.1.2 Techniques to Achieve a Delay

In order to achieve a delay, one must somehow relate real-world 'clock' time to computational time. The way this is done in practice is by assuming that there exists some computational step that takes a minimum amount of time to compute, regardless of the amount of computation

power, and then sequentially iterating this step. Sequentiality is therefore at the heart of delay-based cryptography.

Definitions of Sequentiality There are various approaches to defining what it means for a function to be sequential. The intuition of such a definition should capture that no party is able to compute t iterations of the function in less than a given amount of 'real-world time', for some time-parameter t. In order to formalise this intuition, however, one must choose a computation model. The two most common such models are to consider time as either an arithmetic circuit of depth t, without a bound on the breadth of the circuit; or as t steps of a parallel Turing machine. For the remainder of this section, we do not make explicit any model of computation in order to keep our discussion generic. However, in later chapters of this thesis, we will be using the latter model. In practice, primitives are often modelled assuming the adversary has an additional computational advantage to account for potentially faster machinery. However, this intuition does not translate easily into all cryptographic frameworks.

In particular, in the universal composability (UC) model introduced by Canetti [42], this concept is difficult to capture. As such, a notion of time is instead obtained by providing parties with access to a *global clock*, which partitions time into rounds. In each round, parties can do various computations, before sending a message to the global clock indicating that they are ready for the next round. We will discuss the UC framework in Section 2.3, and for now restrict our attention to the standard model, which we shall describe as follows:

We say that a computation of an output y from an input x is *sequential* if it can be computed in t steps with overwhelming probability, and yet any adversary $\mathcal{A}$ bounded by at most $t-1$ sequential steps, utilising polynomially many parallel machines, cannot compute y from x with more than negligible probability.

To build any such construction that satisfies this, we need an *iterated sequential function* (ISF); that is a function with the same domain and codomain, which takes some minimal

time to compute, that it is iterated t times. We shall illustrate the current methods of building such iterated sequential functions next.

Approaches to building an iterated sequential function Consider a cryptographic hash function H, which maps elements from a set S into the same set S. By sampling an input $x \in S$, we can achieve an ISF trivially by computing t hashes: $y = H(H(H(\cdots H(x))))$. This follows on from standard properties of a cryptographic hash function [93]: By the second pre-image resistance and the collision resistance of H, the only way to verify the output y is to spend another t time on the same computation.

For an ISF to be practical there is a requirement to be able to verify the computation is correct in time significantly faster than t. Therefore, it is desirable to introduce some structure to enable an alternative method for computing or verifying the solution. Having this property facilitates a computational asymmetry in the amount of time taken to iteratively calculate the solution and to efficiently verify the solution. In this chapter, we will refer to this property as *inversion*, noting that a function with fast inversion will have more efficient decryption/verification.

We now explore the existing methods in the literature for computing a delay.

- *Repeated squaring.* The original and most prevalent method of computing a delay is using repeated squaring in an RSA group, as first proposed by Rivest, Shamir and Wagner in [136]. This requires a trusted party generating an RSA modulus $N = pq$, where p and q are primes, and sampling a random input x from $\mathbb{Z}_N^*$. The delay is achieved by repeatedly squaring x and reducing modulo N.

The current state of theory is that this leads to the most practical applications of many primitives, however there is always a discussion on how the modulus N is generated. First, note that if a party knows the factorisation of the modulus, then they know $\phi(N) = (p-1)(q-1)$. This allows them to dramatically speed up the computation x^{2^t} as follows: they

first compute $e = 2^t \bmod \phi(N)$, and then compute $x^e \bmod N$. This is significantly faster than simply squaring and reducing modulo N [136].

Therefore in an untrusted setting, it is desirable that no party learns the factorisation of N. This has led to extensive research into multi-party computation protocols for the trustless generation of an RSA modulus. We discuss the current state of research in Section 2.1.3.

As an alternative approach to solving this problem, it was proposed in [155] to use class groups of an imaginary quadratic field instead of an RSA modulus, as this does not require a trusted third party. However, this is a non-standard security assumption, lacking the rigorous cryptanalysis of the RSA assumption.

- *Hash-based techniques* An interesting alternative approach to computing a delay was proposed by Mahmoody et al. in [114], which relies on a series of hash and xor functions. The main drawback of this technique is that to verify the computation is correct requires the same number or hash and xor functions, making it inefficient. On the other hand, the minimal assumptions required by this scheme (e.g., no setup) make this a useful scheme for proving theoretical results, as seen in [9].

Another interesting hash-based technique for computing a delay is using a *directed acyclic graph*, a technique proposed by Mahmoody et al. in [115]. The idea here is to compute multiple chains of hash functions, to create a graph. Upon publication of this graph, parties can run a verification protocol to verify that a certain amount of time was spent computing the graph. A construction with such public verification is known as a proof of sequential work; we shall discuss this primitive in Section 2.2.6.

- *Randomised encodings* In [24], Bitansky et al. introduce the notion of *non-parallelising languages*, which are languages which require at least t time to evaluate. They then go on to show that if one assumes the existence of such languages, then one can build a time-lock puzzle based upon randomised encodings. This paper provides a fascinating study as a novel approach to building a delay, but is unfortunately theoretical rather than practical in nature.

We note that in this thesis, we shall be using the repeated squaring assumption in an RSA group.

Considerations on assumptions and the state of the art We have seen that there are many approaches to building a sequential function, however many of them rely on heavy cryptographic primitives such as indistinguishability obfuscation, which makes them effectively unusable [1] We now aim to distill the current state of those delay-based assumptions which are most practical.

Accuracy of these assumptions In each of the identified techniques the underlying assumption is that the overall computation cannot be sped up by parallelism. However, it is possible each sequential step may be parallelised to some extent, and it is also the case that a faster machine will compute each step faster than a slower machine. This means in practice an adversary is generally assumed to have a high level of parallelism, and an additional computational advantage to account for potentially faster machinery. Due to a lack of benchmarking, such advantages seem to be chosen arbitrarily, and hence when we discuss the future work in the conclusion of this thesis (Section 7.3), we call for rigorous benchmarking to provide the community a better understanding of how much of a speed-up can be achieved with dedicated hardware.

Community trust Although hard to measure empirically, it seems that exponentiation in a finite group is the most trusted delay technique used in practice. This is possibly due to its reliance on traditional cryptographic building blocks such as RSA, and the longstanding RSW assumption [136]. A recent work by Katz et al. analysed the security of time-lock puzzles in the strong algebraic group model, showing that speeding up a time-lock puzzle is at least as hard as factoring N [94]. Alternative methods such as replacing the RSA group

[1] IO is well known to be impractical at the current time, due to issues such as long evaluation time, heavy memory requirements, and nonstandard assumptions [13, 89].

with a class group lack the same level of community trust, and require significantly more research until it can be considered trusted.

The area of isogeny-based cryptography is significantly younger than RSA, and hence suffers from a similar lack of community trust. Indeed, this lack of trust seems like it may be warranted in this case: An important key exchange protocol known as SIDH (Supersingular Isogeny Diffie-Hellman), which had recently advanced to the fourth round of NIST's ongoing Post-Quantum Cryptography standardization process, was broken in a recent paper by Castryck et al [47]. This result has damaged the reputation of isogeny-based cryptography.

Therefore, it is with good reason that sequential squaring in an RSA group is the de facto method of computing a delay. However, there is always a question in such schemes of who should generate the RSA modulus. The advent of blockchain technologies has elevated the interest in trustless primitives, which has in turn led to various research into generating an RSA modulus without a trusted party, which we discuss next.

2.1.3 Trustless Generation of an RSA Modulus

For any delay-based construction that is based upon the RSW time-lock assumption, a group must be generated in order for the repeated squaring and reduction to take place. It is necessary that the party computing the delay does not know any trapdoor which will speed up the computation.

The most common approach is to use an RSA group. The trapdoor in an RSA group is the Euler phi function $\phi(N) = (p-1)(q-1)$, where $N = pq$. When an RSA group is used, N can be generated in one of two ways: In a trusted setting, a trusted party will generate an RSA modulus N, and pass it to solving parties, who will compute the delay [53, 111, 136].

The alternative is to trust a group of random, potentially anonymous parties to run the setup. In this case, one settles for an efficiency/trust trade-off. To ensure a certain standard of

efficiency, the group will need to be relatively small, but if they all collaborate they can break the security guarantees of the construction.

For this approach, an expensive multi-party computation (MPC) ceremony is required. Research into MPC has recently developed with significant performance improvements. In 2018, Frederiksen et al. [69] provided an implementation for the malicious two-party setting. Using server grade hardware connected via a 40.0 Gbps network link, they were able to achieve average runtimes of 35 seconds - this was not practically efficient, and only supports the 2-party case. Also in 2018, Hazay et al. [85] introduced a method to compute an RSA modulus using a threshold encryption scheme in the two-party setting, and they prove security against malicious attacks. They offer multiple optimisations, and in the best possible case, the average CPU time required to compute an RSA composite is 15 minutes.

In 2020, Chen et al. [49] introduced a new multi-party protocol for the distributed generation of biprime RSA moduli, which improved upon the models of Frederiksen et al. and Hazay et al. by removing security issues (such as information leakage) found in the former, and eliminated some significant security assumptions (such as the use of additive homomorphic encryption) in the latter.

In 2021, Chen et al. [50] extended this work, to produce Diogenes: the first implementation of a multi-party generation of an RSA modulus supported by thousands of parties. The per-party communication cost of Diogenes grows logarithmically in the number of parties, and their security model allows for a malicious adversary to corrupt all but one of the parties. Further, they implemented this with as many as 4000 parties. An example timing that the authors give is to generate a 2048-bit modulus among 1,000 parties, their passive protocol executed in under 6 minutes and their active variant ran in under 25 minutes. These are realistic timings, making this multi-party generation of an RSA modulus a practical approach. We view this as the state-of-the-art construction.

The downside of this construction, and indeed of approaches requiring active participants [54], is the ease with which an adversary can perform a denial-of-service attack on the generation. Denial of service can be achieved by corrupting only a single party. Whilst cheaters can be removed and the protocol re-started, an adversary corrupting a large number of parties can repeatedly perform this attack and greatly delay the generation of N. Therefore, the practicality of its use can be concretely impacted upon.

To conclude, whilst this approach has been made feasible in recent years, significant implementation challenges still exist in practice.

2.2 Delay-Based Primitives

The idea of associating clock time to computational time is the basis of many delay-based primitives.

We shall categorise such primitives into two classes: those which allow for the recomputation of an input, which include time-lock puzzles, timed-release encryption, and timed signatures; and those that do not, such as proofs of sequential work and verifiable delay functions. We shall expand upon this in the Conclusion of this thesis, explicitly in Section 7.2, we illustrate the various relationships between each class of primitive in Fig. 7.1, and suggest some potential relations between other primitives for future work.

In the following sections, we provide intuitions of the major delay-based primitives. Note that while we do not provide formal definitions here, in the subsequent chapters we will formally define the primitives we use, namely timed-release encryption and verifiable delay functions.

2.2.1 Time-lock Puzzles and Time-lock Encryption

The natural starting point of delay-based cryptography is *time-lock puzzles* (TLPs), which were introduced by Rivest et al. in 1996 [136], in what is credited as the first work to formally discuss the idea of a cryptographic delay. In a TLP, an encryptor takes as input a string s and a time parameter t, and outputs a puzzle Z. The decryptor then spends t time running a sequential computation on the puzzle Z to re-construct the string s. In the original construction of Rivest et al., the method for obtaining a time-delay is repeated squaring in an RSA group. Explicitly, the encryptor samples an RSA modulus $N = pq$, where p and q are large primes, and chooses a string s, which they suggested could be a key to a symmetric encryption scheme. The encryptor then randomly samples r and computes the puzzle $Z = s + r^{2^t} \bmod N$. The solver is then given Z and r, allowing for the computation of r^{2^t} in t sequential steps, and hence the solver can learn the string s. Note that using the trapdoor $\phi(N)$, the encryptor can construct the puzzle significantly faster than the solver can recompute it.

Whilst this is not the only method of constructing a cryptographic delay, it is certainly the most popular, and it is this construction that gave rise to many of the primitives that we shall see later in this section. Other notable methods for computing TLPs include the encoding-based construction of Bitansky et al. [24], as discussed in Section 2.1.2, and the UC-based construction of Baum et al. [16], as we shall discuss in Section 2.3. However, the construction of Rivest et al. remains the most relevant.

Whilst advances in constructions have been limited in recent years, the theory of TLPs has continued to advance. Some of the most important recent work has included the introduction of *non-malleable* TLPs [70], TLPs which protect against tampering attacks, by ensuring that a TLP cannot be 'mauled' into another, related message. This has strong applications to auctions and fair coin-flipping, by removing the need for a setup.

As discussed in Section 2.1.2, a recent work by Katz et al. [94] analysed the security of time-lock puzzles, showing that in the strong algebraic group model, speeding up a time-lock puzzle is at least as hard as factoring N.

A fundamental requirement for a TLP is that the generation of the puzzle should be significantly faster than solving the puzzle. This time-gap between generation and solving puzzles makes constructions far more practical, but rules out simpler constructions. If one can accept a linear time difference, then a relaxation of TLPs is *time-lock encryption* (TLE).

There is a lack of consistency in the definitions of TLE and TLPs, within the literature, with some sources claiming they are interchangeable [53]. We do not agree with this, and posit the following:

A timed-release encryption scheme need only be correct, secure and sequential, whilst a time-lock puzzle must additionally have the property that the time spent on puzzle generation is significantly faster than the time spent on solving the puzzle. What this means in practice is that every TLP is also a TLE, but the inverse is not true.

A pertinent example of a TLE scheme is that of Mahmoody et al., which relies on iterating a hash and xor function [114]. In this work, the authors set out to build time-lock puzzles in the random-oracle model. They in fact provide an impossibility result showing that such a time-lock puzzle is impossible due to the required time gap between puzzle generation and puzzle solving. They instead build a scheme using a repeated hash-and-xor technique, which does not qualify as a time-lock puzzle, due to the slow puzzle generation. The way to verify this puzzle is to repeatedly hash and xor t times to go from the solution to the input. Note that this is not a TLP due to the linear gap in solving and generating a puzzle.

Initially this may seem like a weaker construction than that of Rivest et al. due to the longer generation of puzzles. However, this construction does not require an RSA group to be generated, which leads to fewer assumptions, as discussed in Section 2.1.2. This is useful in proving theoretical results, especially in the decentralised setting [9].

2.2.2 Homomorphic Time-lock Puzzles

Malavolta et al. posit that the main drawback of using time-lock puzzles in applications such as auctions or voting is the need to solve many puzzles before being able to compute a function over the time-encrypted messages [118]. They propose *homomorphic time-lock puzzles* to reduce the number of puzzles that require solving to just one (for example the highest bid in an auction). A homomorphic time-lock puzzle consists of a set of puzzles with an underlying property allowing for any party to compute a function on the set of puzzles without learning any of the underlying messages. This means that only the relevant puzzle needs to be decrypted rather than all of the puzzles.

Homomorphic time-lock puzzles can be either linearly, multiplicatively, or fully homomorphic. Moving from left to right across these three types of homomorphism, the puzzles gain more functionality, but become harder to construct in an efficient manner.

The RSW time-lock assumption is the basis of the Malavolta et al. construction and it is augmented with techniques from homomorphic encryption to construct both linearly and multiplicatively homomorphic TLPs. They additionally show that fully homomorphic time-lock puzzles can be built using indistinguishability obfuscation, which is unfortunately impractical at the current time. A follow up work by Brakerski et al. in 2019 built fully homomorphic TLPs from standard assumptions [36], representing significant progress. In 2022, Liu et al. built a scheme which significantly improved the practicality of linearly and multiplicatively homomorphic TLPs [109]. Explicitly, they built a multiplicatively HTLP scheme which computes solutions over $\mathbb{Z}_N^*$, and implemented linearly homomorphic TLP schemes, showing that they run efficiently in practice, and have low storage. What this means in practice is that certain classes of functions can be computed over the set of TLPs efficiently, and others are of theoretical interest but are currently lacking an efficient construction. This is well-illustrated by the observation that voting requires linearly homomorphic TLPs, and

so can be instantiated in practice, whereas auctions rely on fully homomorphic encryption, which is not practically efficient at the current time.

2.2.3 Timed-Release Encryption

Timed-release encryption was first mentioned by May in 1993 [120], with the idea of sending a message and a release time to a trusted agent, who would transfer the message at this release time. Various constructions were proposed using such a trusted agent [48, 119], and in 2008, Cheon et al. [51] proved that the security of this concept is equivalent to that of identity-based encryption. In [129], an interesting generalisation of TRE known as time-specific encryption (TSE) was introduced by Paterson et al. In TSE, a time-server is used to enable decryption of a message within a specified time interval $[t_0, t_1]$, broadening the scope of applications.

In current times however, the use of a trusted agent is generally seen as something to be avoided. Timed-release encryption is now often seen as a combination of public-key encryption with a delay [53, 111]. On a high level, this works by making the encryption key public, and encoding the decryption key as the solution to a sequential computation.

As such, in more recent times there have been some attempts to build such timed-release encryption schemes that do not rely on a trusted server. An interesting line of research has been to use the bitcoin protocol [126] with witness encryption, where one must show the solution to a hard problem in order to decrypt a message [107, 108]. These schemes use bitcoin as the hard problem, on the basis that after a certain amount of time, the correct number of blocks will have been mined, and importantly also made public. This means that any party can then obtain a decryption key for example, without having to do a lot of work themselves. As is often the case, the main issue of such schemes is their reliance on heavy cryptographic primitives such as witness encryption that are unworkable in practice.

At ESORICS 2021, Chvojka et al. introduced the idea of taking a TLP and using its solution in the key generation of a public-key encryption (PKE) scheme [53]. They use this to define a timed-release encryption (TRE) scheme where multiple parties encrypt a message to the public key of the PKE scheme. Then upon solving the puzzle they can reconstruct the secret key and decrypt all of the messages. The authors explain how to achieve this generically using standard TLP and PKE primitives.

In Chapter 4, we shall give the formal definitions of a TRE following Chvojka et al. [53], before presenting a concrete construction of a TRE scheme, using the RSA cryptosystem and the RSW time-lock assumption as our building blocks. Then, in Chapter 5, we introduce a new variant on TRE which we term TRE with *implicit authentication*, providing security definitions and a construction of this primitive.

2.2.4 Delay Encryption

Delay Encryption, introduced by Burdges et al. in 2021 [39], is a primitive which offers a delay-based analogue to identity-based encryption. In identity-based encryption, there exists a master public and private key pair, which are used to authenticate identities and generate each parties' private key [34]. In delay encryption, there is a session ID, and a session key instead of this key pair. The session key is encoded as the solution to a sequential computation, allowing any party who runs the sequential computation to decrypt all messages posted to the session ID.

Unfortunately, the construction of DE presented in [39], which is currently the only published construction [2], comes with two significant challenges for implementation: (i) The storage requirements needed to compute the decryption key is huge - a delay of one hour

[2]There is also an unpublished construction by Loe et al. [110] which is shown to run efficiently, but relies upon a trusted setup.

requires 12 TiB of storage; (ii) The time taken to run setup grows proportionally to the delay, which is very expensive.

Therefore, a more practical construction of delay encryption would be an interesting research problem.

2.2.5 Timed Commitments and Timed Signatures

In 2000, Boneh and Naor introduced the notion of a *timed commitment* [35], a commitment scheme in which the message can be 'forced' open by completing a sequential calculation taking a prescribed length of time t. They additionally introduced the analogous *timed signature*, a commitment to a signature which can also be forced open through similar sequential calculation. A key application of timed commitments is fair contract signing: using timed signatures allows multiple distrusting parties to commit to a signature, with a guarantee that if any party quits the protocol, the relevant signature can be forced open. Whilst timed commitments work well in the two party case for such applications, they do not scale well, as the number of sequential computations grows with the number of parties.

Early literature which also focused on the timed-release of signatures and other time-sensitive information worked on the basis that there was a slow and partial release of the information. That is, the signature would be released in small portions a bit at a time [27, 60, 67]. Furthermore, early delay-based signature literature ensured that standard digital signature schemes could be leveraged into the constructions [75] [76]. This was to ensure backward compatibility and interoperability with well-known digital signatures schemes such as RSA and DSA.

Modern delay-based signature schemes also provide a property known as *well-formedness* [76, 150]. Well-formedness gives the party solving the time-lock puzzle a guarantee that the secret information will indeed be released when the puzzle is solved. This provides

assurance to the party solving the puzzle that they will not commit a large amount of work without a guarantee that the secret information will be released. The theory of verifiable timed signatures (VTS) was formalised in 2020 by Thyagarajan et al. [150], as a practical improvement upon a timed signature. VTS schemes are built from a delay perspective on homomorphic time-lock puzzles [118]. They also rely on digital signature schemes [92], non-interactive zero-knowledge proofs [138], and threshold secret sharing [144]. The homomorphic time-lock puzzles are used to aggregate the challenges of each aborting party into a single puzzle, which allows solvers to only solve one puzzle rather than many. The VTS constructions are compatible with standard BLS, Schnorr, and ECDSA digital signature schemes.

Timed-commitments and timed-release digital signatures have applications in pseudonymous secure computation [92] and non-malleable commitments which can mitigate concurrent person-in-the-middle attacks [106]. Furthermore, verifiable timed signatures also have specific applications in payment channel networks used in cryptocurrencies [117], multi-signature transactions which are used so that multiple signatures are required to authenticate transactions [32], and also in fair multi-party computation so that fairness in blockchains can be achieved by financially penalising parties which abort protocol execution [99].

We now move on to the second class of primitives, which contains proofs of sequential work and verifiable delay functions. These primitives are fundamentally different to those we have discussed so far: Rather than use the output of the delay to encode a message, the following primitives simply prove that a delay has occurred.

2.2.6 Proofs of Sequential Work

In 2013, Mahmoody et al. introduced 'publicly verifiable proofs of sequential work' [115]. The motivation for this work was to provide a computational timestamping technique for a document. They defined a non-interactive timestamping scheme based upon a 'hashgraph', a directed acyclic graph of repeated hashes. This technique is also used in other areas of cryptography, notably including in decentralised consensus protocols [12, 122, 145]. Their scheme was based upon creating a hashgraph for which it is slow for an adversary to compute any feasible alternative solution, whilst also being efficiently verifiable by a member of the public. These properties capture the essence of a PoSW: a computation which takes at least a set length of time t to compute, and which can be publicly verified significantly faster than t. The drawback of this construction, and indeed of the definition of the primitive, is that solutions are *not* unique, meaning that it is possible for a legitimate solution to be 'mangled' into another solution that verifies as correct.

In 2018, Cohen and Pietrzak introduced an alternative, similar construction [55], which has less requirements on the structure of the graph, and is more efficient than Mahmoody's construction. However, currently it seems that this construction has little bearing on the status of theory since the introduction of verifiable delay functions, which as we shall see in Section 7.2 are themselves proofs of sequential work.

In 2017, Lenstra and Wesolowski introduced a slow-timed hash function, which they named Sloth [104]. Sloth makes use of some number theoretic properties, where an evaluator is required to solve a chain of the following puzzles: Sample an input x from a group $\mathbb{Z}_p$, which is chosen such that $p \equiv 3 \bmod 4$. The evaluator is challenged to compute $\sqrt{x} \equiv x^{\frac{p+1}{4}}$.

This downside to such a construction is that the output is a t-bit number (for hardness parameter t), and hence verifying this output is time-consuming. However, this novel paper

advanced the theory of proofs of sequential work, and can be seen as a precursor to verifiable delay functions (VDFs), which we explain next.

2.2.7 Verifiable Delay Functions

The most prominent primitive introduced in recent years in the context of delay is the *verifiable delay function* (VDF). They were first introduced by Boneh et al. in 2018 [32], in what is now considered a seminal paper. A VDF is characterised by a delay in the form of an iterated sequential function, such that each input has a unique output that can be efficiently and publicly verified.

Upon comparison with the previous section, one can view a VDF as a *unique* proof of sequential work. A good illustration of the importance of uniqueness can be seen in a randomness beacon. Building a VDF-based randomness beacon critically relies on the uniqueness (or function) property of VDFs. Recall that a PoSW is not unique, and hence can have multiple outputs. This means that if one is to instead use a PoSW, then a malicious prover can compute multiple outputs, and select the PoSW output that yields the best beacon. This is a very important application, which we shall discuss at length in Section 2.4.3.

In [32], along with defining and motivating the primitive, Boneh et al. introduced various candidate approaches to constructing a VDF, such as using incrementally verifiable computation and injective rational maps, along with the drawbacks of each approach.

Shortly after the publication of this work, three VDF candidates were proposed: Wesolowski [155] and Pietrzak [130] each proposed an RSW-based VDF, and De Feo et al. [68] proposed a VDF based upon pairings over supersingular isogenies over elliptic curves. We briefly discuss each of these constructions.

Wesoloski's and Pietrzak's VDFs were designed concurrently, and are similar in nature: Both of these constructions are based upon repeated squaring and reduction in an RSA group

modulo N (as an interesting aside, this can be seen as contributing to the research into the multi-party generation of an RSA modulus, as discussed in Section 2.1.3). The solver then engages in an interactive protocol to prove to a verifier that they computed the correct solution. Where the two constructions differ however, is their verification procedures. Each uses a different succinct public-coin argument in order to verify the output. Wesolowski's protocol has a stronger security assumption (if it is secure then so is Pietrzak's VDF), smaller proof storage and faster verification. On the other hand, Pietrzak's VDF constructs the proof in significantly fewer operations. We refer the interested reader to [33] for a detailed discussion and comparison of the two constructions. De Feo et al.'s VDF on the other hand takes a different approach to computing the delay and verifying the VDF, utilising techniques from post-quantum cryptography. Their approach to computing a delay is to use BLS signatures together with isogeny graphs over supersingular elliptic curves in order to produce a slow function, that can quickly be verified. Unfortunately this construction suffers from a trusted setup, and implementation challenges such as very large storage requirements [68].

The importance of the VDF primitive is underlined by how quickly the subject has advanced in a short time, with multiple papers discussing VDF variants [64, 66, 122], and VDF-based applications [101, 102, 141]. In 2019, Ephraim et al. introduced the notion of a *continuous* VDF (cVDF) [66]. The cVDF model presented by Ephraim et al. introduces the notion of a *state*, which is an intermediate point within the computation that can be verified. In contrast, with a standard VDF verification is only possible at the end of the computation. These states enable two key applications that a standard VDF is lacking. Firstly, at any state the solving party can pass the computation on to another party, who can efficiently verify the state and take over the computation. Secondly, by running the verification procedure at each state, trusted public randomness can be extracted at regular intervals, creating an efficient randomness beacon from one input. We shall discuss randomness beacons further in Section 2.4.3.

There exists an interesting impossibility result by Mahmoody et al. [116], where the authors look into whether one can take a black box hash function, and use it to build a VDF. In this work it is proven that no perfectly unique VDF (i.e., a VDF with only one solution that will verify as correct for each input) can be constructed in the random oracle model.

In Section In Chapter 6, we build a novel continuous VDF, and resulting randomness beacon, leveraging a trusted setup in order to improve upon the efficiency of Ephraim et al., and hence improving upon the efficiency of the randomness beacon.

2.2.8 Miscellaneous Primitives with a Delay Component

There exist primitives for which a delay constitutes an essential component in terms of enabling functionality. We mention these for completeness.

For instance, in *break-glass encryption* [139] a user encrypts their data to the cloud, and in case of an emergency, such data can be detectably recovered without the use of any cryptographic secret. To trigger this one-time request, the user is contacted by the cloud on an alert address, and if after a prescribed delay there is no answer, this is interpreted as permission to “break the glass" and access the data. The delay in this setting is simply wall-clock time, i.e., it is not determined by some computation.

Time plays a key role also in *a posteriori openable* public-key encryption (APOPKE) [38], a primitive designed to provide a key to “open" encrypted messages that fall within a specific time window, the main application being lawful interception of encrypted messages under investigation. While seemingly close to the TRE and TSE lines of work, APOPKE addresses a specific scenario and comes with its own security definitions. Its realisation involves neither a time-server nor some computational delay, but is instead based on algebraic techniques and standard cryptographic building blocks.

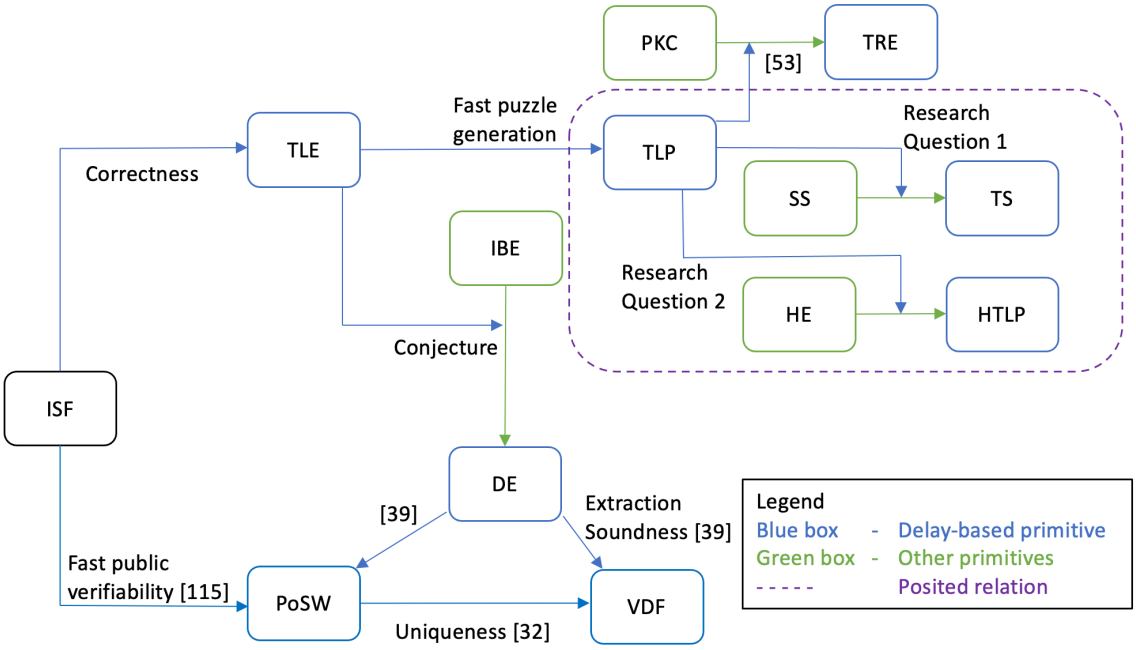

Fig. 2.1 Identified and conjectured relations between delay-based primitives.

2.2.9 Conclusion

We conclude this section by presenting Fig. 7.1, which provides a visual guide to the major primitives discussed, along with how they link together.

In the cases where there is no established link between primitives in the literature, we conjecture potential links. In Chapter 7, we discuss these links in detail, and use these to offer directions for future work.

2.3 Universal Composability

In [42], Canetti introduced the universal composability framework, to prove cryptographic protocols secure in a modular fashion. In this framework, any protocol Π will consist of n parties $P = \{P_1, \cdots P_n\}$, and an adversary A. All parties, as well as the adversary are run by interactive Turing machines (ITMs). The adversary has the power to corrupt a subset $I \in P$ of parties. The UC framework also takes into account an additional ITM known as the

environment, which accounts for any leakage of information, such as through side-channels, that will occur in complex protocols with multiple sessions running simultaneously.

The concept of this framework is to define an ideal functionality, which captures which properties a given primitive (for example a time-lock puzzle) should achieve. Parties exchange messages via these ideal functionalities, which securely compute all computation before passing it back to the party. A protocol for instantiating such primitives is then defined as secure in the following way: The ideal functionality is said to live in the 'ideal world', and the protocol is said to live in the 'real world'. In the real world, there is an adversary who is given the power to corrupt parties, intercept messages etc. in keeping with traditional cryptographic models. In the ideal world, there is instead a simulator who acts as the adversary, with the power to corrupt parties. Additionally, in both worlds there exists the 'environment'. The environment represents information leaked through side channels, such as the number of messages sent across a channel. A protocol is said to be secure, if the environment cannot distinguish between the real world adversary attacking the protocol, and the ideal world simulator attacking the functionality.

This model allows one to substitute the protocols for their functionalities when proving the security of a more complex primitive.

What is highly interesting, is that the standard notions of time discussed in Section 2.1 do not apply to this setting.

Time in UC Recently, there have been efforts to model time-based primitives such as TLPs and TLEs in the universal composability model [9, 17, 16]. In contrast to what we have seen in previous sections, Baum et al. prove that in order to build UC-secure TLPs, one *must* use the random oracle model [16].

Baum et al. introduce TLPs via a ticker functionality in TARDIS [16], which works by splitting time into units known as *clock ticks*. The global clock advances a tick only when all

honest parties have submitted a command which indicates that they have been activated in the current tick. During each tick, a party may interact with any number of functionalities to send messages.

In [9], Arapinis et al. introduce a UC TLE scheme named Astrolabous, which relies on a global clock [6] rather than the TARDIS model. This global clock introduces a synchronised notion of time for all parties, who may 'read' the time from the global clock at any stage.

Where the models of TARDIS and Astrolabous diverge, is how the clock is managed: In the TARDIS model, the 'ticker' provides ticks to each functionality on behalf of the environment, which means that parties do not need to observe the time elapsed by the ticker. They instead see events that are triggered by elapsed time. On the other hand, in Astrolabous, parties explicitly read the time of the global clock to see what round it is.

The key difference between these models can be seen as follows: In the ticker model of TARDIS [17, 16] time is not synchronised but progresses identically for all entities. In Astrolabous [9], the model relies on the stronger assumption that time is synchronised among all entities, and uses an existing approach to using a global clock. The former assumption appears to be weaker, as strict synchronisation is a hard task. On the other hand, the Astrolabous model allows for the generic group model to be avoided, which allows the construction to be applied more widely.

In recent years, the first time-based primitives in the UC framework have been published, allowing for various techniques to be moved to a composably secure setting [6, 8]. This provides stronger security guarantees in complex protocols, and it is our hope that more time-based primitives are constructed in the UC framework.

2.4 Applications

In each of the following three chapters of this thesis, we will introduce a new construction. In this section we discuss the applications which motivate these constructions. Explicitly, we will explore the use of delay in auctions, whistleblowing and randomness beacons.

In Chapters 4 and 6, we show that one can leverage a trusted setup in order to build novel, efficient constructions, which can be applied to auctions and randomness beacons.

In Chapter 5, we instead introduce a new setting of delay-based cryptography, in which we use the same trusted setting to provide fine-grained control of the release of data, using whistleblowing as a use case.

2.4.1 Auctions

Sealed-bid auctions allow bidders to secretly submit a bid for some goods without learning the bids of any other party involved until the end of the auction. The challenge of building a fair, efficient, and cryptographically secure auction has been of interest to the cryptographic community for decades [25, 37, 39, 73, 90]. It is a common motivating example for delay-based primitives, and was mentioned as an application for time-lock puzzles by Rivest et al. in 1996 [136].

A common approach to constructing sealed-bid auctions is to implement a *commit-and-reveal* solution using an append-only bulletin board, such as a blockchain [73]. Such solutions consist of two phases: a bidding phase, where parties commit to a bid and post their commitment on the bulletin board; and an opening phase where parties reveal their bids. However, the main drawback of this approach is that parties are not obliged to open their bids, which is particularly problematic in certain auction variants where it is necessary to learn the second highest bid as well as the first [11, 28, 153]. For an auction to be transparent and fair

it is desirable that each party must open their commitments to the bid once the bidding phase has ended.

One can replace the commitments in the above approach with time-lock puzzles, by requiring that each party encrypts their bid as the solution to a time-lock puzzle, placing the puzzle on the bulletin board rather than the commitment. This means that in the opening phase if a party does not reveal their bid it can instead be opened by computing the solution to the relevant puzzle. There exist multiple various constructions that fit this approach to auctions, and we view the non-malleable TLP construction of Freitag et al. [70] as the state of the art, as it requires no setup. However, this method does not scale well because it leads to many different time-lock puzzles being solved, which is computationally expensive. In recent years, various primitives that we discussed in Section 2.2 have been applied to solve this problem more efficiently. We will now outline these approaches.

In 2019 [118], Malavolta et al. used homomorphic time-lock puzzles (see Section 2.2.2) to improve upon this concept. Their insight is that the tallier then uses techniques from homomorphic encryption to evaluate a computation over the set of puzzles to determine the winning bidder. This leads to only the relevant puzzle being solved rather than the entire set of puzzles. Whilst this is a very elegant solution, the application relies on fully homomorphic TLP constructions, but all current constructions of homomorphic TLPs are based on indistinguishability obfuscation (IO) [39, 118]. IO aims to obfuscate programs to make them unintelligible whilst retaining their original functionality [14]. However, IO is known to be impractical with no construction efficiently implementable at the time of writing [89].

In 2021, Burdges et al. described auctions as a key motivating example for their Delay Encryption primitive [39] (see Section 2.2.4).Where time-lock puzzles require each bidder to encrypt their bid against a unique time-lock puzzle, Delay Encryption instead requires bidders to encrypt their bid to the public *session ID*. Bidders can then run the sequential

computation to extract the session key. Once the session key has been extracted all bids can be decrypted, thus replacing the opening phase described in the commit-and-reveal paradigm. This works well in the context of auctions as in the opening phase, after only one slow computation, all bids can be decrypted. When compared to HTLPs, this is an improvement as rather than performing a computation to learn the one relevant puzzle, and then decrypting it, in DE the computation can be computed directly and then used to open every encryption. This seems like an ideal approach on the primitive level, but as described in Section 2.2.4, there are instantiation challenges with the only one candidate construction to date.

The goal of the approaches outlined above is to utilise a time-delay to solve the auction problem in a scalable, and trustless manner. This improves upon the efficiency of Rivest's solution by ensuring that only one sequential computation (namely the puzzles containing the highest bid in [118], and the delay computed in [39]) needs to be run, rather than one for each bid. However, the HTLP approach is limited in the scope of its application, and DE is impractical.

