SoK: Delay-based Cryptography

Liam Medley
Information Security Group
Royal Holloway, University of London
United Kingdom
liam.medley.2018@rhul.ac.uk

Angelique Faye Loe
Information Security Group
Royal Holloway, University of London
United Kingdom
angelique.loe.2016@rhul.ac.uk

Elizabeth A. Quaglia
Information Security Group
Royal Holloway, University of London
United Kingdom
elizabeth.quaglia@rhul.ac.uk

Abstract—**In this work, we provide a systematisation of knowledge of delay-based cryptography, in which we discuss and compare the existing primitives within cryptography that utilise a time-delay. 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, and analyse topics including trust, decentralisation and concrete methods to implement a delay. We then survey the existing primitives, discussing their security properties, instantiations and applications. We make explicit the relationships between these primitives, identifying a hierarchy and the theoretical gaps that exist. We end this systematisation of knowledge by highlighting relevant future research directions within the field of delay-based cryptography, from which this area would greatly benefit.**

Index Terms—**Time, Time-lock Puzzle, Delay Function**

I. Introduction

Timed-release cryptography is the notion of encrypting a message "to the future", as first proposed by May in 1993, in a post on the Cypherpunks mailing list [70]. In this post, May discusses applications including sending money into the future to avoid taxation, or sending a message 'in the event of death'. Since the Cypherpunks mailing list was released the cited applications and the methods suggested to achieve this have been extensively augmented and modernised. However, 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 was provided in 1996 by Rivest, Shamir and Wagner in their seminal work "Time lock puzzles and timed-release crypto" [78]. 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.

Paper overview

In the remainder of Section I, we will discuss how a delay is used in cryptography, the techniques used to implement a delay, and the obstacles that exist when using such techniques in practice. In Section II, we discuss the major delay-based primitives, and important constructions thereof. In Section III, we provide a discussion of how such delay-based primitives fit into the universal composability framework of Canetti [23]. In Section IV, we analyse the relations between these primitives, providing a hierarchical structure to them, illustrating the branching points between early primitives, and showing which additional cryptographic tools are used to build the more complex delay-based primitives. Furthermore, we identify gaps in the theoretical understanding of how some of these primitives are related. In Section V, we highlight the two main applications of the primitives we discussed, illustrating how the delay improves upon the state of the art. In Section VI, we pose relevant research questions and areas of study that we believe will improve the field of delay-based cryptography. We draw our conclusions in Section VII.

A. *Why a delay is useful*

Fairness and the reduction of 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 [47]. 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. 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, this is a very strong assumption, which leads to the natural question of who should be trusted to perform this role.

By using a delay, it becomes possible to reduce the role of the trusted party (typically limiting the trusted party to only generating 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.

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 [30] 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 [18] showed that this impossibility result could be circumvented using timed commitments. By using a timed-commitment if a party chose to unfairly abort then the remaining party could compute a sequential computation in order to obtain the output.

- *A modern approach to building randomness beacons.* A randomness beacon was first proposed by Rabin in 1983 [76] to remove the need for trusted intermediary parties in protocols such as contract signing. Interest in randomness beacons has recently seen a sharp rise, 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. We refer the reader to the recent SoK paper [77] for a discussion on these, and will focus on the approach pertinent to this work: randomness beacons based upon verifiable delay functions (VDF). 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. Additionally, a VDF-based randomness beacon ensures that each pulse of randomness will come at a regular interval, which is listed by NIST in their project on interoperable randomness beacons [79] as a requirement for a randomness beacon. Both of these properties are desired in an ideal randomness beacon, but not achieved by the majority of current techniques [77]. We will discuss randomness beacons built from VDFs further in Section V.

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 [43], [88], and is viewed as unsustainable. By using techniques from delay-based cryptography, a proof of *sequential* work is possible as an alternative to 'classical' proof of work in the style of bitcoin [73]. When using a classical proof of work, the more computational power 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 [1], [54], [65].

