TIDE: A novel approach to constructing timed-release encryption

Anonymous

No Institute Given

Abstract. In ESORICS 2021, Chvojka et al. introduced the idea of taking a time-lock puzzle and using its solution to generate the keys of a public key encryption (PKE) scheme [13]. They use this to define a timed-release encryption (TRE) scheme, in which the secret key is encrypted 'to the future' using a time-lock puzzle, whilst the public key is published. This allows multiple parties to encrypt a message to the public key of the PKE scheme. Then, once a solver has spent a prescribed length of time evaluating the time-lock puzzle, they obtain the secret key and hence can decrypt all of the messages.
In this work we introduce TIDE (TIme Delayed Encryption), a novel approach to constructing timed-release encryption based upon the RSA cryptosystem, where instead of directly encrypting the secret key to the future, we utilise number-theoretic techniques to allow the solver to factor the RSA modulus, and hence derive the decryption key. We implement TIDE on a desktop PC and on Raspberry Pi devices validating that TIDE is both efficient and practically implementable. We provide evidence of practicality with an extensive implementation study detailing the source code and practical performance of TIDE.

Keywords: auctions · time-lock puzzle · timed-release encryption · public key cryptography.

1 Introduction

In 1996, Rivest et al. introduced the notion of sending a message 'to the future' using a time-lock puzzle [38]. This seminal paper is the basis of modern-day delay-based cryptography. Delay-based cryptography is a prominent and wide-ranging subject built around the notion of associating standard 'wall-clock time' with an iterated sequential computation. In modern times, delay-based cryptography is used in the classical sense of encrypting a message to the future using various primitives such as time-lock puzzles [20, 31], timed-release encryption [12, 13] and delay encryption [11]; as well as in alternative applications such as providing a computational proof-of-age of a document, and building a public randomness beacon [9, 35, 41].

In this paper, we introduce TIDE, a novel construction of timed-release encryption. TIDE is particularly suited to the application of sealed-bid auctions, providing a practical and efficient solution to Vickrey auctions, as we shall explore next.

1.1 Sealed-bid 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. In a sealed-bid second-price auction, known as a Vickrey auction, the highest bidder wins the goods but pays the price of the second highest bid [1, 7, 40]. The challenge of building a fair, efficient, and cryptographically secure Vickrey auction has been of interest for decades [5, 10, 11, 23, 28]. A common approach to constructing sealed-bid auctions is to implement a *commit and reveal* solution using an append-only bulletin board, e.g., a blockchain [23]. 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 the example of Vickrey auctions where it is necessary to learn the second highest bid as well as the first [1,7]. 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.

By replacing the commitments with time-lock puzzles one can obtain an elegant method of solving this problem [38]. Each party encrypts their bid as the solution to a time-lock puzzle. Therefore, in the opening phase if a party does not reveal their bid it can instead be opened by computing the solution to the puzzle. However, this method does not scale well becaue it leads to many different time-lock puzzles being solved, which is computationally expensive. Recently there has been research into solving this problem more efficiently.

At CRYPTO 2019 [31], Malavolta et al. suggest that each party encrypts their bid as a time-lock puzzle, as in the classical method suggested by Rivest et al. Their insight is that the tallyer 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 and all current constructions of homomorphic TLPs are based on indistinguishability obfuscation (IO) [11, 31]. IO aims to obfuscate programs to make them unintelligible whilst retaining their original functionality [3]. However, IO is known to be impractical with no construction efficiently implementable at the time of writing [27].

At EUROCRYPT 2021 Burdges et al. introduce Delay Encryption [11], a primitive which offers an alternative approach to solving this problem, using a delay-based analogue to identity-based encryption. 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 a public *session ID*. This session ID acts as a bulletin board, meaning anyone who knows this session ID can efficiently encrypt messages to it. All messages encrypted to the session ID can be efficiently decrypted by any party who knows a secret *session key*. The key feature of DE is a slow and sequential Extract algorithm, which outputs a session key after a prescribed amount of time. This time delay defines the

bidding phase of the auction in which parties may encrypt bids to the session ID. 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. In DE rather than solving multiple time-lock puzzle the Extract algorithm is run once which outputs a *session key*.

This seems to be an ideal solution, however the construction of DE presented in [11] comes with two significant practical disadvantages: (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. These two factors make this construction problematic from a practical standpoint.

The goal of the approaches outlined above is to utilise a time-delay to solve the auction problem in a scalable manner. This improves upon the efficiency of Rivest's solution by ensuring that at most two sequential computation (namely the puzzles containing the two highest bids in [31], and Extract in [11]) needs to be run, rather than one for each bid. However, the approaches so far have practical problems with the instantiation of their proposed candidate.

At ESORICS 2021 Chvojka et al. introduce the idea of taking a TLP and using its solution in the key generation of a public key encryption (PKE) scheme [13]. 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, but no concrete instantiation is provided.