If one is willing to compromise on the decentralisation and utilise a trusted setup, TRE can be used instead. In 2021, Chvojka et al. suggest using TRE (see Section 2.2.3) to provide a more efficient decryption method, in a similar vein to the described application of DE. By using straightforward public key decryption to decrypt all of the bids, this is a very efficient approach. In Chapter 4, we construct and implement such a TRE scheme, showing it runs efficiently on consumer-grade hardware.

To conclude, there have been various modern improvements over the original approach of Rivest et al. However, we still lack an optimal solution, that is, one that does not rely on a trusted setup, and yet shares an efficient decryption method for all messages. For an auction in which only the highest bid is required, then Malavolta's HTLP approach is the current state of the art for a decentralised auction. On the other hand, if one is content with a trusted setup, then TRE is the most efficient primitive.

2.4.2 Whistleblowing

A novel extension of TRE which we shall introduce in Chapter 5 is the primitive *TRE with implicit authentication*, which allows an encryptor to fine-tune the release of information, whilst providing the recipient with guarantees regarding the legitimacy of the material. We demonstrate that this can be used to introduce safety mechanisms for vulnerable people, using whistleblowers as an illustrative use-case.

A whistleblower is a person who leaks sensitive information on a prominent individual or organisation. Obtaining and sharing the sensitive information associated with whistleblowing can carry great risk to the individual or party revealing the data. In Chapter 5, we explore the idea of incorporating a time delay in the context of whistleblowing, with the goal of providing a cryptographic solution that can improve upon and be used in conjunction with current tools. Explicitly, we introduce a primitive known as TRE with implicit authentication, which ensures that an adversary cannot encrypt a ciphertext of a chosen message, and claim it was from a whistleblower. Additionally, it allows the whistleblower to separate the actions of distributing ciphertexts and distributing the 'challenge', from which the decryption key is derived.

It is our hope that our work in this topic this will lead to further research and, eventually, some publicly-accessible technology which will allow whistleblowing to be practiced in a safer manner.

2.4.3 VDF-based Randomness Beacons

A high-entropy source of public randomness is a necessary component of many cryptographic protocols, including secret sharing and key distribution [29, 58]. Since 2011 NIST have been running a competition to build a trusted randomness beacon, with the objective of promoting the availability of trusted public randomness as a public utility. A randomness beacon allows

a group of parties to use some shared randomness in a protocol, each with the guarantee that none of the other parties has any prior-knowledge of the output. Common applications include running a lottery and random sampling. Random sampling can be used for selecting patients in clinical trials or selecting officials in audits, for instance.

A modern approach to building a randomness beacon utilises verifiable delay functions (VDFs), discussed in Section 2.2.7. Whilst VDFs can also be used to construct consensus protocols and in timestamping, their flagship application is indeed to provide a publicly verifiable randomness beacon [56, 101].

On a high level, a VDF samples a pseudo-random input, and uses the output of the delay function to extract randomness.We use the canonical example from [32] to illustrate this: Say we take the value of a stock price at the end of a trading session as input: the value of a given stock is assumed to be difficult to predict, and hold some entropy. However, a powerful adversary may be able to alter the value this stock finishes at, by making some large trades. One can employ a verifiable delay function with a time parameter high enough that by the time it has been evaluated on any candidate values, trading has finished for the day. This ensures that no party can compute the effect of altering the input whilst trading is still live, and hence the output of the delay function will remain indistinguishable from random. We next describe the most relevant protocols which build upon a VDF to build an RB.

In 2018, Drake et al. [65] proposed a smart contract which uses a VDF to produce random values. This approach requires multiple parties, known as beacon chain proposers to each contribute some local randomness. Drake assumes that there exists a global clock and splits up time into regular *epochs* of 1024 seconds. Each of these epochs is split into 8-second blocks, each of which is ran by a beacon chain proposer. These proposers each commit to some local entropy and reveal it at the end of their block, where it is then broadcast as randomness. This randomness is then used to sample a later beacon proposer. This idea was

presented as a post on Ethereum research forum, and is currently lacking a rigorous security analysis.

In 2020, Schindler et al. proposed randrunner [141], an interactive RB construction in which all participants are given a unique trapdoor in setup. Then consecutive rounds of randomness are evaluated where in each round an input is sampled and also one *leader* is chosen, whose trapdoor allows them to evaluate the VDF faster than any other party. The output of the VDF evaluated in each round is hashed to obtain a pulse of randomness which is the beacon output, then another input and leader are chosen for the subsequent round. All parties are encouraged to try to solve the VDF, even if they do not have the trapdoor, which leads to a large amount of computational expenditure, particularly as the number of participants grow. Additionally, this construction suffers from various attacks, such as the adversary being able to corrupt the round leaders, and withhold output, whilst working on subsequent rounds.

In 2022, Lee et al. proposed HeadStart [102], with the novel idea of having multiple parties computing VDFs simultaneously in a *contribution phase*. During the contribution phase each party in the protocol contributes some randomness before a third party known as the organiser uses these inputs to provide a verifiably random result.

An alternative approach to improve efficiency is to use a continuous VDF, as discussed in Section 2.2.7. Ephraim et al. [66] build a randomness beacon from a continuous VDF by extracting randomness at regular intervals. Recall from Section 2.2.7 that a cVDF consists of multiple states where verification can occur.

In the construction of [66], an initial state state_0 is generated during the setup procedure. Then two algorithms are ran in parallel on every state: algorithm Tick takes a state state_i, and outputs the next state state_{i+1}. Algorithm Tock takes the state state_i and outputs a pulse of randomness. In practice the randomness is obtained with a cryptographically secure hash function. Verification can be performed on both the computation of Tick and of Tock, to

show that the state was computed correctly, and to show that the randomness was correctly computed from the state.

Ephraim et al. presented a construction based upon their cVDF in [66], however the concrete RB resulting from this approach has a verification time which grows in $O(\log t)$, and hence scales badly as the time parameter increases.

In Chapter 6, we demonstrate that one can leverage a trusted setup to build a cVDF which scales more efficiently, hence leading to a more effient randomness beacon.

Chapter 3

Preliminaries

Contents

In this chapter, we introduce some important cryptographic and number theoretic preliminaries that will be used in the subsequent chapters.

3.1 Cryptography

We have discussed the cryptographic ideas most relevant to this thesis in the previous chapter, and so here we restrict our attention to the building blocks that we require for the subsequent chapters, rather than giving a broad cryptographic overview.

3.1.1 The RSW time-lock assumption

As discussed in Chapter 2, the RSW time-lock assumption [136] is core to a number of notable constructions using a cryptographic delay in the latest literature [32, 66, 70, 118, 130, 155]. In this thesis, we rely upon the security of this assumption in each of our constructions. Here, we provide an explicit definition of the RSW time-lock assumption, and the square and multiply algorithms in the RSA setting. We shall use these in each of the subsequent three chapters.

Algorithm 3.1.1: Square and Multiply [57]

```
input  : (a,b,N), // a,b,N ∈ ℕ, a^b mod N
d := 1
B := bin(b) // b in binary
for j ∈ B do
    d := d^2 mod N
    if j = 1 then
        d := da mod N
    end
end
output : d
```

Definition 3.1.1 ***RSW Time-Lock Assumption***: *Let $N = pq$ where p and q are distinct odd primes. Uniformly select $x \in \mathbb{Z}_N^*$, where $\mathbb{Z}_N^* = \{x \mid x \in (0,N) \wedge gcd(x,N) = 1\}$. Then set the seed term as $x_0 := x^2 \bmod N$. If a probabilistic polynomial time (PPT) adversary $\mathcal{A}$ does not know the factorisation of N or group order $\phi(N)$ then calculating $x_t \equiv x_0^{2^t} \bmod N$*

is a non-parallelizable calculation that will require t sequential modular exponentiations calculated with the Algorithm 3.1.1 Square and Multiply [136].

3.1.2 Provable Security

Throughout this thesis, whenever we present definitions and proofs, we will do so within a *provable security* framework. Provable security refers to the practice of modelling a cryptographic system with an adversary, commonly denoted by $\mathcal{A}$. Proofs in provable security aim to show that an adversary cannot break a given security property without solving an underlying problem, which is assumed to be hard [18, 62, 148]. From a formal perspective, this is a significant improvement over the alternative (which is providing intuitive arguments, and showing that known attacks do not work), as one gets a guarantee that a protocol is secure according to a rigorous definition. However, if the definition is inaccurate, or the hardness assumption is wrong, then insecure protocols can be 'proved secure'. This has led to criticism of the use of provable security, notably including [96, 97], where Koblitz and Menezes criticise various aspects of provable security, claiming that in many cases the security assurances it provides are false. However, this proved to be highly controversial, with many leading cryptographers writing to the editor of the journal that published these claims, in order to strongly disagree with the criticism [1]. We believe that provable security, whilst not flawless, is the most pragmatic approach to assessing the security of protocols. We do not think it should be a replacement for practical testing and analysis by the community, but is a very useful addition to this.

Security models Provable security generally models definitions as either *game-based*, or *simulation-based*. In this thesis, we shall present game-based definitions, which define formally when the adversary obtains inputs, and the outputs they must provide at each step.

The security of each protocol is measured by the probability of the adversary winning the relevant game.

The alternative is to use simulation-based definitions, where one compares the security in the 'real world' to that of an 'ideal world', where a protocol is secure by definition. The goal in simulation-based security is to show that security in the real world is no worse than in the ideal world. An important framework within the simulation model is the universal composability framework of Canetti [42], which we described in Section 2.3. We refer the interested reader to this section for a brief delay-oriented introduction, and to the following works for various proposed frameworks [42, 87, 100].

3.1.3 The Random Oracle Model

A random oracle is a theoretical object in cryptography which takes as input a query, and provides a random number as output. In modern cryptography, the random oracle model typically refers to using this model in order to obtain rigorous security proofs of protocols, and then later implementing the oracle using a hash function. It has now been 30 years since the use of random oracles was first suggested for use in cryptography [20], and this is now widely considered a standard technique.

The downside of the random oracle model lies in the disparity between an ideal random oracle, and any real life implementation using a hash function. It is clear that this powerful tool introduces a strong assumption, which has indeed shown not to hold in certain cases [43]. However, it should be noted that circumstances in which security does hold in the random oracle model, but does not in an implementation with a hash function tend to be artificial and unrealistic. Additionally, attempts to avoid the use of this model in proofs of existing protocols has led to security weaknesses not present in the original protocols as proved under

the random oracle model [98]. Despite the criticism, we view this model as an important method for bridging theory and practice in cryptography.

3.1.4 Cryptographic Primitives

Throughout the remainder of this thesis, we shall be using some tools from public-key cryptography. Here, we shall define a *public-key encryption scheme*, and *security against chosen-plaintext attacks*, which will be important for the security proofs that appear in Chapters 4 and 5. We additionally define a *cryptographic hash function*, which shall be used throughout the thesis, and a *pseudo-random number generator*, which is important for the randomness beacon construction of Chapter 6.

Definition 3.1.2 *A public-key encryption (PKE) scheme is a triple of PPT algorithms* $(\mathsf{Gen}, \mathsf{Enc}_{pk}, \mathsf{Dec}_{sk})$ *such that:*

1. Gen takes as input security parameter 1^λ, and outputs a key pair consisting of a public-key pk and a secret key sk.

2. $\mathsf{Enc}_{\mathsf{pk}}$ takes as input the public-key pk, and a message m and outputs a ciphertext c.

3. $\mathsf{Dec}_{\mathsf{sk}}$ takes as input the secret key sk, and a ciphertext c and outputs either a message m, or the symbol $\perp$, denoting failure.

Further, it is required that $\mathsf{Dec}_{\mathsf{sk}}(\mathsf{Enc}_{\mathsf{pk}}(m)) = m$ with overwhelming probability.

Chosen Plaintext Attack A standard security property in public-key cryptography is the notion of security against chosen plaintext attacks (CPA security). We say that a PKE scheme is CPA secure if any PPT adversary cannot win the following experiment: The adversary chooses two equal length messages and passes them to the challenger. The challenger returns the ciphertext of one of these messages, and challenges the adversary to distinguish which

message was encrypted. We make this formal in the CPA security game. This game-based definition will be the standard format for the formal security definitions that we use in this thesis.

CPA Security Game

1. $\mathcal{C}$ runs $\mathsf{pk},\mathsf{sk} \leftarrow \mathsf{Gen}(1^\lambda)$, to generate a public-key pair, and passes pk to $\mathcal{A}$.
2. $\mathcal{A}$ chooses two messages m_0, m_1 of the same length, and passes these to $\mathcal{C}$.
3. $\mathcal{C}$ chooses a bit $b \leftarrow \{0,1\}$ uniformly at random. $\mathcal{C}$ runs $c \leftarrow \mathsf{Enc}_{\mathsf{pk}}(m_b)$, and passes this to $\mathcal{A}$.
4. $\mathcal{A}$ outputs a bit b'.
5. $\mathcal{A}$ wins if $b = b'$.

A PKE scheme is CPA secure if $\mathcal{A}$ wins with probability at most $1/2 + \mathsf{negl}(\lambda)$.

Cryptographic Hash Functions Throughout this thesis we assume the existence of secure cryptographic hash functions, which we implement in later chapters using the very common algorithms SHA-2 and SHA-3. We define a hash function assuming the property collision resistance as follows.

Definition 3.1.3 *A cryptographic hash function H, with output length l, is a pair of PPT algorithms (Gen, H):*

1. Gen takes as input security parameter 1^λ, and outputs a key s.
2. H takes as input the key s and a string $x \in \{0,1\}^*$, and outputs a string $H^s(x) \in \{0,1\}^{l(n)}$

We additionally assume that the hash functions are collision resistant, which states that given a message m such that $H(m) = z$, a PPT adversary can't find a second message m' such that $H(m') = z$ with more than negligible probability. We make this explicit in the collision resistance game.

Pseudorandom Number Generators The final cryptographic object we shall introduce here is a pseudo-random generator (PNRG). A PRNG takes a small amount of true randomness, and uses this to generate a large amount of pseudorandomness.

Collision Resistance Game

1 $\mathcal{C}$ runs $s \leftarrow \mathsf{Gen}(1^\lambda)$, and samples a uniformly random $x \in \{0,1\}^*$, and passes these to $\mathcal{A}$.
2 $\mathcal{A}$ runs a PPT algorithm on s and x, and outputs a value x'.
3 $\mathcal{A}$ wins if $x' \neq x$ and $H^s(x) = H^s(x')$.
H is collision resistant if $\mathcal{A}$ wins with probability at most $\mathsf{negl}(\lambda)$.

Definition 3.1.4 *A pseudo-random generator (PNRG) t is a deterministic algorithm G with the following properties:*

1. ***Expansion*** *For a polynomial l, and an input $s \in \{0,1\}^n$, the output of G has length $l(n)$, where $l(n) > n$.*

2. ***Pseudorandomness:*** *For any PPT algorithm D, there exists a negligible function negl such that the following holds for uniformly chosen $r \in \{0,1\}^{l(n)}$, and uniformly chosen $s \in \{0,1\}^n$:*

$$|Pr[D(r) = 1] - Pr[D(G(s)) = 1]| < negl(n)$$

We next describe the PNRG that we will use to instantiate the constructions in the following chapters of this thesis.

Blum Blum Shub The Blum Blum Shub pseudo random number generator (BBS CSPRNG) [26], is a PNRG based upon the repeated squaring function that we have discussed in Chapter 2, and in Section 3.1.1. It begins by generating an RSA modulus N, and selecting $x \in \mathbb{Z}_N^*$. Then the seed value $x_0 \equiv x^2 \bmod N$ is sampled. To produce a string of t bits, the least significant bit is extracted from each term $x_i \equiv x_{i-1}^2 \bmod N$ for $i \in (1,\ldots,t)$.

3.1.5 Classical vs Quantum Cryptography

The setting of this thesis is classical cryptography, which is to say that we do not consider security in the presence of a quantum computer. A quantum computer is a machine that utilises techniques from quantum mechanics to allow many computations to be carried out simultaneously [63]. If a large, efficient quantum computer is ever built, then many classical cryptographic schemes would be broken, including notably the RSA algorithm used in this thesis. However, despite a large amount of effort by both industry and academia, the current state of the art is largely experimental and impractical [156].

In this thesis, we look at delay-based cryptography with the intention of improving the practicality and efficiency of current tools. Whilst work on delay-based cryptography which is quantum-secure has recently begun [113], the techniques are not yet developed enough to improve upon the efficiency of classical techniques. We hence view it as outside the scope of this thesis, but hope that this subject receives further attention in the future.

3.2 Number Theory

In this section we review the number theory required for the constructions we provide in Chapters 4, 5 and 6.

3.2.1 The Chinese Remainder Theorem

A well known result which we use in our constructions is the Chinese Remainder Theorem (CRT) [146]. The CRT is a number theoretic result regarding finding a unique solution to a system of linear congruences with specific properties. It also provides an explicit formula for finding the unique solution which can be calculated in polynomial time. It is defined as follows:

Theorem 3.2.1 *Let the following be a system of linear congruences:*

$y \equiv \alpha_1 \mod n_1$

$y \equiv \alpha_2 \mod n_2$

$\vdots$

$y \equiv \alpha_k \mod n_k$

Where $y, n_i \in \mathbb{Z}^+$, $\alpha_i \in \mathbb{Z}$, and α_i are arbitrary integers and each n_i is pairwise coprime $\forall i \in \{1, \ldots, k\}$. Then, this system of linear congruences is guaranteed to have a unique solution mod N:

$$y = \sum_{i=1}^{k} \alpha_i N_i N_i^{-1} \mod N \tag{3.1}$$

Where $N = \prod_{i=1}^{k} n_i$, $N_i = \frac{N}{n_i}$, and N_i^{-1} is the multiplicative inverse of N_i mod n_i, i.e. $N_i N_i^{-1} \equiv 1 \mod n_i$. We also recall that each N_i^{-1} can be found in polynomial time using the Extended Euclidean Algorithm.

Proof: Proof can be found in [93]. □

The CRT can also be thought of from a different perspective. That is, given y and the moduli n_i, find the solutions for each α_i. When the CRT is considered in the latter manner an equivalent statement known as the Chinese Remainder Theorem Isomorphism is used. In this thesis, we are concerned in applying the results of the CRT in the RSA setting, namely the case where $N = pq$, where p and q are distinct odd primes. Therefore, we now present the CRT Isomorphism in this specific case.

Definition 3.2.1 *The Chinese Remainder Theorem Isomorphism.*

In the case of $N = pq$, where p and q are distinct odd primes, the Chinese Remainder

Theorem Isomorphism is denoted by

$$\mathbb{Z}_N^* \simeq \mathbb{Z}_p^* \times \mathbb{Z}_q^* \tag{3.2}$$

Simply stated, each $y \in \mathbb{Z}_N^*$ *is equivalent (isomorphic) to a tuple* $([y \mod p], [y \mod q])$.

3.2.2 The Fermat-Euler Theorem

Next, we present the Fermat-Euler Theorem [154].

Theorem 3.2.2 *Fermat-Euler Theorem. Let* N *be an odd prime, or let* $N = pq$*, where* p *and* q *are distinct odd primes. If* $gcd(a,N) = 1$*, then* $a^{\phi(N)} \equiv 1 \bmod N$*, where* $\phi(N) = N - 1$ *if* N *is prime or* $\phi(N) = (p-1)(q-1)$ *if* $N = pq$.

Proof: Proof can be found in [78]. □

In the constructions which we will present in subsequent chapters, we will use $\phi(N)$ as a trapdoor in Algorithm 3.1.1, in order to allow a trusted party to run a setup algorithm which runs significantly faster than the sequential computation of a solver who does not know $\phi(N)$. We prove this in the following corollary.

Corollary 3.2.1 *Let* $x_0 \in \mathbb{Z}_N^*$*. If the group order* $\phi(N)$ *is known, then calculating* x_t *such that* $x_t \equiv x_0^{2^t} \bmod N$ *can be done in* $\log_2 N$ *binary operations.*

Proof: Let $x_t \equiv x_0^{2^t} \bmod N$. If the exponent 2^t is reduced mod $\phi(N)$ we have $2^t = \alpha\phi(N) + \beta$, where β is the remainder of 2^t after the $\phi(N)$ modular reduction. Then, by Theorem 3.2.2 we have $x_t \equiv x_0^{2^t} \equiv x_0^{2^t \bmod \phi(N)} \equiv x_0^{\alpha\phi(N)+\beta} \equiv x_0^{\phi(N)^\alpha} x_0^\beta \equiv 1^\alpha x_0^\beta \equiv x_0^\beta \bmod N$. The number of bits in β is $O(\log N)$, and β is input into line 2 of Algorithm 3.1.1. □

3.2.3 Quadratic Residues and the Jacobi Symbol

Our constructions in the following chapters each have an RSA modulus N that can be factored. In order to prove that this is possible, we use quadratic residues, and the Jacobi symbol. As such, we now define each of these.

Definition 3.2.2 ***Quadratic Residues*** *in $\mathbb{Z}_N^*$ are numbers r that satisfy congruences of the form:*

$$x^2 \equiv r \mod N \tag{3.3}$$

If an integer x exists such that the preceding congruence is satisfied, we say that r is a quadratic residue of N. If no such x exists we say that r is a quadratic non-residue of N.

The Jacobi symbol, denoted $\mathcal{J}_N(r)$, is a function which defines the quadratic character of r in Equation 3.3. The Jacobi Symbol can be calculated in polynomial time using Euler's Criterion.

Theorem 3.2.3 *Euler's Criterion can be used to calculate the Jacobi Symbol of the number r in Equation 3.3 for a prime modulus p. If $\mathtt{gcd}(r,p) = 1$, then:*

$$\mathcal{J}_p(r) = r^{\frac{p-1}{2}} = \begin{cases} +1, \text{if } r \in \mathcal{QR}_p \\ -1, \text{if } r \in \mathcal{QNR}_p \end{cases} \tag{3.4}$$

Where $r \in \mathcal{QR}_p$ indicates that r is a quadratic residue of p and $r \in \mathcal{QNR}_p$ indicates that r is a quadratic non-residue of p.

Proof: Proof can be found in [93]. □

When the modulus is a prime number, if the Jacobi symbol evaluates to $+1$ then r is always a quadratic residue and if the Jacobi symbol evaluates to -1 then r is always a

quadratic non-residue. The Jacobi symbol is more complex when the modulus is a composite number $N = pq$.

Corollary 3.2.2 *(Of Theorem 3.2.3). Euler's Criterion can be used to calculate the Jacobi Symbol of the number r in Equation 3.3 for a composite modulus N if the factorisation of N is known.*

Proof: Proof can be found in [93]. □

Algorithm 3.2.1 shows how to determine the quadratic character of r for composite N using Theorem 3.2.3 and Corollary 3.2.2. When N is composite the quadratic character of r can take three formats. If the Jacobi symbol evaluates to -1 then r is always a quadratic non-residue, denoted $\mathcal{QNR}_N^{-1}$. However, if the Jacobi symbol evaluates to $+1$ then r can either be a quadratic residue, denoted $\mathcal{QR}_N$ or a quadratic non-residue denoted $\mathcal{QNR}_N^{+1}$.

Algorithm 3.2.1: Calculating $\mathcal{J}_N(r)$ for composite N.

input : (r, p, q)
$\mathcal{J}_p(r) := r^{\frac{p-1}{2}} \bmod p$
$\mathcal{J}_q(r) := r^{\frac{q-1}{2}} \bmod q$
if $\mathcal{J}_p(r) = 1 \wedge \mathcal{J}_q(r) = 1$ **then**
  $x := \mathcal{QR}_N$
else if $\mathcal{J}_p(r) = -1 \wedge \mathcal{J}_q(r) = -1$ **then**
  $x := \mathcal{QNR}_N^{+1}$
else
  $x := \mathcal{QNR}_N^{-1}$
end
output : x

Quadratic residues and quadratic non-residues for composite N have a distinct distribution in $\mathbb{Z}_N^*$, which can be described as follows.

Theorem 3.2.4 *The cardinality of $\mathcal{QR}_N$, $\mathcal{QNR}_N^{+1}$, and $\mathcal{QNR}_N^{-1}$ for composite $N = pq$, where p and q are distinct primes is as follows:*

$$\begin{aligned} \left|\mathcal{QR}_N\right| &= \frac{\left|\mathbb{Z}_N^*\right|}{4} = \frac{\phi(N)}{4}. \\ \left|\mathcal{QNR}_N^{+1}\right| &= \frac{\left|\mathbb{Z}_N^*\right|}{4} = \frac{\phi(N)}{4}. \\ \left|\mathcal{QNR}_N^{-1}\right| &= \frac{\left|\mathbb{Z}_N^*\right|}{2} = \frac{\phi(N)}{2}. \end{aligned} \tag{3.5}$$

Where, $\left|\mathbb{Z}_N^*\right| = \phi(N) = (p-1)(q-1)$, and $\phi(N)$ is Euler's totient function.

Proof: Proof can be found in [93]. □

Next, we discuss how to calculate preceding terms of the seed term $x_0 \in \mathcal{QR}_N$ in an RSW time-lock sequence. To calculate the subsequent term of x_0 in the sequence evaluate $x_1 \equiv x_0^{2^1} \bmod N$ by inputting $(x_0, 2^1, N)$ into Algorithm 3.1.1.

If the factorisation of N is known Theorem 3.2.3 can be used in conjunction with the Chinese Remainder Theorem (CRT) to calculate the term x_{-1} in polynomial time.

We first note that in each of our constructions, we will specify a particular type of RSA modulus, known as a Blum integer [26]. A Blum integer $N = pq$, is the product of two Gaussian primes. A Gaussian prime has the property $p \equiv 3 \bmod 4$. Using a Blum integer allows us to utilise some number theoretic properties to factor the modulus N after a sequential computation in each of these constructions.

Theorem 3.2.5 *Let p be a Gaussian prime. For any $r \in \mathbb{Z}_p^*$, if $\mathcal{J}_p(r) = +1$, then finding α such that $\alpha \equiv \sqrt{r} \bmod p$ can be found by calculating $\alpha \equiv r^{\frac{p+1}{4}} \bmod p$.*

Proof: Let $\alpha = r^{\frac{p+1}{4}} \bmod p$. Then $\alpha^2 \equiv (r^{\frac{p+1}{4}})^2 \equiv r^{\frac{2p+2}{4}} \equiv r^{\frac{p+1}{2}} \bmod p$. Next, let $\frac{p+1}{2} = 1 + \frac{p-1}{2}$. Therefore, by Euler's Criterion (Theorem 3.2.3) $\alpha^2 \equiv r^1 r^{\frac{p-1}{2}} \equiv r \bmod p$. In subsequent chapters, we will refer to α as the principal square root of $r \bmod p$. □

Example 3.2.1 *Let $N = 67 \cdot 139 = pq = 9313$. Given the seed $x_0 = 776 \in \mathcal{QR}_N$, the square root of $x_0 \bmod N$, denoted by $x_{-1} = \sqrt{x_0}$, can be found as follows:*

- *calculate* $\alpha \equiv x_0^{\frac{p+1}{4}} \equiv x_0^{17} \equiv 21 \bmod p$
- *calculate* $\beta \equiv x_0^{\frac{q+1}{4}} \equiv x_0^{35} \equiv 9 \bmod q$
- *calculate* $x_{-1} = \alpha q(q^{-1} \bmod p) + \beta p(p^{-1} \bmod q) = 128862$

Then α and β are calculated using Theorem 3.2.5 and x_{-1} is calculated using the CRT. Note that $(q^{-1} \bmod p)$ and $(p^{-1} \bmod q)$ are calculated using Euclid's Extended Algorithm. To verify correctness, note that $128862^2 \equiv 776 \equiv x_0 \bmod N$. We provide formal analysis of this in Section 4.4.

If $r \in \mathcal{QR}_N$ then the CRT implies that there are four distinct solutions to Equation 3.3.

Theorem 3.2.6 *For all $N = pq$, where p and q are distinct odd primes, each $r \in \mathcal{QR}_N$ has four distinct solutions.*

Proof: Proof can be found in [93]. □

If N is a Blum integer, then the four square roots of each $r \in \mathcal{QR}_N$ has specific properties. That is, two of the square roots of r are quadratic non-residues with Jacobi symbol -1, one square root is a quadratic non-residue with Jacobi symbol $+1$, and one square root is a quadratic residue. We prove this next.

Theorem 3.2.7 *Let N be a Blum integer. Then for all $r \in \mathcal{QR}_N$, if $x^2 \equiv x'^2 \equiv r \bmod N$, where $x \neq \pm x'$, then without loss of generality $\mathcal{J}_N(\pm x) = -1$, and $\mathcal{J}_N(\pm x') = +1$. That is $\pm x \in \mathcal{QNR}_N^{-1}$, $x' \in \mathcal{QR}_N$ and $-x' \in \mathcal{QNR}_N^{+1}$. We refer to $x' \in \mathcal{QR}_N$ as the principal square root of r mod N.*

Proof: If N is a Blum integer, then $N \equiv 1 \bmod 4$. By Theorem 3.2.6 every $x_0 \in \mathcal{QR}_N$ has four distinct square roots $\pm x$ and $\pm x'$. As $N \equiv 1 \bmod 4$, by the law of quadratic reciprocity $\mathcal{J}_N(x) = \mathcal{J}_N(-x)$ and $\mathcal{J}_N(x') = \mathcal{J}_N(-x')$. It must be the case that $x^2 \equiv x'^2 \bmod N$, which implies $(x-x')(x+x') \equiv 0 \bmod N$, which implies $(x-x') \mid N$ and $(x+x') \mid N$. That is, without loss of generality $(x-x') = k \cdot p$ and $(x+x') = \ell \cdot q$, where $k, \ell \in \mathbb{N}$. Therefore, $\mathcal{J}_p(x) = \mathcal{J}_p(x')$ and $\mathcal{J}_q(x) = \mathcal{J}_q(-x')$. As $p \equiv 3 \bmod 4$, the law of quadratic reciprocity tells us $\mathcal{J}_p(-1) = -1$, we have $\mathcal{J}_q(x) \cdot \mathcal{J}_p(-1) = \mathcal{J}_q(-x') \cdot \mathcal{J}_p(-1)$. This implies that $\mathcal{J}_N(-x) = \mathcal{J}_N(x')$ or written another way $\mathcal{J}_N(x) \neq \mathcal{J}_N(x')$.

Without loss of generality, eliminate the two roots with $\mathcal{J}_N$ equal to -1, say $\mathcal{J}_N(x) = \mathcal{J}_N(-x) = -1$. This leaves $\mathcal{J}_N(x') = \mathcal{J}_N(-x') = +1$. It is the case that only one of $-x'$ or x' has $\mathcal{J}_p = \mathcal{J}_q = 1$ as $p \equiv 3 \bmod 4$. Therefore, without loss of generality, it is only x' that has the property $\mathcal{J}_N(x') = +1$ and $x' \in \mathcal{QR}_N$ [26]. $\square$

Chapter 4

TIDE: A Novel Approach to Constructing Timed-Release Encryption

Contents

The work in this chapter appears in [111], which was a joint work with Angelique Faye Loe, Christian O'Connell, and Elizabeth A. Quaglia. In this chapter, we analyse the definition of timed-release encryption according to [53], and propose an alternative, more flexible definition. We then construct a timed-release encryption scheme using the cryptographic building blocks of the RSA-OAEP encryption scheme, and the Blum-Blum-Shub random number generator. We prove this scheme cryptographically secure, before providing an implementation in python, and an accompanying efficiency analysis of this implementation. My contributions to this work include the analysis of the definitions, the security modelling, the framing of the construction as a TRE scheme, and the applications.

In this chapter, we design a novel construction of a timed-release encryption scheme, which we name TIDE (TIme-Delayed Encryption). We build TIDE using the longstanding cryptographic tools of the RSW time-lock assumption and RSA encryption, details of which can be found in Sections 3.1.1 and 3.2 respectively. We shall begin with a discussion of the definition of timed-release encryption.

4.1 Definitions of Timed-Release Encryption

In Chapter 2, we introduced the notion of timed-release encryption (TRE). In both this chapter and in Chapter 5, we shall be working with the TRE primitive.

Recall from Section 2.2.3 that TRE has a 'classical' definition, which is reliant on a trusted third party and a time-server, and a modern definition according to Chvojka et al. [53]. For the remainder of this thesis, we shall be using the latter when we refer to TRE. The

intuition behind this type of TRE scheme is to combine a delay with a public-key encryption scheme, in order to enable encryption and, after a delay, decryption of multiple messages, whilst requiring a single sequential computation. A TRE scheme consists of four algorithms: Setup, Solve, Encrypt, Decrypt. Setup generates the public parameters, which comprise of a public encryption key, and information allowing a solver to derive a decryption key after a sequential computation. Solve uses the public parameters to evaluate a sequential computation in order to recover the decryption key. Encrypt is used to encrypt a message m under the public encryption key, and outputs a ciphertext c. Decrypt uses the decryption key to decrypt a ciphertext c.

We begin this chapter by presenting the formal definition of TRE according to [53]. Note that in [53] the authors offer a generalised version of this definition, to incorporate what they term *sequential timed-release encryption*. We do not need sequential TRE for this construction, and as such we simplify our definition to specify the “non-sequential" case.

4.1.1 Chvojka et al.’s Definition

We begin by presenting the definition of (sequential) TRE according to Chvojka et al., before analysing this definition. This will help to motivate why we next provide slightly different game-based security definitions, which we shall use for the remainder of this chapter and the subsequent chapter. Our definitions offer more flexibility, and are in our opinion easier to use. Additionally, our definitions are consistent with those we shall see in the later chapters of this thesis.

Definition 4.1.1 *A sequential timed-release encryption scheme with message space $\mathcal{M}$ is a tuple of algorithms TRE = (Setup, Enc, Solve, Dec) with the following syntax:*

- $\left(\mathrm{pp}_{e,i}, \mathrm{pp}_{d,i}\right)_{i\in[n]} \leftarrow \mathrm{Setup}\left(1^{\lambda}, (T_i)_{i\in[n]}\right)$ is a probabilistic algorithm which takes as input a security parameter 1^{λ} and a set of time hardness parameters $(T_i)_{i\in[n]}$ with

$T_i < T_{i+1}$ for all $i \in [n-1]$, and outputs set of public encryption parameters and public decryption parameters $\mathrm{PP} := \left(\mathrm{pp}_{e,i}, \mathrm{pp}_{d,i}\right)_{i\in[n]}$. It is required that Setup runs in time $\mathrm{poly}\left((\log T_i)_{i\in[n]}, \lambda\right)$.

- $s_i \leftarrow \mathrm{Solve}\left(\mathrm{pp}_{d,i}, s_{i-1}\right)$ is a deterministic algorithm which takes as input public decryption parameters $\mathrm{pp}_{d,i}$ and a solution from a previous iteration s_{i-1}, where $s_0 := \perp$, and outputs a solution s_i. It is required that Solve runs in time at most $(T_i - T_{i-1}) \cdot \mathrm{poly}(\lambda)$.

- $c \leftarrow \mathrm{Enc}\left(\mathrm{pp}_{e,i}, m\right)$ is a probabilistic algorithm that takes as input public encryption parameters $\mathrm{pp}_{e,i}$ and message $m \in \mathcal{M}$, and outputs a ciphertext c.

- $m/\perp \leftarrow \mathrm{Dec}\left(T_i, s_i, c\right)$ is a deterministic algorithm which takes as input a hardness parameter T_i, a solution s_i and a ciphertext c, and outputs $m \in \mathcal{M}$ or $\perp$.

A sequential timed-release encryption scheme is also required to be correct and secure.

Intuitively, a TRE scheme is *correct* if any encrypted message can be decrypted using the corresponding decryption key with overwhelming probability; security guarantees that any adversary cannot distinguish between two different ciphetexts, when bounded by t sequential steps. This can be viewed as a delay-based analogue of the notion of ciphertext security seen in Chapter 3.

We now introduce the definitions given in [53], noting that by design the following definitions are not consistent with others in this thesis: this is in order to draw a contrast between the original definitions and the game-based definitions we provide in the next section.

Correctness A sequential timed-release encryption scheme is *correct* if for all $\lambda, n \in \mathbb{N}$, for all sets of hardness parameters $\left(T_j\right)_{j\in[n]}$ such that $\forall j \in [n-1] : T_j < T_{j+1}$, for all $i \in [n]$

and for all messages $m \in \mathcal{M}$ it holds:

$$\Pr\left[m = m' : \begin{array}{r} \mathrm{PP} \leftarrow \mathrm{Setup}\left(1^\lambda, \left(T_j\right)_{j\in[n]}\right), s_i \leftarrow \mathrm{Solve}\left(\mathrm{pp}_{d,i}, s_{i-1}\right) \\ m' \leftarrow \mathrm{Dec}\left(T_i, s_i, \mathrm{Enc}\left(\mathrm{pp}_{e,i}, m_i\right)\right) \end{array}\right] = 1.$$

Security A sequential timed-release encryption scheme is *secure* with gap $0 < \varepsilon < 1$ if for all polynomials n in λ there exists a polynomial $\tilde{T}(\cdot)$ such that for all sets of polynomials $\left(T_j\right)_{j\in[n]}$ such that $\forall j \in [n] : T_j(\cdot) \geq \tilde{T}(\cdot)$, for all $i \in [n]$ and every polynomial-size adversary $\mathcal{A} = \left\{\left(\mathcal{A}_{1,\lambda}, \mathcal{A}_{2,\lambda}\right)\right\}_{\lambda\in\mathbb{N}}$ there exists a negligible function negl$(\cdot)$ such that for all $\lambda \in \mathbb{N}$ it holds

$$\mathbf{Adv}_{\mathcal{A}}^{\mathrm{TRE}} = \left|\mathsf{Pr}(b = b') : \left[\begin{array}{r} PP \leftarrow \mathsf{Setup}\left(1^\lambda, (T_j)_{j\in[n]}\right) \\ (i, m_0, m_1, \mathrm{st}) \leftarrow \mathcal{A}_{1,\lambda}(PP) \\ b \xleftarrow{\mathrm{s}} \{0,1\}; c \leftarrow \mathsf{Encrypt}\left(\mathsf{pp}_{e,i}, m_b\right) \\ b' \leftarrow \mathcal{A}_{2,\lambda}(c, \mathrm{st}) \end{array}\right] - \frac{1}{2}\right| \leq \mathsf{negl}(\lambda).$$

It is required that $|m_0| = |m_1|$ and that the adversary $\mathcal{A}_\lambda = \left(\mathcal{A}_{1,\lambda}, \mathcal{A}_{2,\lambda}\right)$ consists of two circuits with total depth at most $t^\varepsilon(\lambda)$ (i. e., the total depth is the sum of the depth of $\mathcal{A}_{1,\lambda}$ and $\mathcal{A}_{2,,\lambda}$).