B. *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 for such a definition should capture that no party can compute the function in less than t '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[1]. 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 [23], this concept is difficult to capture. As such, notions of time are instead given 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 III, and for now restrict our attention to the standard model, which we shall define 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.

[1]For the purpose of this work, we do not need to choose a model of computation to follow, so that our discussion can be kept generic.

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: $y = H(H(H(\cdots H(x))))$. By 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 work 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 [78]. This requires a trusted party generating an RSA modulus $N = pq$, where p and q are primes, and sampling an input $x \leftarrow_R \mathbb{Z}_N^*$. The delay is achieved by 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 $e = 2^t \pmod{\phi(N)}$, and then compute $x^e \pmod{N}$. This is significantly faster than simply squaring and reducing modulo N [78].

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 I-C.

As an alternative approach to solving this problem, it was proposed in [90] 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.

- *Isogeny-based techniques* One of the most modern alternative methods of computing a delay is using random isogeny walks over elliptic curves, together with BLS signatures [22], [39]. This can be viewed on a high level as plugging a slow function into a fast function - the former acts as the delay, and the latter the verification. Currently there are practical implementation challenges, as we shall discuss in Sections II-D and II-G, but as the field of isogeny-based cryptography is rapidly growing, this is a technique that may improve as the theory advances.

- *Hash-based techniques* An interesting alternative approach to computing a delay was proposed by Mahmoody et al. in [64], 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 [4].

Another interesting hash-based technique for computing a delay is using a *directed acyclic graph*, a technique proposed by Mahmoody et al. in [65]. 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.

- *Randomised encodings* In [10], 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.

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, as we discuss in Section V-B. 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 in Section VI, we call for rigorous benchmarking to provide the community with 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 [78]. 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 [51]. Alternative methods such as replacing the RSA group with a class group lacks the same level of community trust, and requires significantly more research until it can be considered trusted.

Hash functions have been widely and consistently used, leading to the assumption that any delay-based protocol which inherently relies on their security [4], [64], [65] would acquire community trust. However, it is not as evident with use of hash function as it is in the case of repeated squaring how much one can expedite each step of a hash function through parallelisation. The efficiency of such an approach would rely on the hash function in use and the available computing power. As a result, we believe that this method holds the second-highest level of community trust following the repeated squaring technique.

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. [24]. 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.

C. 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 [29], [62], [78].

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. [40] 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. [46] 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. [26] 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. [27] 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 [30], 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.

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

II. 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. The basis of our argument for splitting these primitives up is that in [64], Mahmoody et al.

showed a black-box separation between proofs of sequential work and time-lock puzzles. We shall expand upon this in Section IV, where we show the various relationships between each class of primitive in Fig. 1.

A. 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 [78], 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 $Z = s + r^{2^t} \pmod N$.

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 to 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. [10], as discussed in Section I-B, and the UC-based construction of Baum et al. [9], as we shall discuss in Section III. 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 [41], 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. We will discuss this in the context of auctions in Section V.

As discussed in Section I-B, a recent work by Katz et al. [51] 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 [29]. 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 [64]. 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 I-B. This is useful in proving theoretical results, especially in the decentralised setting [4].

B. 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 [68]. 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.

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 [19], representing significant progress. In 2022, Liu et al. built a scheme which significantly improved the practicality of linearly and multiplicatively homomorphic TLPs [60]. 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.

C. *Timed-release encryption*

Timed-release encryption was first mentioned by May in 1993 [70], 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 [25], [69] using such a trusted agent, and in 2008, Cheon et al. [28] proved that the security of this concept is equivalent to that of identity-based encryption. In [74], 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 [29], [62], [63]. 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 [73] with witness encryption, where one must show the solution to a hard problem in order to decrypt a message [58], [59]. 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.

In 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 [29]. 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 [62], Loe et al. provide a concrete instantiation of this primitive.