In this work we present TIDE, a novel, efficient and easily implementable approach to building a TRE scheme to solve the scalability problems in Vickrey auctions. TIDE seamlessly integrates RSA encryption into a TLP using powerful results from number theory. On top of being a concrete construction, TIDE subtly differs in its approach to that in [13] in the way the secret key is derived.We provide further insights on how TIDE works next.

1.2 Technical Overview

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 [38], for an RSA modulus N. This assumption was first introduced in 1996 by Rivest et al. [38], and has been used to build a variety of cryptographic constructions [9, 20, 31, 35, 41]. TIDE deviates from previous literature by using number theoretic techniques to utilise the output $x^{2^t} \bmod N$ in a novel way. Previous approaches used squaring solely for its sequential properties, i.e., the final output is used only to guarantee a delay. For example, in time-lock puzzles, the solution to the puzzle is precomputed using a trapdoor, in order to hide a message as the product of the solution and the message [20, 31, 38]. This allows one to trivially obtain the message upon computing the delay, and hence solving the puzzle.

In the context of verifiable delay functions, the RSW assumption is used to prove that a certain amount of clock time has taken place. This is achieved by the solver computing repeated squarings upon a randomly sampled element of $\mathbb{Z}_N^*$, and computing a proof in order to mathematically prove to a verifier that t squarings have taken place [9, 35, 41].

In TIDE the output of the computation expands beyond guaranteeing a delay. Namely 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 [36].

By carefully setting up the system we provide the user with value x and ensure that the output of the 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 a 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 and security definitions in Section 4, where we follow the definition of TRE by Chvojka et al [13].

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 [36]. 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, the element x is efficiently sampled such that $\mathcal{J}_N(x) = -1$, where $\mathcal{J}_N(x)$ is the Jacobi symbol [29]. 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 Gen 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 [36]. The simplicity of RSA encryption and decryption makes TIDE a conceptually simple approach to sealed-bid auctions, whilst the underlying number theoretic techniques allow the functionality to be very efficient and practical.

1.3 Related Work

Alternatives to Vickrey auctions The most common style of auction is the sealed first-bid auction in which the highest bidder wins and pays the amount they bid for the goods. From a cryptographic perspective this is more straightforward to implement, making it easier to include additional properties

in such schemes. For example, research has been done into sealed first-bid auction schemes where the bids of losers remain hidden, by requiring bidders to run a protocol computing the highest bid, and ensuring that only the highest bid is opened [5, 39].

In a well-known paper 'Secure Multiparty Computation Goes Live' [8], 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 is computed, allowing transactions to be be made at this price point.

Both of these examples rely on a very different framework to that of TIDE: they requires multiple parties being online at the same time carrying out a protocol. As such, whilst linked by the application of auctions, we view such work as tangential. We now turn our attention to delay-based cryptography which is more closely related to our work.

Encrypting a message to the future Time-lock puzzles (TLPs) were first introduced in the seminal paper of Rivest et al. [38] as a way to encrypt messages to the future. They suggested various applications for this, including sealed-bid auctions and key escrow schemes. The method they use to build the delay is sequentially squaring in a finite group of unknown order, which is known as the RSW time-lock assumption. Recently, there have been some alternative approaches to building TLPs. Rather than using repeated squaring new TLPs include using witness encryption and bitcoin [30], randomised encodings [4], and random isogeny walks over elliptic curves [19].

The RSW time-lock assumption has been used as the base of various constructions of *verifiable delay functions* (VDFs) [9, 35, 41]. In a VDF a solver computes a delay similarly to a time-lock puzzle, but rather than decrypting a message at the end, the solver instead proves that they have spent the prescribed amount of time on the computation. This proof of elapsed time has primarily found use in randomness beacons which are used in blockchain design [14].

In 2019 De Feo et al. introduced a VDF based upon isogeny walks [19], which in 2021 they extended to a *delay encryption* (DE) scheme. DE is similar to a time-lock puzzle, but rather than proving that time has elapsed, instead a *session key* is derived. This can be seen as similar to the decryption key described in the technical overview of TIDE, in Section 1.2. Indeed DE as a primitive is very similar to the notion of timed-release encryption where we align our TIDE construction. The key difference between the two primitives is that DE uses notions from identity-based encryption and thus avoids using a trapdoor in the setup phase.

Recall in Section 1.1 that timed-release encryption (TRE) is another delay-based primitive, whose traditional definition combines public-key encryption with a time-server [12, 32]. Messages can be encrypted to a public key and decryption requires a trapdoor which is kept confidential by a time-server until at an appointed time. In a recent paper by Chvojka et al [13] TRE was defined

generically with a view to improving versatility and functionality. Whilst we build a timed-release encryption scheme following the definitions of Chvojka et al., our scheme does not require a time-server.