4.1.2 Our Game-Based Definition

We now give our definition, which separates the public decryption parameter of Definition 4.1.1 into a *decryption key* and a *challenge*. This nuance introduces a finer grained control of information, which we will show in Chapter 5 leads to interesting new applications. Additionally, we provide definitions which are easy to work with, and consistent with the

rest of this thesis. In Section 4.1.3, we discuss at length the differences between the two definitions.

In what follows, we will refer to algorithms 'taking t time to compute', and 'bounding computation time by t'. In both cases, we are referring to evaluating a polynomial sized arithmetic circuit of depth at most t, an approach we discussed in Chapter 2.

Definition 4.1.2 (TRE) *A sequential timed-release encryption scheme with message space $\mathcal{M}$ is a tuple of algorithms TRE = (Setup, Enc, Solve, Dec) with the following syntax.*

- $(\mathsf{pk},\mathsf{sk},P) \leftarrow \mathsf{Setup}\left(1^{\lambda},t\right)$ is a probabilistic algorithm which takes as input a security parameter 1^{λ} and a time hardness parameter t, and outputs a public encryption parameter pk, a secret key sk, and a puzzle P. Setup must run in time poly $(\log t,\lambda)$.
- $\mathsf{sk} \leftarrow \mathsf{Solve}(\mathsf{pk},P,t)$ is a deterministic algorithm which takes as input a public key pk, a puzzle P, and a time parameter t, and which outputs a secret key sk. Solve must run in time at most $t \cdot \mathsf{poly}(\lambda)$.
- $c \leftarrow \mathsf{Encrypt}\,(\mathsf{pk},m)$ is a probabilistic algorithm that takes as input public encryption parameter pk and message $m \in \mathcal{M}$, and outputs a ciphertext c.
- $m/\perp \leftarrow \mathsf{Decrypt}\,(\mathsf{sk},c)$ is a deterministic algorithm which takes as input a secret key sk and a ciphertext c, and outputs $m \in \mathcal{M}$ or $\perp$.

We now provide game-based security definitions of correctness and security. In the security games that follow, $\mathcal{E}$ is the encryptor, $\mathcal{D}$ is the decryptor, and $\mathcal{A}$ is the (PPT) adversary.

Correctness We model our correctness game as an interaction between the encryptor and the decryptor, where the encryptor runs Setup and then Encrypt on a message m, obtaining a ciphertext c. The decryptor then runs Decrypt on the ciphertext c. A TRE is correct if the decryptor obtains the message m with overwhelming probability.

Correctness Game

1. $\mathcal{E}$ outputs the public key, secret key and puzzle: $(\mathsf{pk},\mathsf{sk},P) \leftarrow \mathsf{Setup}\left(1^{\lambda},t\right)$.
2. $\mathcal{E}$ computes the ciphertext on message m: $c \leftarrow \mathsf{Encrypt}(\mathsf{pk},m)$.
3. $\mathcal{E}$ passes the public parameters, challenge, time parameter and ciphertext to $\mathcal{D}$.
4. $\mathcal{D}$ recovers the secret key: $\mathsf{sk} \leftarrow \mathsf{Solve}(\mathsf{pk},P,t)$.
5. $\mathcal{D}$ decrypts the ciphertext: $m' \leftarrow \mathsf{Decrypt}(\mathsf{sk},c)$.

   A TRE scheme is correct if $m = m'$ with probability at least $1 - \mathsf{negl}(\lambda)$.

Security In our TRE game, *security* is defined as an indistinguishability game as follows: Suppose an adversary $\mathcal{A}$ chooses two messages of the same length, m_0 and m_1, and sends them to the encryptor $\mathcal{E}$. $\mathcal{E}$ chooses one of these messages at random, which it encrypts and sends to $\mathcal{A}$. The adversary then gets polynomial time to preprocess upon this ciphertext before receiving the challenge C. Upon receiving the challenge, the adversary must then make a guess *before* t sequential steps are computed. A TRE scheme is *secure* if no PPT adversary $\mathcal{A}$ can gain an advantage in guessing which message was chosen by $\mathcal{E}$. This is made precise in the Security game.

Security Game

1. $\mathcal{E}$ outputs the public key, secret key and puzzle: $(\mathsf{pk},\mathsf{sk},P) \leftarrow \mathsf{Setup}\left(1^{\lambda},t\right)$.
2. $\mathcal{A}$ chooses two messages of the same length (m_0,m_1), and passes them to $\mathcal{E}$.
3. $\mathcal{E}$ selects $b \in \{0,1\}$ uniformly at random, and encrypts m_b as $c \leftarrow_{\mathrm{R}} \mathsf{Encrypt}(pk,m_b)$.
4. $\mathcal{A}$ runs a preprocessing algorithm $\mathcal{A}_0$ on the public parameter and the ciphertext, and stores $\mathsf{st} \leftarrow \mathcal{A}_0(pp,c)$.
5. $\mathcal{E}$ sends t and C to $\mathcal{A}$.
6. $\mathcal{A}$ runs a PPT algorithm $\mathcal{A}_1$ which outputs $b' \leftarrow \mathcal{A}_1(\mathsf{st},C,t)$, where $\mathcal{A}_1$ must run in fewer than t sequential steps.

   $\mathcal{A}$ wins the game if $b = b'$. A TRE-IA scheme is secure if $\mathcal{A}$ wins the game with probability no greater than $\frac{1}{2} + \mathsf{negl}(\lambda)$.

4.1.3 Discussion

In this section we discuss the differences in the two definitions, and the motivation for our construction.

Comparison of definitions The first thing to discuss about Definition 4.1.1 is that it defines a generalisation of TRE known as 'sequential' TRE, in which a set of puzzles are defined in the setup, each with their own public encryption key, and each allowing for the decryption functionality at a different time. Each such puzzle takes as input the solution of the previous puzzle (represented by s_{i-1}), and the time taken to solve each individual puzzle i is the difference in the two time parameters $T_i - T_{i-1}$.

Chvojka et al. note that the above definition also defines "non-sequential" TRE, by setting $n = 1$. In that case the value T_i is not needed as an input for Dec algorithm, in contrast to sequential TRE. Therefore, the first way in which we deviate from the original definition is by removing all of the parameters that relate to sequential TRE, as we will be working with non-sequential TRE. Additionally, we deviate slightly from the definition of [53] by including a puzzle P and a secret key in the output of the Setup algorithm, rather than combining these two elements into a single public decryption parameter. We do this in order to emphasise the fact that in our construction, one can separate the release of the public key, which enables encryption; and the release of the puzzle, which starts the sequential computation and hence the delay. In our definition, these do not have to happen simultaneously, as is the case in [53]. This allows for a finer control over the release of information. Whilst in certain scenarios, e.g. auctions, this is not necessary, we will show in Chapter 5 how this can be used to enable novel applications such as whistleblowing. As such, we provide a definition which we can build directly upon in the next chapter. We would like to note that this difference in the two definitions is largely semantic, and has little bearing on the security of the scheme. Indeed, by amalgamating the puzzle and the secret key into the tuple (P, sk), and calling this the decryption parameter, we reach the standard definition of [53]. The one difference that occurs as a result of this separation is that an adversary may obtain ciphertexts before they receive the challenge, and spend this time trying to decrypt such ciphertexts. We model this in the security game by providing the adversary with a preprocessing step once the adversary

receives the ciphertext, but before they receive the message, in a style reminiscent of VDFs [32].

Finally, we note that the computation model used by Chvojka et al. includes a gap parameter of ε to represent a hardware advantage that the adversary may have over an honest solving party. Whilst we acknowledge that such a gap is likely to exist, we follow works such as [32, 68, 130, 155] in leaving this implicit rather than explicit. We have discussed the underling assumptions of various approaches to computing a delay, along with the potential gap due to a speed-up in hardware in Chapter 2.

TIDE In the field of timed-release encryption, there is no concrete construction of the modern style of TRE, which is practically efficient and implementable. In this chapter, we build TIDE from the standard, well-trusted building blocks of RSA-OAEP and repeated squaring in an RSA group. We use interesting number theoretic techniques to allow a solver to factor the RSA modulus N, and hence derive the RSA-OAEP decryption key. This approach to delay-based encryption may be of independent interest, and we demonstrate in the following chapters that it can be used in other scenarios. TIDE is particularly useful in the application of sealed-bid auctions, where the various alternative approaches each come with drawbacks, as we discussed in Section 2.4.1. Furthermore, our construction and definitions lay the groundwork for an extension to the TRE primitive known as TRE with implicit authentication, which enables novel applications. This shall be the subject of Chapter 5.

4.2 Technical Overview

In this section, we provide an intuition of our construction, before introducing all of the cryptographic and number-theoretic building blocks that we use to implement this. We endeavour to provide references to the earlier sections within this thesis in which we discuss each of the relevant underlying cryptographic and number theoretic techniques.

TIDE relies on the RSW time-lock assumption, which states that it is hard to compute $x^{2^t} \bmod N$ in fewer than t sequential steps [136], for an RSA modulus N. This assumption was first introduced in 1996 by Rivest et al. [136], and has been used to build a variety of cryptographic constructions [32, 70, 118, 130, 155]. TIDE deviates from previous literature by using number theoretic techniques to utilise the output $x^{2^t} \bmod N$ in a novel way. Explicitly, TIDE provides exactly the information required to factor the RSA modulus N. TIDE achieves this by incorporating a theorem of Fermat and Rabin, which states that if x and x' are known such that $x^2 \equiv x'^2 \bmod N$, where $x \not\equiv \pm x' \bmod N$, then the non-trivial factors of N can be recovered in polynomial time [131]. By carefully setting up the system we provide the user with value x and ensure that the output of the delay (implemented with sequential squaring) reveals x'. Therefore knowledge of x and x' can be used to factor N in polynomial time. Then we combine this with a standard RSA encryption scheme using N and an encryption exponent as the public key. Once a solving party computes the delay they can derive the secret key and hence can decrypt all messages. Therefore, our construction can be seen as a natural integration of an RSW-based time-lock puzzle and the RSA encryption scheme. We formalise this in terms of syntax in Section 4.3 and give security proofs in Section 4.4. We now highlight the key technical details.

The key insight of TIDE is contained in the generation of the public key and puzzle, as this allows us to use the relevant theorem of Rabin [131]. N is chosen to be a particular class of RSA modulus known as a Blum integer $N = pq$, which has the property that $p \equiv q \equiv 3 \bmod 4$. The puzzle consists of three different elements, $P = (x, x_0, x_{-t})$. First, an element x is efficiently sampled such that $\mathcal{J}_N(x) = -1$, where $\mathcal{J}_N(x)$ is the Jacobi symbol, as defined in Section 3.2. Next, the seed x_0 is calculated as $x_0 \equiv x^2 \bmod N$. Crucial to TIDE is the term x_{-t}, where $x_{-t}^{2^t} \equiv x_0 \bmod N$. Now, any party wishing to solve the puzzle sequentially calculates the term $x_{-1} := x' \equiv \sqrt{x_0}$ by repeated squaring. The term x' has the property $\mathcal{J}_N(x') = +1$. This is crucial, as in Setup x was chosen such that $\mathcal{J}_N(x) = -1$. Therefore,

the solving party obtains the term $x^2 \equiv x'^2 \equiv x_0 \bmod N$, where $x \neq x' \bmod N$. Thus, the party obtains all four square roots of x_0. Therefore, Solve can recover the non-trivial factors of N in polynomial time using the result from Rabin [131].

4.2.1 Implementation details

We use OAEP, the Optimal Asymmetric Encryption Padding [21] with RSA, to ensure that all encryptions achieve IND-CPA security (Section 3.1.4). On a high level, OAEP processes each plaintext using two random oracles in order to ensure that inspection of the resulting ciphertexts offers no method of distinguishing the underlying plaintext used for encryption. It has been shown that RSA-OAEP is secure under the RSA assumption [72], although this proof takes place in the random oracle model. As we discuss in Section 3.1.3, this model relies on the assumption that existing hash functions can securely instantiate a true random oracle. As we discussed in the relevant section, we believe that whilst it would be better to avoid using this model, it has stood the test of time in practice, and is extremely useful for obtaining rigorous security proofs.

We note that in this following construction, we shall fix the RSA exponent to be $e := 2^{16} + 1 = 65537$. This is in order to make the construction concrete, allowing us to provide accurate runtimes in the implementation analysis given in Section 4.5. $e = 65537$ was chosen as it conforms to many existing hardware and software specifications, and is widely seen as a compromise between being small enough to run efficiently, and large enough to offer security advantages over lower values of e such as 3 [31, 86]. It if of course possible to use other values of e, and 65537 should be seen as an example, used to provide realistic timings for the implementation study.

4.3 Our Construction

TIDE consists of the following four algorithms.

- $(\mathsf{sk}, \mathsf{pk}, P) \leftarrow \mathsf{Setup}(1^\lambda, t)$ takes as input a security parameter 1^λ and time parameter t and ouputs a secret key sk, public key pk, and a puzzle P. The secret key consists of the factors of $\mathsf{sk} := (p,q)$ and the public key consists of an RSA modulus N and fixed encryption exponent $e := 2^{16}+1 = 65537$. The puzzle is set to $P := (x, x_0, x_{-t})$, where $x^2 \equiv x_0 \bmod N$, $\mathcal{J}_N(x) = -1$, and where $x_{-t}^{2^t} \equiv x_0 \bmod N$.

- $\mathsf{sk} \leftarrow \mathsf{Solve}(\mathsf{pk}, P, t)$ takes as input the public key pk, puzzle P, and time parameter t and outputs the secret key $\mathsf{sk} := (p,q)$, where $N = pq$.

- $c \leftarrow \mathsf{Encrypt}(\mathsf{pk}, m)$ takes as input a public key $\mathsf{pk} := (N,e)$ and a message m and outputs a ciphertext c.

- $\{m, \perp\} \leftarrow \mathsf{Decrypt}(\mathsf{sk}, c)$ takes as input the secret key $\mathsf{sk} := (p,q)$ and a ciphertext c as input and outputs a message m or error $\perp$.

We now provide the four algorithms, along with the details of each.

Setup $\mathcal{E}$ runs $(\mathsf{sk}, \mathsf{pk}, P) \leftarrow_{\mathrm{R}} \mathsf{Setup}(1^\lambda, t)$ to generate the secret key, public key, and puzzle as seen on Algorithm 4.3.1 Setup. The function $\mathtt{prime}(j)$ on lines 3 and 4 is the Miller-Rabin Monte Carlo algorithm [125] which generates j bit Gaussian primes. That is, $p \leftarrow_{\mathrm{R}} \mathtt{prime}(j)$. This guarantees that N, which is calculated on line 6, is a Blum integer. Additionally, the second condition of the while loop on line 2 ensures that e does not divide $(p-1)(q-1)$, to ensure the pair (N,e) is not lossy. The nuances of this are explained in Section 4.4. Setup then enters a while loop. The purpose of the while loop is to find an x such that $x \in \mathcal{QNR}_N^{-1}$. The logic statement on line 8 condenses the conditional statements in lines 3, 5 and 7 of Algorithm 3.2.1 using De Morgan's laws [80]. Once a suitable x is found, x_0 is

set to $x^2 \bmod N$. Once x is sampled and x_0 is computed, the term x_{-t} is calculated, where $x_{-t}^{2^t} \equiv x_0 \bmod N$. To calculate x_{-t} in polynomial time, Euler's Criterion, the Fermat-Euler Theorem and the Chinese Remainder Theorem (CRT) must be applied (all of which are described in Section 3.2: Number Theory).

Next, α_t is calculated, where α_t is the tth square root of $x_0 \bmod p$. To complete the calculation of the term x_{-t}, the CRT is used on line 16, where the terms $(q^{-1} \bmod p)$ and $(p^{-1} \bmod q)$ are calculated using Euclid's Extended Algorithm (EEA). Theorem 3.2.5 tells us that $\alpha \equiv \sqrt{x_0} \equiv x_0^{\omega} \bmod p$, where $\omega = \frac{p+1}{4}$. Let α_t be the tth square root of $x_0 \bmod p$. For example, if $t = 2$, then $\alpha_2 \equiv \sqrt{\sqrt{x_0}} \equiv (x_0^{\omega})^{\omega} \equiv x_0^{\omega^2}$. Therefore, $\alpha_t \equiv x_0^{\omega^t} \bmod p$. Note that the exponent ω^t, for large t will make calculating $x_0^{\omega^t} \bmod p$ computationally infeasible. Therefore, the Fermat-Euler Theorem is used so the exponent ω^t can be reduced $\bmod (p-1)$. Next, β_t is calculated, where β_t is the tth square root of $x_0 \bmod q$. β_t is calculated in a similar fashion as α_t, except ω is set to $\frac{q+1}{4}$.

The puzzle P is set to the tuple (x, x_0, x_{-t}) and then $\mathcal{E}$ passes (pk, P, t) to $\mathcal{D}$.

Solve $\mathcal{D}$ runs $\mathsf{sk} \leftarrow \mathsf{Solve}(\mathsf{pk}, P, t)$ to solve the challenge, as seen on Algorithm 4.3.2 Solve. First Solve calculates the term x' in $t-1$ sequential steps by evaluating $x_{-t}^{2^{t-1}} \bmod N$. This is where the sequential calculation takes place using Algorithm 3.1.1 with inputs $(x_{-t}, 2^{t-1}, N)$. The term x' is guaranteed to be in $\mathcal{QR}_N$ by Definition 3.2.2. $\mathcal{D}$ now has $x \in \mathcal{QNR}_N^{-1}$ and $x' \in \mathcal{QR}_N$. Therefore, x must be distinct from x', and we have $x^2 \equiv x'^2 \equiv x_0 \bmod N$. Finally, using the result from Theorem 4.4.4, Solve calculates $\mathsf{gcd}(x - x', N)$ to recover one factor p' of N using Euclid's Extended Algorithm. Next, $\frac{N}{\mathsf{gcd}(x-x',N)}$ is calculated to recover the other factor q'.

Encrypt $\mathcal{E}$ runs $c \leftarrow \mathsf{Encrypt}(\mathsf{pk}, m)$ as seen in Algorithm 4.3.3 Encrypt. Encrypt takes as input the public key $\mathsf{pk} := (N, e)$ and encrypts a message m using RSA-OEAP encryption and outputs the ciphertext c. Using RSA-OAEP, parties can encrypt messages to the modulus N and encryption exponent e. This means that messages can be decrypted using the Decrypt

Algorithm 4.3.1: Setup run on security parameter 1^λ and time parameter t to create the secret key sk, public key pk and puzzle P.

input : $1^\lambda, t, e := 65537$
$p, q := 2$
while $p = q \wedge e \nmid (p-1)(q-1)$ **do**
  $p := \texttt{prime}(\frac{\lambda}{2})$
  $q := \texttt{prime}(\frac{\lambda}{2})$
end
$N := pq$
$\mathcal{J}_p(x), \mathcal{J}_q(x) := 1$
while $\neg(\mathcal{J}_p(x) = 1 \wedge \mathcal{J}_q(x) \neq 1) \wedge \neg(\mathcal{J}_p(x) \neq 1 \wedge \mathcal{J}_q(x) = 1)$ **do**
  $x := \mathcal{U}(2, N)$
  $\mathcal{J}_p(x) := x^{\frac{p-1}{2}} \bmod p$
  $\mathcal{J}_q(x) := x^{\frac{q-1}{2}} \bmod q$
end
$x_0 := x^2 \bmod N$
$\alpha_t := x_0^{\frac{p+1}{4}^t \bmod p-1} \bmod p$
$\beta_t := x_0^{\frac{q+1}{4}^t \bmod q-1} \bmod q$
$x_{-t} := \alpha_t q(q^{-1} \bmod p) + \beta_t p(p^{-1} \bmod q) \bmod N$
$P := (x, x_0, x_{-t})$
output : $(\mathsf{sk}, \mathsf{pk}, P, t)$

Algorithm 4.3.2: Solve runs on the public key, puzzle, and time parameter pk, P, t to recover the secret key sk.

input : $\mathsf{pk} := (N, e), P = (x, x_0, x_{-t}), t$
$x' := x_{-t}^{2^{t-1}} \bmod N$
$p' := \mathsf{gcd}(x - x', N)$
$q' := \frac{N}{p'}$
$\mathsf{sk} := (p', q')$
output : sk

algorithm only after Solve has recovered the secret key sk. Note that the Solve and Encrypt algorithms are not sequential. The Encrypt algorithm can be run by any solver using pk prior to the Solve algorithm recovering the sk.

Algorithm 4.3.3: Encrypt runs on a message public key pk and message m, to produce ciphertext c.

input : $\mathsf{pk} := (N, e), m$

1 $k_0, k_1, G, H \leftarrow \texttt{params}(1^\lambda)$ `// OAEP parameters`

2 $m' := m \,||\, 0^{k_1}$ `// Zero pad to` $n - k_0$ `bits`

3 $r := \text{rand}(k_0)$ `// Generate a random` k_0 `bit number`

4 $X := m' \oplus G_{n-k_0}(r)$ `// Hash` r `to length` $n - k_0$

5 $Y := r \oplus H_{k_0}(X)$ `// Hash` X `to length` k_0

6 $m'' := X \,||\, Y$ `// Create message object`

7 $c := m''^e \bmod N$ `// RSA encrypt`

output : c

Decrypt $\mathcal{D}$ runs $\{m, \perp\} \leftarrow \mathsf{Decrypt}(\mathsf{sk}, c)$ as seen in Algorithm 4.3.4 Decrypt. Decrypt takes as input the secret key $\mathsf{sk} := (p, q)$ and decrypts ciphertext c using RSA-OEAP decryption, and either outputs the message m or an error $\perp$. First, Decrypt recovers the decryption exponent d on lines $2, 3, 4$, where Euclids Extended Algorithm is used. Finally, the RSA-OEAP decryption algorithm removes the padding and randomness added during the encryption to recover the message m.

4.4 Security

In order to prove the security of TIDE, we must first define a new hardness assumption. Informally, this states that the terms x, x_0 and x_{-t} provide a negligible advantage to factoring a Blum integer N when the computational time is bounded by t.

Definition 4.4.1 (BBS Shortcut Assumption) *Let the RSA Assumption be that for any* $N \leftarrow_R \mathsf{Gen}\left(1^\lambda\right)$ *and* $e = 65537$*, it is hard for any probabilistic polynomial-time algorithm to find the e-th root modulo N of a random* $y \leftarrow_R \mathbb{Z}_N^*$ *[135].*

Algorithm 4.3.4: Decrypt runs on secret key sk and ciphertext c, to produce message m.

input : $\mathsf{sk} := (p', q'), c$
$k_0, k_1, G, H \leftarrow \texttt{params}(1^\lambda)$ `// OAEP parameters`
$N := p'q'$
$\phi(N) := (p'-1)(q'-1)$
$d := e^{-1} \bmod \phi(N)$ `// recover` d `using EEA`
$m'' := c^d \bmod N$
$X := \lfloor c'' \cdot 2^{-k_0} \rfloor$ `// Extract` X
$Y := m'' \bmod 2^{k_0}$ `// Extract` Y
$r := Y \oplus H_{k_0}(X)$ `// Recover` r
$m' := X \oplus G_{n-k_0}(r)$ `// Recover padded message`
$m := m' \cdot 2^{-k_1}$ `// Remove padding`
output : m

The BBS Shortcut Assumption states that given (N', e) *and terms* (x, x_0, x_{-t})*, where* $N' \leftarrow_R \mathsf{Gen}\left(1^\lambda\right)$ *is a randomly sampled Blum integer,* $e = 65537$*,* x *is a randomly sampled integer such that* $x \in \mathcal{QNR}_N^{-1}$*,* $x_0 := x^2 \bmod N$*, and* x_{-t} *is the term* $t+1$ *steps before* x_0 *in a BBS_CSPRNG sequence, it is no easier to find the e-th root of a random* $y' \leftarrow_R \mathbb{Z}_{N'}^*$ *than to find the e-th root modulo N of a random* $y \leftarrow_R \mathbb{Z}_N^*$ *in a standard RSA instance, without learning the factors* $N' = pq$*.*

We now analyse this security assumption, in order to relate it to the RSA assumption that RSA with OAEP relies on [72].

Recall that $P = (x, x_0, x_{-t})$ consists of a randomly sampled integer x, and two terms x_0, x_{-t} which by construction are part of the BBS-CSPRNG sequence, and hence are pseudorandom. As we will see in Lemma 4.4.1, the relation between these integers exactly relates to the evaluation of the BBS-CSPRNG sequence, which allows N' to be factored, and cannot be evaluated in time less than t, for some $t \in \mathbb{N}$. The crux of the assumption is that x_{-t} is only related to the terms x and x_0 by the repeated squaring property, which allows the Blum integer N' to be factored. Therefore, we assume that these values cannot be used together to break

the RSA assumption using an alternative method to repeated squaring.

The key insight of TIDE is the technique we use to factor the RSA modulus N. We use Fermat's factorisation method, a technique to factor an odd composite number $N = pq$ in exponential time [59]. The method requires finding x and x' such that $x^2 - x'^2 = N$ is satisfied. Then the left-hand side can be expressed as a difference of squares $(x - x')(x + x') = N$.

Fermat's method can be extended to finding x and x' to satisfy the following weaker congruence of squares condition $x^2 \equiv x'^2 \bmod N$, where $x \not\equiv \pm x'$. This congruence can be expressed as $(x - x')(x + x') \equiv 0 \bmod N$. Finding a congruence of squares forms the basis for several sub-exponential sieving-based factorisation algorithms [59]. However, if x and x' in a congruence of squares are known, then factoring N can be done in polynomial time, as we shall see.

We now prove TIDE is a timed-release encryption scheme satisfying correctness and security, beginning with the correctness of TIDE.

Theorem 4.4.1 *TIDE is correct.*

First, suppose Algorithm 4.3.1 Setup has been run, such that the following parameters have been generated: a public key N, puzzle $P = (x, x_0, x_{-t})$ and time parameter t, and a secret key $\mathsf{sk} = (p, q)$. Next, let a ciphertext c be computed on a message m following the correctness game, Game 5. Then, consider the following statement:

For any message $m \in \{0,1\}^*$, $\mathsf{Decrypt}\Big(\mathsf{Encrypt}\big(\mathsf{N}, m\big), (p, q)\Big)$ outputs m, where Encrypt and Decrypt are described in Algorithms 4.3.3 Encrypt and 4.3.4 Decrypt respectively. This corresponds to the statement that the RSA cryptosystem with OAEP is correct, which is known to be true [72].

What remains is to prove that Solve outputs $\mathsf{sk} = (p, q)$. We structure this proof as a sequence of arguments based on the preliminaries given in Section 3.2.

First we must prove that Algorithm 4.3.1 Setup correctly selects the term x such that $x \in \mathcal{QNR}_N^{-1}$.

Corollary 4.4.1 *(Of Theorem 3.2.4). The while loop on lines 8-12 of Algorithm 4.3.1 Gen selects $x \in \mathcal{QNR}_N^{-1}$ with overwhelming probability.*

Proof: The while loop on lines 8-12 of Algorithm 4.3.1 Gen selects a quadratic non-residue with Jacobi Symbol equal to -1 by running a series of Bernoulli trials with probability $\mathrm{P}\left(x = \mathcal{QNR}_N^{-1}\right) = \frac{1}{2}$. This forms a geometric distribution $G \sim \mathrm{Geo}(\frac{1}{2})$. Therefore, we can expect to find $x \in \mathcal{QNR}_N^{-1}$ in $\mathbb{E}\{G\} = 2$ trials. As the number of trials increases, the probability of repeatedly failing to sample an $x \in \mathcal{QNR}_N^{-1}$ tends to 0. □

Second, we recall from Theorem 3.2.5 that $\alpha \equiv \sqrt{r} \bmod p$ can be found by calculating $\alpha \equiv r^{\frac{p+1}{4}} \bmod p$. We use this in conjunction with the CRT to prove that Algorithm 4.3.1 Setup correctly calculates the term x_{-t}, which is the tth principal square root of x_0.

Theorem 4.4.2 *The Algorithm 4.3.1 Setup correctly calculates the tth principal square root x_{-t} of the seed x_0.*

Proof: Let $\omega = \frac{p+1}{4}$. If Algorithm 4.3.1 Setup provides the seed term $x_0 \in \mathcal{QR}_N$, then, by Theorem 3.2.5, the tth principal square root of $x_0 \bmod p$ is $\alpha_t := x_0^{\omega^t} \bmod p$ and the tth principal square root of $x_0 \bmod q$ is $\beta_t := x_0^{\omega^t} \bmod q$. Then, the Chinese Remainder Theorem (Chapter 3 Theorem 3.2.1) is used to calculate:

$x_{-t} := [\alpha_t q(q^{-1} \bmod p) + \beta_t p(p^{-1} \bmod q)] \bmod N.$ □

Third, we prove that Algorithm 4.3.2 Solve correctly calculates the term $x' \in \mathcal{QR}_N$ using Algorithm 3.1.1. To see this, we first show in Theorem 4.4.3 that Algorithm 3.1.1 Square and Multiply correctly calculates the term x_i, where $x_i \equiv x_0^{2^i} \bmod N$, and then we show in Theorem 4.4.4 that Algorithm 4.3.2 Solve calculates $\mathtt{gcd}(x' - x, N)$ to recover a non-trivial factor of N [131].

Theorem 4.4.3 *Algorithm 3.1.1 Square and Multiply correctly calculates the term x_i, where $x_i \equiv x_0^{2^i} \bmod N$.*

Proof: The input to calculate the term x_i in Algorithm 3.1.1 Square and Multiply is $(x_0, 2^i, N)$, where $x_0 \in \mathcal{QR}_N$ is the seed term, and $N = pq$, where p and q are distinct odd primes. By Definition 3.2.2, selecting $x_0 \in \mathcal{QR}_N$ can be done by uniformly selecting $x \in \mathbb{Z}_N^*$ and setting $x_0 \equiv x^2 \bmod N$. Consider the base case when $i := 1$. The algorithm proceeds as follows: d is set to 1 and the exponent $b := 2^1$ is set to the binary string $B = 10$. Next, the algorithm enters the for loop on the first iteration. On the first iteration j is the first digit of B, which is 1. Next $d := 1$ is squared to output 1. Then the first conditional **if** statement is met as $j = 1$, therefore $d := 1 \cdot x_0 = x_0 \bmod N$, and the first iteration of the loop is done. On the second iteration j is the second digit of B, which is 0. Next, as d was set to x_0 on the first iteration d is now set to $x_0^2 \bmod N$ on the second iteration. The first conditional **if** statement is not met, and the loop terminates as the final digit of B was processed. The algorithm then returns $d := x_1 \equiv x_0^2 \equiv x_0^{2^1} \bmod N$, as required. Therefore, the base case is true. By the inductive hypothesis we claim that for any $i := k$, the loop invariant of Algorithm 3.1.1 returns the term $x_0^{2^k} \bmod N$ after k iterations. Therefore after k iterations, where b was set to 2^{k+1}, Algorithm 3.1.1 will have $d := x_0^{2^k} \bmod N$, and j will be the final digit of $B := 10\ldots0$. For any k, the variable B will be a binary string starting with the digit 1 followed by a trail of k digits equal to 0. Therefore, after the first iteration of the for loop all remaining $j \in B$ will be 0. Thus, at the $k+1$ iteration of the for loop d will be set to $x_k^2 \bmod N$, and by definition $x_k^2 \equiv x_{k+1} \equiv x_0^{2^{k+1}} \bmod N$. Finally, Algorithm 3.1.1 will terminate at the $k+1$ iteration as the final digit of B was processed, and the algorithm will return $d := x_0^{2^{k+1}} \bmod N$. $\square$

Theorem 4.4.4 *Let N be a Blum integer. If x and x' are known such that $x^2 \equiv x'^2 \bmod N$, where $x \not\equiv \pm x' \bmod N$, then the non-trivial factors of N can be recovered in polynomial time.*

Proof: (Theorem 4.4.4.) As x and x' are distinct we have $x^2 \equiv x'^2 \bmod N$. This implies that $pq \mid x^2 - x'^2$. As p and q are both prime this indicates that $p \mid (x-x')(x+x')$ and $q \mid (x-x')(x+x')$. Also, because p is prime it must be the case that $p \mid (x-x')$ or $p \mid (x+x')$. Similarly, it must be the case that $q \mid (x-x')$ or $q \mid (x+x')$. Without loss of generality, assume that $p \mid (x-x')$ is true and that $q \mid (x-x')$ is true. This implies that $pq \mid (x-x')$, which indicates that $x \equiv x' \bmod N$. This is a contradiction because x and x' are distinct. Then it must be the case that $p \mid (x-x')$ and $q \nmid (x-x')$. Therefore, one of the factors of N can be recovered by calculating $p' := \mathsf{gcd}(x-x', N)$ using Euclid's Extended Algorithm, and the other factor of N can be recovered by calculating $q' := \frac{N}{\mathsf{gcd}(x-x',N)} = \frac{N}{p'}$. □

We now prove that Solve outputs $\mathsf{sk} = (p,q)$, and hence Theorem 4.4.1: the correctness of TIDE.

Proof: (Theorem 4.4.1) For any pk, sk, and puzzle P generated by Setup, we show that sk can be recovered by Solve. More precisely, let $N = pq, P := (x, x_0, x_{-t}), t$ be output by Setup, before being input into Algorithm 4.3.2 Solve. Algorithm 4.3.2 Solve will calculate the term x' by entering the following parameters $(x_{-t}, 2^{t-1}, N)$ into Algorithm 3.1.1, which will output $x' := x_{-t}^{2^{t-1}} \bmod N$. The term x' is guaranteed to be correct by Theorem 4.4.3 and is guaranteed to be in $\mathcal{QR}_N$ by Definition 3.2.2, and hence we have that $x \in \mathcal{QNR}_N^{-1}$ and $x' \in \mathcal{QR}_N$.

This guarantees that x must be distinct from x'. Therefore, by Theorem 4.4.4, calculating $p' = \mathsf{gcd}(x - x', N)$ will recover one factor of N using Euclid's Extended Algorithm, and the other factor can be recovered by calculating $q' = \frac{N}{\mathsf{gcd}(x-x',N)}$.

□

Theorem 4.4.5 *TIDE is a secure TRE scheme under the RSW, RSA and BBS-shortcut assumptions.*

Proof: In order to prove TIDE secure, we show that no adversary can win the security game in Section 4.1.2 with more than negligible probability. That is to say, we show that two messages encrypted using public key (N, e) are indistinguishable under a chosen plaintext attack, when the adversary is bounded by t computation time.

We first prove the following statement.

Lemma 4.4.1 *Given any (N, P, t) output by Algorithm 4.3.1 Setup, the RSA modulus N cannot be factored in time less than t, with more than negligible probability.*

Proof: Let N be a random Blum integer and P be a puzzle output by Algorithm 4.3.1 Setup. Note from Algorithm 4.3.1 that $P = (x, x_0, x_{-t})$, where $x \in \mathcal{QNR}_N^{-1}$, $x_0 \equiv x^2 \bmod N$, and x_{-t} is the tth square root of x_0. To factor N in time less than t, a pair of integers (p^*, q^*) must be computed, such that $p^* \neq 1, q^* \neq 1$, and $p^* q^* = N$, in less than t sequential steps.

We split the proof into two parts: i) Attempts to compute an x', where $x' \equiv \sqrt{x_0} \bmod N$ and $x' \in \mathcal{QR}_N$, and ii) Attempts to recover the non-trivial factors of N using a method that does not use x'.

We start by proving part (i): that computing x' in time less than t reduces to the RSW time-lock assumption. If Solve is honestly run, then $x' := x_0^{2^{t-1}} \bmod N$ is calculated using Algorithm 3.1.1 with the input $(x_{-t}, 2^{t-1}, N)$. By the RSW time-lock assumption calculating x' using Algorithm 3.1.1 requires $t-1$ sequential steps. Finding a PPT algorithm $\mathcal{E}_{<t}$ to

evaluate x' in less than $t-1$ sequential steps contradicts the RSW time-lock assumption. Therefore, it is not possible to compute an x' in fewer than $t-1$ steps with non-negligible probability.

Next, we prove part (ii): With overwhelming probability, an adversary can't recover the non-trivial factors of N in less than t time without computing x'. We show that factoring N faster than sequential squaring (i.e, in fewer than t sequential steps) reduces to an open problem. First note that N is a Blum integer, which is an RSA modulus that is the product of Gaussian primes. Therefore, we assume N cannot be factored by any PPT algorithm with more than negligible probability.

Giving $\mathcal{A}$ either (N,x,x_0,t) or (N,x_{-t},t) also reduces to a standard factoring assumption, as seen in Section 4 of Rabin [131]. What remains is to show that giving an adversary all of the puzzle P does not allow them to factorise N. To see this, note that x_0 can be trivially obtained from x, and that by construction x_{-t} and x_0 are terms in a BBS_CSPRNG sequence [26]. Knowledge of these terms does not allow factorisation of N faster than sequential squaring unless $x_{-t}^{2^{\lambda(\lambda(N))}}$ mod N is calculated efficiently. This is an open problem given by Theorem 9 of Blum et al. [26, 71, 81].