D. *Delay Encryption*

Delay Encryption, introduced by Burdges et al. in 2021 [22], 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 [17]. 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 [22], 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 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.

E. *Timed commitments and timed signatures*

In 2000, Boneh and Naor introduced the notion of a *timed commitment* [18], 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 [12], [34], [38]. Furthermore, early delay-based signature literature ensured that standard digital signature schemes could be leveraged into the constructions [44] [45]. 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* [45], [85]. 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. [85], as a practical improvement upon a timed signature. VTS schemes utilise homomorphic time-lock puzzles [68] in order to achieve a delay. They also rely on digital signature schemes [50], non-interactive zero-knowledge proofs [80], and threshold secret sharing [83]. The

[2]There is also an unpublished construction by Loe et al. [61] which is shown to run efficiently, but relies upon a trusted setup.

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 [50] and non-malleable commitments which can mitigate concurrent person-in-the-middle attacks [57]. Furthermore, verifiable timed signatures also have specific applications in payment channel networks used in cryptocurrencies [67], [87], multi-signature transactions which are used so that multiple signatures are required to authenticate transactions [15], and also in fair multi-party computation so that fairness in blockchains can be achieved by financially penalising parties which abort protocol execution [53].

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

F. Proofs of sequential work

In 2013, Mahmoody et al. introduced 'publicly verifiable proofs of sequential work' [65]. 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 [6], [71], [84]. 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 drawbacks 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 [31], 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 IV are themselves proofs of sequential work.

In 2017, Lenstra and Wesolowski introduced a slow-timed hash function, which they named Sloth [56]. 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 $p \equiv 3 \mod 4$. The evaluator is challenged to compute $\sqrt{x} \equiv x^{\frac{p+1}{4}}$.

The 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.

G. 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 [15], 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 V-A.

In [15], 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 [90] and Pietrzak [75] each proposed an RSW-based VDF, and De Feo et al. [39] 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 I-C). 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 [16] 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

TABLE I

Primitive	Functionality	Distinctive Feature
TLE	Encryption	Minimal assumptions
TLP	Encryption	Fast puzzle generation
HTLP	Encryption	Evaluation over multiple puzzles
TRE	Encryption	Delayed public-key encryption
DE	Encryption	Delayed key encapsulation mechanism
TS	Signature	Delayed Authentication
PoSW	Proof of Delay	Public verification
VDF	Proof of Delay	Uniqueness

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 [39].

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 [35], [37], [71], and VDF-based applications [54], [55], [82].

In 2019, Ephraim et al. introduced the notion of a *continuous* VDF (cVDF) [37]. 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 V.

We end this section by mentioning an interesting impossibility result by Mahmoody et al. [66], 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.

H. 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* [81] 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) [21], 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.

I. Conclusion

We conclude this section by presenting Table I, which provides an overview of the key properties of each primitive. In particular, we highlight the key functionality of each primitive, and what distinguishes them from the others.

III. Universal Composability

In [23], 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 I 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 [4], [8], [9]. 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 [9].

Baum et al. introduce TLPs via a ticker functionality in TARDIS [9], which works by splitting time into units known as *clock ticks*. The global clock advances a tick only when all 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 [4], Arapinis et al. introduce a UC TLE scheme named Astrolabous, which relies on a global clock [52] 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 [8], [9] time is not synchronised but progresses identically for all entities. In Astrolabous [4], 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 [1], [3]. This provides stronger security guarantees in complex protocols, and it is our hope that more time-based primitives are constructed in the UC framework.

IV. Relating Delay-Based Primitives

One of the goals of this work is to provide an overview and understanding of how delay-based primitives are related.

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 I-B.