1.4 Contributions

In our work we design a novel and theoretically efficient variant of a time-lock puzzle by utilising RSA encryption and decryption to obtain a simple and efficient construction. We provide a security and efficiency analysis of our construction, proving that TIDE is cryptographically secure under the TRE notions [13]. We analyse the theoretical and practical efficiency of our scheme, proving that it has concrete theoretical advantages over the alternative proposals for Vickrey auctions and demonstrate that it is significantly more practical than current candidates. We present evidence of the practicality of our scheme by providing an implementation study using Raspberry Pi devices and a desktop PC, showing that TIDE can be run efficiently on consumer grade hardware. In particular, we show that when using a 2048-bit modulus, TIDE takes approximately one second to setup on a desktop PC and 30 seconds on a Raspberry Pi.

2 Preliminaries: Assumptions and Number Theory

In this section we review the time-lock assumption and number theory required to construct TIDE. For well-known theorems we refer to the relevant sources and we prove the other theorems in Section 4 and Appendix B.

The RSW time-lock assumption [38] is core to a number of notable constructions using a cryptographic delay in the latest literature [9, 18, 20, 31, 35, 41].

Definition 1. ***RSW time-lock assumption****: Let $N = pq$ where p and q are distinct odd primes. Uniformly select $x \in \mathbb{Z}_N^*$, where $\mathbb{Z}_N^* = \{x \,|\, x \in (0, N) \wedge \mathsf{gcd}(x, N) = 1\}$. 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 2.1 Square and Multiply [38].*

Secondly we note that the modulus used in our TIDE will be a Blum integer [6]. A Blum integer $N = pq$, is the product of two Gaussian primes. A Gaussian prime has the property $p \equiv 3 \bmod 4$.

Next, we provide the definition of quadratic residues.

Definition 2. ***Quadratic Residues*** *in $\mathbb{Z}_N^*$ are numbers r that satisfy congruences of the form:*

$$x^2 \equiv r \mod N \tag{1}$$

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.

Algorithm 2.1: Square and Multiply [16]

input : (a, b, N), // $a, b, N \in \mathbb{N}$, $a^b \bmod N$
$d \coloneqq 1$
$B \coloneqq \mathsf{bin}(b)$ // b in binary
for $j \in B$ **do**
  $d \coloneqq d^2 \bmod N$
  if $j = 1$ **then**
    $d \coloneqq da \bmod N$
  end
end
output: d

The Jacobi symbol, denoted $\mathcal{J}_N(r)$, is a function which defines the quadratic character of r in Equation 1. The Jacobi Symbol can be calculated in polynomial time using Euler's Criterion.

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

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

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.

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 1. *(Of Theorem 1). Euler's Criterion can be used to calculate the Jacobi Symbol of the number r in Equation 1 for a composite modulus N if the factorisation of N is known.*

Algorithm 2.2 shows how to determine the quadratic character of r for composite N using Theorem 1 and Corollary 1. 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 $\mathcal{QNR}_N^{+1}$.

Quadratic residues and quadratic non-residues for composite N have a distinct distribution in $\mathbb{Z}_N^*$.

Algorithm 2.2: Calculating $\mathcal{J}_N(r)$ for composite N.

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

Theorem 2. *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:*

$$\begin{aligned} \left|\mathcal{QR}_N\right| &= \frac{\left|\mathbb{Z}_N^*\right|}{4} = \frac{\phi(N)}{4}. \\ \left|\mathcal{QNR}_N^{+1}\right| &= \frac{\left|\mathbb{Z}_N^*\right|}{4} = \frac{\phi(N)}{4}. \\ \left|\mathcal{QNR}_N^{-1}\right| &= \frac{\left|\mathbb{Z}_N^*\right|}{2} = \frac{\phi(N)}{2}. \end{aligned} \tag{3}$$

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

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 2.1.

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

Theorem 3. *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 $\alpha \equiv r^{\frac{p+1}{4}} \bmod p$.*

Example 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 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.

If $r \in \mathcal{QR}_N$ then the CRT implies that there are four distinct solutions to Equation 1.

Theorem 4. *For all $N = pq$, where p and q are distinct odd primes, each $r \in \mathcal{QR}_N$ has four distinct solutions.*

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.

Theorem 5. *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 $r \bmod N$.*

Finally, we discuss a method to factor a Blum integer N in polynomial time if specific information is provided.

Fermat's factorisation method is a technique to factor an odd composite number $N = pq$ in exponential time [17]. 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 [17]. However, if x and x' in a congruence of squares are known, then factoring N can be done in polynomial time.

Theorem 6. *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. Proofs for Theorems 1, 2, 4, and Corollary 1 can be found in [29]. Proofs for Theorems 3 and 6 can be found in Section 4. The proof for Theorem 5 can be found in Appendix B.