Therefore, the only way a PPT algorithm could factorise N given (pk,P,t) with non-negligible probability is to sequentially evaluate x' and subsequently recover the factors by calculating $p' := \mathtt{gcd}(x-x',N)$ and $q' = \frac{N}{p'}$. □

Now that we have shown that factoring can't be done with non-negligible probability without spending at least t computational time, we can obtain a proof by contradiction using various security assumptions.

Assume that there exists an adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ who can gain a non-negligible advantage in the $\mathbf{Adv}_{\mathcal{A}}^{\mathrm{TRE}}$ game defined in Definition 6. For this to hold, it must be the case that an algorithm D exists such that both of the following occur with non-negligible probability: a) D can decrypt, and b) D runs in time less than t.

Lemma 4.4.1 tells us that if the adversary $\mathcal{A}$ factors a Blum integer N output by Algorithm 4.3.1 in time less than t with more than negligible probability, then the RSW time-lock assumption is broken, and hence we have a contradiction. Now suppose that $\mathcal{A}$ gains a non-negligible advantage in the TRE security game without factoring. As the underlying encryption scheme is IND-CPA secure, for an adversary to gain a non-negligible advantage in distinguishing between the messages m and m' without using the additional information contained in P would break the IND-CPA security of RSA-OAEP, and hence contradict the RSA assumption. Finally, if an adversary manages to gain a non-negligible advantage *using* the information contained in P then this contradicts the BBS shortcut assumption presented in Definition 4.4.1. Therefore, no such algorithm D can exist, and hence it is not possible for an adversary to win the security game with more than negligible probability.

□

A note on lossiness

We end this security section with a short discussion on lossiness in TIDE. Lossiness refers to the loss of information that occurs when using a function for which the image is smaller than the domain, as we illustrate in the following example. In RSA, a 'lossy key' refers to a pair (N,e) such that $e|\phi(N)$. There is information loss when using a lossy key rather than a 'standard key' (i.e. one where e does not divide $\phi(N)$), as the map $e \to e^x$ becomes e-to-one on $\mathbb{Z}_N^*$, rather than one-to-one [147].

This is particularly relevant when using a fixed value of e, as we do in our implementation, as using a fixed value of e (and explicitly $e = 65537$) as part of a lossy key has been shown to weaken the security of RSA-based signature schemes such as FDH signatures [91], and more importantly to repeated squaring when using a Blum integer [143]. There exists significant work in the literature on the difficulty of distinguishing between the cases when $\gcd(e,\phi(N)) = 1$ and when $e|\phi(N)$ in the case when $e < N^{1/4}$, which is known as the ϕ-

hiding assumption [41, 143]. This is particularly relevant in the case where N is generated through a distributed setup, as we discussed in Section 2.1.3. However, due to the trusted setup of TIDE, we are able to include a check that e does not divide $(p-1)(q-1)$ in Algorithm 4.3.1, and hence ensure a lossy key is not chosen.

4.5 Implementation

In this section we describe the implementation and performance analysis of our TIDE construction. The software implementation is written in Python 3 and the code is publicly available at https://github.com/wsAJMYbR/tide.git.

Our testing platform consisted of two different hardware environments: a Raspberry Pi cluster, and a desktop PC. The Pi cluster consisted of four Raspberry Pi 3 Model B computers networked together. Each Pi node utilises a quad-core 1.2 GHz CPU, with 1 GB of available memory. This enabled us to run experiments on four different modulus sizes in parallel. The use of Raspberry Pi devices provides an affordable and ubiquitously available device with a consistent configuration. This facilitates the replication of our experiments and comparison with other delay-based schemes. Furthermore, as Raspberry Pi devices are lower power, they represent a lower bound for hardware that may reasonably be expected to be used in practice outside of embedded applications.

We also executed performance tests on a consumer grade desktop PC. The machine used a quad-core 3.2 GHz Intel i5 processor, with 16 GB of available memory. We wished to confirm that the statistical properties remained constant over different hardware types. Additionally, this dataset provides a more pragmatic view of performance on commercial hardware.

Figure 4.1 demonstrates our first experiment, which shows how the run time of Setup, Solve, Encrypt and Decrypt is impacted by the time parameter t for a 2048 bit modulus

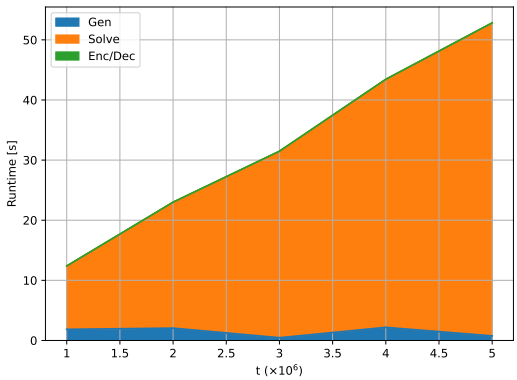

Fig. 4.1 The impact of adjusting parameter t on the run time of Setup, Solve, Encrypt, and Decrypt algorithms when run on a desktop PC with a 2048 bit modulus. The primary effect is on Solve, which displays a linear increase.

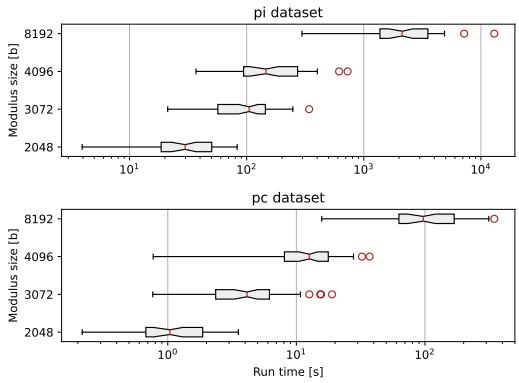

Fig. 4.2 The spread of setup time across modulus sizes and machines. Setup time increases in response to an increase in modulus size. The dispersion of run times is similar across different devices.

when run on a desktop PC. The figure shows that as t increases, the run time for Solve also increases in a predictable linear manner. The linear variance in run time of Algorithm 4.3.2 Solve as t varies supports the RSW time-lock assumption in Definition 3.1.1. Furthermore, we see that Algorithm 4.3.1 Setup and Algorithms Encrypt and Decrypt remain consistently low, regardless of the size of t. This is expected as both algorithms reduce the parameter t by the group order using the Fermat-Euler Theorem 3.2.2. We also observe that Setup has minor variations in the run time when compared to Solve and Encrypt and Decrypt. This is a result of the randomised nature of Setup in comparison to the deterministic behaviour of Solve, Encrypt, and Decrypt. This experiment confirms that in practice, increasing the time parameter leads to a linear increase in the time of Solve, and has a negligible effect upon the other algorithms, which is exactly what is desired for a TRE construction.

For our next experiments we select $t = 5 \times 10^6$ to provide a total run time appropriate for repeat testing. We performed experimentation over four modulus sizes $m \in \{2048, 3072, 4096, 8192\}$ bit, selected to cover common modulus sizes in use. For each modulus size, we run 70 experiments, which allows us to estimate values with a 90% confidence

interval with a 10% margin for error. The 8192 bit modulus is included as an edge case to demonstrate performance at the upper bound present in real world applications.

In Figure 4.2, we plot the spread of run times for the Setup algorithm. The primary metric of interest is the spread of the stochastic algorithm. For both the Pi and PC datasets, an increase in the modulus size increases the median run time. However, there is some overlap between modulus sizes, particularly between $m = 3072$ bit and $m = 4096$ bit. This discrepancy can be attributed to more efficient computation afforded when $\log_2 m \in \mathbb{N}$. In particular, the Miller-Rabin primality implementation can use a fast Fourier transform which is most efficient when dealing with powers of two [127]. The dispersion of the data points follows a similar pattern across both data sets, with an offset in median speed afforded by the relative difference in processor speed.

In Figure 4.3 we plot the run times of the Solve and Encrypt and Decrypt algorithms for both datasets against the modulus size. We use the run time means as a metric to eliminate variations caused by other processes on the machine, which we assume to be Gaussian. This leaves us with a more accurate indication of the run time of the deterministic algorithms. As with the Setup algorithm, we see similar increases in run time as a function of modulus size for both Solve and Encrypt and Decrypt. However, we note the large difference between the run times of Encrypt and Decrypt and Solve. Above each bar we plot the ratio of the run time of Encrypt and Decrypt to the run time of Solve. We see that, while there is a small increase in the ratio as the modulus size increases, the difference between the two remains marked. Even at the edge case, when Solve runs in excess of four hours on the Pi when $m = 8192$ bit, Encrypt and Decrypt don't exceed 30 s. For most practical cases, Encrypt and Decrypt often results in sub-second evaluations, demonstrating practicality even in constrained environments. As Solve factors N, our TIDE construction is single-use for each setup. However, as we have seen in our experiments, this property is not an obstacle for practical use in applications such as auctions. Even in more computationally constrained

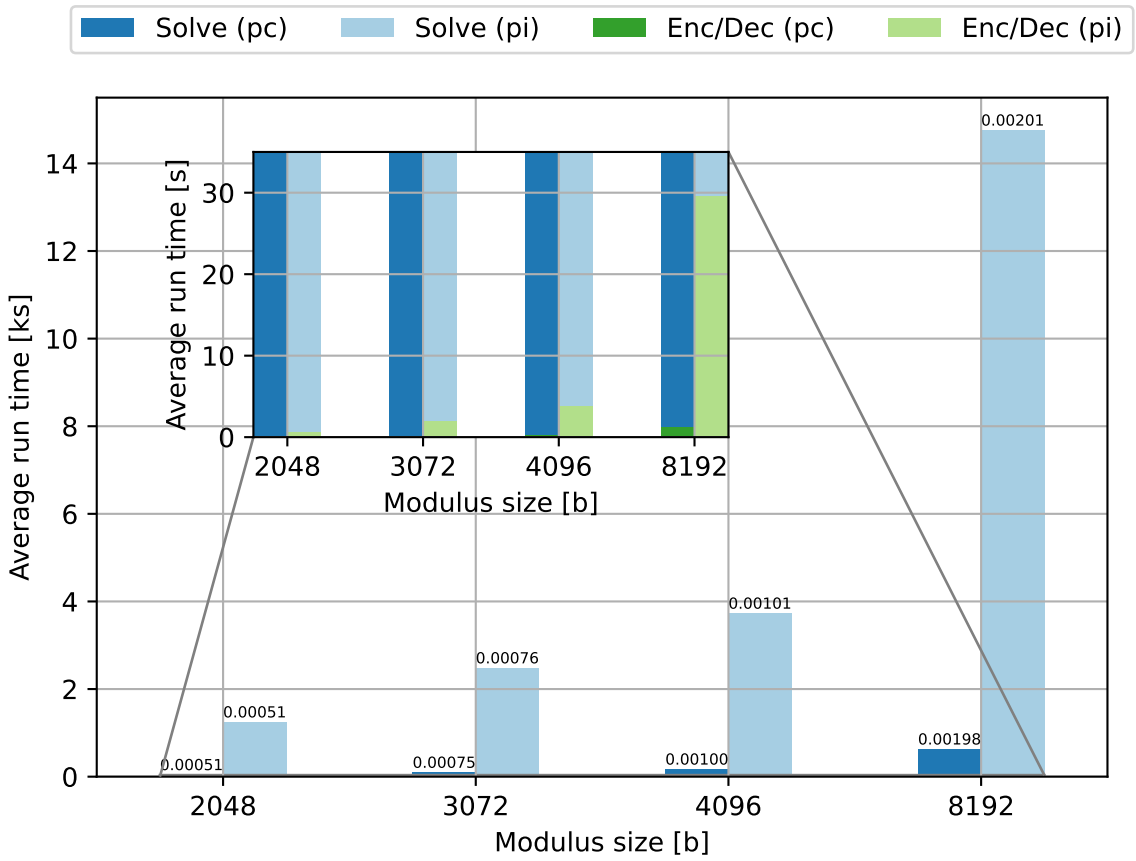

Fig. 4.3 Encrypt and Decrypt take significantly less time to run than Solve across modulus sizes. The inset shows a zoomed in view of the bar chart which is necessary for the effect of Encrypt and Decrypt to be observed. Above each bar is the ratio Solve to Encrypt and Decrypt run time.

environments such as the Raspberry Pi, the Solve and Encrypt and Decrypt algorithms do not require an impractical time cost. We would recommend for a standard use case to use $m \leq 4096$ to keep the setup run time within an appropriate bound. This leaves the value for t as the primary parameter dictating the length of the delay. As we saw in Figure 4.1, the value for t can be set with reasonable accuracy to introduce a desired delay for the target hardware.

4.6 Application to Auctions

In this section, we summarise how TIDE can be used in practice for the application of auctions, and the benefits this has when compared to other approaches to auctions. We will first discuss the main alternative to delay-based auctions, and the trade-offs that arise when compared to delay-based auctions, before briefly discussing how TIDE fits into the delay-based literature of auctions.

One common approach to designing auction protocols is to use secure multi-party computation techniques to compute an auction where all parties submit an input, and together compute a shared output. This is demonstrated in a well-known paper 'Secure Multiparty Computation Goes Live' [30], where such techniques from multi-party computation were used to implement a nation wide double auction in Denmark. In a double auction, sellers indicate how much of an item they are willing to sell at certain price points, whilst buyers indicate how much of the same item they are willing to buy at each price point. Using this information, the *market clearing price*, i.e., the price per unit of this item that parties agree upon is computed, allowing transactions to be be made at this price point.

This approach allows advanced functions such as the market clearing price to be computed upon inputs, whilst providing a higher level of privacy than that of TIDE, due to the inputs of each party remaining secret. However, this MPC approach utilises a different framework to that of TIDE, which comes with different assumptions: explicitly, it is required that all parties are online at the same time when carrying out the protocol, and often a significant computational burden is placed upon participating parties. As such, whilst linked by the application of auctions, we view such work as tangential.

We will instead restrict our attention to sealed-bid auctions, giving a brief summary here, and referring the reader to the exposition in Chapter 2. Sealed-bid auctions allow bidders to secretly submit a bid for some goods without learning the bids of any other party involved

until the end of the auction. Importantly, this setting circumvents the requirement for all bidders to be online at the same time. Time-lock puzzles (Section 2.2.1), can be used to submit bids without requiring a setup to be performed, however such bids must be decrypted individually, leading to scaling issues. Homomorphic time-lock puzzles (Section 2.2.2) and delay encryption (Section 2.2.4) offer solutions which address this scaling issue, but a practical construction that can be used for auctions has yet to be built.

In this chapter, we demonstrated that TIDE is a timed-release encryption scheme that improves upon these scaling issue at the cost of a trusted setup. TIDE provides a solution which is based upon well known, trusted cryptographic building blocks, using number-theoretic results in order to derive the secret decryption key. TIDE runs efficiently in practice: We demonstrated in Section 4.5 that TIDE has a setup that takes on average 1 to 2 seconds on a consumer-grade desktop PC for a 2048-bit RSA modulus, and negligible time taken for encryption and decryption. Importantly, only one delay is required regardless of the number of bids placed in the auction. By releasing the encryption key in the public parameters, all interested parties can bid in the auction using Algorithm 4.3.3. Algorithm 4.3.2 will then provide the relevant decryption key, which can be used to recover all bids efficiently.

This efficiency, combined with the publicly available code, makes this tool a practical choice in the context of delay-based auctions.

4.7 Conclusion

In this chapter we presented TIDE, a novel, efficient and easily implementable approach to building a TRE scheme.

TIDE integrates RSA encryption into an RSW-based TLP using powerful results from number theory. On top of being a concrete construction, the novelty of TIDE lies in its ap-

proach to deriving the secret key. We have shown that TIDE is demonstrably implementable, and practically efficient in the case where a trusted setup is tenable.

Future work The definitions which we introduced in Section 4.1.2 contain a nuance in the Setup algorithm, which can be used to enable further potential applications, which have yet to be explored in the context of delay-based encryption. We develop this idea in the next chapter, in which we present a follow-up work where we introduce the notion of *TRE with implicit authentication*. We will show that this new variant of timed-release encryption allows for finer grained control of data, and we will demonstrate that this enables delay-based cryptography to be applied to applications such as whistleblowing.

Additionally, the technique of using BBS with RSA-OAEP in order to factor an RSA modulus after a delay may be of use in further delay-based constructions. One can use this technique to allow the party who runs the delay to receive a reward, in order to incentivise the solver to run the computation. We will use this technique again in each of the following chapters, demonstrating that it is applicable in various scenarios.

Chapter 5

Timed-Release Encryption with Implicit Authentication and Applications

Contents

In this chapter, we build upon the work of the previous chapter by expanding the definition of TRE, introducing a novel security notion which we term implicit authentication. This allows us to construct a scheme which offers the encryptor greater flexibility over the release

of information, which we propose as a method for improving the safety of vulnerable parties such as whistleblowers. The work in this chapter appears in [112], which was a joint work with Angelique Loe, Christian O'Connell, and Elizabeth Quaglia.

5.1 Introduction

In 2013, Edward Snowden leaked highly classified information from the National Security Agency [140, 152], at great personal risk. Other recent cases of whistleblowing include the Panama papers [128], the Paradise papers [22], and the Pandora papers [105]. Leaking information subjected the whistleblowers to personal danger due to the power and influence of the organisations whose data was leaked. In the case of the Panama papers, the whistleblower claimed their 'life was in danger' [77].

In this chapter, we construct a cryptographic tool based on timed-release encryption [53], which can augment existing tools for whistleblowers, such as SecureDrop [3]. Our goal is to introduce techniques from delay-based cryptography into the the release of sensitive information, which we believe has potential to make the practice of whistleblowing safer. We model our solution on the Edward Snowden case, in which all classified material was destroyed before arriving in Russia, in order "To protect himself from Russian leverage" [2].

We propose a construction which offers the concept of delay-based encryption for whistleblowers, to allow them to rely on cryptographic assurances rather than the trust of a journalist or ombudsman. The technique we introduce allows sensitive information to be encrypted in such a way that a) there is a predictable delay between the *receipt of the ciphertext* encapsulating the leaked information and the *release* of the information, and b) there is no way an adversary can forge a chosen document to insert alongside the genuine documents. The motivation behind using a delay in this context is to afford the whistleblower time to destroy all classified material after encapsulating the material, and hence ensure their safety. In the

Snowden case, the delay would have allowed the passage to a safe harbour country without the sensitive information being decrypted until a specified time.

The core idea of our approach is to have two separate keys, an encryption key and decryption key, the latter being encoded as the solution to a *challenge*. This challenge will be similar to the puzzle used in Chapter 4, and has the property that the delay starts once the challenge is distributed. The whistleblower generates a public key pair and keeps the encryption key secret, encapsulating its corresponding decryption key with a time-delay, such that it takes at least t time to recover. We provide the whistleblower with the ability to encrypt and distribute ciphertexts under the encryption key, without 'starting the clock' on the time-delay. At a time of their choosing, the whistleblower can distribute the challenge, upon which a sequential computation taking time t will output the decryption key for the ciphertexts. Due to the asymmetric nature of the encryption key and decryption key, once the decryption key is recovered, the whistleblower will still hold the exclusive ability to encrypt more data at a later date.

We formalise this through the introduction of a security property which we term *implicit authentication*, in order to provide the journalist receiving the leaked information with assurance that an adversarial party cannot encrypt a document of their choosing under the encryption key.

Chapter Layout In the remainder of this section we detail our methodology in approaching this problem, outlining the security goals we desire and providing an overview of our construction, before discussing relevant related work. In Section 5.2 we formally define the primitive *TRE with implicit authentication* (TRE-IA), giving game-based definitions of the required security properties. In Section 5.3 we present our construction for a TRE-IA scheme, which is based upon the BBS-random number generator and RSA-OAEP encryption. In Section 5.4 we prove our construction is secure under the definitions given in Section 5.2.

In Section 5.5, we conclude this chapter by discussing our ideas for how this work can be further developed in the future, in order to be used and trusted in practice.

5.1.1 Technical Overview

The goal of this chapter is to explore how a time-delay can be used by vulnerable parties such as whistleblowers in order to make the distribution of sensitive material safer. In order to do so, we introduce a novel construction based upon a clear set of properties, which we can implement in practice and which may augment existing whistleblower tools. Therefore, we define the following security goals that we believe may be helpful to a whistleblower based on the real life cases of the Panama papers leak [77] and the Edward Snowden leak [140, 152].

1. An *adjustable time delay*: this will allow the whistleblower to destroy all materials that can be used against them. It also allows the whistleblower to have a configurable amount of time to reach a place of safety.

2. *Maximum flexibility*: this will allow the whistleblower to determine a) when they can encrypt and distribute messages and b) when they can 'start the clock' for evaluating the delay. This is achieved through the separation of the ciphertexts and the *challenge*. When the challenge is distributed this starts the clock.

3. *Implicit authentication*: this ensures that no other entity can generate a document of their choice to insert into the leak.

Property 1 can be useful for a whistleblower to protect themselves from the dangers associated with carrying sensitive material. Property 2 allows a whistleblower to gather various different pieces of evidence over time and encrypt and distribute this evidence to journalists. The whistleblower can also ensure that the journalists cannot yet leak the

material until a time delay has passed, thus mitigating risks to the personal safety of the whistleblower. We believe it is crucial that the whistleblower remains in control of all aspects of the system, and by giving the whistleblower the freedom to distribute ciphertexts *without* 'starting the clock' on the time delay, we minimise the trust placed in journalists, and provide the whistleblower with fine-grained control of when the documents are leaked.

Property 3 ensures that once the decryption key has been derived, it cannot be used by third parties to obtain the encryption key to encrypt their own messages. Without this property, it is possible for a third party to choose and encrypt their own fake material, and claim it is from the whistleblower.

We now describe the methodology of how we designed our construction.

Building our construction. Our base property, 1, can be achieved using various primitives that we discussed in Chapter 2, most notably time-lock puzzles (TLPs) and timed-release encryption (TRE). We will start our discussion with TLPs.

Recall that a time-lock puzzle [136] encrypts a message to the future, in such a way that once a solver spends a predictable amount of time evaluating the encrypted message, they obtain the plaintext message.

One could think of using the naive approach of simply encrypting each message as a TLP and passing it to a journalist. This achieves property 1, however it limits the whistleblower to the condition that they encrypt all materials at once, and allows adversarial parties to impersonate the whistleblower. As we wish for the whistleblower to have a finer control of the encryption process we see that the latter method has limitations. For example, the whistleblower may wish for multiple messages to be encrypted. This is a reasonable assumption as the Panama papers exposed over eleven million leaked documents [128] and the Paradise papers exposed over thirteen million leaked documents [22]. Using only a TLP results in the loss of property 2, that is, the whistleblower loses the element of maximum flexibility. It is more appropriate to follow the TRE approach of Section 2.2.3 and Chapter 4,

using a symmetric key as the solution to a TLP, and then distribute ciphertexts separately. This gives us a solution which is close to ideal, as ciphertexts can be distributed to journalists, with guarantees both time delay and flexibility as to when the whistleblower starts the clock. In other words, we can achieve properties 1 and 2 using this approach, however we are missing the implicit authentication property described in 3.

Our approach to fixing this problem is to require that the encryption key and decryption key are different, and more importantly, that one *cannot derive the encryption key from the decryption key*. If this is the case, a whistleblower can encapsulate the decryption key of a public-key encryption scheme, whilst keeping the encryption key secret.

As a starting point, we use the TRE definitions given in Section 4.1.2. However, we note that our definitions, and indeed those of Chvojka et al. [53], do not give us the desired property of a secret encryption key, nor do they necessarily imply that it is impossible to derive the encryption key from the decryption key.

Therefore, to achieve these goals simultaneously we propose a variant of timed-release encryption, which we term TRE with implicit authentication (TRE-IA) and instantiate this with a construction based upon the BBS-CSPRNG random number generator [26], and the RSA-OAEP encryption scheme [21].

5.1.2 Related Work

Delay-Based Primitives We have discussed at length in Chapter 2 the various delay-based primitives that exist in the literature. In this section we will briefly discuss how these primitives relate to TRE-IA, and to our proposed applications. We have already discussed in the previous section the downsides associated with using TLPs both individually, and augmented with a symmetric key encryption scheme: In essence, TRE-IA improves upon this by using distinct (asymmetric) keys for encryption and decryption, as opposed to one

symmetric key. In both primitives decryption keys are recovered after the delay, in order to decrypt the ciphertext. But in TRE-IA only the decryption key is recovered. This leads to a different functionality: the whistleblower has control over what information is leaked (i.e., encrypted) as the sole holder of the encryption key.

We also note that the main difference between TRE and TRE-IA is that the former requires encryption to be public rather than only a prerogative of the whistleblower: In TRE-IA it is specified that the encryption key cannot be derived by the solver even when the decryption key is recovered. In a TRE-IA scheme assurance is provided through the property of implicit authentication that only the whistleblower can encrypt to the classic notion of a *public key* in a standard PKE scheme. In the following section, we will introduce the definitions of TRE-IA, and provide a more detailed comparison with the TRE definitions given in Chapter 4.

DE (Section 2.2.4) is distinct from TRE-IA for a similar reason: DE allows anyone to encrypt to the session ID, compared to just the whistleblower being in control of encryption in TRE-IA. Finally, the remaining delay-based primitives, i.e. PoSW and VDFs do not generate a ciphertext, or indeed have any method for encrypting data, and so are clearly distinct from TRE-IA.

Signcryption A cryptographic primitive offering a similar property to implicit authentication (IA) is Signcryption [157–159]. IA provides the receiving party with assurance that the encrypted documents were sent by the whistleblower. The IA property states that only the holder of the private encryption key can generate a legitimate ciphertext that can be correctly decrypted to a chosen message. In a similar fashion the concept of Signcryption was introduced to provide a single computation that would simultaneously provide the authenticity from a digital signature scheme (DSS), and the confidentiality from a public-key encryption (PKE) scheme. However, Signcryption does not consider the property of delay which is crucial to our TRE-IA scheme.

Tools for whistleblowers There exists a variety of tools which can provide protection to whistleblowers. The Tor [5] browser can be used to navigate the Internet anonymously, PGP [160] keys and encrypted email services can support the secure communication between a whistleblower and the investigative journalist, as well as end-to-end encrypted messaging services such as Signal [4]. Closely aligned to our intended end-goal is the SecureDrop [3] submission system, which enables whistleblowers to securely deliver documents containing leaked information. The novelty of our proposed solution in this space is the introduction of a delay, which provides, when combined with encryption, a time 'bubble' within which the whistleblower can reach safety. Indeed, we see TRE-IA as an addition to a whistleblower's toolbox featuring the property of a time-lock delay. In this context, we recognise that no tool is perfect and solutions which guarantee the safety of the whistleblower should be grounded in reality. Our proposed scheme represents a first, technological step towards introducing delay as a form of protection to whistleblowers.

We conclude this section by noting that the whistleblower use case is an illustrative example of how and why TRE-IA could be useful. We envisage TRE-IA being useful in further applications where a delay and the ability to control the release of sensitive information could help users in at risk situations.

5.2 Defining TRE-IA

We now provide the definition and properties of TRE-IA, which can be seen as an extension of the TRE primitive, and in particular of the definitions that we gave in 4.1.2. We deviate from the standard security model of TRE (see [53] and 4.1.2) in the following way: (i) to fit in with our model of the encryptor alone knowing the encryption key, we require that the encryptor runs setup. (ii) we separate the Setup algorithm into Setup and Gen to allow the encryptor more flexibility on when they choose the time parameter. (iii) we introduce the

new notion of implicit authentication. We provide an intuition of this security property, along with a detailed description of the game in the following section.

We follow the notation of Chapter 4, using $\mathcal{E}$ to denote the encryptor, $\mathcal{D}$ for the decryptor, and $\mathcal{A}$ for the PPT adversary. $\leftarrow_{\mathrm{R}}$ represents a probabilistic algorithm, and $\leftarrow$ represents a deterministic algorithm.

Definition 5.2.1 *A timed-release encryption scheme with implicit authentication is a tuple of algorithms TRE = (Setup, Enc, Solve, Dec) with the following syntax.*

$(\mathsf{IA.Setup}, \mathsf{IA.Gen}, \mathsf{IA.Enc}, \mathsf{IA.Solve}, \mathsf{IA.Dec})$, defined as follows.

- $pp, td \leftarrow_{\mathrm{R}} \mathsf{IA.Setup}(1^{\lambda})$. IA.Setup is an algorithm run by $\mathcal{E}$ that takes as input the security parameter 1^{λ} and outputs the public parameter pp and trapdoor td. $\mathcal{E}$ must keep td private. IA.Setup runs in time $\mathsf{poly}(\lambda)$.

- $e, d, C, t \leftarrow_{\mathrm{R}} \mathsf{IA.Gen}(pp, td, t)$. IA.Gen is an algorithm run by $\mathcal{E}$ that takes as input pp, td, t and outputs an encryption key e, decryption key d, and a public challenge C. IA.Gen runs in time $\mathsf{poly}(\lambda)$.

- $c \leftarrow_{\mathrm{R}} \mathsf{IA.Enc}(m, e, pp)$. IA.Enc is an algorithm run by $\mathcal{E}$ that takes as input a message m, encryption key e, and public parameter pp and outputs ciphertext c. IA.Enc runs in time $\mathsf{poly}(\lambda)$.

- $d \leftarrow \mathsf{IA.Solve}(pp, C, t)$. IA.Solve is an algorithm run by $\mathcal{D}$ that takes as input the parameters pp, C, t and outputs the decryption key d. IA.Solve requires t sequential steps to recover d with a run time of $(t)\mathsf{poly}(\lambda)$.

- $\{m', \perp\} \leftarrow \mathsf{IA.Dec}(c, d, pp)$. IA.Dec is an algorithm run by $\mathcal{D}$ that takes the ciphertext c, decryption key d, and public parameter pp and outputs plaintext m' or error $\perp$. IA.Dec runs in time $\mathsf{poly}(\lambda)$.

A TRE-IA scheme must satisfy the properties of *correctness*, *security*, and *implicit authentication*.

Correctness As with a TRE scheme, a TRE-IA scheme is *correct* if any encrypted message can be decrypted using the corresponding decryption key with overwhelming probability. Namely, in the context of TRE-IA, using the decryption key with a legitimate ciphertext in $\mathsf{IA.Dec}$ will recover the original message input into $\mathsf{IA.Enc}$. This is made precise in the Correctness game.

Correctness Game

1. $\mathcal{E}$ outputs the public parameter and trapdoor: $pp, td \leftarrow_{\mathrm{R}} \mathsf{IA.Setup}(1^{\lambda})$.
2. $\mathcal{E}$ outputs an encryption key, decryption key, challenge, and time parameter: $e, d, C, t \leftarrow_{\mathrm{R}} \mathsf{IA.Gen}(pp, td, t)$.
3. $\mathcal{E}$ computes the ciphertext on message m: $c \leftarrow_{\mathrm{R}} \mathsf{IA.Enc}(m, e, pp)$.
4. $\mathcal{D}$ recovers the decryption key: $d \leftarrow \mathsf{IA.Solve}(pp, C, t)$.
5. $\mathcal{D}$ decrypts the ciphertext: $m' \leftarrow \mathsf{IA.Dec}(c, d, pp)$.

A TRE-IA scheme is correct if $m = m'$ with probability $1 - \mathsf{negl}(\lambda)$.

Security In TRE-IA, *security* is defined as an indistinguishability game, as follows: Suppose an adversary $\mathcal{A}$ chooses two messages of the same length m_0 and m_1 and sends them to the encryptor $\mathcal{E}$. $\mathcal{E}$ chooses one of these messages at random, which it encrypts and sends to $\mathcal{A}$. The adversary then gets polynomial time to preprocess upon this ciphertext before receiving the challenge C. The adversary must then make a guess *before* t sequential steps are computed. A TRE-IA scheme is *secure* if no PPT adversary $\mathcal{A}$ can gain an advantage in guessing which message was chosen by $\mathcal{E}$. The key difference between this game when compared to the standard TRE security game given in the previous chapter is that the adversary is given polynomial time to preprocess on the two ciphertexts before gaining access to the challenge. This is made precise in the Security game.

Implicit authentication is the new property we introduce in TRE, and it captures an adversary being unable to *forge* a ciphertext for a message of their choice, hence providing an

Security Game

1 $\mathcal{E}$ outputs the public parameter and trapdoor $pp, td \leftarrow_{\mathrm{R}} \mathsf{IA.Setup}(1^{\lambda})$.
2 $\mathcal{E}$ outputs an encryption key, decryption key, challenge, and time parameter: $e, d, C, t \leftarrow_{\mathrm{R}} \mathsf{IA.Gen}(pp, td, t)$.
3 $\mathcal{A}$ selects two messages of the same length (m_0, m_1) for $\mathcal{E}$.
4 $\mathcal{E}$ uniformly selects $b \in \{0,1\}$, and encrypts m_b as $c \leftarrow_{\mathrm{R}} \mathsf{IA.Enc}(m_b, e, pp)$.
5 $\mathcal{A}$ runs a preprocessing algorithm $\mathcal{A}_0$ on the public parameter and the ciphertext, and stores $\mathsf{st} \leftarrow \mathcal{A}_0(pp, c)$.
6 $\mathcal{E}$ sends t and C to $\mathcal{A}$.
7 $\mathcal{A}$ runs a PPT algorithm $\mathcal{A}_1$ which outputs $b' \leftarrow \mathcal{A}_1(\mathsf{st}, C, t)$, where $\mathcal{A}_1$ must run in fewer than t sequential steps.

A TRE-IA scheme is secure if $b = b'$ with probability $\frac{1}{2} + \mathsf{negl}(\lambda)$.

implicit guarantee that ciphertexts are authentic. In the context of our motivating application, this property ensures that a malicious party cannot insert a document of their choice into the leak provided by a genuine whistleblower.

The idea of modelling security against a ciphertext forgery is inspired by the notions of plaintext integrity and ciphertext integrity [19, 23] in the symmetric encryption setting. More specifically, plaintext integrity states that it should be infeasible to produce a ciphertext decrypting to any message which the sender has not encrypted, and ciphertext integrity requires that it be infeasible to produce a ciphertext not previously produced by the sender, regardless of whether or not the underlying plaintext is 'new' [19].

However, these existing notions do not directly map to the asymmetric-key setting primitives such as TRE, since the adversary gains access to the secret decryption key after time t. This represents a challenge because it allows the adversary to select elements from the ciphertext space with non-negligible probability, and decrypt them to obtain a plaintext, and present this as a forgery. Whilst any message obtained this way will be not necessarily be 'meaningful', this approach makes a simple analogue of either ciphertext authenticity or plaintext authenticity difficult.

To overcome this, we took the approach of modelling our implicit authentication game as an encryption analogue of *selective* forgery [103, 124], a property used in digital signature schemes where an adversary first commits to a target message m^* and is later challenged to forge a signature for this target message. The key difference in our implicit authentication game is that the adversary is instead asked to output an encryption of the target message, rather than a signature.

We model the implicit authentication game as follows: We first ask the adversary to output a message m^* that they wish to encrypt. The adversary is then given the decryption key, and access to an encryption oracle. Finally, the adversary is asked to output a ciphertext c to the challenger. The adversary wins the game if c decrypts to the message m^*.

We make this precise in the Implicit Authentication game.

Implicit Authentication Game

1. $\mathcal{E}$ outputs a key pair e,d, the public parameters pp, a trapdoor td, and a challenge C: $pp,td \leftarrow_{\mathrm{R}} \mathsf{IA.Setup}(1^\lambda)$, $e,d,C,t \leftarrow_{\mathrm{R}} \mathsf{IA.Gen}(pp,td,t)$.
2. $\mathcal{E}$ sends the public parameter pp to the adversary $\mathcal{A}$.
3. $\mathcal{A}$ returns a target message m^* to $\mathcal{E}$.
4. $\mathcal{E}$ sends $\mathcal{A}$ the challenge C, time parameter t, and the decryption key d.
5. $\mathcal{A}$ is also given access to the encryption oracle $\mathcal{O}^{\mathsf{Enc}}$, which takes as input a message $m' \neq m^*$, and returns $c' \leftarrow \mathsf{IA.Enc}(m',e,pp)$ if the message is valid, and $\perp$ otherwise.
6. $\mathcal{A}$ returns a ciphertext c to $\mathcal{E}$.
7. $\mathcal{A}$ wins the game if $m^* \leftarrow \mathsf{IA.Dec}(c,d,pp)$.

A TRE-IA scheme has implicit authentication if $\mathcal{A}$ wins the game with probability no greater than $\mathsf{negl}(\lambda)$.

5.3 Construction of a TRE-IA scheme

In this section we provide a concrete construction of a TRE with implicit authentication. As in our TIDE construction in Chapter 4, the delay element of our TRE-IA is derived from the construction of the Blum Blum Shub CSPRNG [26], and the encryption component comes

from the RSA-OAEP PKE scheme. For the sake of brevity, we shall refer to our construction as the BBS-TRE.

Recall that implicit authentication states that without access to the encryption key an adversary should not be able to forge a ciphertext for a message of their choice. When RSA is used in practice it is standard procedure to use $e = 65537$ as the encryption key [15]. This does not allow for implicit authentication, as an adversary can guess this. Using any 'standard' fixed encryption key, or a key from a fixed small set will allow an adversary to guess this key, and hence encrypt a message as a ciphertext with more than negligible probability. As such, in contrast to our TRE construction presented in Chapter 4, we design our TRE-IA construction to choose e at random, to ensure that we obtain the implicit authentication property whilst still conforming to the NIST SP-800-56B standard for random public exponent key pair generation [15], Section 6.3.2. Using the BBS-CSPRNG provides an elegant solution to integrating random keys in a TRE-IA setup.