An iteratively sequential function is defined as a function which maps from a given group into itself. Additionally, it

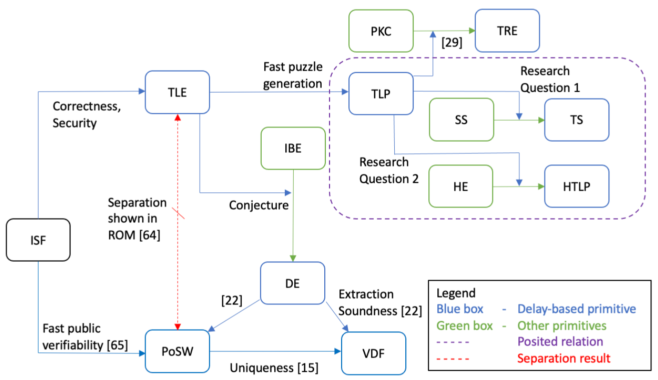

Fig. 1. Identified and conjectured relations between delay-based primitives.

must satisfy *sequentiality*: For a time hardness 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 II. We provide a mapping of the relation between these primitives in Fig. 1. Recall that the separation between the two classes comes from a result by Mahmoody et al. in [64], 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 [22], 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 [65]. Boneh et al. [15] 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.

Delay encryption is described as the identity-based analogue to time-lock encryption [22], and in Section 2.1 of [22], 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 II-A, 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 [29], [62]. It is shown by Chvojka et al. [29] 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 II-E, many standard signature schemes [44], [45] 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 [68], 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 VI.

V. Applications

Often there are multiple primitives and constructions which share the same motivating application. In this section we look in-depth at two of the most prominent applications of delay-based primitives, clarifying in which scenarios certain primitives are better suited.

We start by outlining the main themes of these applications. The two main aims of modern cryptographic research are i) to reduce trust and ii) to improve efficiency, and commonly a trade-off arises between the two.

Trust One theme we see repeatedly in this section is that of trust. The main distinguishing factor between many of the modern delay-based primitives and the older primitives is that the modern primitives aim to reduce trust as much as possible. Conversely, the majority of the older primitives and constructions assume a trusted setup, or indeed a trusted agent such as a time server.

For use in the decentralised space there is extensive research into building cryptography with the minimum trust assumptions possible. Delay-based cryptography is no exception to this and indeed there exists various lines of research which use delay-based cryptography to avoid the use of a trusted party [15], [18], [55], [68], [78]. One highly active research area is using MPC to generate an RSA modulus without a trusted third party, as we discussed in I-C, another is randomness beacons, which we shall discuss in Section V-A.

Efficiency One of the key efficiency goals for many delay-based primitives is to ensure that only one delay needs to be computed, rather than many. An example of this can be found in the applications of voting and auctions, where if one is to use standard TLPs, each bid or vote must be encoded as a separate puzzle. As such there have been various works on alternative methods which provide the desired ‘solve one get many for free’ property of only one delay computation [22], [29], [62], [68]. Also of importance for the majority of cases to be practical is for inversion to be fast: one should be able to verify that the computation is correct without having to spend a significant amount of time on recomputation.

We next explore in greater detail two of the most prominent applications of delay-based primitives, highlighting the tensions between the need to reduce trust, whilst ensuring the constructions remain practical.

A. *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 [14], [33]. Since 2011 NIST have been running a competition to build a trusted randomness beacon (RB), 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 II-G. 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 [32], [54].

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 [15] 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. [36] 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 [82], 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 [55], 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 II-G.

Ephraim et al. [37] build a randomness beacon from a continuous VDF by extracting randomness at regular intervals. Recall from Section II-G that a cVDF consists of multiple states where verification can occur.

In the construction of [37], 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 [37], 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.

B. 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 [11], [20], [22], [42], [49]. 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 [78].

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 [42]. 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 [5], [13], [89]. 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 puzzle. There exist multiple various constructions that fit this approach to auctions, and we view the non-malleable TLP construction of Freitag et al. [41] 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 II have been applied to solve this problem more efficiently. We will now outline these approaches.