3 Our Construction

In this section we give the concrete details of our construction TIDE. Formally, TIDE is a TRE scheme and we provide a formal exposition of its security properties in Section 4. In our TRE scheme $\mathcal{C}$ is the Challenger, $\mathcal{S}$ is the Solver, and $\mathcal{A}$ is the Adversary. In the context of Vickery auctions, $\mathcal{C}$ can be though of as the auctioneer and $\mathcal{S}$ can be thought of as a Bidder. As is customary, multiple bidders are participate in an auction.

A TRE scheme consists of four algorithms: Gen, Solve, Encrypt, Decrypt. Gen and Solve provide the time-lock element of the scheme: Gen generates a secret key,

public key and a puzzle, Solve takes the puzzle and recovers the corresponding secret key. Encrypt can be ran by multiple parties (bidders) simultaneously using the public key. Solve can be run by any party i.e. any bidder or third party can run this algorithm. Once Solve has terminated, the Solver can then use the secret key to decrypt all of the bids encrypted with Encrypt by using the Decrypt algorithm. We now outline the details of the four TIDE algorithms.

- $(\mathsf{sk}, \mathsf{pk}, P, t) \leftarrow \mathsf{Gen}(1^\kappa, t)$ takes as input a security parameter 1^κ and time parameter t and ouputs a secret key sk, public key pk, puzzle P, and time parameter. 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 $\perp$.

Algorithm 3.1: Gen run on security parameter 1^κ and time parameter t to create the secret key sk, public key pk and puzzle P.

input : $1^\kappa, t$
$p, q := 1$
while $p = q$ **do**
  $p := \mathtt{prime}(\frac{\kappa}{2})$
  $q := \mathtt{prime}(\frac{\kappa}{2})$
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: $(\mathsf{sk}, \mathsf{pk}, P, t)$

1) $\mathcal{C}$ runs $(\mathsf{sk}, \mathsf{pk}, P, t) \leftarrow_{\mathrm{R}} \mathsf{Gen}(1^{\kappa}, t)$ to generate the secret key, public key, and puzzle as seen on Algorithm 3.1 Gen. The function $\mathtt{prime}(j)$ on lines 3 and 4 is the Miller-Rabin Monte Carlo algorithm [33] 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. Gen 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 2.2 using De Morgan's laws [25]. 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.

Next, α_t is calculated, where α_t is the t^{th} 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 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 t^{th} 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 t^{th} 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 $\mathcal{C}$ securely stores sk and passes (pk, P, t) to $\mathcal{S}$ who must solve:

$$\text{Given } (\mathsf{pk} \coloneqq (N, e), P \coloneqq (x, x_0, x_{-t}), t), \text{ find the factors of } N.$$

2) $\mathcal{S}$ (or any party) runs $\mathsf{sk} \leftarrow \mathsf{Solve}(\mathsf{pk}, P, t)$ to solve the challenge, as seen on Algorithm 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 2.1 with inputs $(x_{-t}, 2^{t-1}, N)$. The term x' is guaranteed to be in $\mathcal{QR}_N$ by Definition 2. $\mathcal{S}$ 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 6, 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'.

3) $\mathcal{S}$ runs $c \leftarrow \mathsf{Encrypt}(\mathsf{pk}, m)$ as seen in Algorithm 3.3 Encrypt. Encrypt inputs the public key $\mathsf{pk} \coloneqq (N, e)$ and encrypts a message m using RSA-OEAP encryption and outputs the ciphertext c. First Encrypt outputs the RSA-OAEP parameters k_0, k_1, G, H, where k_0 and k_1 are constants used for padding and G and H are hashing algorithms modelled as random oracles. Using RSA-OAEP, parties can encrypt messages to this modulus and encryption exponent. This means that messages can only be decrypted using the Decrypt 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 (Bidder) using pk prior to the Solve algorithm recovering the sk.

Algorithm 3.2: Solve runs on the public key, puzzle, and time parameter pk, P, t to recover the secret key sk.

input : $\mathsf{pk} \coloneqq (N, e), P = (x, x_0, x_{-t}), t$

$x' \coloneqq x_{-t}^{2^{t-1}} \bmod N$

$p' \coloneqq \mathsf{gcd}(x - x', N)$

$q' \coloneqq \frac{N}{p'}$

$\mathsf{sk} \coloneqq (p', q')$

output: sk

Algorithm 3.3: Encrypt runs on a message public key pk and message m, to produce ciphertext c.