Next, we provide the notation required for the exposition of our BBS-TRE. In the pseudo code of our algorithms $:=$ indicates assignment, $=$ indicates equality, $\neq$ indicates inequality, $()$ indicates a tuple, and $//$ denotes a comment. The function $\texttt{prime}(j)$ outputs a random j-bit Gaussian prime. The function $\mathcal{U}(a,b)$ uniformly selects of an integer that is between $a,b \in \mathbb{Z}$, where $a < b$ and a,b are inclusive. Also, the symbol $\wedge$ indicates logical conjunction (and).

Our BBS-TRE is summarised as follows:

$$
\begin{aligned}
pp := (N, k_0, k_1, G, H), td := \phi(N) \quad &\leftarrow_{\mathrm{R}} \quad \mathsf{IA.Setup}(1^\lambda) \\
e := d^{-1} \bmod \phi(N), d := x_0^{2^{t-1} \bmod \phi(N)} \bmod N, \quad & \\
C := (x_0, x_t), t \quad &\leftarrow_{\mathrm{R}} \quad \mathsf{IA.Gen}(pp, td, t) \\
c \quad &\leftarrow_{\mathrm{R}} \quad \mathsf{IA.Enc}(m, e, pp) \\
d := \sqrt{x_t} \quad &\leftarrow \quad \mathsf{IA.Solve}(pp, C, t) \\
\{m, \perp\} \quad &\leftarrow \quad \mathsf{IA.Dec}(c, d, pp)
\end{aligned}
$$

In our BBS-TRE IA.Setup outputs the public parameters N, k_0, k_1, G, H. The first parameter is the RSA modulus N which is a Blum integer. A Blum integer is the product of two Gaussian primes i.e. $N = pq$, where $p \equiv q \equiv 3 \bmod 4$ [26]. The modulus being a Blum integer is a key requirement for the correctness of our scheme. The parameters k_0, k_1, G, H are the RSA-OAEP parameters which can be seen in detail in Algorithm 5.3.3. IA.Setup also outputs the trapdoor $\phi(N) := (p-1)(q-1)$ and keeps this parameter private.

Next, the IA.Gen algorithm outputs the encryption and decryption keys, the challenge (x_0, x_t) and the time parameter t. The term x_0 is a randomly sampled quadratic residue of N, denoted $x_0 \in \mathcal{QR}_N$. The decryption key d is calculated with the trapdoor using Algorithm 3.1.1 with the parameters $(x_0, 2^{t-1} \bmod \phi(N), N)$. If $\gcd(d, \phi(N)) = 1$, then x_t is set to $d^2 \bmod N$ and the encryption key e is set to $d^{-1} \bmod \phi(N)$. Next the IA.Enc algorithm takes as inputs a message m and the encryption key e and the public encryption parameters pp, and computes the ciphertext c using the RSA-OEAP PKE scheme. The BBS-TRE deviates from a traditional RSA-OAEP PKE scheme as the encryption key e remains private to ensure the property of implicit authentication.

Next, the IA.Solve algorithm sequentially calculates the decryption key d using Algorithm 3.1.1 with the parameters $(x_0, 2^{t-1}, N)$. The RSW time-lock assumption tells us that finding the term d will require $t-1$ sequential modular exponentiations to calculate if the trapdoor

$\phi(N)$ is not known. Finally, the IA.Dec algorithm takes as inputs the ciphertext c and the decryption key d and the public parameters pp, and outputs either the message m or an error $\perp$ using the RSA-OEAP PKE scheme. We now provide the full details of our BBS-TRE algorithms.

Setup $\mathcal{E}$ runs $pp, td \leftarrow_{\mathrm{R}} \mathsf{IA.Setup}(1^{\lambda})$ to generate the public parameter and trapdoor, as seen in Algorithm 5.3.1. The function $\texttt{prime}(j)$ on lines 4 and 5 randomly generates j bit primes $p \equiv q \equiv 3 \bmod 4$. We represent this with the notation $p \leftarrow_{\mathrm{R}} \texttt{prime}(j)$. This guarantees that N, which is calculated on line 7, is a Blum integer. The trapdoor is set to $\phi(N) := (p-1)(q-1)$. Next it runs the function $\texttt{params}(1^k)$ which outputs the parameters for the RSA-OAEP PKE scheme. The parameters k_0, k_1 are integers fixed by RSA-OEAP, and the parameters G, H are cryptographically secure hashing functions. The public parameter is set to $pp := (N, k_0, k_1, G, H)$. The public parameter can be released to $\mathcal{D}$ after IA.Setup is run, but the trapdoor must remain private.

Gen $\mathcal{E}$ runs $e, d, C, t \leftarrow_{\mathrm{R}} \mathsf{IA.Gen}(pp, td, t)$ to generate the encryption and decryption keys and the challenge, as seen in Algorithm 5.3.2. The attentive reader may note that this algorithm is similar to the Setup algorithm in our construction TIDE, presented in Chapter 4. However, there are some nuances between the constructions, most importantly that in our TRE-IA construction we first compute the decryption key d at random, and then use this to derive a random exponent e. This is highly important for the property of implicit authentication, as we discussed in Section 5.2. We now describe in detail the steps of this algorithm. First, IA.Gen sets the variable gcd to 0. Next, IA.Gen enters a while loop to generate an appropriate encryption and decryption exponent e, d for RSA-OAEP. This is done by first uniformly selecting $x \in \mathbb{Z}_N^*$ and computing $x_0 \equiv x^2 \bmod N$. Then IA.Gen evaluates $d \equiv x_0^{2^{t-1} \bmod \phi(N)} \bmod N$. The decryption key d is calculated using Algorithm 3.1.1 with the parameters $(x_0, 2^{t-1} \bmod \phi(N), N)$. Note that $\mathcal{E}$ is able to reduce the exponent

$2^{t-1} \bmod \phi(N)$ using the trapdoor. The while loop runs until the decryption key d computed on line 6 is coprime to $\phi(N)$. That is, until $\mathsf{gcd} := \mathtt{gcd}(d, \phi(N)) = 1$. Once this is found the while loop exits and the term $x_t := d^2 \bmod N$ is calculated and the encryption key e is calculated using the extended Euclidean Algorithm. In Theorem 5.4.1 we prove that the while loop will terminate. Furthermore, we prove that in expectation the number of iterations the while loop will require to generate a challenge such that $\mathsf{gcd} := \mathtt{gcd}(d, \phi(N)) = 1$ is $\frac{\pi^2}{3} \approx 3.3$. Finally, the challenge C is set to the tuple (x_0, x_t). IA.Gen then outputs the encryption and decryption keys e, d the challenge C, and the time parameter t. The challenge that $\mathcal{D}$ must solve to recover the decryption key is: for seed x_0, find d such that $d \equiv \sqrt{x_t} \bmod N$. The encryption key e must remain private, and C and t must only be released to $\mathcal{D}$ once $\mathcal{E}$ would like the decryption key d to be extracted under the RSW time-lock assumption by using the IA.Solve algorithm.

Algorithm 5.3.1: $\mathcal{E}$ runs IA.Setup on security parameter 1^λ to output public parameter pp and trapdoor td.

```
   input  : 1^λ
1  p := 0
2  q := 0
3  while p ≠ q do
4  |   p := prime(λ/2)
5  |   q := prime(λ/2)
6  end
7  N := pq
8  φ(N) := (p − 1)(q − 1)
9  k_0, k_1, G, H ← params(1^λ)
   output : pp := (N, k_0, k_1, G, H), td := φ(N)
```

Encrypt $\mathcal{E}$ runs $c \leftarrow_{\mathrm{R}} \mathsf{IA.Enc}(m, e, pp)$ on a message m, in order to generate a ciphertext c, as seen in Algorithm 5.3.3. IA.Enc is the encryption algorithm of the RSA-OAEP PKE scheme. We provide comments to explain each step of this well-known algorithm. One of

the key properties of a TRE-IA scheme is that the ciphertexts can be released to the decryptor independently of the challenge and time parameter output by IA.Gen.

Algorithm 5.3.2: $\mathcal{E}$ runs IA.Gen on public parameter, trapdoor, and time parameter pp, td, t to create the encryption and decryption exponents and the challenge e, d, C, t.

input : $pp, \phi(N), t$ // $t \in \mathbb{N}$
1 $\mathsf{gcd} := 0$
2 **while** $gcd \neq 1$ **do**
3 $\quad x \leftarrow_{\mathrm{R}} \mathbb{Z}_N^*$
4 $\quad x_0 := x^2 \bmod N$
5 $\quad d := x_0^{2^{t-1} \bmod \phi(N)} \bmod N$
6 $\quad \mathsf{gcd} := \mathsf{gcd}(d, \phi(N))$
7 **end**
8 $x_t := d^2 \bmod N$
9 $e := d^{-1} \bmod \phi(N)$ // EEA
10 $C := (x_0, x_t)$
output : e, d, C, t

Solve $\mathcal{D}$ runs $d \leftarrow \mathsf{IA.Solve}(pp, C, t)$ to evaluate the challenge and output the decryption key as seen in Algorithm 5.3.4. First IA.Solve calculates the term $\sqrt{x_t}$ by entering the parameters $(x_0, 2^{t-1}, N)$ into Algorithm 3.1.1. By the RSW time-lock assumption it will take $t-1$ sequential steps to calculate d because the trapdoor is not known by the decryptor $\mathcal{D}$.

Next, IA.Solve checks if $\sqrt{x_t}^2 \bmod N = x_t$ is true. If the condition is true, then d is set to $\sqrt{x_t}$ and output and the algorithm terminates.

Algorithm 5.3.3: $\mathcal{E}$ runs IA.Enc on (m, e, pp) to output ciphertext c.

input : m, e, pp

// $pp := (N, k_0, k_1, G, H)$

1 $m' := m \,||\, 0^{k_1}$ // Zero pad to $n - k_0$ bits

2 $r := \texttt{rand}(k_0)$ // Random k_0 bit number

3 $X := m' \oplus G_{n-k_0}(r)$ // Hash r to length $n - k_0$

4 $Y := r \oplus H_{k_0}(X)$ // Hash X to length k_0

5 $m'' := X \,||\, Y$ // Create message object

6 $c := m''^e \bmod N$ // RSA encrypt

output : c

Algorithm 5.3.4: $\mathcal{D}$ runs IA.Solve to evaluate pp, C, t and output the decryption key d.

input : pp, C, t

// $pp := (N, k_0, k_1, G, H)$, $C := (x_0, x_t)$

1 $\sqrt{x_t} := x_0^{2^{t-1}} \bmod N$

2 **if** $\sqrt{x_t}^2 \bmod N = x_t$ **then**

3 $\quad d := \sqrt{x_t}$

4 **end**

output : d

Decrypt $\mathcal{D}$ runs $\{m, \perp\} \leftarrow \mathsf{IA.Dec}(c, d, pp)$ to output the plaintext message m or $\perp$, as seen in Algorithm 5.3.5. IA.Dec is the decryption algorithm of the RSA-OEAP PKE scheme. As with IA.Enc, each step of the algorithm is described on the comments of each line. By the correctness of the RSA-OAEP PKE scheme, if the parameter d extracted by IA.Solve under the RSW time-lock assumption has the property $ed \equiv 1 \bmod \phi(N)$ (line 9 of IA.Gen), then the message m will be recovered. Else, IA.Dec will output $\perp$.

Algorithm 5.3.5: $\mathcal{D}$ runs IA.Dec on (c, d, pp) to recover message m or output $\perp$.

input : c, d, pp
// $pp := (N, k_0, k_1, G, H)$
1 $m'' := c^d \bmod N$
2 $X := \lfloor m'' 2^{-k_0} \rfloor$ // Extract X
3 $Y := m'' \bmod 2^{k_0}$ // Extract Y
4 $r := Y \oplus H_{k_0}(X)$ // Recover r
5 $m' := X \oplus G_{n-k_0}(r)$ // Recover padded message
6 $m := m' 2^{-k_1}$ // Remove padding
output : $\{m, \perp\}$

We have presented a concrete construction for a TRE-IA based on a RSW TLP and the RSA-OAEP PKE scheme. We have done this by setting up an RSA modulus which is a Blum integer, generating a TLP challenge and a PKE key-pair, then time-locking the decryption key using the TLP. We then integrated our encryption and decryption exponents (the PKE key pair) into the RSA-OAEP scheme for the encryption of a message and the decryption of the ciphertext respectively. In the next section we provide a formal security analysis of our scheme.

5.4 Security Analysis

In this section we provide the security analysis of the BBS-TRE scheme presented in Section 5.3. This comprises of proving correctness, security and implicit authentication.

We first provide proof of the correctness of our scheme. The outline of our proof will be as follows: first we will prove that IA.Gen will terminate and hence generate a suitable decryption key d as the solution to the challenge D with overwhelming probability. Second we will prove that IA.Solve will correctly output the decryption key d, third we will prove that the decryption key d is unique because N is a Blum integer, and finally we will prove that the decryption key will correctly return the original message m when it is used to decrypt a ciphertext c generated with the encryption exponent e.

For the first element of our correctness proof we must prove that the while loop in $\mathsf{IA.Gen}$ will terminate and generate a suitable challenge and decryption key d. We note that this step is something that was not necessary in the correctness proof of TIDE, and is required due to computing the key pair d and e at random, rather than fixing e.

Theorem 5.4.1 *The while loop in $\mathsf{IA.Gen}$ will in expectation take $\frac{\pi^2}{3}$ trials to generate a suitable challenge and decryption key d.*

Proof: The probability of two randomly selected integers being coprime is $\frac{6}{\pi^2}$ [84], Theorem 33. The Blum integer $N = pq$ generated with $\mathsf{IA.Setup}$ is randomly selected using the Miller Rabin Monte Carlo algorithm [125]. Next, the $\phi(N)$ is calculated as $(p-1)(q-1)$. Therefore, $\phi(N)$ is always even.

Each iteration of the while loop in $\mathsf{IA.Gen}$ is a Bernoulli trial. In our Bernoulli trial N and hence $\phi(N)$ are randomly selected and the integer d on line 5 of $\mathsf{IA.Gen}$ is also randomly selected. We model d as a random integer as it is an output of the BBS CSPRNG. In each trial, if $\mathsf{gcd}(d, \phi(N)) = 1$ the outcome is a success, otherwise if $\mathsf{gcd}(d, \phi(N)) \neq 1$ then the outcome is a failure. Therefore, in each trial, the probability of selecting two random integers which are coprime when one integer is even is $\frac{6}{2\pi^2} = \frac{3}{\pi^2}$.

Finally, the probability distribution of the number of Bernoulli trials required until one success is achieved forms a Geometric distribution $G \sim \mathsf{Geo}(\frac{3}{\pi^2})$. Therefore, in expectation, the number of Bernoulli trials required until a suitable challenge and decryption key d is selected such that $\mathsf{gcd}(d, \phi(N)) = 1$ is $\mathbb{E}(G) = \frac{\pi^2}{3} \approx 3.3$ trials. Therefore it is clear that this while loop will terminate in polynomial time with overwhelming probability.

□

The second part of proving the correctness of our BBS-TRE construction is to prove that the $\mathsf{IA.Solve}$ algorithm correctly calculates the decryption key d.

Lemma 5.4.1 *The IA.Solve algorithm in the BBS-TRE correctly outputs the decryption key* $d := x_{t-1}$.

Proof: Suppose encryptor $\mathcal{E}$ honestly generates a random public parameter, challenge and time parameter $pp := (N, k_0, k_1, G, H), (C := (x_0, x_t), t)$ and presents these to an honest $\mathcal{D}$. Next, suppose $\mathcal{D}$ selects the legitimate evaluation algorithm IA.Solve to evaluate (C, t). The IA.Solve algorithm will calculate the decryption key d by entering the following parameters $(x_0, 2^{t-1}, N)$ into Algorithm 3.1.1, which will output $d := {x_0}^{2^{t-1}} = x_{t-1} \bmod N$. IA.Solve will correctly output the BBS term x_{t-1} with overwhelming probability due to the correctness of Algorithm 3.1.1 noted in Theorem 4.4.3. Therefore, the IA.Solve algorithm will correctly output $d := x_{t-1}$. □

Next we must prove that the decryption key $d := x_{t-1} = \sqrt{x_t} \bmod N$ output by IA.Solve is unique. First we must recall that d by definition of being a term in a BBS CSPRNG sequence is a quadratic residue of the modulus N.

Therefore, we must prove that the solution d to the following equation is unique:

$$d := \sqrt{x_t} \bmod N \tag{5.1}$$

This challenge arises because the Chinese Remainder Theorem isomorphism indicates that when $N = pq$, where p, q are distinct odd primes, that Equation 5.1 has four distinct solutions [93]. That is, $\pm a \equiv \pm b \equiv \sqrt{x_t} \bmod N$, where $a \neq b$.

Theorem 5.4.2 *If $N = pq$ is a Blum integer, then the decryption key d output by IA.Solve is unique.*

Proof: If $N = pq$ is a Blum integer with $p \equiv q \equiv 3 \bmod 4$, then $N \equiv 1 \bmod 4$. By the Chinese Remainder Theorem isomorphism every $r \in \mathcal{QR}_N$ has four distinct square roots $\pm a$

and $\pm b$. As N is a Blum integer, by the law of quadratic reciprocity $\mathcal{J}_N(a) = \mathcal{J}_N(-a)$ and $\mathcal{J}_N(b) = \mathcal{J}_N(-b)$, where $\mathcal{J}_N$ is the Jacobi symbol. It must be the case that $a^2 \equiv b^2 \bmod N$, which implies $(a-b)(a+b) \equiv 0 \bmod N$, which implies $(a-b) \mid N$ and $(a+b) \mid N$. That is, without loss of generality $(a-b) = kp$ and $(a+b) = \ell q$, where $k, \ell \in \mathbb{N}$. Therefore, $\mathcal{J}_p(a) = \mathcal{J}_p(b)$ and $\mathcal{J}_q(a) = \mathcal{J}_q(-b)$. As $p \equiv 3 \bmod 4$, the law of quadratic reciprocity tells us $\mathcal{J}_p(-1) = -1$, we have $\mathcal{J}_q(a)\mathcal{J}_p(-1) = \mathcal{J}_q(-b)\mathcal{J}_p(-1)$. This implies that $\mathcal{J}_N(-a) = \mathcal{J}_N(b)$ or written another way $\mathcal{J}_N(a) \neq \mathcal{J}_N(b)$.

Without loss of generality, eliminate the two roots with $\mathcal{J}_N$ equal to -1, say $\mathcal{J}_N(b) = \mathcal{J}_N(-b) = -1$. This leaves $\mathcal{J}_N(a) = \mathcal{J}_N(-a) = 1$. It is the case that only one of $-a$ or a has $\mathcal{J}_p = \mathcal{J}_q = 1$ as $p \equiv 3 \bmod 4$. Therefore, it is this one that is the only quadratic residue modulo N [26].

Returning to our BBS-TRE, by Lemma 5.4.1 the term $d := x_{t-1} = \sqrt{x_t} \bmod N$ is correctly calculated by IA.Solve and by definition it is a term in a BBS sequence. Therefore, d is a quadratic residue of the modulus N. Therefore, d is the only one of the four distinct square roots of x_t that is a quadratic residue of N. □

For the final element of the correctness proof of our BBS-TRE construction we must prove that the decryption key d will correctly recover the message in an RSA-OAEP scheme.

Corollary 5.4.1 *The BBS-TRE is correct.*

Proof: By Theorem 5.4.1 we know that IA.Gen will terminate and output a suitable RSW TLP challenge (C,t). By Theorem 4.4.3 and Lemma 5.4.1 we know that the decryption key d will be recovered by IA.Solve. By Theorem 5.4.2 we know that the decryption key d against the modulus N is unique. From the Fermat-Euler Theorem [78] we know that the decryption key d calculated on line 5 of IA.Gen is the same as the decryption key recovered by IA.Solve on line 1. From the correctness of the Extended Euclidean Algorithm [93] we know that e calculated on line 9 of IA.Gen is the multiplicative inverse of d. Finally, from the correctness

of the RSA-OEAP scheme [21] we know that $\mathsf{IA.Dec}$ will correctly recover the message m using decryption key d from the ciphertext c output by $\mathsf{IA.Enc}$ using encryption key e with overwhelming probability. □

Next we prove that our scheme satisfies the security property. We begin by showing that the adversary is unable to compute d with non negligible probability in fewer than $t-1$ steps. We then go on to show that an adversary who wins the TRE-IA security game without computing d can also break the standard IND-CPA game for RSA-OAEP, which is known to be secure.

Theorem 5.4.3 *Our BBS-TRE requires $t-1$ sequential steps to recover the decryption key d.*

Proof: Suppose we run the security game using our TRE-IA scheme as follows: $\mathcal{E}$ honestly generates a random public parameter pp and generates the encryption key, decryption key, challenge, and time parameter e, d, C, t. Next $\mathcal{A}$ selects two messages of the same length, m_0 and m_1, for $\mathcal{E}$ to encrypt. $\mathcal{E}$ uniformly selects $b \in \{0,1\}$ and encrypts m_b, and sends the resulting ciphtertext c, along with the public parameter pp to $\mathcal{A}$. $\mathcal{A}$ runs a PPT algorithm $\mathcal{A}_0$ which pre-processes on pp and c, storing any output in a state $\mathsf{st} \leftarrow \mathcal{A}_0(pp, c)$. $\mathcal{E}$ sends the challenge and time parameter to $\mathcal{A}$. Finally, $\mathcal{A}$ runs a PPT algorithm $\mathcal{A}_1$ which runs in fewer than $t-1$ sequential steps, and outputs a bit $b' \leftarrow \mathcal{A}_1(\mathsf{st}, C, t)$.

First, we prove that computing d without using the challenge and time parameter is infeasible with overwhelming probability. We recall from Rabin that finding $d = \sqrt{x_t} \bmod N$ (i.e. taking square roots $\bmod N$) without the challenge and time parameter C, t is equivalent to factoring N [131]. Therefore, as N is an RSA modulus, it cannot be factored by any PPT algorithm with more than negligible probability. Therefore, we have that any PPT algorithm

that does not use C,t cannot recover the unique decryption key d with more than negligible probability.

We next show that giving $\mathcal{A}$ the challenge and time parameter $C := (x_0, x_t), t$ does not allow them to take square roots modulo N faster than sequential squaring. To see this, note that by construction x_0 and x_t are the seed term and t^{th} term in a BBS CSPRNG sequence [26]. Under the Generalised BBS assumption [35], knowledge of these terms does not allow finding $d = \sqrt{x_t}$ faster than sequential squaring unless $x_0^{2^{\lambda(\lambda(N))}}$ mod N is calculated efficiently, where $\lambda(N)$ is the Carmichael function [44]. Finding $x_0^{2^{\lambda(\lambda(N))}}$ efficiently is an open problem given by Theorem 9 of Blum et al. [26, 71, 81].

Finally, we have that computing d in fewer than $t-1$ sequential steps breaks the RSW time-lock assumption: To see this, recall from Lemma 5.4.1 that $\mathsf{IA.Solve}$ correctly outputs the decryption key d in $t-1$ sequential steps, and we know from Theorem 5.4.2 that the decryption key d is unique. Therefore, the existence of an algorithm that computes d in less than $t-1$ sequential steps with non-negligible probability contradicts the RSW time-lock assumption given in Definition 3.1.1.

□

Theorem 5.4.3 proves that the adversary cannot recover d in less than t sequential steps to win the Security game. Therefore, to conclude our security proof we must demonstrate that the adversary cannot guess b in less than t sequential steps without knowledge of the decryption key.

Theorem 5.4.4 *Our BBS-TRE scheme is secure.*

Proof: We first assume for a contradiction that a PPT adversary $\mathcal{A}$ can win the Security game with a non-negligible advantage.

Let IND-CPA^{RSA} be the standard IND-CPA game for RSA-OAEP [21]. We now recall the well-known result that RSA with OAEP padding is IND-CPA secure under the RSA

assumption [72]. We will show that any adversary who can break the Security game can also win the IND-CPA^{RSA} game.

When choosing the two messages in the Security game, the adversary $\mathcal{A}$ has the same available information as they do in the IND-CPA^{RSA} game: the public parameters. Similarly, any preprocessing that can be done in the TRE-IA security game can also be done in the standard RSA-OAEP setting. Therefore, any non-negligible advantage obtained by the adversary in the TRE-IA game prior to receiving (C,t) could also be used to obtain the same advantage in the IND-CPA^{RSA} game, which is known to be secure. Therefore, it is impossible for an adversary to gain a non-negligible advantage prior to receiving the challenge and time parameter.

We have shown in Theorem 5.4.3 that an adversary bounded by $t-1$ sequential steps cannot obtain the decryption key d with more than negligible probability. However, if an adversary runs an algorithm which provides a non-negligible advantage without knowledge of the decryption key d, then they are able to distinguish between two RSA-OAEP ciphertexts, which were chosen independently of the challenge. Hence the adversary would also break the underlying IND-CPA^{RSA} game with the same algorithm, which is known to be impossible.

□

We now prove that our scheme has the property of implicit authentication.

Theorem 5.4.5 *Our TRE scheme provides the implicit authentication property.*

Proof: Suppose that the encryptor runs the $\mathsf{IA.Setup}$ and $\mathsf{IA.Gen}$ algorithms. Let $\mathcal{A}$ receive the RSA modulus N and the OAEP parameters (k_0,k_1,G,H), and let m^* be the target message it outputs.

Now let $\mathcal{A}$ receive the challenge C, the time parameter t, the decryption key d and have access to the encryption oracle $\mathcal{O}^{\mathsf{enc}}$.

In our construction d is chosen at random on lines 3 - 5 of $\mathsf{IA.Gen}$. As we are working in $\mathbb{Z}_N^*$ there is only one multiplicative inverse of d, which is the encryption key e calculated on line 9 of $\mathsf{IA.Gen}$. To derive e from d requires knowledge of $\phi(N)$, as $e := d^{-1} \bmod \phi(N)$. In order to learn $\phi(N)$, the adversary would need to factor the RSA modulus N, which is a well-known hard problem [131, 135]. Therefore, unless the adversary can factor N, they cannot guess e with more than negligible probability. Therefore with overwhelming probability the adversary will not learn the trapdoor $\phi(N)$, and hence will not be able to derive e from d.

Using the encryption oracle $\mathcal{O}^{\mathsf{enc}}$ $\mathcal{A}$ can obtain polynomially many ciphertexts. Recall from [21] that RSA-OAEP has ciphertext indistinguishability under chosen-plaintext attack, which guarantees indistinguishability between encryptions of messages. This property guarantees in particular that the adversary has no advantage in identifying a ciphertext that will allow them to win the IA game.

The adversary can choose random elements from the ciphertext space, and decrypt them using the decryption key d. However, without knowledge of the encryption key, any such ciphertext will decrypt to a random element of the message space.

As the size of the ciphertext space is exponential (explicitly it is the magnitude of $\mathbb{Z}_N^*$), and the adversary runs a PPT algorithm, there is a negligible chance of correctly guessing a ciphertext which decrypts to the target message m^*. Therefore the adversary will not win the implicit authentication game with greater than negligible probability.

□

5.5 Conclusion

In this chapter we introduced a variant of TRE, which we call timed-release encryption with implicit authentication (TRE-IA). Implicit authentication is formally introduced with a

game-based definition, and we provide a concrete implementation with this property. Our implementation of a TRE-IA uses the BBS CSPRNG and RSA-OAEP PKE as the building blocks, as in the construction TIDE seen in Chapter 4.

We have discussed the additional applications that this work may be applied to, using the setting of whistleblowing to motivate our work. We believe that this direction of using delay-based crypto in the setting of vulnerable parties has the potential be useful in practice. It is our hope that this work is expanded upon in the future, in order to make the lives of such individuals safer.

Future work As our suggested application of whistleblowing is a sensitive topic, and the use of delay in this context is new, we believe that there is a lot of scope for follow-up work, from both a technical and non-technical perspective.

From a non-technical perspective, we believe that a social-science based review into whistleblowers and other vulnerable people would be of great benefit. We envisage such a review involving interviews which discuss how and why such a tool could be useful, and what features they would like to be included.

From a technical point of view, as well as making any changes which are identified by a review as mentioned above, we also believe that it is of great importance that any such tool should be easy to use and trusted by potential users, with no required technical knowledge. Therefore we believe that a tool with a clear, easy-to-use interface would be highly useful, along with external cryptanalysis of both this work, and any work that builds upon this concept.

Chapter 6

Continuous Verifiable Delay Functions and Randomness Beacons

Contents

In this chapter, we discuss the modern state of cryptographic randomness, and the role delay-based cryptography plays. This leads us to examine the primitive of verifiable delay functions, and an extension known as continuous verifiable delay functions. We discuss the definitions and applications of this primitive, arguing that the original definitions provided in [66] are non-standard, and that their heavy parameterisation make this powerful primitive difficult to use. We therefore offer definitions that are in keeping with the rest of the literature, and hence easier to use and understand. We then provide a construction following these

definitions, using a trusted setup to achieve a highly efficient construction. We then extend this to a randomness beacon, and discuss the efficiency of various approaches to building a randomness beacon, and how our construction fits into the literature.

6.1 Introduction

Often a group of parties wish to use shared common randomness in a protocol, each with the guarantee that none of the other parties has any prior knowledge of the output.

Common examples within cryptography include secret sharing and key distribution [29, 58], and examples outside of cryptography include running a lottery and random sampling, such as patients in clinical trials and officials in audits [32, 137].

In cryptography, a *randomness beacon* (RB) [132] is the primitive used to regularly output values that can be used for such applications. One of the fundamental properties of such a primitive is that it should not be biased, i.e., the parties running it should not have influence over the final outcome. One natural approach to generating shared randomness between parties is to agree to use a hash function on a source of entropy, such as a particular stock index. However, this approach is not free from bias, as the price of the stock could be biased by a powerful trader. An influential trader could compute the hash values of the current price, a slightly higher price and a slightly lower price, with the goal of biasing the price towards the most favourable outcome.

To address this issue, in CRYPTO 2018, Boneh et al. introduced the concept of a Verifiable Delay Function (VDF) [32]. VDFs ensure that rather than directly computing the hashed price of the stock, one can instead evaluate a sequential function that takes a prescribed amount of time to compute. The idea is that no party can bias the output of the random beacon because they cannot calculate the result in the given time window. In our example, this enables us to guarantee that by the time the trader has evaluated the sequential

function on the proposed values of the stock, the market will have closed, and so they will not know the effect of altering the stock price.

Critically, for any practical application it is also necessary that parties wanting to use this randomness are able to quickly verify that a given amount of time has elapsed, without re-computing the entire function. Therefore a VDF must have the property that a party can publicly verify that a delay has happened significantly faster than the time taken to compute the delay.

One downside of a VDF is that for each delay computed, only one *pulse* of randomness can be extracted. In EUROCRYPT 2020, Ephraim et al. [66] introduced the concept of a *continuous* VDF (cVDF), in order to address this. A cVDF can be used to verify intermediate states of the computation, rather than only the final output of the sequential computation, allowing for randomness to be extracted at each interval.

The definitions given in [66] are non-standard, built upon a chain of nested definitions, which are heavily parameterised. This makes the definitions reliant on a certain computational model, which in turn makes them somewhat difficult to use. For example it is not trivial for one to an extend a VDF to a cVDF, as the syntax is materially different. The authors instantiate this primitive using an iterated extension of a VDF protocol introduced by Pietrzak at ITCS19 [130]. However, this construction has some drawbacks in terms of efficiency. For example, a significant amount of time is required for an evaluator without many parallel cores to compute the proof. Additionally, the storage of the proof and the time spent computing the verification protocol grow large as the length of delay increases.

In this work, we provide new definitions in keeping with the VDF literature, and we use these to introduce a new approach to build a cVDF, and resulting randomness beacon. In this construction, we leverage a precomputation phase to significantly improve the storage and verification costs.

6.1.1 Related Work

As a result of the rapid development of blockchain technology in recent years, many RBs have been proposed in recent years, based on primitives such as Publicly Verifiable Secret Sharing (PVSS) schemes [45, 46, 95, 142, 149], Threshold Crypto-Systems [40, 52, 83], Verifiable Random Functions (VRF) [61, 79], as well as the approach used in this paper, Verifiable Delay Functions (VDFs) [32, 65, 66, 82, 141]. We shall now give a brief discussion of some recent works.

Interactive RBs Perhaps the most common approach to building an RB is to build an interactive RB. Typically, these RBs are used in blockchain technologies, including notably DFINITY [83], Algorand [79], and Ouroboros Praos [61]. We will briefly discuss some of the most relevant literature around interactive RBs.

Collaborative RBs The majority of the works cited in this section – with the notable exception of the VDF-based RBs – can be classified as collaborative RBs, i.e., those in which participants work together to generate randomness. In such RBs, the goal is generally to minimise the computational and communication complexity of each participant. Many of the current protocols offer different trade-offs: One approach is to relay all messages through a single node in order to reduce the communication cost, including for instance the DFINITY protocol [83] and Randhound [149]. This reduces the communication cost to one broadcast, i.e., $O(n)$, where n is the number of participants. Alternatively, some protocols aim to maximise decentralisation, by avoiding a central node. Examples of such protocols include Ouroboros [95] and SCRAPE [45], and whilst such protocols are more decentralised, this comes at a significantly higher communication cost of $O(n^3)$ [134].

Competitive RBs An alternative approach is to use competitive proof of work as a basis for a randomness beacon [82, 133]. The idea is for parties to compete to find solutions to

cryptographic puzzles, in the well-known proof of work style used in Bitcoin [126], using these puzzles as randomness. Using proof of work allows the beacon to achieve both scalability, meaning a large number of participants can join, as well as achieving the fairness property, meaning the influence of each participant is roughly equal. However, such a beacon is computationally intensive. The effect of such proof of work schemes on the environment is well-documented [74, 151], and widely viewed as unsustainable. Additionally, it is always the case that a powerful miner or group of miners can influence the outcome of the beacon by not publishing a solution they view as undesirable. A recent systematisation of knowledge paper by Raikwar et al. [134] discusses in greater detail the approaches we mention here to building an interactive RB.

We now discuss VDF-based RBs, the subject of this chapter. We begin by highlighting the advantages of using this primitive, before discussing concrete details. The key property a VDF-based randomness beacon gives, when compared to alternative approaches, is the *timeliness* property, which states that each pulse of randomness will come at a regular interval, which is desirable in most applications. This property is listed by NIST in their project on interoperable randomness beacons [137] as a property a randomness beacon should have.

Another advantage of a VDF-based RB is that there is no need for multiple parties to reach an agreement on what the output is, which is a requirement of the interactive RBs discussed earlier. Furthermore, there is no requirement for a synchronous network, which can be unrealistic in practice, but required by many schemes [133].

Finally, VDF-based RBs have a strong bias-resistance property as the randomness cannot be altered once the input has been chosen, unlike the collaborative and competitive approaches discussed previously, where parties can choose not to publish undesirable solutions [82].

VDF-based RBs A modern approach to building a randomness beacon utilises verifiable delay functions (VDFs), first introduced at CRYPTO 2018 by Boneh et al [32]. A VDF is a tool used to provide a delay in a decentralised setting. Whilst VDFs can be used to construct consensus protocols and in timestamping, their flagship application is to provide a publicly verifiable randomness beacon [56, 101]. A VDF requires a solver to compute an iterated sequential function on an input, where any interested entity can efficiently verify that the output is correct, and hence the prescribed amount of time was spent on the computation. In recent years various VDF schemes have been proposed [66, 68, 130, 155]. Most relevant to this chapter is Pietrzak's VDF [130], where a solver uses repeated squaring to calculate a delay and additionally calculate a proof. The solver then engages in an interactive protocol to prove to a verifier that they computed the correct solution.

We next describe some of the most relevant protocols which use a VDF to build an RB.

In 2018, Drake et al. [65] proposed a smart contract which uses a VDF to produce random values. This approach is a collaborative RB, as discussed earlier - it requires multiple parties, known as beacon chain proposers, each contributing some local randomness. Drake assumes that there exists a global clock, and splits up time into regular *epochs* of 1024 seconds. Each of these epochs is split into 8-second blocks, each of which is ran by a beacon chain proposer. These proposers each commit to some local entropy, and reveal it at the end of their block, where it is then broadcast as randomness. This randomness is then used to sample a later beacon proposer. This idea was presented as a post on Ethereum research forum and lacks a rigorous security analysis.

In 2020, Schindler et al. proposed randrunner [141], a competitive RB construction in which all participants are given a unique trapdoor in setup. Then consecutive rounds of randomness are evaluated, where in each round an input is sampled and also one *leader* is chosen, whose trapdoor allows them to evaluate the VDF faster than any other party.

The output of the VDF evaluated in each round is hashed to obtain a pulse of randomness, which is the beacon output, before another input and leader are chosen for the subsequent round. All parties are encouraged to try to solve the VDF, even if they do not have the trapdoor, which leads to a large amount of computational expenditure, particularly as the number of participants grow. Additionally, this construction suffers from various attacks, such as the adversary being able to corrupt the round leaders, and withhold output, whilst working on subsequent rounds. An alternative approach is to use a continuous VDF.

Continuous VDFs In EUROCRYPT 2019, Ephraim et al. introduced the notion of a *continuous* VDF (cVDF) [66]. The cVDF model presented by Ephraim et al. introduces the notion of a *state*, which is an intermediate point within the computation that can be verified. In contrast, when using a standard VDF verification is only possible at the end of the computation. These states enable two key applications that a standard VDF is lacking. Firstly, at any state the solving party can pass the computation on to another party, who can efficiently verify the state and take over the computation. Secondly, by running the verification procedure at each state, trusted public randomness can be extracted at regular intervals, creating an efficient randomness beacon from one input. This is very well-suited for the notion of an RB in which a pulse of randomness is output at regular intervals.