In 2019 [68], Malavolta et al. used homomorphic time-lock puzzles (see Section II-B) 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 fully homomorphic TLPs are based on indistinguishability obfuscation (IO) [22], [68]. IO aims to obfuscate programs to make them unintelligible whilst retaining their original functionality [7]. However, IO is known to be impractical with no construction efficiently implementable at the time of writing [48].

In 2021, Burdges et al. described auctions as a key motivating example for their Delay Encryption primitive [22] (see Section II-D).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 II-D, 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 [68], and the delay computed in [22]) 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 II-C) 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. This is illustrated by Loe et al. in [62], where the authors construct and implement a practical 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.

VI. Research questions and directions for future work

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 is worth noticing 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 the 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 [2].

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. [86] introduces a smart contract for clients to outsource a sequential computation to powerful servers, which may use specialised hardware. Early research into such hardware includes [72].

Alternative iterated sequential functions

As we discussed in Section I-B, 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 [22], [39]. 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

A significant achievement would be to construct a protocol which shares the efficiency of a public-key TRE scheme such as [62], whilst maintaining the minimal trust assumptions of TLPs such as Astrolabous [4]. 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 IV 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 1 can be translated into UC, and the effect that altering the underlying assumptions such as global synchronicity has upon such primitives.

VII. Conclusion

In recent years, there has been a dramatic surge in delay-based cryptographic research. We present the first systematisation of knowledge (SoK) for the existing efforts on delay-based primitives and constructions. This SoK provides a comprehensive review of the trust assumptions and nuances that separate the primitives, and the impact of such nuances on standard delay-based applications. We review the underlying techniques that exist in the literature to construct a delay, and discuss their practicality and ways they can be improved. We discuss the existing primitives along with the state-of the art constructions. In Section IV, we analyse how these primitives are linked to each other, identifying relations and highlighting gaps in the theoretical knowledge we have so far. Finally, 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.

References

[1] 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*. Springer, 2020.

[2] VDF Alliance. https://www.vdfalliance.org/news/open-vdf-asic-introduction, 2020.

[3] M. Arapinis, Á. Kocsis, N. Lamprou, L. Medley, and T. Zacharias. Universally composable simultaneous broadcast against a dishonest majority. In *ACM Symposium on Principles of Distributed Computing*, 2023.

[4] 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*, 2021.

[5] L. Ausubel. A generalized vickrey auction. *Econo0 metrica*, 1999.

[6] L. Baird. The swirlds hashgraph consensus algorithm: Fair, fast, byzantine fault tolerance. 2016.

[7] 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.

[8] C. Baum, B. David, R. Dowsley, J. Nielsen, and S. Oechsner. Craft: Composable randomness beacons and output-independent abort mpc from time. *Cryptology ePrint Archive*, 2020.

[9] 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*, 2021.

[10] 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.

[11] 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.

[12] M. Blum. How to exchange (secret) keys. In *CCM Transactions on Computer Systems, 1(2):175–193*, 1983.

[13] A. Blume and P. Heidhues. All equilibria of the vickrey auction. *Journal of economic Theory*, 114(1):170–177, 2004.

[14] C. Blundo, A. De Santis, and U. Vaccaro. Randomness in distribution protocols. *Information and Computation*, 1996.

[15] 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.

[16] D. Boneh, B. Bünz, and B. Fisch. A survey of two verifiable delay functions. IACR Cryptology ePrint Archive, 2018.

[17] D. Boneh and M. Franklin. Identity-based encryption from the weil pairing. In *CRYPTO 2001: 21st Annual International Cryptology Conference*. Springer, 2001.

[18] D. Boneh and M. Naor. Timed commitments. In *Annual international cryptology conference*. Springer, 2000.

[19] 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*. Springer, 2019.

[20] F. Brandt. Auctions. In *Handbook of Financial Cryptography and Security*, pages 75–84. Chapman and Hall/CRC, 2010.