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

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

$m' \coloneqq m \mid\mid 0^{k_1}$ `// Zero pad to` $n - k_0$ `bits`

$r \coloneqq \mathrm{rand}(k_0)$ `// Generate a random` k_0 `bit number`

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

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

$m'' \coloneqq X \mid\mid Y$ `// Create message object`

$c \coloneqq m''^e \bmod N$ `// RSA encrypt`

output: c

Algorithm 3.4: Decrypt runs on secret key sk and ciphertext c, to produce message m.

input : $\mathsf{sk} \coloneqq (p', q'), c$

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

$N \coloneqq p'q'$

$\phi(N) \coloneqq (p' - 1)(q' - 1)$

$d \coloneqq e^{-1} \bmod \phi(N)$ `// recover` d `using EEA`

$m'' \coloneqq c^d \bmod N$

$X \coloneqq \lfloor c'' \cdot 2^{-k_0} \rfloor$ `// Extract` X

$Y \coloneqq m'' \bmod 2^{k_0}$ `// Extract` Y

$r \coloneqq Y \oplus H_{k_0}(X)$ `// Recover` r

$m' \coloneqq X \oplus G_{n-k_0}(r)$ `// Recover padded message`

$m \coloneqq m' \cdot 2^{-k_1}$ `// Remove padding`

output: m

4) $\mathcal{S}$ runs $\{m, \perp\} \leftarrow \mathsf{Decrypt}(\mathsf{sk}, c)$ as seen in Algorithm 3.4 Decrypt. Decrypt inputs the secret key $\mathsf{sk} := (p, q)$ and decrypts ciphertext c using RSA-OEAP encryption and recovers the message m or outputs an error $\perp$. Decrypt also outputs the same RSA-OAEP parameters k_0, k_1, G, H as Encrypt. Next, Decrypt recovers the decryption exponent d on lines $2, 3, 4$, where Euclids Extended Algorithm is used. Finally, the RSA-OEAP decrypt algorithm removes the padding and randomness added during the encryption to recover the message m.

Implementation and Performance Analysis. We implement TIDE on a desktop PC and a cluster of raspberry Pis. We show how the timings of Algorithm 3.2 Solve grows linearly with the time parameter t, whilst the other algorithms grow by $O(1)$ in the time parameter. We provide timings with several RSA moduli of practical relevance, and note in particular that with a modulus size of 2048, the average time to run Solve was one second on a desktop PC. The full description of the implementation and our results can be found in Appendix A.

4 Security

We provide a security analysis of our construction. To this end, we recall the formal definition of Timed-Release Encryption (TRE), following Chvojka et al. [13][1], along with the definitions of correctness and security for a TRE scheme.

Definition 3. *A timed-release encryption scheme with message space $\mathcal{M}$ is a tuple of algorithms TRE = (Gen, Solve, Encrypt, Decrypt) defined as follows.*

- $(\mathsf{pk},\mathsf{sk}, P, t) \leftarrow \mathsf{Gen}\,(1^\kappa, t)$ *is a probabilistic algorithm which takes as input a security parameter 1^κ and a time hardness parameter t, and outputs a public encryption parameter pk, a secret key sk, and a puzzle P. We require that Gen runs in time poly $((\log t), \kappa)$.*
- $\mathsf{sk} \leftarrow \mathsf{Solve}(\mathsf{pk}, P, t)$ *is a deterministic algorithm which takes as input a public key pk, a puzzle P, and a time parameter t, and outputs a secret key sk. We require that Solve runs in time at most $t \cdot \mathsf{poly}(\kappa)$.*
- $c \leftarrow \mathsf{Encrypt}\,(\mathsf{pk}, m)$ *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 $\perp$.*

Definition 4 (Correctness).
A TRE scheme is correct *if for all $\kappa \in \mathbb{N}$ and hardness parameter t, it holds that*

$$\Pr\left[m = m' : \begin{array}{r}(\mathsf{pk},\mathsf{sk}, P, t) \leftarrow \mathsf{Gen}\,(1^\kappa, t)\,, \mathsf{sk} \leftarrow \mathsf{Solve}\,(\mathsf{pk}, P, t) \\ m' \leftarrow \mathsf{Decrypt}\,(\mathsf{sk}, \mathsf{Encrypt}\,(\mathsf{pk}, m))\end{array}\right] = 1$$

[1] In [13] they offer a generalised version of this definition, to incorporate what they define *sequential timed-release encryption*. This is beyond the scope of this work, and we instead specify the "non-sequential" case.

Definition 5 (Security). *A timed-release encryption scheme is secure with gap* $0 < \epsilon < 1$ *if for all polynomials* n *in* κ *there exists a polynomial* $\tilde{t}(\cdot)$ *such that for all polynomials* t *fulfilling that* $t(\cdot) \geq \tilde{t}(\cdot)$*, and every polynomial-size adversary* $\mathcal{A} = \{(\mathcal{A}_{1,\kappa}, \mathcal{A}_{2,\kappa})\}_{\kappa \in \mathbb{N}}$ *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\kappa \in \mathbb{N}$ *it holds*

$$\mathbf{Adv}_{\mathcal{A}}^{\mathrm{TRE}} = \left| Pr \left[\begin{array}{r} \mathsf{pk}, P \leftarrow \mathsf{Gen}\,(1^\kappa, t) \\ (m_0, m_1, \mathrm{st}) \leftarrow \mathcal{A}_{1,\kappa}(\mathsf{pk}, P) \\ b = b' : b \xleftarrow{\$} \{0,1\}; c \leftarrow \mathsf{Encrypt}\,(\mathsf{pk}, m_b) \\ b' \leftarrow \mathcal{A}_{2,\kappa}(c, \mathrm{st}) \end{array} \right] - \frac{1}{2} \right| \leq \mathsf{negl}(\kappa)$$

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

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.