Crucially, Ephraim et al. show that if you have a secure cVDF, i.e., one that is adaptively sound, sequential, and correct, you can build a secure randomness beacon from this primitive. Ephraim et al. build their randomness beacon using the Tick/Tock paradigm, shown in Figure 6.1. In such a beacon, an initial state state_0 is generated during the setup procedure (The reader will see in the diagram that Ephraim et al. call the relevant algorithm init, which takes input x_0). Then, two algorithms are ran in parallel on every state: algorithm Tick takes a state state_i, and outputs the next state state_{i+1}. Algorithm Tock takes the state state_i and outputs a pulse of randomness (denoted in the diagram as x_i). Verification can then be performed on

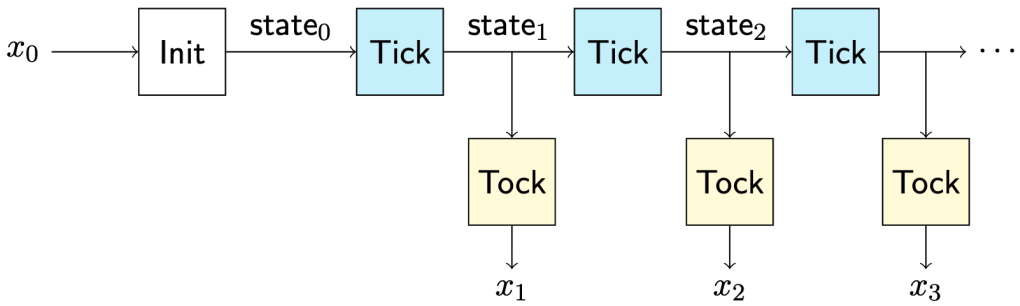

Fig. 6.1 The Tick/Tock paradigm as illustrated by Ephraim et al. in [66], where each x_i represent a pulse of randomness.

both the computation of Tick and of Tock, to show that the state was computed correctly, and that the randomness was correctly computed from the state. This is a simple and effective approach to building a randomness beacon.

The downside of this approach is that the definitions given in [66] utilise a very specific computational model, namely one that uses many non-standard parameters and invokes multiple nested definitions. This is a departure from the rest of the delay-based literature, and we believe that this makes the definitions more difficult to use, as one must first familiarise themself with this model. As such, we provide more generic definitions, in keeping with the literature, broadly following the original VDF definition provided in [32].

Additionally, the cVDF construction in [66], along with the associated RB, has a verification time which grows in $O(\log t)$, and hence scales badly as the time parameter increases.

In this chapter, we introduce a precomputation phase in order to build a significantly more efficient cVDF, and resulting public randomness beacon, at the cost of a trusted setup.

6.1.2 Contributions of this Chapter

We begin in 6.2 by analysing the definition of a cVDF, in a similar manner to the TRE discussion in Chapter 4. We then propose a new game-based definition that is in keeping

with current literature, and we show that our definition of a cVDF is a generalisation of a standard VDF. We construct a continuous VDF (Section 6.4), providing an efficiency analysis and an implementation study (Section 6.5) showing that it has concrete advantages over the existing constructions. We argue that avoiding some trust when setting up a VDF based randomness beacon is impractical at the current time, and that trusting a small group of anonymous parties is not necessarily better than trusting an entity such as NIST. Under the assumption that this trusted third party runs the setup procedure, we build and implement a randomness beacon which is both efficient to run in terms of storage and computation costs, and additionally extremely fast for external parties to verify. We show that under our definition, any secure cVDF also yields a randomness beacon (Section 6.7) in the style of Ephraim et al. We then implement this, showing that whilst the setup is trusted, our randomness beacon runs significantly more efficiently than that of Ephraim et al., and indeed all current cVDF-based randomness beacons. We provide rigorous game-based security definitions for a cVDF (Section 6.2.3), and Finally, in Section 6.6 we prove our scheme secure under the definitions given in 6.2.

6.2 Definitions

In this section, we begin by recalling the definitions given by Ephraim et al. in [66], and discussing why we think that these definitions could be made more practical. We will then provide the definition of a standard VDF following Boneh et al. in [32], before extending this definition to capture a continuous verifiable delay function.

6.2.1 Ephraim's definitions

We structure the definitions in the same manner as Chapter 4, which is to say we begin by giving the cVDF definitions provided by Ephraim, in the original style, rather than

one consistent with the rest of this thesis. We do this to highlight the differences in these definitions, and how our definitions are advantageous.

Definition 6.2.1 (Continuous Verifiable Delay Function) *A continuous verifiable delay function is a tuple (Setup, Gen, Eval, Verify) such that (Setup, Gen, Eval) is a* $(1, B, l, \varepsilon)$*-iteratively sequential function, (Setup, Eval, Verify) is a B-sound function, and is correct from an honest start.*

This definition relies on the following:

Definition 6.2.2 (Correctness from an Honest Start) *For every* $\lambda \in \mathbb{N}$*, pp in the support of Setup*$\left(1^{\lambda}\right)$*,* v_0 *in the support of Gen* (pp)*, and* $T \in \mathbb{N}$*, it holds that*

$$\mathsf{Verify}\left(pp, (v_0, T), \mathsf{Eval}^{(T)}(pp, v_0)\right) = 1 .$$

Definition 6.2.3 (Iteratively Sequential Function) *Let* $D, B, l : \mathbb{N} \to \mathbb{N}$ *be functions and let* $\varepsilon \in (0,1)$*. A tuple of algorithms (Setup,Gen,Eval) is a* (D, B, l, ε)*-iteratively sequential function if Setup and Gen are PPT, Eval is deterministic, and the following properties hold.*

- i) Iteratively sequential. The tuple $(\mathsf{Setup}, \mathsf{Gen}, \mathsf{Eval}^{(\cdot)})$ is a (D, B, l, ε)-sequential function.
- ii) Length Bounded. There exists a polynomial m such that for every $\lambda \in \mathbb{N}$ and $x \in \{0,1\}^*$, it holds that $|\mathsf{Eval}(\mathsf{pp}, x)| \leq m(\lambda)$. We define $\mathsf{Eval}(\cdot)$ to be the function that takes as input pp, and (x, t) and represents the t-wise composition given by

$$\mathsf{Eval}^{(T)}(\mathsf{pp}, x) := \mathsf{Eval}(\mathsf{pp}, \cdot) \circ \cdots \circ \mathsf{Eval}(\mathsf{pp}, \cdot)(x)$$

Which requires us to recall the following definitions:

Definition 6.2.4 (B-Soundness) *For every non-uniform algorithm $A = \{A_\lambda\}_{\lambda\in\mathbb{N}}$ such that size $(A_\lambda) \in$ poly$(B(\lambda))$ for all $\lambda \in \mathbb{N}$, there exists a negligible function negl such that for every $\lambda \in \mathbb{N}$ an adversary $\mathcal{A}$ cannot win the soundness game with more than negligible probability.*

1 $\mathsf{pp} \leftarrow \mathsf{Gen}\left(1^\lambda\right)$ $\mathcal{V}$ generates public parameters;
2 $(x,y) \leftarrow A_\lambda(\mathsf{pp})$ $\mathcal{A}$ runs an algorithm before outputting a pair (x,y);
$\mathcal{A}$ wins if Verify outputs accept and $\mathsf{Eval}(\mathsf{pp},x) \neq y$.

Game 6.2.1: Soundness Game

Definition 6.2.5 (Sequential Function) *Let $D,B,\ell : \mathbb{N} \to \mathbb{N}$ and let $\varepsilon \in (0,1)$. A (D,B,ℓ,ε) -sequential function is a tuple (Setup, Gen, Eval) where Setup and Gen are PPT, Eval is deterministic, and the following properties hold:*

- i) Honest Evaluation. There exists a uniform circuit family $\{C_{\lambda,t}\}_{\lambda,t\in\mathbb{N}}$ such that $C_{\lambda,t}$ computes $\mathsf{Eval}\left(1^\lambda,\cdot,(\cdot,t)\right)$, and for all sufficiently large $\lambda \in \mathbb{N}$ and $D(\lambda) \leq t \leq B(\lambda)$, it holds that $\mathsf{depth}\left(C_{\lambda,t}\right) = t \cdot \ell(\lambda)$ and $\mathsf{width}\left(C_{\lambda,t}\right) \in \mathsf{poly}(\lambda)$
- ii) Sequentiality. For all non-uniform algorithms $\mathcal{A}_0 = \{\mathcal{A}_{0,\lambda}\}_{\lambda\in\mathbb{N}}$ such that $\mathsf{size}\left(\mathcal{A}_{0,\lambda}\right) \in \mathsf{poly}(B(\lambda))$ for all $\lambda \in \mathbb{N}$, there exists a negligible function negl such that for every $\lambda \in \mathbb{N}$, the adversary cannot win the Sequentiality Game with more than negligible probability.

1 $\mathsf{pp} \leftarrow \mathsf{Setup}\left(1^\lambda\right)$ generates public parameters;
2 $\mathsf{Eval}(\mathsf{pp},(x,t)) = y$ runs an algorithm before outputting a pair (x,y);
3 $\mathcal{A}_1 \leftarrow \mathcal{A}_{0,\lambda}(\mathsf{pp}) \wedge \mathsf{depth}(\mathcal{A}_1) \leq (1-\varepsilon)\cdot t \cdot \ell(\lambda)$;
4 $x \leftarrow \mathsf{Gen}\left(1^\lambda,\mathsf{pp}\right)$;
$\mathcal{A}$ wins if $(t,y) \leftarrow \mathcal{A}_1(x) \wedge t \geq D(\lambda)$.

Game 6.2.2: Sequentiality Game

As the reader can see from the various nested definitions and the number of parameters, this is a rather complex approach to defining a cVDF. Our goal when redefining a cVDF is to make the definitions as generic and as close to the standard VDF definitions as possible. We therefore avoid nested definitions, and reduce the number of parameters in order to become syntactically consistent with the rest of the VDF literature. Before this however, we shall introduce the original definition of a verifiable delay function [32] as a point of reference.

6.2.2 Verifiable Delay Functions

We now provide the definition and properties of a generic VDF [32, 130].

We follow Boneh et al. [32] in defining a VDF as the following tuple of algorithms: (Setup, Eval, Verify).

Definition 6.2.6 (VDF Definition) *A VDF is the following tuple of algorithms (Setup, Eval, Verify):*

- $\mathsf{pp} \leftarrow_{\mathrm{R}} \mathsf{Setup}(1^{\lambda})$. Setup takes as input security parameter 1^{λ} and outputs the public parameter pp. Setup runs in time $\mathsf{poly}(\lambda)$.
- $y \leftarrow \mathsf{Eval}(\mathsf{pp}, x, t)$. Eval takes as input the public parameter pp, and a challenge x, and evaluates a solution y in t sequential steps. To qualify as a VDF, the solution y must be unique.
- $\{\text{accept}, \text{reject}\} \leftarrow \mathsf{Verify}(\mathsf{pp}, x, t, y)$. Verify takes as input the public parameter pp, a solution y, the seed x, and time parameter t. Verify runs in time $\mathsf{polylog}(t)$ and $\mathsf{poly}(\lambda)$.

Additionally, a VDF must be correct, sound, and sequential, [32, 130].

- A VDF is *correct* if a solution output by Eval is accepted by Verify with overwhelming probability. This is made precise in the VDF correctness game.

VDF Correctness Game

1 $\mathcal{V}$ generates random public parameter, challenge, and chooses the time parameter: $x, t, \mathsf{pp} \leftarrow_{\mathrm{R}} \mathsf{Setup}(1^{\lambda})$.
2 $\mathcal{P}$ runs the legitimate algorithm Eval to generate solution: $y \leftarrow \mathsf{Eval}(\mathsf{pp}, x, t)$.
3 $\mathcal{V}$ verifies the solution: $\{\text{accept}, \text{reject}\} \leftarrow \mathsf{Verify}(\mathsf{pp}, x, t, y)$.
$\mathcal{P}$ wins if Verify outputs accept.
A VDF is correct if $\mathcal{P}$ wins with probability $1 - \mathsf{negl}(\lambda)$.

- A VDF is *sound* if any solution y' output by any algorithm $\mathcal{E}$, where $y' \neq y \leftarrow \mathsf{Eval}$, has a negligible probability of being accepted by Verify. This is made precise in the VDF soundness game.

VDF Soundness Game

1 $\mathcal{V}$ generates random public parameter, challenge, and chooses the time parameter: $x, t, \mathsf{pp} \leftarrow_{\mathrm{R}} \mathsf{Setup}(1^{\lambda})$.
2 $\mathcal{A}$ selects a PPT algorithm $\mathcal{E}$, to generate solution: $y' \leftarrow \mathcal{E}(\mathsf{pp}, x, t)$, where $y' \neq y \leftarrow \mathsf{Eval}(\mathsf{pp}, x, t)$.
3 $\mathcal{V}$ verifies the solution: $\{\text{accept}, \text{reject}\} \leftarrow \mathsf{Verify}(\mathsf{pp}, x, t, y')$.
$\mathcal{A}$ wins if Verify outputs accept.
A VDF is sound if $\mathcal{A}$ wins with probability $\mathsf{negl}(\lambda)$.

- A VDF is *sequential* if the following is true. Suppose $\mathcal{A}$ is provided with a public parameter pp and is given polynomially bounded time to precompute on this public parameter, prior to being provided with a random challenge x. The property of sequentiality follows if $\mathcal{A}$ is unable to compute an output y' by an algorithm $\mathcal{E}_{<t}$ which takes less that t sequential steps to calculate, such that $y' = y \leftarrow \mathsf{Eval}(\mathsf{pp}, x, t)$. This is made precise in the VDF sequentiality game.

The value x is often called the *seed*, or simply the *input* [32, 130, 155]. We use the word challenge, as this is the most consistent with this thesis. In the original definition provided by Boneh et al., it is not specified how the challenge x is sampled. Other definitions such as [130] include an additional algorithm Gen which makes this sampling explicit. We believe both approaches are valid, and we choose to follow Boneh et al. in leaving it implicit. In

VDF Sequentiality Game

1 $\mathcal{V}$ generates random public parameter: $\mathsf{pp} \leftarrow_{\mathrm{R}} \mathsf{Setup}(1^\lambda)$.
2 $\mathcal{A}$ selects a PPT algorithm $\mathcal{E}_p$ to pre-process $\mathcal{L} \leftarrow \mathcal{E}_p(\mathsf{pp}, t)$, where $\mathcal{E}_p$ runs in time $O(\mathsf{poly}(t, \lambda))$.
3 $\mathcal{V}$ generates random challenge x.
4 $\mathcal{A}$ selects a PPT algorithm $\mathcal{E}_{<t}$, where $\mathcal{E}_{<t}$ runs in less than t sequential steps to generate solution: $y' \leftarrow \mathcal{E}_{<t}(\mathcal{L}, \mathsf{pp}, x, t)$.
$\mathcal{A}$ wins if $y' = y \leftarrow \mathsf{Eval}(\mathsf{pp}, x, t)$.
A VDF is sequential if $\mathcal{A}$ wins with probability $\mathsf{negl}(\lambda)$.

the correctness and soundness games in both our VDF games and our cVDF games, we will output the challenge x and initial state on the same line as the public parameters for ease of notation, but note that generally speaking they are sampled independently of the setup algorithm.

6.2.3 A New cVDF Definition

We now provide our game-based definitions for a cVDF, where we utilise the syntax of standard VDFs, aiming to keep our definitions consistent with those presented in Section 6.2.2. After this, we will discuss the difference between the two definitions.

Definition 6.2.7 (cVDF Definition) *For any $k \in \mathbb{Z}$, a $k-$continuous VDF is the following tuple of algorithms:*

- $\mathsf{pp} \leftarrow_{\mathrm{R}} \mathsf{cVDF.Setup}(1^\lambda, t, k)$. $\mathsf{cVDF.Setup}$ takes as input a security parameter 1^λ, a time parameter t, and a continuity parameter k; and outputs the public parameter pp. $\mathsf{cVDF.Setup}$ runs in time $\mathsf{poly}(\lambda)$.

- $\mathsf{state}_{i+1} \leftarrow \mathsf{cVDF.Eval}(\mathsf{pp}, \mathsf{state}_i)$. $\mathsf{cVDF.Eval}$ takes as input the public parameter pp and a state state_i and evaluates the next state state_{i+1} in a set number of sequential steps k.

- $\mathsf{accept}, \mathsf{reject} \leftarrow \mathsf{cVDF.Verify}(\mathsf{pp}, \mathsf{state})$. $\mathsf{cVDF.Verify}$ takes as input a state and the public parameter, and outputs either accept or reject. $\mathsf{cVDF.Verify}$ must run in time at most $\mathsf{polylog}(t)$ and $\mathsf{poly}(\lambda)$.

Additionally, a cVDF needs to satisfy the following three properties.

- *Correctness*: Any state_{i+1} output by running $\mathsf{cVDF.Eval}$ on a state state_i is accepted with overwhelming probability.
- *Soundness*: An adversarial party cannot compute a state $\mathsf{state}'_{i+1} \neq \mathsf{state}_{i+1}$ that is accepted by $\mathsf{cVDF.Verify}$ with more than negligible probability. In other words, each state is computationally unique.
- *Sequentiality*: Given an honestly sampled state_0, adversarial parties cannot compute any state state_i in time $i \cdot k$, where k is the number of iterative steps between states.

We formalise these properties in the following security games.

Algorithm 6.2.6: cVDF Correctness Game

1 $\mathcal{V}$ chooses the time parameter t and continuity parameter k, and generates the public parameters and an initial state: $\mathsf{state}_0, \mathsf{pp} \leftarrow_{\mathrm{R}} \mathsf{cVDF.Setup}(1^\lambda, t, k)$.
2 $\mathcal{A}$ chooses a state i.
3 $\mathcal{P}$ runs i iterations of $\mathsf{cVDF.Eval}$ to generate solution:
$\mathsf{state}_i \leftarrow \mathsf{cVDF.Eval}^{(i)}(\mathsf{pp}, \mathsf{state}_0, t)$.
4 $\mathcal{V}$ verifies the solution: $\{\mathsf{accept}, \mathsf{reject}\} \leftarrow \mathsf{cVDF.Verify}(\mathsf{pp}, \mathsf{state}_0, \mathsf{state}_i, t)$.
$\mathcal{P}$ wins if $\mathsf{cVDF.Verify}$ outputs accept.
A VDF is correct if $\mathcal{P}$ wins with probability $1 - \mathsf{negl}(\lambda)$.

6.2.4 cVDF Implies a VDF

We now show that in our definitions, a cVDF with $k = t$ is also a VDF according to definition 6.2.6.

Algorithm 6.2.7: cVDF Soundness Game

1 $\mathcal{V}$ chooses the time parameter t and continuity parameter k, and generates the public parameters and an initial state: $\mathsf{state}_0, \mathsf{pp} \leftarrow_{\mathrm{R}} \mathsf{cVDF.Setup}(1^\lambda, t, k)$.
2 $\mathcal{A}$ selects a PPT algorithm $\mathcal{E} \neq \mathsf{Eval}$, to choose a state state_i and generates solution: $\mathsf{state}'_{i+1} \leftarrow \mathcal{E}(\mathsf{pp}, \mathsf{state}_i, t)$, where $\mathsf{state}'_{i+1} \neq \mathsf{state}_{i+1} \leftarrow \mathsf{cVDF.Eval}(\mathsf{pp}, \mathsf{state}_i, t)$.
3 $\mathcal{V}$ verifies the solution: $\{\mathsf{accept}, \mathsf{reject}\} \leftarrow \mathsf{cVDF.Verify}(\mathsf{pp}, \mathsf{state}_i, t, \mathsf{state}'_{i+1})$.
$\mathcal{A}$ wins if cVDF.Verify outputs accept.
A VDF is sound if $\mathcal{A}$ wins with probability at most $\mathsf{negl}(\lambda)$.

Algorithm 6.2.8: cVDF Sequentiality Game

1 $\mathcal{V}$ chooses the time parameter t and continuity parameter k, and generates the public parameter. $\mathsf{pp} \leftarrow_{\mathrm{R}} \mathsf{cVDF.Setup}(1^\lambda, t, k)$.
2 $\mathcal{A}$ selects a PPT algorithm $\mathcal{E}_p$ to pre-process $\mathcal{L} \leftarrow \mathcal{E}_p(\mathsf{pp}, t, k)$, where $\mathcal{E}_p$ runs in time $O(\mathsf{poly}(t, \lambda))$.
3 $\mathcal{V}$ generates a random initial state state_0.
4 $\mathcal{A}$ chooses a state i, and selects a PPT algorithm $\mathcal{E}_{<i \cdot k}$, where $\mathcal{E}_{<i \cdot k}$ runs in fewer than $i \cdot k$ sequential steps to generate state: $\mathsf{state}'_i \leftarrow \mathcal{E}_{<i \cdot k}(\mathsf{pp}, \mathsf{state}_0, t)$.
$\mathcal{A}$ wins if $y' = y \leftarrow \mathsf{cVDF.Eval}(\mathsf{pp}, x, t)$.
A VDF is sequential if $\mathcal{A}$ wins with probability at most $\mathsf{negl}(\lambda)$.

Lemma 6.2.1 *A cVDF satisfying Definition 6.2.7, with $k = t$ is also a VDF according to Definition 6.2.6.*

Proof: When $k = t$, the only states that exist are the initial state state_0, and the final state state_1. This is equivalent to x and y in the standard VDF definitions. We therefore build a VDF using the same algorithms as a cVDF with $k = t$.

Correctness in the cVDF setting states that state_1 is verified with overwhelming probability, which implies correctness of the VDF.

cVDF soundness implies that for any state, an adversary cannot generate a false state that will be accepted by Verify. Clearly the case where there is only one state that can be verified is a special case of this, and hence VDF soundness is implied by cVDF soundness.

Similarly to the soundness property, cVDF Sequentiality guarantees that that no state i can be computed by the adversary in fewer than $i \cdot k$ sequential steps. By setting k to t, we

immediately have the VDF sequentiality property, and so we have that a cVDF with $k = t$ is also a standard VDF. □

Note that due to the lack of a continuity parameter k, this result is not possible when using the definition of [66].

6.2.5 Discussion

Upon comparison of the cVDF definitions given in Section 6.2.1 and Section 6.2.3, the first thing to note is that the latter definition is a single definition, which a candidate construction can be easily shown to satisfy using standard delay-based notions. Juxtaposed with this is the former definition, within which a construction must be shown to satisfy various non-standard notions such as B-Soundness. Additionally, the definitions given in Section 6.2.1 are heavily parameterised, which has the drawback of restricting the user to a specific compuational model. We believe that a definition for a primitive should be as generic as possible, in order to allow users who work within various computational models (discussed in 2.1.2) to be able to use these definitions. Our definitions have fewer parameters, and are significantly closer to the generic VDF definitions presented in Section 6.2.2. Indeed, we demonstrate in Lemma 6.2.1 that our definitions are an extension of a standard VDF. Note that the extra additional security parameters D, B, l, ε are generally not used in delay-based primitives [32, 39, 118, 136], and hence we view the definitions of [66] as over-parameterised.

We note that Ephraim et al.'s definition requires a cVDF to be verifiable after every evaluation of the iterated function used in $\mathsf{cVDF.Eval}$, however we generalise this to allow for a smaller number of states. One can see this as each state being k iterations of the underlying iterated sequential function, where k is represented by the continuity parameter. This allows one to obtain a trade-off between the efficiency of the setup, and the number of

pulses of randomness that are output, and hence build a more practical randomness beacon. We demonstrate this in Section 6.5.

6.3 Technical Overview of our Construction

A VDF is a sequential function which takes a predictable amount of time to evaluate and has an output which is very fast to verify. As discussed in Section 6.1, a cVDF is an extension of this which introduces the notion of a state. A state is an intermediate point in the computation which can be verified, as discussed in Section 6.1.1. In the model of [66], there is a state at every step of the underlying iterated function. However, this seems inefficient in practice, due to the requirements of applications such as a randomness beacon: NIST suggest a pulse of randomness once a minute [137]. Verifying multiple states every second is therefore something that reduces efficiency, which may be unnecessary. Indeed, we will see in Section 6.7.4 that 2^{25} iterations of repeated squaring and reducing runs in a matter of minutes on consumer-grade hardware.

We now give an overview of our candidate cVDF and how we extend it to build a randomness beacon. In our construction we follow the outline of the NIST project on interoperable randomness beacons [137] and assume the existence of a trusted third party, who runs the precomputation phase in the algorithm Setup.

We instantiate our cVDF by generating an RSA modulus N, and an input $x \in \mathbb{Z}_N$ to be uniformly selected by the trusted third party. The trusted third party will know the factors $pq = N$ and will also specify a cryptographic hash function $\mathcal{H}_0$.

Recall that a cVDF introduces multiple states which can all be verified efficiently. During this setup, we require that this trusted party computes each of these states, which can be computed in time $\mathsf{polylog}(\lambda, t)$. Each of these states is then hashed with $\mathcal{H}_0$, and placed in an array hash_0.

For a solving party to evaluate the cVDF, they start with the input x, and repeatedly run a square and reduce modulo N algorithm until they reach each state. They then publish the state, which can be easily publicly verified, as an external party need only hash the state and compare it against the array hash_0.

To extend this to a randomness beacon, we introduce a second hash function $\mathcal{H}_1$.

In the evaluation stage, when the solving party reaches a state, $\mathcal{H}_0$ is used to verify the state as before, and $\mathcal{H}_1$ is used to extract the randomness from the state, with the pulse $H_1(\mathsf{state}_i)$ being the beacon output.

Now, for any external party to publicly verify a state, they need only hash it with $H_0(\mathsf{state}_i)$ and compare it against value $\mathsf{hash}_0[i]$ in the array, to ensure that the state was correctly computed, and then hash it with $H_1(\mathsf{state}_i)$ to extract the pulse of randomness.

The simplest approach to selecting $\mathcal{H}_1$ is to require the trusted party to choose $\mathcal{H}_1$ as part of the Setup algorithm. The alternative is for a group of external parties to select $\mathcal{H}_1$ using multi-party computation, in order to reduce trust in the trusted third party (we discuss this further in Section 6.7.1).

In our construction, we implement a solution where the evaluator stores all of the intermediate states, in order for all previous states to be verified. This is in order to fulfill the NIST requirement that any past pulse is publicly accessible [137]. Note that it would be trivial to store previous states instead on, for instance, a public append-only ledger, which would remove any storage requirements from the evaluator. However, given that the memory requirements are very low (storing 100000 pulses comes to 25.6MB), we let the evaluator store these pulses, simplifying our construction.

Finally, we note that we use a setup based upon the BBS-CSPRNG, as seen in Chapters 4 and 5. This allows us the additional option of factoring the RSA modulus at the end of the computation, which is advantageous as it allows an evaluator to be rewarded, and makes it easy to perform regular, public verification of the setup procedure.

6.4 cVDF Design and Implementation

There is currently only one concrete cVDF construction in the literature, which was proposed in [66]. This construction has a significant efficiency loss as the time parameter increases, as we shall see in Section 6.5. As such, in this section we will build an alternative cVDF construction, where we shall lean on a trusted setup to build a highly efficient cVDF, at the price of a more centralised setup.

We build a cVDF using the BBS setup and repeated squaring that has been a theme of this thesis, having also been used in Chapters 4 and 5. We will explore the use of a trusted setup and precomputation phase, which allows us to optimise the efficiency of the cVDF, giving us a very fast verification. We will then provide a discussion of the trust vs efficiency trade-off obtained when compared with the existing cVDF constructions.

We then provide the full detail of our cVDF construction. The Python code for our construction can be found in https://github.com/acnsrandom/randombeacon.

6.4.1 cVDF Overview

A summary of the inputs and outputs for the cVDF.Setup, cVDF.Verify, and cVDF.Eval algorithms of our cVDF construction are as follows:

$$
\begin{aligned}
\mathsf{pp} &:= [N, t, k, C, \mathsf{cp}, \mathsf{hash}_0], \\
\mathsf{state} &:= [x_{-t}] \quad \leftarrow_{\mathrm{R}} \quad \mathsf{cVDF.Setup}(1^\lambda, t, k) \\
\mathsf{state} &:= [x_{-t}, x_i, \ldots] \quad \leftarrow \quad \mathsf{cVDF.Eval}(\mathsf{pp}, \mathsf{state}, k) \\
\mathsf{V} &:= \{\mathsf{Accept}, \mathsf{Reject}\} \quad \leftarrow \quad \mathsf{cVDF.Verify}(\mathsf{pp}, \mathsf{state})
\end{aligned}
$$

cVDF.Setup computes the public parameter pp and the *initial* state. Note that due to the inclusion of a precomputation phase in the setup, we include the generation of the initial state in the Setup algorithm. The public parameter pp contains the modulus N, the time parameter

t, the continuity parameter k, the challenge $C := (x, x_0)$, where $x \in \mathbb{Z}_N^*$ is uniformly selected and $x_0 \equiv x^2 \bmod N$, an array of *checkpoint* values cp which contains $k+1$ elements, and a hash table hash_0 which also contains $k+1$ elements. As an example, let $t = 100$ and $k = 5$, then $\mathsf{cp} = [100, 80, 60, 40, 20, 0]$ and $(x_{-100})^{2^{100}} = x_0 \bmod N$. Next, the initial $\mathsf{state} = [x_{-100}]$ is set. Finally, let $\mathcal{H}_0$ be a hash function, then:

$$\mathsf{hash}_0 = [\mathcal{H}_0(x_{-100}), \mathcal{H}_0(x_{-80}), \ldots, \mathcal{H}_0(x_{-20}), \mathcal{H}_0(x_0)].$$

We have additionally included as a separate output the initial state, which consists of an array with the single term x_{-t}, where $x_{-t}^{2^t} \equiv x_0 \bmod N$ is calculated using the Chinese Remainder Theorem. In order to remain precisely within definition 6.2.7, we could consider including the initial state within the public parameter pp. However, we have separated it out for clarity. Note that generating the initial state during setup makes the pre-processing step in the sequentiality game obsolete in our case, as there will likely be no separation between the release of the public parameter and the initial state in practice.

$\mathsf{cVDF.Verify}$ takes the public parameter pp and validates the soundness of the *current* state. If the current state is sound, $\mathsf{cVDF.Verify}$ outputs Accept, else it outputs Reject.

$\mathsf{cVDF.Eval}$ takes the public parameter pp and uses $\mathsf{cVDF.Verify}$ to validate the soundness of the *current* state. If the current state is sound $\mathsf{cVDF.Eval}$ proceeds to calculate the *next* state in $\frac{t}{k}$ sequential steps. If the current state is not sound, $\mathsf{cVDF.Eval}$ will not calculate the next state.

6.4.2 cVDF Construction

In this section we provide the full details of the algorithms in our cVDF, namely $\mathsf{cVDF.Setup}$, $\mathsf{cVDF.Verify}$, and $\mathsf{cVDF.Eval}$.

Algorithm 6.4.1: cVDF.Setup

```
input  : (1^λ, t, k)
cp := GenCheckPoints(t, k)
N, p, q := GenModulus(1^λ)
C := GenChallenge(N, p, q)
preimage := GenPreImage(N, p, q, C, cp)
hash_0 := GenHashTable(preimage)
output : pp := [N, t, k, C, cp, hash_0], state := [preimage[0]]
```

First cVDF.Setup generates the checkpoint array cp using the time and continuity parameters t and k in the algorithm GenCheckPoints.

Algorithm 6.4.2: GenCheckPoints

```
input  : (t, k)
if t mod k = 0 then
    x := t/k
    cp := [] // empty array
    y := k
    while |cp| < k + 1 do
        cp||(x · y)
        y := y − 1
    end
end
output : cp
```

GenCheckPoints first checks if $t \bmod k = 0$. Next, variable x is set to $\frac{t}{k}$. In a toy example, if $t := 100$ and $k := 5$, then $x := \frac{100}{5} = 20$. Then the empty array cp is created. Next, the variable y is initially set to k, that is $y := 5$. Next, while the size of the array cp is less than $k+1$ then generate the checkpoint values and append them to cp. Initially the size of the cp array is 0 which is less than $5+1$. Then, the array cp is appended with the element $x \cdot y$, which in our example is initially $20 \cdot 5 = 100$. Next, y is decremented by 1. In our example y is then set to 4. On the second iteration the array cp has a length of 1 which is less than $5+1$, therefore cp is appended with $20 \cdot 4 = 80$ and the variable y is decremented to 3. This continues until cp $:= [100, 80, 60, 40, 20, 0]$ and the array is output.

Algorithm 6.4.3: GenModulus

```
input  : (1^λ)
x := 0
while x < 1 do
    p := φ(1^λ)
    q := φ(1^λ)
    if p ≠ q then
        N := p · q
        x := x + 1
end
output : N, p, q
```

Next, cVDF.Setup generates the modulus N from the primes p and q using the algorithm GenModulus. GenModulus takes as input the security parameter 1^λ. It first sets variable x to 0, then enters a while loop which runs until x is equal to 1.

In the while loop the primes p and q are generated using the function $\phi(1^\lambda)$. This function uses the Miller-Rabin primality test to generate the relevant primes and ensures that p and q are congruent to 1 mod 4 [125]. This ensures that N is a Blum integer which is needed to prove the sequentiality of our construction. If $p \neq q$ then N is set to $p \cdot q$ and x is incremented by 1 thus exiting the while loop. GenModulus then outputs N, p, q.

Algorithm 6.4.4: GenChallenge

```
input  : N, p, q
J_p(x), J_q(x) := 1
while ¬(J_p(x) = 1 ∧ J_q(x) ≠ 1) ∧ ¬(J_p(x) ≠ 1 ∧ J_q(x) = 1) do
    x := U(2, N)
    J_p(x) := x^((p−1)/2) mod p
    J_q(x) := x^((q−1)/2) mod q
end
x_0 := x^2 mod N
C := [x, x_0]
output : C
```

Next, cVDF.Setup runs the GenChallenge algorithm with inputs N, p, q to generate the challenge C. First, GenChallenge sets the Jacobi symbol for $\mathcal{J}_p(x)$ and $\mathcal{J}_q(x)$ to 1. Next,

GenChallenge enters a while loop. The while loop will find an x such that $x \in \mathcal{QNR}_N^{-1}$ using the logic statement on line 2. A suitable x is found by uniformly selecting x in the range 2 through N (both not inclusive) such that $\mathcal{J}_p(x) = 1$ and $\mathcal{J}_q(x) = -1$ or $\mathcal{J}_p(x) = 1$ and $\mathcal{J}_q(x) = -1$. When a suitable x is found x_0 is set to $x^2 \bmod N$ and the challenge C is set to the array $[x, x_0]$ and output.

Algorithm 6.4.5: GenPreImage

input : N, p, q, C, cp
1 preimage := []
2 **for** *each* $i \in cp$ **do**
3 $\quad \alpha_i := x_0^{\frac{p+1}{4}^{i} \bmod p-1} \bmod p$
4 $\quad \beta_i := x_0^{\frac{q+1}{4}^{i} \bmod q-1} \bmod q$
5 $\quad x_{-i} := \alpha_i q(q^{-1} \bmod p) + \beta_i p(p^{-1} \bmod q) \bmod N$
6 $\quad$ preimage$||x_{-i}$
7 **end**
output : preimage

Next, cVDF.Setup runs the GenPreImage algorithm with inputs N, p, q, C, cp to generate the preimage array. First, the empty array preimage is created. Next, a for loop is run which iterates through each item in the checkpoints array cp. Returning to the toy example, let cp $:= [100, 80, 60, 40, 20, 0]$. On the first iteration of the for loop $i := 100$. The term x_{-100} is calculated, where $(x_{-100})^{2^t} \equiv x_0 \bmod N$. To calculate x_{-100} in polynomial time, Euler's Criterion, the Fermat-Euler Theorem and the Chinese Remainder Theorem are applied. Euler's Criterion and the Fermat-Euler Theorem are used on lines 3 and 4 to calculate the 100th square root of $x_0 \bmod p$ and $x_0 \bmod q$ respectively. Next, these terms are used to calculate x_{-100} using the Chinese Remainder Theorem on line 5, and x_{-100} is added to the array preimage. The same is repeated for the terms $x_{-80}, x_{-60}, \ldots, x_{-20}$ until x_0 is reached and added to the preimage array. Once x_0 is added to the preimage array the for loop terminates and GenPreImage outputs this array.

Algorithm 6.4.6: GenHashTable

```
input  : preimage
hash_0 := []
for each j ∈ preimage do
    h := H_0(j)
    hash_0||h
end
output : hash_0
```

Next, cVDF.Setup runs the GenHashTable algorithm with the preimage array as input to generate the hash_0 array. GenHashTable first creates an empty array hash_0. Next, the algorithm iterates through each element in the preimage array and hashes them with the algorithm $\mathcal{H}_0$. Each hashed element is then added to the hash_0 array. When the final term x_0 is hashed and added to the array the GenHashTable algorithm outputs hash_0.

Finally, cVDF.Setup sets the public parameter pp to the array $[N, t, k, C, \mathsf{cp}, \mathsf{hash}_0]$ and sets the *initial* state by populating it with the first term in the preimage array. In the toy example, the initial state would be an array with the term x_{-100}.

Algorithm 6.4.7: cVDF.Eval

```
input  : pp, state, k
seed := state[-1]
V := cVDF.Verify(pp, state)
if |state| < k+1 ∧ V = 'Accept' then
    state_i := seed^(2^(t/k)) mod N
    state||state_i
end
output : state
```

The cVDF.Eval algorithm takes as input the public parameter pp and the *current* state and outputs the next state. First cVDF.Eval sets the seed term as the last element in the state array. Next it runs cVDF.Verify. If the length of the state array is less than $k+1$ and cVDF.Verify validates that the last term in the current state array is sound then cVDF.Eval performs the sequential calculation. The term state_i is set to $\mathsf{seed}^{2^{\frac{t}{k}}} \bmod N$. This is calculated in $\frac{t}{k}$

sequential steps using the Square and Multiply binary exponentiation algorithm, noted in Algorithm 3.1.1. Once state_i is calculated it is added to the state array and output.