[21] X. Bultel and P. Lafourcade. A posteriori openable public key encryption. In *ICT Systems Security and Privacy Protection*, 2016.

[22] J. Burdges and L. De Feo. Delay Encryption. In *40th Annual International Conference on the Theory and Applications of Cryptographic Techniques. EUROCRYPT 2021*, pages 302–326, 2021.

[23] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, 2001.

[24] W. Castryck and T. Decru. An efficient key recovery attack on sidh (preliminary version). *Cryptology ePrint Archive*, 2022.

[25] 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.

[26] M. Chen, Jack Doerner, Y. Kondi, E.Lee, S. Rosefield, A. Shelat, and R. Cohen. Multiparty generation of an rsa modulus. *Journal of Cryptology*, 2022.

[27] 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)*, 2021.

[28] J. Cheon, N. Hopper, Y. Kim, and I. Osipkov. Provably secure timed-release public key encryption. *ACM Transactions on Information and System Security (TISSEC)*, 2008.

[29] 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.

[30] 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*, 1986.

[31] B. Cohen and K. Pietrzak. Simple proofs of sequential work. In *Advances in Cryptology – EUROCRYPT 2018*, 2018.

[32] B. Cohen and K. Pietrzak. The chia network blockchain, 2019.

[33] 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*, 2013.

[34] I.B. Damgard. Practical and provably secure release of a secret and exchange of signatures. In *J. of Crypt., 8(4):201–222*, 1995.

[35] N. Döttling, S. Garg, G. Malavolta, and P. Vasudevan. Tight verifiable delay functions. In *International Conference on Security and Cryptography for Networks*, 2020.

[36] Justin Drake. Minimal vdf randomness beacon. *Ethereum Research*, 2018.

[37] 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.

[38] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. In *Commun. ACM, 28(6):637–647*, 1985.

[39] 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.

[40] 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*. Springer, 2018.

[41] 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.

[42] 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.

[43] U. Gallersdörfer, L. Klaaßen, and C. Stoll. Energy consumption of cryptocurrencies beyond bitcoin. *Joule*, pages 1843–1846, 2020.

[44] J.A. Garay and M. Jakobson. Timed release of standard digital signatures. In *Financial Crypto 2002*. LNCS, 2002.

[45] J.A. Garay and C. Pomerance. Timed fair exchange of standard signatures. In *Financial Crypto 2003*. LNCS, 2003.

[46] 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*, 2019.

[47] 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*, 2022.

[48] 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.

[49] A. Juels and M. Szydlo. A two-server, sealed-bid auction protocol. In *International conference on financial cryptography*, pages 72–86. Springer, 2002.

[50] J. Katz. Digital signatures. 2010.

[51] J. Katz, J. Loss, and J. Xu. On the security of time-lock puzzles and timed commitments. In *Theory of Cryptography: 18th International Conference*. Springer, 2020.

[52] J. Katz, U. Maurer, B. Tackmann, and V. Zikas. Universally composable synchronous computation. In *Theory of Cryptography: 10th Theory of Cryptography Conference*. Springer, 2013.

[53] R. Kumaresan and I. Bentov. How to use bitcoin to incentivize correct computations. In *ACM CCS 2015*. ACM Press, 2015.

[54] 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.

[55] 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.

[56] Arjen K Lenstra and Benjamin Wesolowski. Trustworthy public randomness with sloth, unicorn, and trx. *International Journal of Applied Cryptography*, 2017.

[57] H. Lin, R. Pass, and P. Soni. Two-round and non-interactive concurrent non-malleable commitments from time-lock puzzles. In *58th IEEE Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society Press, 2017.

[58] 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.

[59] J. Liu, T. Jager, S. Kakvi, and B. Warinschi. How to build time-lock encryption. *Designs, Codes and Cryptography*, 2018.

[60] 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*. Springer, 2022.