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, or distinguishing between ciphertexts encrypted to a Blum integer N, when the computational time is bounded by t.

Definition 6 (BBS Shortcut Assumption).

Let the RSA Assumption be that for any $N \leftarrow_R \mathsf{Gen}\,(1^\kappa)$ *and* $e = 65537$*, it is hard for any probabilistic polynomial-time algorithm to find the e-th root modulo* N *of a random* $y \leftarrow_R \mathbb{Z}_N^*$ *[37].*

The BBS Shortcut Assumption states that given (N', e) *and terms* (x, x_0, x_{-t})*, where* $N' \leftarrow_R \mathsf{Gen}\,(1^\kappa)$ *is a randomly sampled Blum integer,* $e = 65537$*,* x *is a randomly sampled integer such that* $x \in \mathcal{QNR}_N^{-1}$*,* $x_0 \coloneqq x^2 \bmod N$*, 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 first factoring* N'*.*

We now analyse this security assumption, in order to relate it to the RSA assumption that RSA with OAEP relies on [22].

Recall $P = (x, x_0, x_{-t})$ 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 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. By the RSW time-lock assumption, we know that this will take t time to evaluate, and hence we assume that $P = (x, x_0, x_{-t})$ are only useful when factoring N'

We now prove TIDE is a timed-release encryption scheme satisfying correctness and security.

Theorem 7. *TIDE is correct.*

Proof. First, 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 3.3 Encrypt and Algorithm 3.4 Decrypt respectively.

This corresponds to the statement that the RSA cryptosystem with OAEP is correct, which is known to be true [22].

Now suppose Algorithm 3.1 Gen 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 $\mathsf{sk} = (p,q)$. What remains is to prove that Solve outputs $\mathsf{sk} = (p,q)$. This proof will require a sequence of arguments based on the Theorems outlined in Section 2.

First we must prove that Algorithm 3.1 Gen correctly selects the term x such that $x \in \mathcal{QNR}_N^{-1}$.

Corollary 2. *(Of Theorem 2). The while loop on lines 8-12 of Algorithm 3.1 Gen selects $x \in \mathcal{QNR}_N^{-1}$ with overwhelming probability.*

Proof. The while loop on lines 8-12 of Algorithm 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.

Second we prove that Algorithm 3.1 Gen correctly calculates the term x_{-t}, which is the t^{th} principal square root of x_0. This proof begins by proving Theorem 3, and subsequently uses the Chinese Remainder Theorem for the final proof.

Proof. (Theorem 3). 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 1) $\alpha^2 \equiv r^1 r^{\frac{p-1}{2}} \equiv r \bmod p$. We refer to α as the principal square root of $r \bmod p$.

Theorem 8. *The Algorithm 3.1 Gen correctly calculates the t^{th} principal square root x_{-t} of the seed x_0.*

Proof. Let $\omega = \frac{p+1}{4}$. If Algorithm 3.1 Gen provides the seed term $x_0 \in \mathcal{QR}_N$, then, by Theorem 3, the t^{th} principal square root of $x_0 \bmod p$ is $\alpha_t := x_0^{\omega^t} \bmod p$ and the t^{th} principal square root of $x_0 \bmod q$ is $\beta_t := x_0^{\omega^t} \bmod q$. Then, the Chinese Remainder Theorem (Appendix B Theorem 11) is used to calculate:
$x_{-t} := [\alpha_t q(q^{-1} \bmod p) + \beta_t p(p^{-1} \bmod q)] \bmod N$.

Third we must prove that the Algorithm 3.2 Solve correctly calculates the term $x' \in \mathcal{QR}_N$ using Algorithm 2.1.

Theorem 9. *Algorithm 2.1 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 2.1 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 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 2.1 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 2.1 will have $d := x_0^{2^k} \bmod N$, and j will be the final digit of $B := 10 \ldots 0$. 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. This means 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 2.1 will terminate at the $k+1$ iteration as the final digit of B was processed, and the algorithm will return $d := x_0^{2^{k+1}} \bmod N$.

Finally, Theorem 6 is proven to show that Algorithm 3.2 Solve calculates $\mathsf{gcd}(x' - x, N)$ to recover a non-trivial factor of N [36].

Proof. (Theorem 6.) 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 7: the correctness of TIDE.