Algorithm 6.4.8: cVDF.Verify

```
input  : pp, state
V := 'Reject'
if state ≤ k + 1 then
    h := H_0(state[−1])
    if h = hash_0[|state| − 1] then
        V := 'Accept'
    end
end
output : V
```

The cVDF.Verify algorithm takes as input the public parameter pp and the *current* state and outputs $\mathsf{V} := \{\mathsf{Accept},\ \mathsf{Reject}\}$. The algorithm first sets V to Reject. Next, if the state is less than $k+1$ then h is set to the hash of the latest hash in the state array by running the function $\mathcal{H}_0(\mathsf{state}[-1])$. If h is equal to the hash at the correct position in the hash_0 array, then V is set to Accept.

In our example, the initial state output by cVDF.Setup contains the element x_{-100}. After cVDF.Eval is first run, the state array will be populated with the terms $[x_{-100}, x_{-80}]$. Therefore, $\mathcal{H}_0(\mathsf{state}[-1])$ will hash the term x_{-80}. Next, recall that: $\mathsf{hash}_0 := [\mathcal{H}_0(x_{-100}), \mathcal{H}_0(x_{-80}), \ldots, \mathcal{H}_0(x_{-20}), \mathcal{H}_0(x_0)]$. Therefore, $\mathsf{hash}_0[|\mathsf{state}|-1]$ will be the element at index 1 in the array, which is the term $\mathcal{H}_0(x_{-80})$. Hence, $h = \mathsf{hash}_0[|\mathsf{state}|-1]$ will evaluate to True and V will be set to Accept.

6.5 Efficiency

In this section we compare our construction against other cVDF constructions. There are currently two published cVDF schemes, both of which are extensions of existing VDF schemes [17, 66]. The former is a generic extension, and the latter an explicit extension.

Therefore, we believe that the most relevant comparison is against the underlying VDF candidates, and hence we give a detailed comparison against the current VDF proposals, and then discuss the cVDF extensions.

6.5.1 Comparison Against VDF Constructions

In this section we compare the efficiency of the cVDF.Setup, cVDF.Eval and cVDF.Verify algorithms of our cVDF against the other RSW-based VDF candidates and the isogeny-based VDF. We also compare the size of the solution and optional proof parameter output by cVDF.Eval of our cVDF against the other VDF candidates.

Currently, there are two classes of VDF, the RSW-based and the isogeny-based. The RSW-based VDFs consist of schemes defined by Wesolowski [155] and Pietrzak [130]. These VDFs are both based upon exponentiation in a group of unknown order. We refer to the framework introduced by De Feo et al. in [68] as the isogeny-based VDF candidate. This VDF is based on supersingular isogenies and pairings in elliptic curve cryptography. We will use the parameter λ to refer to the number of bits of security we expect from each protocol, and we follow [68] in assuming t is super-polynomial in λ. This allows for a meaningful comparison between the VDF candidates, where minimising t is the priority.

Setup As we mentioned in Section 6.2.2, some works include an additional algorithm Gen, whilst others do not. To provide a meaningful comparison we normalise the constructions by unifying the Gen algorithm into the discussion of the cVDF.Setup algorithm. The first step of cVDF.Setup in our cVDF is to generate an RSA modulus N which must be a Blum integer (Section 3.2). The RSW-based VDFs share a similar setup. Indeed, they all require the generation of an RSA modulus. For ease of comparison, we will assume a trusted setup in this phase[1]. In Pietrzak's construction, the primes are required to be safe primes. That is,

[1] We note, however, that the other RSW-based VDFs have an additional option of using a multi-party setup as discussed in Section 6.7.2.

a prime p such that $(p-1)/2$ is also prime [130]. Our cVDF also requires the generation of the checkPoints array, as seen in Algorithm 6.4.2, which has complexity $O(\lambda \log(t/k))$. Therefore, the RSW-based VDFs have a smaller asymptotic complexity of Setup than our cVDF. This is in contrast to the isogeny-based VDF which has a longer setup of $O(t\lambda^3)$ [68]. That is, a notable gap in efficiency is present between the isogeny-based VDF and our cVDF.

The main constraint of our cVDF is that $\mathsf{cVDF.Setup}$ must be run for each new challenge. However, our cVDF can be verified continuously, meaning that each challenge can be verified at multiple different points without the need to generate a new seed.

Eval The definition of a VDF indicates that the time complexity of computing $\mathsf{cVDF.Eval}$ should be t regardless of the amount of computational power used [32]. Note that the same is true for our cVDF, but with parameter t/k, and hence we assume $k = 1$ here for a direct comparison. If a proof is included in $\mathsf{cVDF.Eval}$ the run time is increased.

Our cVDF and the isogeny-based VDF have empty proofs, and hence require t sequential steps for honest evaluation. Contrastingly, the other RSW-based VDFs require a proof in their construction. It takes $O(\sqrt{t})$ group operations to construct Pietrzak's proof, and $O(t)$ operations to construct Wesolowski's proof [33].

In an implementation study by Attias et al. [10], the time spent generating the proof was similar to the evaluation time for both constructions. Both VDFs address this limitation by indicating that the computation of the proofs can be parallelised to reduce the run time. However, the run time overhead of proof computation in $\mathsf{cVDF.Eval}$ is an additional constraint when compared to a VDF with an empty proof.

Verify Our cVDF runs $\mathsf{cVDF.Verify}$ in $O(1) \ll \mathsf{polylog}(t)$. This is much more efficient than the $O(\lambda^4)$ used by the isogeny VDF and Wesolowski's VDF, and the $\log(t)$ used in Pietrzak's VDF [68].

Storage requirements. The RSW-based VDFs require the storage of an RSA modulus, and the challenge parameters. Whilst our cVDF has an empty proof, the RSW-based VDFs

have an additional proof to store. Wesolowski's proof has the size of one group element, whereas Pietrzak's proof has size $\log_2(t)$ elements [33].

The isogeny-based VDF has an empty proof, but still requires the evaluator to use large amounts of storage due to the public parameters. In the case where the evaluator runs the VDF in optimal time, $O(t)$ storage is required. This can be improved upon using a time-memory trade off, meaning $O(t/n)$ storage is required instead, at the cost of increasing evaluation time to $O(t \log n)$. The authors used the parameter $n = 1244$ when benchmarking this scheme [68].

In our scheme, storage can be set to be constant, i.e. the length of one element of the RSA group, by keeping the array of pulses on a public ledger, rather than being stored by the evaluator. We summarise our analysis in Table 6.1.

Table 6.1 Theoretical efficiency comparison of our cVDF against alternative VDF candidates.

VDF	Setup	Sequential Eval	Parallel Eval	Verify
Wesolowski	$O(\lambda^3)$	$O((1+\frac{2}{log(t)})t)$	$O((1+\frac{2}{s\ log(t)})t)$	$O(\lambda^4)$
Pietrzak	$O(\lambda^3)$	$O((1+\frac{2}{\sqrt{t}})t)$	$O((1+\frac{2}{s\sqrt{t}})t)$	$O(\log(t))$
Isogeny	$O(t\log(\lambda))$	$O(t)$	$O(t)$	$O(\lambda^4)$
Our cVDF	$O(\lambda^3\log(t))$	$O(t)$	$O(t)$	$O(1)$

6.5.2 Discussion

Table 6.1 illustrates that the main advantages of our cVDF is its lack of proof and the efficiency of cVDF.Verify. The isogeny-based VDF shares the benefit of having an empty proof. However, it has a Setup algorithm which grows linearly with t, which is very impractical.

Our cVDF and the isogeny-based VDF have the most efficient evaluation due to their lack of proofs. However, both of these candidates have less efficient setups when compared to the RSW-based VDFs: the isogeny-based VDF has a setup of order $O(t)$, making it impractical

for long-running VDFs. Our cVDF has a setup of $O(\lambda^3 \log(t))$, longer than the RSW-based VDFs, but significantly shorter than that of the isogeny-based VDFs, and an additional limitation that it is single-use. The other RSW-based VDFs have the additional strength that challenges with different time parameters can be generated for each setup, which is not possible with our cVDF and the isogeny-based VDF.

However, in the implementation study of Attias et al. [10], the proving time of both RSW-based VDFs was similar to the evaluation time, which is undesirable. By contrast, our cVDF and the isogeny-based VDF excel where fast verification is desired. Both have very efficient verification procedures which are non-interactive.

Ephraim et al.'s cVDF [66] is a generalisation of Pietrzak's protocol, where the length of proof computation and storage grows as more states are added. Asymptotically, the proof size and verification time of this construction both grow in time $O(\log(t))$, as compared to our cVDF where they are constant. The authors state that the proof size is $(z-1)^2(\log_z(t))^2)$ elements, and that verifying a proof requires computing $O(n \cdot \rho)$ squares, where n is the number of elements in the proof, and ρ is a parameter which can be varied leading to various time-memory trade-offs. In [66], they suggest setting ρ to be equal to the security parameter.

In comparison, our construction has a constant verification time, and hence is overall much more efficient.

Baum et al.'s cVDF In [17], Baum et al. give a construction of a randomness beacon by generically building a cVDF from a VDF. As such, the efficiency of this approach clearly relies upon that of the underlying VDF, which perform worse than our approach, as we can be seen in Table 6.1 and in greater detail in Section 6.5.1.

To conclude, whilst our setup procedure takes longer than in the other RSW-based constructions, it is still significantly faster than that of the isogeny-based VDFs. However, our setup benefits from the property that it only needs to be ran once to extract many pulses

of randomness. To compensate for our setup, we have the fastest verification to date, the smallest storage cost for the evaluator (when storing states publicly), and the smallest amount of computation due to the proof being empty. In the randomness beacon application such fast verification is extremely useful, particularly when used by devices without the capability for parallelising computation.

6.6 Security

In this section we prove the *correctness, soundness* and *sequentiality* properties of the cVDF.

6.6.1 Correctness

The correctness proof must show that Algorithm 6.4.8 cVDF.Verify accepts any state i computed by i iterations of algorithm 6.4.7 cVDF.Eval, upon the initial state.

Theorem 6.6.1 *Our cVDF construction is correct.*

Proof: Suppose the trusted third party $\mathcal{T}$ runs Algorithm 6.4.1 cVDF.Setup, and that the adversary $\mathcal{A}$ chooses a target state i.

In Algorithm 6.4.1 Setup, state_i is defined as $x_i \equiv x_0^{2^{i \cdot k}} \bmod N$. The prover $\mathcal{P}$ runs i iterations of Algorithm 6.4.7 cVDF.Eval, to compute $\mathsf{state}_i \leftarrow \mathsf{cVDF.Eval}^{(i)}(\mathsf{pp}, \mathsf{state}_0, t)$, and then publishes state_i. Recall from Theorem 4.4.3 that Algorithm 3.1.1 Square and Multiply correctly calculates the term x_i, where $x_i \equiv x_0^{2^i} \bmod N$ for any $t \in \mathbb{Z}$. As cVDF.Eval calls the Square and Multiply Algorithm, we have that $\mathsf{cVDF.Eval}^{(i)}(\mathsf{pp}, \mathsf{state}_0, t)$ will compute $\mathsf{State}_i \equiv x_0^{2^{i \cdot k}} \bmod N$, and hence State_i is equal to the preimage of $cp[i]$. The Verifier $\mathcal{V}$ runs Algorithm 6.4.8, which computes $H_0(\mathsf{state}_i)$, and outputs accept if $H_0(\mathsf{state}_i) = cp[i]$. As hash functions are deterministic, we have that $H_0(\mathsf{state}_i) = cp[i]$, and hence our cVDF is correct. □

6.6.2 Soundness

Theorem 6.6.2 *Our cVDF construction is sound.*

Proof: Our construction is sound if $\mathcal{A}$ can't win the soundness game presented in Algorithm 6.2.7 with more than negligible probability.

Assume an adversary chooses a target state i, and computes a value $\mathsf{state}'_i \neq \mathsf{state}_i$ which verifies as correct at state i, with greater than negligible probability. cVDF.Verify will accept the value state'_i if and only if $H(\mathsf{state}'_i) = A[t]$. If the adversary first computes state_i, then finding a second state'_i which is accepted by cVDF.Verify with more than negligible probability using state_i would break the collision resistance property described in Definition 3.1.3. Otherwise, for the adversary to find a value which would be accepted by cVDF.Verify with more than negligible probability would break the pre-image resistance of the hash function, as described in Definition 3.1.3.

Therefore, an adversary cannot win the soundness game with more than negligible probability. □

6.6.3 Sequentiality

Recall that the Setup of our construction is based upon the BBS setup used in chapters 4 and 5, and hence we can utilise the security proofs given earlier in this thesis. We recall Lemma 4.4.1 states the following:

Given any (N,P,t) output by Algorithm Gen, the RSA modulus N cannot be factored in time less than t, with more than negligible probability.

In Chapter 4, N is a blum integer, $P = (x, x_0, x_{-t})$, where $x \in \mathcal{QNR}_N^{-1}$, $x_0 \equiv x^2 \bmod N$, and x_{-t} is the tth square root of x_0. One can see that this is equivalent to the pair (C, state_0), in the cVDF construction.

In the proof of Lemma 4.4.1, we proved the following two statements: i) Attempts to compute an x', where $x' \equiv \sqrt{x_0} \bmod N$ and $x' \in \mathcal{QR}_N$, in less than t sequential steps reduce to breaking the RSW time-lock assumption. ii) Attempts to recover the non-trivial factors of N using a method that does not use x' reduces to an open problem given by Theorem 9 of Blum et al. in 1986 [26].

We use these results in the proof of the following theorem.

Theorem 6.6.3 *Our cVDF construction is sequential.*

Proof: Suppose $\mathcal{T}$ honestly generates a random public parameter pp, and initial state state_0, and presents this to a PPT adversary $\mathcal{A}$.

The adversary additionally chooses a state i and a PPT algorithm $\mathcal{E}_{<i\cdot\lambda}$, which runs in less than $i \cdot \lambda$ sequential steps, outputting a state.

By the cVDF sequentiality game described in Algorithm 6.2.8, we see that the adversary wins if cVDF.Verify accepts the output of $\mathcal{E}_{<i\cdot\lambda}$ (pp,state_0).

We now show that if the adversary wins the game with greater than negligible probability, then they are breaking either the RSW-time lock assumption, BBS-shortcut assumption, or the pre-image property of a hash function.

Assume that the adversary runs an algorithm $\mathcal{E}_{<i\cdot\lambda}$ which provides a non-negligible advantage in the cVDF sequentiality game.

As N is a Blum integer, it cannot be factored by any PPT algorithm in time less than t with more than negligible probability, as a result of Lemma 4.4.1, by the BBS shortcut assumption given in Definition 4.4.1. Therefore the adversary cannot learn the trapdoor $\phi(N)$, and speed up the computation. If the adversary calculates the correct value state_i without knowledge of the trapdoor, then as $\mathcal{E}_{<i\cdot\lambda}$ runs in fewer than $i \cdot \lambda$ steps, this contradicts the RSW time-lock assumption.

On the other hand, if the adversary is able to find a solution that verifies as correct without computing state_i, then this contradicts the pre-image resistance property of the hash function as described in Definition 3.1.3. □

6.7 Randomness Beacon

Probably the most important application of VDFs and cVDFs is a randomness beacon, as we described in the introduction of this chapter. In this section, we explain how our cVDF definitions and construction extend to a randomness beacon.

In [66], any secure cVDF is shown to yield a randomness beacon under the Tick/Tock paradigm, as discussed in Section 6.1.1. In this section we adapt the randomness beacon definition of [66] in line with our new cVDF definitions, and provide a construction. We show that any secure cVDF under our definitions yields an RB, and hence our construction is secure.

We begin by defining a randomness beacon, broadly following the definition of [66]. Note that we omit the *Honest Evaluation* property, which states that computing each step must take no more than the allocated number of sequential steps. We instead make explicit in Algorithm Tick that this algorithm takes exactly k sequential steps to compute.

Definition 6.7.1 (Randomness Beacon) *A public randomness beacon consists of four algorithms RB.Setup, RB.Tick, RB.Tock, and RB.Verify:*

- $\mathsf{pp}, G \leftarrow_{\mathrm{R}} \mathsf{RB.Setup}(1^\lambda, t, k)$. RB.Setup is an algorithm that takes as input security parameter 1^λ, a time parameter t, and a continuity parameter k and outputs the public parameter pp and a PRG G. RB.Setup runs in time $\mathsf{poly}(\lambda)$.

- $\mathsf{state} \leftarrow \mathsf{RB.Tick}(\mathsf{pp}, \mathsf{state}, k)$. RB.Tick is an algorithm that takes as input the public parameter pp and the *current* state and evaluates the *next* state in k sequential steps.

- $\mathsf{pulse} \leftarrow \mathsf{RB.Tock}(\mathsf{pp},\ \mathsf{state},\ \mathsf{G})$. $\mathsf{RB.Tock}$ is an algorithm that takes as input the public parameter pp, the current state, and the PRG G, and outputs a pulse of randomness.

- $\mathsf{V} := \{\mathsf{Accept}, \mathsf{Reject}\} \leftarrow \mathsf{RB.Verify}\ (\mathsf{pp}, \mathsf{pulse}, \mathsf{state}, G)$. $\mathsf{RB.Verify}$ is an algorithm that takes as input the public parameters, a pulse and a state, before outputting either accept or reject. $\mathsf{RB.Verify}$ runs in time $\mathsf{polylog}(t)$ and $\mathsf{poly}(\lambda)$.

Additionally, a randomness beacon needs to satisfy the following three properties:

- *Correctness*: For each i, pulse_i can be verified as the pulse of state_i in time $\mathrm{polylog}(i \cdot k)$.

- *Soundness*: An adversary cannot compute a $\mathsf{pulse}'_i \neq \mathsf{pulse}_i$ that is accepted by verify with more than negligible probability. This property states that each pulse is computationally unique.

- *Indistinguishability* Each pulse of randomness extracted at state i should be indistinguishable from random for an adversary bounded by time $i \cdot k$ after being given state_0.

We formalise these properties in the following security games.

Algorithm 6.7.1: RB Correctness Game

1 $\mathcal{V}$ chooses the time parameter t and continuity parameter k, and generates the public parameters, pseudo-random generator and an initial state:
$\mathsf{state}_0, \mathsf{pp}, G \leftarrow_{\mathrm{R}} \mathsf{RB.Setup}(1^\lambda, t, k)$.
2 $\mathcal{A}$ chooses a state i.
3 $\mathcal{P}$ runs i iterations of tick to generate solution: $\mathsf{state}_i \leftarrow \mathsf{RB.Tick}^{(i)}(\mathsf{pp}, \mathsf{state}_0, t)$.
4 $\mathcal{P}$ runs $\mathsf{pulse} \leftarrow_{\mathrm{R}} \mathsf{RB.Tock}(\mathsf{state}_i)$.
5 $\mathcal{V}$ verifies the solution: $\{\mathsf{accept}, \mathsf{reject}\} \leftarrow \mathsf{RB.Verify}(\mathsf{pp}, \mathsf{pulse}, \mathsf{state}_i, G)$.
$\mathcal{P}$ wins if $\mathsf{RB.Verify}$ outputs accept.
A VDF is correct if $\mathcal{P}$ wins with probability $1 - \mathsf{negl}(\lambda)$.

Algorithm 6.7.2: RB Soundness Game

1 $\mathcal{V}$ chooses the time parameter t and continuity parameter k, and generates the public parameters, pseudo-random generator and an initial state:
$\mathsf{state}_0, \mathsf{pp}, G \leftarrow_{\mathrm{R}} \mathsf{RB.Setup}(1^\lambda, t, k)$.

2 $\mathcal{A}$ selects a PPT algorithm $\mathcal{E} \neq \mathsf{Eval}$, to choose a target state_i and pulse_i, and generate a candidate alternative pair $(\mathsf{state}, \mathsf{pulse})' \leftarrow \mathcal{E}(\mathsf{pp}, \mathsf{state}_i, t)$, where $(\mathsf{state}, \mathsf{pulse})' \neq (\mathsf{state}, \mathsf{pulse})$ output by algorithms tick and tock respectively.

3 $\mathcal{V}$ verifies the solution: $\{\mathsf{accept}, \mathsf{reject}\} \leftarrow \mathsf{RB.Verify}(\mathsf{pp}, \mathsf{state}_i, t, \mathsf{pulse}'_i)$.
$\mathcal{A}$ wins if RB.Verify outputs accept.
A VDF is sound if $\mathcal{A}$ wins with probability at most $\mathsf{negl}(\lambda)$.

Algorithm 6.7.3: RB Indistinguishablity Game

1 $\mathcal{V}$ chooses the time parameter t and continuity parameter k, and generates the public parameters, pseudo-random generator and an initial state:
$\mathsf{state}_0, \mathsf{pp}, G \leftarrow_{\mathrm{R}} \mathsf{RB.Setup}(1^\lambda, t, k)$.

2 $\mathcal{A}$ selects a PPT algorithm $\mathcal{E}_p$ to pre-process $\mathcal{L} \leftarrow \mathcal{E}_p(\mathsf{pp}, t, k, G)$, where $\mathcal{E}_p$ runs in time $O(\mathsf{poly}(t, \lambda))$.

3 $\mathcal{A}$ chooses a state i.

4 $\mathcal{V}$ generates a random initial state state_0

5 $\mathcal{P}$ runs i iterations of tick to generate solution: $\mathsf{state}_i \leftarrow \mathsf{RB.Tick}^{(i)}(\mathsf{pp}, \mathsf{state}_0, t)$, and then computes $\mathsf{pulse}_i \leftarrow \mathsf{RB.Tock}(\mathsf{pp}, \mathsf{state}_i, t)$.

6 $\mathcal{V}$ chooses $b \leftarrow_{\mathrm{R}} \{0, 1\}$ at random. If $b = 0$, $\mathcal{V}$ passes pulse_i to $\mathcal{A}$, if $b = 1$, $\mathcal{V}$ samples a random string of the same length, and passes that to $\mathcal{A}$.

7 $\mathcal{A}$ runs algorithm $\mathcal{A}_1$, which is bounded by $i \cdot t$ sequential steps, to output a bit $b' \leftarrow_{\mathrm{R}} \mathcal{A}_1(\mathsf{pp}, \mathsf{state}_0, G)$.
$\mathcal{A}$ wins if $b' = b$.
An RB is indistinguishable if $\mathcal{A}$ wins with probability at most $1/2 + \mathsf{negl}(\lambda)$.

6.7.1 Randomness Beacon from a cVDF

In this section we apply the algorithms of a generic cVDF in order to construct a Randomness Beacon. We follow Ephraim et al. [66] in assuming the existence of a pseudo-random generator (PRG), which we instantiate with a hash function in our implementation.

From the above definition, we see that the RB.Setup algorithm is equivalent to cVDF.Setup, and is used to provide public parameters pp and an *initial* state. The algorithm RB.Tick is equivalent to Eval, and is used on the *current* state to compute the *next* state of the cVDF. RB.Tock is the only additional algorithm that defines the randomness beacon; taking the state output by RB.Tick, and outputting the pulse of randomness. Note that RB.Tick and RB.Tock can be run in parallel, as illustrated in Figure 6.1. Similar to cVDF.Verify, RB.Verify checks the soundness of the current state output by RB.Tick, and additionally checks the soundness of RB.Tock, and outputs Accept or Reject accordingly.

We will provide black-box algorithms which can be used with any cVDF, and then show that using these algorithms with a secure cVDF yields a secure randomness beacon according to Definition 6.7.1. In the next section we will discuss the intricacies of our construction, and our implementation choices.

Algorithm 6.7.4: RB.Setup

input : $1^\lambda, t, k$
1 $\mathsf{pp}, \mathsf{state} := \mathsf{cVDF.Setup}(1^\lambda, t, k)$
2 $\mathcal{H}_1 \leftarrow \mathsf{chooseG}$
output : $\mathsf{pp}, \mathsf{state}, \mathcal{H}_1$

RB.Setup takes as input $1^\lambda, t, k$ and outputs the public parameter pp and state by calling the cVDF.Setup algorithm. It then selects a pseudo-random generator $\mathcal{H}_1$ using the chooseG algorithm.

RB.Tick takes as input pp and the current state and outputs the *next* state by calling the cVDF.Eval algorithm.

Algorithm 6.7.5: RB.Tick

```
input  : pp, state, k
1 state := cVDF.Eval(pp, state, k)
output : state
```

Algorithm 6.7.6: RB.Tock

```
input  : pp, state, H_1
1 pulse := H_1(state)
output : pulse
```

RB.Tock takes as input pp and the current state output by RB.Tick and outputs randomness beacon pulse.

Algorithm 6.7.7: RB.Verify

```
input  : pp, pulse, state, H_1
1 V := cVDF.Verify(pp, state)
2 if V = 'Accept' then
3     if pulse ≠ H_1(State) then
4         V = 'Reject'
5     end
6 end
output : V
```

RB.Verify takes as input pp, the current state and pulse output by RB.Tick and RB.Tock respectively, and outputs V. RB.Verify calls the algorithm cVDF.Verify. If the current state and pulse are both sound, then V is set to Accept, else it is set to Reject.

6.7.2 Trust

In this work, we set out to make the cVDF primitive more usable, and to build a randomness beacon with the fastest verification possible. To do this we use a trusted setup with a precomputation phase, which is stronger than the trust assumptions typically used for VDFs and cVDFs, where there are options to use a (typically expensive) multi-party computation in order to run the setup.

We think this requirement provides an acceptable trade-off, particularly on devices without much computation power, as current VDF constructions all require a certain level of parallelism (see [10]) or storage (see [68]) to run efficiently.

We believe that in practice trusting a third party such as NIST is a reasonable approach, used in much of current cryptography. The alternative is to trust a group of often pseudonymous parties. In this case, one is trusting that they are not working together, as if so they can break all the cryptographic guarantees that the RBs give. Additionally, this approach is less stable, with the current state-of-the art constructions [49, 50, 69, 85] vulnerable to malicious parties being able to carry out denial-of-service attacks on the setup procedure. We refer the reader to Section 2.1.3 for a discussion of this.

If one accepts this trade-off, the construction we propose achieves the most efficient verification procedure to date.

6.7.3 Discussion

The randomness beacon obtained from the cVDF given in Section 6.4 is very efficient, as we shall demonstrate in Section 6.5, but due to the trusted setup of our construction, this comes at the cost of a strong trust assumption. The precomputation phase makes it possible for the trusted party to know the states in advance, but if G is chosen by a different party after the setup procedure, the trusted party will not know the pulses in advance, which is important. We shall now discuss the various approaches to computing the PRG G.

Choosing the PRG

In this section we discuss the potential approaches to choosing the PRG G, which in our construction is instantiated using the hash function $\mathcal{H}_1$. In our implementation, this will be implemented using hash function SHA3-512. However in practice this PRG will need to be chosen by some entity. The simplest approach is to require the trusted third party (TTP) (e.g.

NIST) to specify this in cVDF.Setup. The drawback is that the TTP is able to precompute all random values ahead of time, and if they know what they will be used for, they may be able to bias the outcome.

We can reduce the amount of trust placed in them, by requiring another entity to choose the PRG *after* cVDF.Setup has been run. This means that whilst the TTP will be able to compute the random values ahead of time, they will not be able to bias them anymore. Note that these valuse can also be computed ahead of time in the more common case when a TTP only generates the RSA modulus N, in the alternative RSW-based VDFs [66, 130, 155], as a TTP in this case can use $\phi(N) := (p-1)(q-1)$ as a shortcut to compute the VDF output.

In our implementation, we use $\mathcal{H}_0$ as SHA2-256 to output hash_0 in the public parameter pp in the RB.Setup algorithm and $\mathcal{H}_1$ as SHA3-512 to output the pulse in the RB.Tock algorithm. However, in practice there are various approaches to choosing the second hash function, which lead to slightly different trust models. For example, it is simple to enumerate various different hash functions, and have a group of entities run a simple MPC computation in order to choose a hash function between them.

We note that neither of these approaches are perfect, and an untrusted construction would be preferable. However, our trusted setup is what allows us such an efficient construction.

BBS Setup Recall that the trusted setup of our construction is based upon the BBS-CSPRNG, as in the preceding chapters, which allows an evaluator (or indeed any external party) to factor the RSA modulus N.

This property is beneficial in the context of cVDFs and randomness beacons as it allows for regular public audits of the setup, as any party knowing the factors of N can check that Setup was performed correctly. Additionally, it introduces the option of encapsulating a reward for the evaluator, by encrypting e.g. the key to a bitcoin wallet under the RSA modulus N, as described in [111].

6.7.4 Practical Performance

The theoretical efficiency of our randomness beacon is directly derived from the efficiency of the cVDF, which we discussed in Section 6.5. In this section, we focus instead on the practical performance of our construction, and how this compares with the earlier theoretical efficiency discussion. We are able to draw some comparisons against the other RSW constructions due to an implementation study of Pietrzak's and Wesolowski's VDF protocols in [10], however this is not possible with the isogeny-based VDF, as to the best of our knowledge there is no publicly available implementation.

In order to demonstrate that our construction runs very efficiently on consumer-grade hardware, we benchmark our randomness beacon in Python, using a 2018 MacBook Pro 14. The specifications of the hardware are an Intel core i5 processor, a 2.3 GHz processor, and 8GB of RAM. We ran three experiments, each with 100 iterations of each algorithm.

In all experiments, we set the security parameter to $\lambda = 2048$.

The values for the first experiment were chosen to compare our results against the implementation study of Attias et al. [10] who implemented the schemes of Pietrzak's VDF and Wesolowski's VDF with values of $t = 2^{25}$. We match this, ensuring that each execution of Tick also requires 2^{25} exponentiations. However, given that Attias et al. do not detail the hardware used in this implementation, this comparison should be seen as a heuristic rather than a precise comparison.

We then increased the value of k and the value of t independently, in order to illustrate how changing these parameters impacts the resulting algorithms. Our results are shown in Table 6.2.

We see that increasing k leads to a small increase in Setup time, but does nothing to increase Tock or Verify, which is the desired functionality. Increasing t only affected the value of Tick noticeably, indicating that the time between pulses of randomness scales predictably

Table 6.2 Timings in seconds for each of our RB algorithms

Experiment	Setup	Tick	Tock	Verify
$t = 2^{25}, k = 5$	1.830	359.888	0.0002	0.0001
$t = 2^{25}, k = 25$	2.041	73.263	0.0002	0.0001
$t = 2^{26}, k = 25$	1.973	715.419	0.0001	0.0001

well as t grows. Explicitly, this shows that the asymptotic values given in Table 6.1 are not hiding any large constants.

In contrast, the protocols of Wesolowski and Pietrzak lose more time in practice than the asymptotic values indicated in Table 6.1 might suggest. Both evaluation and verification times were significantly longer than the theoretical optimum, as shown in the VDF implementation of Attias et al. [10]. In this implementation the time taken for the evaluator to compute the proof was shown to be similar to the amount of time spent on the sequential computation. Extending Piertzak's scheme to Ephraim et al.'s cVDF further reduces the efficiency, increasing the time spent on both the proof evaluation and verification time significantly. This is a significant drop in efficiency compared to our implementation, in which no time is spent computing the proof (as it is empty), and the verification procedure is a negligible constant.

Therefore, our implementation demonstrates that our construction is the most efficient VDF-based RB for use in consumer grade hardware.

6.7.5 RB Security

By design of the tick-tock paradigm, the security of the randomness beacon follows naturally from the underlying cVDF. We give a generic result for any cVDF using the algorithms provided in Section 6.7.1.

Theorem 6.7.1 *Algorithms 6.7.4 - 6.7.7 yield a secure randomness beacon when instantiated with a secure cVDF, and a secure PRG.*

Proof: *Correctness* Correctness follows immediately from the correctness of the underlying cVDF.

Soundness We show that any algorithm $\mathcal{E}$ which can win the RB soundness game presented in Algorithm 6.7.2 can also be used to break the cVDF soundness game presented in 6.2.7.

Consider an algorithm $\mathcal{E}$ which gains a non-negligible advantage in the RB soundness game. Recall from the RB soundness game that $\mathcal{E}$ outputs a tuple $(\mathsf{state}, \mathsf{pulse})'$, which is not equal to the tuple $(\mathsf{state}, \mathsf{pulse})$ for any state.

We first look at the case when $\mathsf{state}_i = \mathsf{RB.Tick}(\mathsf{state}_{i-1})$, and then the case where $\mathsf{state}_i \neq \mathsf{RB.Tick}(\mathsf{state}_{i-1})$.

If $\mathsf{state}_i = \mathsf{RB.Tick}(\mathsf{state}_{i-1})$, then we must have that $\mathsf{pulse} \neq \mathsf{RB.Tock}(\mathsf{State})$, as otherwise $(\mathsf{state}, \mathsf{pulse})' = (\mathsf{state}, \mathsf{pulse})$. In this case, we have that $\mathsf{RB.Verify}$ will be set to reject on line 4. Therefore we must have that $\mathsf{state}_i \neq \mathsf{RB.Tick}(\mathsf{state}_{i-1})$. However, for such an algorithm to be accepted by $\mathsf{RB.Verify}$ with non-negligible probability, it must be the case that $\mathsf{cVDF.Verify}$ accepts an incorrect state with non-negligible probability. Therefore we can use this algorithm to break the soundness of the underlying cVDF.

Indistinguishability Inspecting the first 5 lines of the $\mathsf{RB.Indistinguishability}$ game, we see that this is identical to the cVDF sequentiality game, with the inclusion of sampling a PRG.

If the challenger is able to use an algorithm $\mathcal{A}_1$ bounded by $i \cdot t$ sequential steps to learn the output of State_i with more than negligible probability, then they can use this same algorithm to break the sequentiality of the underlying cVDF. Therefore this is not possible. This means the adversary is unable to compute $G(\mathsf{State}_i)$ with more than negligible probability. Therefore, an adversary able to win Game 6.7.3 with more than negligible probability would

also be able to distinguish between the output of the PRG and a random string with more than negligible probability, contradicting Definition 3.1.4.

□

Corollary 6.7.1 *Algorithms 6.7.4 - 6.7.7 yield a secure randomness beacon when instantiated with the cVDF Algorithms 6.4.1, 6.4.8 and 6.4.7.*

6.8 Conclusion

We discussed the cVDF definitions proposed by [66], and provided new definitions that we believe to be easier to use, and more flexible due to the continuity parameter.

We then built a novel cVDF, which offers a materially different alternative to the construction of Ephram et al: Our cVDF enjoys extremely fast public verification, small storage requirements and an empty proof, however this comes with the clear drawback of the trusted setup. From this we built a randomness beacon following Ephraim et al.'s framework, and concretely instantiate it, providing a cVDF-based RB, with the fastest public verification to date, and an option to factor the RSA modulus at the end of the computation.

Future work We believe that the cVDF primitive is ideal for use in randomness beacons. It is our hope that our new cVDF definitions and RB extension will make it easier for researchers to build new cVDFs, with resulting randomness beacons.

There is currently a lack of consensus when determining what properties are essential and desirable for a randomness beacon. The strongest approach from a trust perspective seems to be combining various beacons together, so that the output is unbiased as long as at least one of the underlying beacons is unbiased. However, this approach comes with the natural drawback of being no faster than the least efficient beacon. We believe that our construction will work well in combination with another VDF-based beacon, which has a distributed setup,

as discussed in Section 2.1.3. The additional verification cost of our cVDF-based beacon would add an insignificant overhead when combined with other VDF-based beacons (see the analysis in Section 6.5), but ensuring that the output is unbiased provided the trusted party does not collude with those running the distributed setup.

We believe that a valuable contribution would be a framework which defines a randomness beacon, including the various properties that are either necessary or desirable in order for the beacon to function well.

We additionally think that further work needs to be done in developing and maintaining a public randomness beacon, with the end goal a publicly accessible, efficient and trustworthy beacon. This work has shown that a beacon with a highly efficient verification is possible, and a natural follow-up question is whether such speeds can be achieved without a precomputation phase. Indeed, an interesting line of theoretical research would be to explore the optimal efficiency that can be achieved under certain assumptions. For example, establishing a bound on the fastest possible beacon with a completely distributed setup would give the research community a concrete goal to aim for when proposing constructions.

Chapter 7

Conclusion

Contents

In this chapter, we summarise the work in this thesis, before discussing potential directions for future work. We make explicit the relationships between the various primitives discussed throughout this thesis, identifying a hierarchy based upon existing results, and highlighting the theoretical gaps that exist.

7.1 Summary of this thesis

In this thesis, we began in Chapter 2 by presenting the first systematisation of knowledge of delay-based primitives and constructions. Chapter 2 can serve as a starting point to explore delay-based cryptography, and is designed to help researchers or practitioners pick the correct primitive and construction for their application. We provided a comprehensive review of the trust assumptions and setup procedures used in delay-based cryptography, and highlighted

the nuances that separate each of the delay-based primitives. In this chapter, we reviewed the underlying techniques that exist in the literature to construct a delay, and discussed their practicality, and the level of community trust in each technique. We additionally discussed some of the key applications of delay-based cryptography, along with the state-of the art constructions.

In Section 7.2, we analyse how the primitives discussed in Chapter 2 are linked to each other theoretically, identifying relations and highlighting gaps in the theoretical knowledge we have so far. In Section 7.3 we propose promising research directions for the future design of delay-based cryptographic techniques and protocols. It is our hope that this inspires and aids in exciting new research in this field.

In Chapter 3, we provide the preliminary material including number theory and cryptographic primitives that we then used in the subsequent chapters.

In Chapter 4, we discuss timed-release encryption, a method of incorporating public-key encryption into delay-based cryptography. We discuss the recent definition given in [53], and introduce an alternative definition with a nuance which improves the flexibility of the primitive. We propose a new construction TIDE, which takes a novel approach to utilising a cryptographic delay, by ensuring that the output of the computation allows for the factorisation of an RSA modulus. Combined with standard RSA-OAEP encryption, we obtain a practical, efficient construction, which we show fits the application of sealed-bid auctions.

In Chapter 5, we exploit the abovementioned nuance in the definition of timed-release encryption, in order to build an extension which we term *timed-release encryption with implicit authentication*. This new primitive offers an atypical approach to the use of public key encryption, where the encryption key is kept secret, and the decryption key is encrypted

with a delay. When combined with a new privacy property named implicit authentication, we demonstrate that primitive offers an encryptor fine-grained control over the release of information, allowing the encryptor to separate the actions of distributing ciphertexts, and computing a delay that provides the decryption key. We frame this primitive in the context of whistleblowers, using the Edward Snowden case as a key motivating example.