[61] A. Loe, L. Medley, C. O'Connell, and E. Quaglia. A practical verifiable delay function and delay encryption scheme. *IACR Cryptol. ePrint Arch.*, 2021.

[62] 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*, 2022.

[63] A. Loe, L. Medley, C. O'Connell, and E. Quaglia. Applications of timed-release encryption with implicit authentication. *14th International Conference on Cryptology in Africa, AFRICACRYPT 2023*, 2023.

[64] M. Mahmoody, T. Moran, and S. Vadhan. Time-lock puzzles in the random oracle model. In *Advances in Cryptology – CRYPTO 2011*, 2011.

[65] 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*, 2013.

[66] 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.

[67] G. Malavolta, P. Moreno-Sanchez, A. Kate, M. Maffei, and S. Rav. Concurrency and privacy with payment-channel networks. In *ACM CCS 2017*. ACM Press, 2017.

[68] G. Malavolta and S. Thyagarajan. Homomorphic time-lock puzzles and applications. In *Annual International Cryptology Conference*, pages 620–649. Springer, 2019.

[69] W. Mao. Timed-release cryptography. In *International Workshop on Selected Areas in Cryptography*, pages 342–357. Springer, 2001.

[70] T. May. Timed-release crypto. *http://www. hks. net. cpunks/cpunks-0/1560. html*, 1993.

[71] L. Medley and E. Quaglia. Collaborative verifiable delay functions. In *International Conference on Information Security and Cryptology (INSCRYPT 2021)*, 2021.

[72] A. Mert, E. Öztürk, and E. Savaş. Low-latency asic algorithms of modular squaring of large integers for vdf evaluation. *IEEE Transactions on Computers*, 2020.

[73] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[74] K. Paterson and E. Quaglia. Time-specific encryption. In *Security and Cryptography for Networks: 7th International Conference, SCN 2010*, 2010.

[75] K. Pietrzak. Simple verifiable delay functions. In *10th Innovations in Theoretical Computer Science Conference, ITCS 201*, pages 601–615, San Diego, California, 2019.

[76] M. Rabin. Transaction protection by beacons. *Journal of Computer and System Sciences*, 1983.

[77] M. Raikwar and D. Gligoroski. Sok: Decentralized randomness beacon protocols. *arXiv preprint arXiv:2205.13333*, 2022.

[78] 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.

[79] R.Peralta, H.Booth, L.Brandão, J Kelsey, and C. Miller. Interoperable Randomness Beacons. *NIST*, 2019.

[80] A. De Santis, S. Micali, and G. Persiano. Non-interactive zero-knowledge proof systems. In *Conference on the Theory and Application of Cryptographic Techniques*, 1987.

[81] A. Scafuro. Break-glass encryption. In *PKC '19: 22nd IACR International Conference on Practice and Theory of Public-Key Cryptography*, 2019.

[82] 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.

[83] A. Shamir. How to share a secret. 2010.

[84] W. Silvano and R. Marcelino. Iota tangle: A cryptocurrency to communicate internet-of-things data. *Future generation computer systems*, 2020.

[85] 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.

[86] S. Thyagarajan, T. Gong, A. Bhat, A. Kate, and D. Schröder. Opensquare: Decentralized repeated modular squaring service. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021.

[87] S. Thyagarajan, G. Malavolta, F. Schmid, and D. Schröder. Verifiable timed linkable ring signatures for scalable payments for monero. In *Computer Security–ESORICS 2022: 27th European Symposium on Research in Computer Security*, 2022.

[88] J. Truby. Decarbonizing Bitcoin: Law and policy choices for reducing the energy consumption of Blockchain technologies and digital currencies, 2018.

[89] W. Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *The Journal of finance*, 16(1):8–37, 1961.

[90] B. Wesolowski. Efficient Verifiable Delay Functions. In *Advances in Cryptology – EUROCRYPT 2019*, page 379–407, Darmstadt, Germany, 2019.