Proof. (Theorem 7) For any pk, sk, and puzzle generated by Gen, we show that sk can be recovered by Solve. More precisely, let $N = pq, P := (x, x_0, x_{-t}), t$ be output by Gen, before being input into Algorithm 3.2 Solve. Algorithm 3.2 Solve will calculate the term x' by entering the following parameters $(x_{-t}, 2^{t-1}, N)$ into Algorithm 2.1, which will output $x' := x_{-t}^{2^{t-1}} \bmod N$. The term x' is guaranteed to be correct by Theorem 9 and is guaranteed to be in $\mathcal{QR}_N$ by Definition 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 6, 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 10. *TIDE is a secure TRE scheme under the RSW, RSA and BBS-shortcut assumptions.*

To prove TIDE secure, we show that two messages encrypted using public key (N, e) are indistinguishable under a chosen plaintext attack, where the adversary is bounded by t computation time. We first note that the underlying encryption scheme is RSA with OAEP padding, which is IND-CPA secure [22]. In our proof we provide a reduction from the TRE security of TIDE to IND-CPA security of RSA with OAEP. Explicitly, this requires proving that giving an adversary the additional parameters of P and t, and bounding their computation time by t offers a negligible advantage over the standard RSA-OAEP game.

We first prove the following statement.

Lemma 1. *Given any (N, P, t) output by Algorithm 3.1 Gen, 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 3.1 Gen. Note from Algorithm 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 t^{th} 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 $x' \in \mathcal{QR}_N$, in less than t sequential steps, 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. Specifically, if Solve is honestly run, then $x' := x_0^{2^{t-1}} \bmod N$ is calculated using Algorithm 2.1 with the input $(x_{-t}, 2^{t-1}, N)$. By the RSW time-lock assumption calculating x' using Algorithm 2.1 requires $t-1$ sequential steps. Once x' is calculated, Algorithm 3.2 Solve recovers the factors of N by calculating $p' := \mathsf{gcd}(x - x', N)$ and $q' = \frac{N}{p'}$.

Next, suppose there exists a PPT algorithm $\mathcal{E}_{<t}$ to evaluate x' in less than $t-1$ sequential steps. Finding such an x' using $\mathcal{E}_{<t}$ reduces to the RSW time-lock assumption and we obtain a contradiction. Therefore, it is not possible to recover $p^* := \mathsf{gcd}(x - x', N)$ without sequentially evaluating x' with non-negligible probability.

Next, we prove part (ii): that factoring N faster than sequential squaring 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.

Next, 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 [36]. 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 [6]. 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. [6, 21, 26].

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 $q' = \frac{N}{p'}$.

We now use this result to obtain a reduction from the TRE security of TIDE to the standard RSA IND-CPA security.

Proof (sketch). (Theorem 10) We start by assuming that there exists an adversary $\mathcal{A} = (\mathcal{A}_{1,\kappa}, \mathcal{A}_{2,\kappa})$ who can gain a non-negligible advantage in the $\mathbf{Adv}_{\mathcal{A}}^{\mathrm{TRE}}$ game defined in Definition 5.

We use Lemma 1 and Definition 6 to show that if the adversary wins the game by factoring N we obtain a contradiction based on the RSW assumption, and if they win the game without factoring N, we obtain a contradiction based on the RSA and BBS-shortcut assumptions.

Recall from Lemma 1 that if the adversary $\mathcal{A}$ factors a Blum integer N output by Algorithm 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, recall that RSA with OAEP padding is IND-CPA-secure under the RSA assumption [22]. Suppose $\mathcal{A}$ gained a non-negligible advantage in the TRE security game without factoring. As the underlying encryption scheme is IND-CPA secure, to distinguish between the messages m and m' with any advantage would require decrypting one of the messages, and hence taking an e-th root modulo N. By the BBS shortcut assumption presented in Definition 6, any adversary who gains an advantage in the TRE security game could also gain the same non-negligible advantage in the standard IND-CPA game for RSA-OAEP, and hence break the RSA assumption. This gives us another contradiction. Therefore TIDE is secure under the RSW, RSA and BBS-shortcut assumptions.

5 Conclusion

In this work we introduced TIDE, a new TRE construction which seamlessly integrates the RSA cryptosystem into a time-lock puzzle using powerful number-theoretic concepts. TIDE challenges a solver to factor a special class of RSA modulus, known as a Blum integer. Parties may encrypt to this RSA modulus,

and any solver who factors the modulus may easily decrypt all encrypted messages. We demonstrated that this property makes TIDE well-suited to sealed-bid auctions: We compared TIDE to the most recent constructions for sealed-bid auctions, showing that TIDE has advantages both in terms of practicality and efficiency. We proved security of TIDE in the TRE framework introduced by Chvojka et al, and we implemented TIDE on both a Raspberry Pi and on a desktop PC, showing that it is indeed a practical construction.