We switch our focus in Chapter 6 away from encryption, instead looking at the primitive of verifiable delay functions, which allow a solver to compute a delay such that a member of the public may efficiently verify that the delay occurred. This primitive is particularly useful in the setting of public randomness, which we discuss at length. In particular, we focus on an extension known as a *continuous* verifiable delay function, which allows for public verification at intermediary points within the computation. We provide new definitions consistent with the style of standard delay-based primitives, and constructed a cVDF and resulting randomness beacon. We take the approach of requiring a trusted party to precompute the states in advance, in order to obtain a large speed-up in public verification, arguing that the advantage the trusted party gets is similar to that in the more standard approach of having the trusted party compute the public parameters.

7.2 Relating Delay-Based Primitives

One of the goals of this thesis is to provide the reader with an overview and understanding of how delay-based primitives are related. The reader will note that the primitives which we studied in depth, TRE and VDFs, are fundamentally different: the former contains a notion of encryption, allowing for the incorporation of techniques from public-key cryptography, whilst the latter simply proves that a delay has indeed been computed. In this section, we shall clarify this difference, making explicit the known relations between each of the delay-based

primitive that we discussed and classified in Section 2.2. Furthermore, we will highlight as research questions some gaps in the literature between other primitives.

In our study of this field, we have identified that the core of modern delay-based primitives is an *iterated sequential function*, as we discussed in Section 2.1.2.

An iteratively sequential function is defined as a function which maps from a given group into itself. Additionally, it must satisfy *sequentiality*: For a time parameter t, the following hold: i) an adversary with polynomially many processors cannot compute a valid solution in less than t sequential steps with more than negligible probability. ii) any party can compute a valid solution in t steps with overwhelming probability.

From an iterated sequential function, we can branch into the two classes of primitives identified in Section 2.2: those which allow for the recomputation of an input, which include time-lock puzzles, timed-release encryption, and timed signatures; and those that do not, such as proofs of sequential work and verifiable delay functions. We provide a mapping of the relation between these primitives in Fig. 7.1. Recall that the separation between the two classes comes from a result by Mahmoody et al. in [114], and is based on the assumption that one is using the random oracle model. Interestingly, it is claimed that DE implies a weak form of TLPs [39], which suggests that it may be possible to circumvent this separation result of Mahmoody et al. outside the random oracle model. We believe this question warrants further study.

We shall start by discussing the lower branch in the diagram. In order to build a proof of sequential work from an iterated sequential function, it is required that there is a method for another party to publicly verify that the computation is correct, which must be significantly faster than the time taken to compute the iterated sequential function [115]. Boneh et al. [32] show that if one takes a proof of sequential work and imposes the condition that the iterated sequential function is unique, this satisfies the requirement of a VDF.

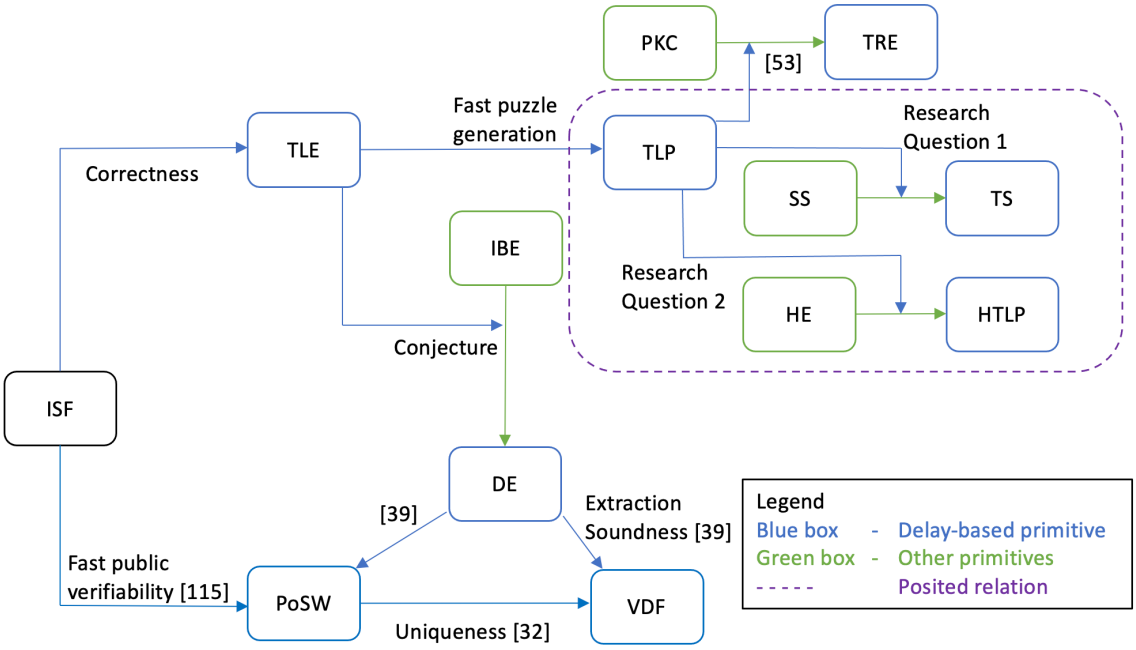

Fig. 7.1 Identified and conjectured relations between delay-based primitives.

Delay encryption is described as the identity-based analogue to time-lock encryption [39], and in Section 2.1 of [39], various relationships between DE and other primitives are discussed. Explicitly, they state that DE implies PoSW, and that DE with the *extraction soundness* property implies a VDF. Extraction soundness can be seen as a uniqueness property of the session key (used for decryption) of a DE scheme, which intuitively provides the uniqueness to build a VDF rather than a PoSW.

These relations are not formally proven, and, in particular, it is not shown how to build DE from specific cryptographic primitives. Furthermore, from a functionality standpoint, we conjecture that given an identity-based encryption scheme and a time-lock encryption scheme, it could be possible to construct a delay encryption scheme. These relations would have interesting implications, and necessitate further formal investigations.

We now turn our attention to the upper branch of the diagram. In what follows, we assume that there is a party who generates the puzzle, and another party who solves the puzzle. A time-lock encryption scheme requires that the generating party inputs a value s

whilst generating the puzzle, and that this value s is output upon solving the puzzle. As discussed in 2.2.1, in order to build a time-lock puzzle from a time-lock encryption scheme, there is an additional requirement that generating the puzzle is much faster than solving the puzzle.

We now reach another branching point in our diagram.

In the diagram we refer to the modern perspective of timed-release encryption (TRE), which incorporates public-key cryptography into time-lock puzzles without using a trusted agent [53, 111]. It is shown by Chvojka et al. [53] that if one has a CPA-secure public-key encryption scheme and a time-lock puzzle, one can build a TRE scheme in the following way. Recall that a public-key encryption scheme has a key generation algorithm which takes some randomness as input. Chvojka et al. build their black-box TRE scheme by using the input value s of the time-lock puzzle as the randomness of the key generation algorithm, and publishing the resulting public key. This allows all parties to encrypt to the public key, and upon solving the TLP, they can run key generation using the puzzle output s to obtain the secret key, and hence decrypt all the messages.

We now move to the purple box of the diagram, that represents our posited relationships between the outstanding primitives.

As we discussed in Section 2.2.5, many standard signature schemes [75, 76] have been adapted to construct timed-signatures. However, as far as the authors are aware, it is yet to be proven that a time-lock puzzle and a generic digital signature scheme can be combined to make a timed-signature scheme. Therefore we pose the following natural research question: Can one build a generic timed-signature scheme from a time-lock puzzle and generic signature scheme?

In [118], the authors show how to build homomorphic time-lock puzzles from cryptographic primitives including puncturable pseudorandom functions, trapdoor encryption, probabilistic obfuscation and indistinguishability obfuscation. These primitives are used

variously to build linearly homomorphic, multiplicatively homomorphic, and fully homomorphic time-lock puzzles. Therefore we believe that there exists a strong relation between HE and TLPs, and HTLPs. We pose an open question as to whether one can build a (linearly, multiplicatively or fully homomorphic) HTLP by composing in some way a TLP and an HE scheme.

To conclude, our analysis in this section shows how delay-based primitives fit together, identifying a hierarchy among them as well as theoretical gaps that would be interesting to address, given some of the implications they may have. We believe this is indeed a worthy pursuit, as we discuss further in Section 7.3.

7.3 Research Questions and Directions for Future Work

In the conclusions of Chapters 4, 5 and 6, we discuss potential future work related to the primitive and application of the respective section. In this section we supplement these with research questions related to the wider field of delay-based cryptography, and that hence have a broader scope.

Rigorous benchmarking for accurate parameterisation One thing that is lacking in the field of delay-based cryptography is a publicly available, rigorous benchmarking of the RSW time-lock assumption.

It should be mentioned that there is a limit to how useful such a procedure can be, due to the the vast range of devices, and the constant improvement in both consumer-grade and state-of-the-art computation power. However, the authors believe that without this, selecting parameters for adversarial advantage will be at best vague, and at worst insecure.

We suggest that an ideal benchmarking procedure would include a range of devices, including small IoT devices (e.g., Raspberry Pi), various consumer grade hardware, and

specialist hardware such as an ASIC. Research into specialist hardware was proposed by the VDF alliance, along with [7]. Additionally, an up-to-date online resource of the fastest algorithms and specialist hardware currently available for computing the iterated sequential functions would also be desirable. Early research into such hardware includes [123].

Alternative iterated sequential functions As we discussed in Section 2.1.2, if one wishes to compute a delay, the alternative methods to RSW are limited, and largely impractical. A relevant question therefore, is to ask (not for the first time) how one can use an alternative method to the RSW time-lock assumption, whilst still retaining a practically efficient construction.

Currently, the most promising alternative seems to be based on isogeny-based cryptography [39, 68]. A method for altering the underlying method of BLS signatures and isogeny walks in order to circumvent or mitigate the computationally expensive trusted setup and the high storage costs would be very beneficial for the area of delay-based cryptography. Another interesting line of research would be to see if there is some way of altering a known sequential computation to give the resulting output some mathematical properties which allow for faster inversion of the computation.

Concrete constructions of primitives Many cryptographic protocols aim to balance the conflicting goals of decentralisation and efficiency. A significant achievement would be to construct a protocol which shares the efficiency of a public-key TRE scheme such as [111], whilst maintaining the minimal trust assumptions of TLPs such as Astrolabous [9]. In particular, it would be desirable to have an untrusted, yet practically implementable method of decrypting n time-encrypted messages with significantly less than n sequential computations, say $\log(n)$ decryptions.

A related research question is to build schemes for the delay encryption and fully homomorphic TLP primitives, which can be implemented in an efficient manner.

Theoretical results To facilitate the design of optimally efficient constructions, it would be beneficial to derive theoretical bounds on efficiency under various assumptions. Therefore we believe a highly rewarding line of study would be into what the best efficiency one can hope to achieve in various settings, an example of which could be an impossibility result to demonstrate what the optimal decryption we can hope for in a setting without public-key infrastructure.

The research questions we pose in Section 7.2 illustrate some gaps in the theoretical knowledge of how the delay-based primitives fit together. Exploring whether or not the relationships we suggest hold, and providing formal security reductions would be of benefit to the community. Another avenue would be to further explore the various delay-related primitives in the UC framework - for example it would be interesting to see which relationships in Figure 7.1 can be translated into UC, and the effect that altering the underlying assumptions such as global synchronicity has upon such primitives.

References

[1] Letters to the editor. *Notices of the AMS*, 54(12):1454–1455, 2007.

[2] Edward Snowden's Motive Revealed: He Can 'Sleep at Night', https://www.nbcnews.com/feature/edward-snowden-interview/edward-snowdens-motive-revealed-he-can-sleep-night-n116851, 2014.

[3] SecureDrop Whistleblower Submission System, https://securedrop.org, 2021.

[4] Signal Messaging, https://signal.org/en, 2021.

[5] The Tor Project, https://www.torproject.org, 2021.

[6] A. Abadi, M. Ciampi, A. Kiayias, and V. Zikas. Timed signatures and zero-knowledge proofs—timestamping in the blockchain era. In *Applied Cryptography and Network Security: 18th International Conference, ACNS 2020*, pages 335–354. Springer, 2020.

[7] VDF Alliance. https://www.vdfalliance.org/news/open-vdf-asic-introduction, 2020.

[8] M. Arapinis, Á. Kocsis, N. Lamprou, L. Medley, and T. Zacharias. Universally composable simultaneous broadcast against a dishonest majority and applications. In *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing*, pages 200–210, 2023.

[9] M. Arapinis, N. Lamprou, and T. Zacharias. Astrolabous: A universally composable time-lock encryption scheme. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 398–426. Springer, 2021.

[10] V. Attias, L. Vignerii, and V. Dimitrov. Implementation study of two verifiable delay functions. In *2nd International Conference on Blockchain Economics, Security and Protocols*, 2021.

[11] L. Ausubel. A generalized vickrey auction. *Econo0 metrica*, 1999.

[12] L. Baird. The swirlds hashgraph consensus algorithm: Fair, fast, byzantine fault tolerance. *Swirlds Tech Reports SWIRLDS-TR-2016-01, Tech. Rep*, 34:9–11, 2016.

[13] S. Banescu, M. Ochoa, N. Kunze, and A. Pretschner. Idea: benchmarking indistinguishability obfuscation–a candidate implementation. In *Engineering Secure Software and Systems: 7th International Symposium*, pages 149–156. Springer, 2015.

[14] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang. On the (im) possibility of obfuscating programs. In *Annual international cryptology conference*, pages 1–18. Springer, 2001.

[15] E. Barker, L. Chen, A. Roginsky, A. Vassilev, R. Davis, and S. Simon. Sp 800-56b rev. 2, recommendation for pair-wise key-establishment using integer factorization cryptography. *ITL Computer Security Resource Center*, 2019.

[16] C. Baum, B. David, R. Dowsley, J. Nielsen, and S. Oechsner. Tardis: a foundation of time-lock puzzles in uc. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 429–459. Springer, 2021.

[17] C. Baum, B. David, R. Dowsley, J. Nielsen, and S. Oechsner. Craft: Composable randomness beacons and output-independent abort mpc from time. *IACR International Conference on Public-Key Cryptography*, pages 439–470, 2023.

[18] M. Bellare. Practice-oriented provable-security. In *International workshop on information security*, pages 221–231. Springer, 1997.

[19] M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In *Advances in Cryptology—ASIACRYPT 2000: 6th International Conference on the Theory and Application of Cryptology and Information Security Kyoto, Japan, December 3–7, 2000 Proceedings 6*, pages 531–545. Springer, 2000.

[20] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[21] M. Bellare and P. Rogaway. Optimal Asymmetric Encryption. In *Advances in Cryptology EUROCRYPT '94*, pages 92–111, 1994.

[22] P. Berglez and A. Gearing. The panama and paradise papers. the rise of a global fourth estate. *International Journal of Communication*, 12:20, 2018.

[23] F. Berti, F. Koeune, O. Pereira, T. Peters, and F. Standaert. Ciphertext integrity with misuse and leakage: definition and efficient constructions with symmetric primitives. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pages 37–50, 2018.

[24] N. Bitansky, S. Goldwasser, A. Jain, O. Paneth, V. Vaikuntanathan, and B. Waters. Time-lock puzzles from randomized encodings. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, pages 345–356, 2016.

[25] E. Blass and F. Kerschbaum. Borealis: Building block for sealed bid auctions on blockchains. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, pages 558–571, 2020.

[26] L. Blum, M. Blum, and M. Shub. A Simple Unpredictable Pseudo-Random Number Generator. In *Journal on Computing*, volume 15, page 364–383, 1986.

[27] M. Blum. How to exchange (secret) keys. In *CCM Transactions on Computer Systems, 1(2):175–193*, 1983.

[28] A. Blume and P. Heidhues. All equilibria of the vickrey auction. *Journal of economic Theory*, 114(1):170–177, 2004.

[29] C. Blundo, A. De Santis, and U. Vaccaro. Randomness in distribution protocols. *Information and Computation*, 131(2):111–139, 1996.

[30] P. Bogetoft, D. Christensen, I. Damgård, M. Geisler, T. Jakobsen, M. Krøigaard, J. Nielsen, J. Nielsen, K. Nielsen, J. Pagter, et al. Secure multiparty computation goes live. In *International Conference on Financial Cryptography and Data Security*, pages 325–343. Springer, 2009.

[31] D Boneh. Twenty years of attacks on the rsa cryptosystem. *Notices of the AMS*, 46(2):203–213, 1999.

[32] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch. Verifiable Delay Functions. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference*, pages 757–788, Santa Barbara, CA, USA, 2018.

[33] D. Boneh, B. Bünz, and B. Fisch. A survey of two verifiable delay functions. IACR Cryptology ePrint Archive, 2018.

[34] D. Boneh and M. Franklin. Identity-based encryption from the weil pairing. In *CRYPTO 2001: 21st Annual International Cryptology Conference*, pages 213–229. Springer, 2001.

[35] D. Boneh and M. Naor. Timed commitments. In *Annual international cryptology conference*, pages 236–254. Springer, 2000.

[36] Z. Brakerski, N. Döttling, S. Garg, and G. Malavolta. Leveraging linear decryption: Rate-1 fully-homomorphic encryption and time-lock puzzles. In *Theory of*

Cryptography: 17th International Conference, TCC 2019, pages 407–437. Springer, 2019.

[37] F. Brandt. Auctions. In *Handbook of Financial Cryptography and Security*, pages 75–84. Chapman and Hall/CRC, 2010.

[38] X. Bultel and P. Lafourcade. A posteriori openable public key encryption. In *ICT Systems Security and Privacy Protection*, pages 17–31, 2016.

[39] J. Burdges and J. DeFeo. Delay Encryption. In *40th Annual International Conference on the Theory and Applications of Cryptographic Techniques. EUROCRYPT 2021*, pages 302–326, 2021.

[40] C. Cachin, K. Kursawe, and V. Shoup. Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography. *Journal of Cryptology*, pages 219–246, 2005.

[41] C. Cachin, S. Micali, and M. Stadler. Computationally private information retrieval with polylogarithmic communication. In *Advances in Cryptology—EUROCRYPT'99: International Conference on the Theory and Application of Cryptographic Techniques*, pages 402–414. Springer, 1999.

[42] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145. IEEE, 2001.

[43] R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. *Journal of the ACM (JACM)*, 51(4):557–594, 2004.

[44] R. Carmichael. Note on a new number theory function. In *Bulletin of the American Mathematical Society*, pages 232–238, 1910.

[45] I. Cascudo and B. David. Scrape: Scalable randomness attested by public entities. In *International Conference on Applied Cryptography and Network Security*, pages 537–556. Springer, 2017.

[46] I. Cascudo and B. David. Albatross: publicly attestable batched randomness based on secret sharing. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 311–341. Springer, 2020.

[47] W. Castryck and T. Decru. An efficient key recovery attack on sidh (preliminary version). *Cryptology ePrint Archive*, 2022.

[48] J. Cathalo, B. Libert, and J. Quisquater. Efficient and non-interactive timed-release encryption. In *International Conference on Information and Communications Security*, pages 291–303. Springer, 2005.

[49] M. Chen, Jack Doerner, Y. Kondi, E.Lee, S. Rosefield, A. Shelat, and R. Cohen. Multiparty generation of an rsa modulus. *Journal of Cryptology*, 35(2):12, 2022.

[50] M. Chen, C. Hazay, Y. Ishai, Y. Kashnikov, D. Micciancio, T. Riviere, A. Shelat, M. Venkitasubramaniam, and R. Wang. Diogenes: lightweight scalable rsa modulus generation with a dishonest majority. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 590–607. IEEE, 2021.

[51] J. Cheon, N. Hopper, Y. Kim, and I. Osipkov. Provably secure timed-release public key encryption. *ACM Transactions on Information and System Security (TISSEC)*, 11(2):1–44, 2008.

[52] A. Cherniaeva, I. Shirobokov, and O. Shlomovits. Homomorphic encryption random beacon. *Cryptology ePrint Archive*, 2019.

[53] P. Chvojka, T. Jager, D. Slamanig, and C. Striecks. Versatile and sustainable timed-release encryption and sequential time-lock puzzles. In *European Symposium on Research in Computer Security*, pages 64–85. Springer, 2021.

[54] 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*, pages 364–369, 1986.

[55] B. Cohen and K. Pietrzak. Simple proofs of sequential work. In *Advances in Cryptology – EUROCRYPT 2018*, pages 451–467. Springer, 2018.

[56] B. Cohen and K. Pietrzak. The chia network blockchain, 2019.

[57] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, USA, 3 edition, 2009.

[58] H. Corrigan-Gibbs, W. Mu, D. Boneh, and B. Ford. Ensuring high-quality randomness in cryptographic key generation. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 685–696, 2013.

[59] R. Crandall and C. Pomerance. *Prime Numbers: A Computational Perspective*. Springer-Verlag, New York, NY, USA, 2 edition, 2005.

[60] I. Damgard. Practical and provably secure release of a secret and exchange of signatures. In *J. of Crypt., 8(4):201–222*, 1995.

[61] B. David, P. Gaži, A. Kiayias, and A. Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 66–98. Springer, 2018.

[62] J. Degabriele, K. Paterson, and G. Watson. Provable security in the real world. *IEEE Security & Privacy*, 9(3):33–41, 2010.

[63] D DiVincenzo. Quantum computation. *Science*, 270(5234):255–261, 1995.

[64] N. Döttling, S. Garg, G. Malavolta, and P. Vasudevan. Tight verifiable delay functions. In *International Conference on Security and Cryptography for Networks*, pages 65–84. Springer, 2020.

[65] Justin Drake. Minimal vdf randomness beacon. *Ethereum Research*, 2018.

[66] N. Ephraim, C. Freitag, I. Komardogski, and R. Pass. Continuous Verifiable Delay Functions. In *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, page 125–154, Zagreb, Croatia, 2020.

[67] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. In *Commun. ACM, 28(6):637–647*, 1985.

[68] L. De Feo, S. Masson, C. Petit, and A. Sanso. Verifiable Delay Functions from Supersingular Isogenies and Pairings. In *Advances in Cryptology – ASIACRYPT 2019 – 25th Annual Conference*, pages 248–277, Kobe, Japan, 2019.

[69] T. Frederiksen, Y. Lindell, V. Osheter, and B. Pinkas. Fast distributed rsa key generation for semi-honest and malicious adversaries. In *Annual International Cryptology Conference*, pages 331–361. Springer, 2018.

[70] C. Freitag, I. Komargodski, R. Pass, and N. Sirkin. Non-malleable time-lock puzzles and applications. In *Theory of Cryptography Conference*, pages 447–479. Springer, 2021.

[71] J. Friedlander, C. Pomerance, and I. Shparlinski. Period of the power generator and small values of Carmichael's function. In *American Mathematical Society*, pages 1591–1605. Mathematics of Computation 70 (2001), 2000.

[72] E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern. Rsa-oaep is secure under the rsa assumption. In *Annual International Cryptology Conference*, pages 260–274. Springer, 2001.

[73] H. Galal and A. Youssef. Verifiable sealed-bid auction on the ethereum blockchain. In *International Conference on Financial Cryptography and Data Security*, pages 265–278. Springer, 2018.

[74] U. Gallersdörfer, L. Klaaßen, and C. Stoll. Energy consumption of cryptocurrencies beyond bitcoin. *Joule*, pages 1843–1846, 2020.

[75] J.A. Garay and M. Jakobson. Timed release of standard digital signatures. In *International Conference on Financial Cryptography*, pages 168–182. Springer, 2002.

[76] J.A. Garay and C. Pomerance. Timed fair exchange of standard signatures. In *Financial Crypto 2003*, pages 190–207. Springer, 2003.

[77] J. Garside. Panama Papers: inside the Guardian's investigation into offshore secrets, https://www.theguardian.com/news/2016/apr/16/panama-papers-inside-the-guardians-investigation-into-offshore-secrets, 2016.

[78] C. Gauss. *Disquisitiones Arithmeticae*. Yale University Press, New Haven, CT, USA, 1 edition, 2009.

[79] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th symposium on operating systems principles*, pages 51–68, 2017.

[80] R.L. Goodstein. *Boolean Algebra*. Dover Publications, Mineola, NY, USA, 1 edition, 2007.

[81] F. Griffin and I. Shparlinski. On the linear complexity profile of the power generator. In *IEEE Transactions on Information Theory (Volume: 46, Issue: 6, Sep 2000)*, pages 2159 – 2162, 2000.

[82] R. Han, J. Yu, and H. Lin. Randchain: Decentralised randomness beacon from sequential proof-of-work. *IACR Cryptol. ePrint Arch.*, 2020.

[83] T. Hanke, M. Movahedi, and D. Williams. Dfinity technology overview series, consensus system. *arXiv preprint arXiv:1805.04548*, 2018.

[84] G. Hardy and E. Wright. *An introduction to the theory of numbers*. Oxford university press, 1979.

[85] C. Hazay, G. Mikkelsen, R. Rabin, T. Toft, and A. Nicolosi. Efficient rsa key generation and threshold paillier in the two-party setting. *Journal of Cryptology*, 32:265–323, 2019.

[86] N. Heninger and H. Shacham. Reconstructing rsa private keys from random key bits. In *Annual International Cryptology Conference*, pages 1–17. Springer, 2009.

[87] D. Hofheinz and V. Shoup. Gnuc: A new universal composability framework. *Journal of Cryptology*, 28(3):423–508, 2015.

[88] N. Huber, R. Küsters, T. Krips, J. Liedtke, J. Müller, D. Rausch, P. Reisert, and A. Vogt. Kryvos: Publicly tally-hiding verifiable e-voting. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 1443–1457, 2022.

[89] A. Jain, H. Lin, and A. Sahai. Indistinguishability obfuscation from well-founded assumptions. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 60–73, 2021.

[90] A. Juels and M. Szydlo. A two-server, sealed-bid auction protocol. In *International conference on financial cryptography*, pages 72–86. Springer, 2002.

[91] S. Kakvi and E. Kiltz. Optimal security proofs for full domain hash, revisited. In *Advances in Cryptology–EUROCRYPT 2012*, pages 537–553. Springer, 2012.

[92] J. Katz. Digital signatures. 2010.

[93] J. Katz and Y. Lindell. *Introduction to Modern Cryptography, Second Edition*. CRC Press, Boca Raton, FL, USA, 2 edition, 2014.

[94] J. Katz, J. Loss, and J. Xu. On the security of time-lock puzzles and timed commitments. In *Theory of Cryptography: 18th International Conference*, pages 390–413. Springer, 2020.

[95] A. Kiayias, A. Russell, B. David, and R. Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Advances in Cryptology – CRYPTO 2017*, pages 357–388, 2017.

[96] N. Koblitz. The uneasy relationship between mathematics and cryptography. *Notices of the AMS*, 54(8):972–979, 2007.

[97] N. Koblitz and A.Menezes. Another look at" provable security". *Journal of Cryptology*, 20:3–37, 2007.

[98] N. Koblitz and A. Menezes. The random oracle model: a twenty-year retrospective. *Designs, Codes and Cryptography*, 77:587–610, 2015.

[99] R. Kumaresan and I. Bentov. How to use bitcoin to incentivize correct computations. In *ACM CCS 2015*, pages 30–41. ACM Press, 2015.

[100] R. Küsters, M. Tuengerthal, and D. Rausch. The iitm model: a simple and expressive model for universal composability. *Journal of Cryptology*, 33:1461–1584, 2020.

[101] E. Landerreche, M. Stevens, and C. Schaffner. Non-interactive cryptographic timestamping based on verifiable delay functions. In *International Conference on Financial Cryptography and Data Security*, pages 541–558. Springer, 2020.

[102] H. Lee, Y. Hsu, J. Wang, H. Yang, Y. Chen, Y. Hu, and H. Hsiao. Headstart: Efficiently verifiable and low-latency participatory randomness generation at scale. In *Network and Distributed System Security (NDSS) Symposium 2022*, 2022.

[103] A. Lenstra and I. Shparlinski. Selective forgery of rsa signatures with fixed-pattern padding. In *Public Key Cryptography: 5th International Workshop on Practice and Theory in Public Key Cryptosystems*, pages 228–236. Springer, 2002.

[104] Arjen K Lenstra and Benjamin Wesolowski. Trustworthy public randomness with sloth, unicorn, and trx. *International Journal of Applied Cryptography*, 3(4):330–343, 2017.

[105] M. Liedtke and J. Mattise. Leaked" pandora papers" expose how billionaires and corrupt leaders hide wealth. *Guardian (Sydney)*, 2021.

[106] H. Lin, R. Pass, and P. Soni. Two-round and non-interactive concurrent non-malleable commitments from time-lock puzzles. volume 49, pages 17–196. SIAM, 2020.

[107] J. Liu, F. Garcia, and M. Ryan. Time-release protocol from bitcoin and witness encryption for sat. *Korean Circulation Journal*, 40(10):530–535, 2015.

[108] J. Liu, T. Jager, S. Kakvi, and B. Warinschi. How to build time-lock encryption. *Designs, Codes and Cryptography*, 86:2549–2586, 2018.

[109] Y. Liu, Q. Wang, and S. Yiu. Towards practical homomorphic time-lock puzzles: Applicability and verifiability. In *ESORICS 2022: 27th European Symposium on Research in Computer Security*, pages 424–443. Springer, 2022.

[110] A. Loe, L. Medley, C. O'Connell, and E. Quaglia. A practical verifiable delay function and delay encryption scheme. *IACR Cryptol. ePrint Arch.*, 2021.

[111] A. Loe, L. Medley, C. O'Connell, and E. Quaglia. Tide: a novel approach to constructing timed-release encryption. In *Information Security and Privacy: 27th Australasian Conference, ACISP 2022*, pages 244–264. Springer, 2022.

[112] A. Loe, L. Medley, C. O'Connell, and E. Quaglia. Applications of timed-release encryption with implicit authentication. In *International Conference on Cryptology in Africa*, pages 490–515. Springer, 2023.

[113] I Maffei and AW Roscoe. Delay encryption by cubing. *arXiv preprint arXiv:2205.05594*, 2022.

[114] M. Mahmoody, T. Moran, and S. Vadhan. Time-lock puzzles in the random oracle model. In *Advances in Cryptology – CRYPTO 2011*, pages 39–50. Springer, 2011.

[115] M. Mahmoody, T. Moran, and S. Vadhan. Publicly verifiable proofs of sequential work. In *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science*, pages 373–388, 2013.

[116] M. Mahmoody, C. Smith, and D. Wu. Can verifiable delay functions be based on random oracles? In *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, 2020.

[117] G. Malavolta, P. Moreno-Sanchez, A. Kate, M. Maffei, and S. Rav. Concurrency and privacy with payment-channel networks. In *ACM CCS 2017*, pages 455–471. ACM Press, 2017.

[118] G. Malavolta and S. Thyagarajan. Homomorphic time-lock puzzles and applications. In *Annual International Cryptology Conference*, pages 620–649. Springer, 2019.

[119] W. Mao. Timed-release cryptography. In *International Workshop on Selected Areas in Cryptography*, pages 342–357. Springer, 2001.

[120] T. May. Timed-release crypto. *http://www. hks. net. cpunks/cpunks-0/1560. html*, 1993.

[121] L. Medley, A. Loe, and E. Quaglia. Sok: Delay-based cryptography. In *2023 IEEE Computer Security Foundations Symposium*. IEEE, 2023.

[122] L. Medley and E. Quaglia. Collaborative verifiable delay functions. In *International Conference on Information Security and Cryptology (INSCRYPT 2021)*, pages 507–530. Springer, 2021.

[123] A. Mert, E. Öztürk, and E. Savaş. Low-latency asic algorithms of modular squaring of large integers for vdf evaluation. *IEEE Transactions on Computers*, 71(1):107–120, 2020.

[124] M.Girault and J. Misarsky. Selective forgery of rsa signatures using redundancy. In *Advances in Cryptology—EUROCRYPT'97: International Conference on the Theory and Application of Cryptographic Techniques*, pages 495–507. Springer, 1997.

[125] G. Miller. Riemann's Hypothesis and Tests for Primality. In *Journal of Computer and System Sciences, 13(3)*, pages 300–317, 1976.

[126] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[127] Shyam Narayanan. Improving the speed and accuracy of the miller-rabin primality test, 2014.

[128] J. O'Donovan, H. Wagner, and S. Zeume. The value of offshore secrets: Evidence from the panama papers. *The Review of Financial Studies*, 32(11):4117–4155, 2019.

[129] K. Paterson and E. Quaglia. Time-specific encryption. In *Security and Cryptography for Networks: 7th International Conference, SCN 2010*, pages 1–16. Springer, 2010.

[130] K. Pietrzak. Simple verifiable delay functions. In *10th Innovations in Theoretical Computer Science Conference, ITCS 201*, pages 601–615, San Diego, California, 2019.

[131] M. Rabin. Digitalized signatures and public-key functions as intractable as factorization. In *MIT/LCS/TR-212, MIT Laboratory for Computer Science*, 1979.

[132] M. Rabin. Transaction protection by beacons. *Journal of Computer and System Sciences*, 27(2):256–267, 1983.

[133] M. Raikwar. Competitive decentralized randomness beacon protocols. In *Proceedings of the Fourth ACM International Symposium on Blockchain and Secure Critical Infrastructure*, pages 83–94, 2022.

[134] M. Raikwar and D. Gligoroski. Sok: Decentralized randomness beacon protocols. *Information Security and Privacy: 27th Australasian Conference, ACISP 2022*, pages 420–446, 2022.

[135] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 26(1):96–99, 1983.

[136] R. Rivest, A. Shamir, and D. Wagner. Time-lock puzzles and timed-release crypto. In *MIT/LCS/TR-684, MIT Laboratory for Computer Science*, 1996.

[137] R.Peralta, H.Booth, L.Brandão, J Kelsey, and C. Miller. Interoperable Randomness Beacons. *NIST*, 2019.

[138] A. De Santis, S. Micali, and G. Persiano. Non-interactive zero-knowledge proof systems. In *Advances in Cryptology—CRYPTO'87*, pages 52–72. Springer, 1988.

[139] A. Scafuro. Break-glass encryption. In *PKC '19: 22nd IACR International Conference on Practice and Theory of Public-Key Cryptography*, pages 34–62. Springer, 2019.

[140] W. Scheuerman. Whistleblowing as civil disobedience: The case of edward snowden. *Philosophy & Social Criticism*, 40(7):609–628, 2014.

[141] P. Schindler, A. Judmayer, M. Hittmeir, N. Stifter, and E. Weippl. Randrunner: Distributed randomness from trapdoor vdfs with strong uniqueness. Cryptology ePrint Archive, Paper 2020/942, 2020.

[142] P. Schindler, A. Judmayer, N. Stifter, and E. Weippl. Hydrand: Efficient continuous distributed randomness. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 73–89. IEEE, 2020.

[143] Y. Seurin. On the lossiness of the rabin trapdoor function. In *International Workshop on Public Key Cryptography*, pages 380–398. Springer, 2014.

[144] A. Shamir. How to share a secret. 2010.

[145] W. Silvano and R. Marcelino. Iota tangle: A cryptocurrency to communicate internet-of-things data. *Future generation computer systems*, 112:307–319, 2020.

[146] N. Smart. *Cryptography: an introduction*. 2003.

[147] Adam Smith and Ye Zhang. On the regularity of lossy rsa: Improved bounds and applications to padding-based encryption. In *Theory of Cryptography Conference*, pages 609–628. Springer, 2015.

[148] J. Stern. Why provable security matters? In *Advances in Cryptology—EUROCRYPT 2003: International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4–8, 2003 Proceedings 22*, pages 449–461. Springer, 2003.

[149] E. Syta, P.Jovanovic, E.Kogias, N. Gailly, L.Gasser, I. Khoffi, M. Fischer, and B. Ford. Scalable bias-resistant distributed randomness. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 444–460. IEEE, 2017.

[150] S. Thyagarajan, A. Bhat, G. Malavolta, N. Döttling, A. Kate, and D. Schröder. Verifiable timed signatures made practical. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1733–1750, 2020.

[151] J. Truby. Decarbonizing Bitcoin: Law and policy choices for reducing the energy consumption of Blockchain technologies and digital currencies. *Energy research & social science*, 44:399–410, 2018.

[152] J. Verble. The nsa and edward snowden: surveillance in the 21st century. *Acm Sigcas Computers and Society*, 44(3):14–20, 2014.

[153] W. Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *The Journal of finance*, 16(1):8–37, 1961.

[154] S. Viswanathan, RS. Bhuvaneswaran, S. Ganapathyi, and A. Kannan. Euler phi function and gamma function based elliptic curve encryption for secured group communication. *Wireless Personal Communications*, 125(1):421–451, 2022.

[155] B. Wesolowski. Efficient Verifiable Delay Functions. In *Advances in Cryptology – EUROCRYPT 2019*, page 379–407, Darmstadt, Germany, 2019.

[156] Z. Yang, M. Zolanvari, and R. Jain. A survey of important issues in quantum computing and communications. *IEEE Communications Surveys & Tutorials*, 2023.

[157] Y. Zheng. Digital signcryption or how to achieve cost(signature and encryption) cost(signature) + cost(encryption). In *International Conference of Theory Applied Cryptography Technology (CRYPTO) 97*, pages 165–179. Springer, 1997.

[158] Y. Zheng. A new efficient signcryption scheme in the standard model. In *Security and Communication Networks vol. 8, no 5*, pages 703 – 878, 2015.

[159] Y. Zheng and H. Imai. How to construct efficient signcryption schemes on elliptic curves. In *Proc. of IFIP SEC98 vol. 68, no. 5*, pages 227 – 233, 1998.

[160] P. Zimmerman. Why I Wrote PGP, Essays on PGP. Phil Zimmermann and Associates LLC, 2013.