References

1. L. Ausubel. A generalized vickrey auction. *Econo0 metrica*, 1999.
2. E. Bach and J. Shallit. *Algorithmic number Theory, Vol. 1: Efficient Algorithms.* MIT Press, 1996.
3. 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.*
4. 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.*
5. 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.*
6. L. Blum, M. Blum, and M. Shub. A Simple Unpredictable Pseudo-Random number Generator. In *Journal on Computing.*
7. A. Blume and P. Heidhues. All equilibria of the vickrey auction. *Journal of economic Theory.*
8. 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.*
9. D. Boneh, J. Bonneau, B. Bünz, and B. Fisch. Verifiable Delay Functions. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference.*
10. F. Brandt. Auctions. In *Handbook of Financial Cryptography and Security.*
11. J. Burdges and J. DeFeo. Delay Encryption. In *40th Annual International Conference on the Theory and Applications of Cryptographic Techniques. EUROCRYPT 2021*, 2021.
12. J. Cathalo, B. Libert, and J. Quisquater. Efficient and non-interactive timed-release encryption. In *International Conference on Information and Communications Security.*
13. 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.*
14. B. Cohen and K. Pietrzak. The chia network blockchain, 2019.
15. J. Cook. Computing Legendre and Jacobi symbols, Algorithm for computing Jacobi symbols. https://www.johndcook.com/blog/2019/02/12/computing-jacobi-symbols, 2019.
16. T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, 2009.

17. R. Crandall and C. Pomerance. *Prime numbers: A Computational Perspective*. Springer-Verlag, 2005.
18. 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*.
19. 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*.
20. C. Freitag, I. Komargodski, R. Pass, and N. Sirkin. Non-malleable time-lock puzzles and applications. In *Theory of Cryptography Conference*.
21. J. Friedlander, C. Pomerance, and I. Shparlinski. Period of the power generator and small values of Carmichael's function. In *American Mathematical Society*. Mathematics of Computation 70, 2000.
22. E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern. Rsa-oaep is secure under the rsa assumption. In *Annual International Cryptology Conference*.
23. H. Galal and A. Youssef. Verifiable sealed-bid auction on the ethereum blockchain. In *International Conference on Financial Cryptography and Data Security*.
24. C. Gauss. *Disquisitiones Arithmeticae*. Yale University Press, 2009.
25. R.L. Goodstein. *Boolean Algebra*. Dover Publications, 2007.
26. F. Griffin and I. Shparlinski. On the linear complexity profile of the power generator. In *IEEE Transactions on Information Theory (6, Sep 2000)*, 2000.
27. 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*.
28. A. Juels and M. Szydlo. A two-server, sealed-bid auction protocol. In *International conference on financial cryptography*.
29. J. Katz and Y. Lindell. *Introduction to Modern Cryptography, Second Edition*. CRC Press, 2014.
30. J. Liu, F. Garcia, and M. Ryan. Time-release protocol from bitcoin and witness encryption for sat. *Korean Circulation Journal*.
31. G. Malavolta and S. Thyagarajan. Homomorphic time-lock puzzles and applications. In *Annual International Cryptology Conference*.
32. W. Mao. Timed-release cryptography. In *International Workshop on Selected Areas in Cryptography*.
33. G. Miller. Riemann's Hypothesis and Tests for Primality. In *Journal of Computer and System Sciences, 13(3)*, 1976.
34. Shyam Narayanan. Improving the speed and accuracy of the miller-rabin primality test, 2014.
35. K. Pietrzak. Simple verifiable delay functions. In *10th Innovations in Theoretical Computer Science Conference, ITCS 201*.
36. M. Rabin. Digitalized signatures and public-key functions as intractable as factorization. In *MIT/LCS/TR-212, MIT Laboratory for Computer Science*, 1979.
37. R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*.
38. 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.
39. K. Sako. An auction protocol which hides bids of losers. In *International Workshop on Public Key Cryptography*.
40. W. Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *The Journal of finance*.
41. B. Wesolowski. Efficient Verifiable Delay Functions. In *Advances in Cryptology – EUROCRYPT 2019*.

A Practicality and Implementation

In this section we detail why TIDE is the most practical solution to the issue in sealed-bid auctions detailed in the introduction when compared to previous solutions. Recall that the purpose of using time-lock puzzles in the context of auctions is to ensure that when a party does not reveal their bid it can be recovered at some computational cost by solving the relevant puzzle. The problem with this approach is that it does not scale well as the the number of bidders increase. The recent works by Malavolta et al. [31] and Burdges et al. [11] each introduce a method to ensure that only one long computation needs to be solved, regardless of the number of bidders. TIDE shares this property, as Solve only needs to be run once per auction, and as such is more efficient than the standard TLP approach.

Malavolta et al. advocate using homomorphic time-lock puzzles, to determine homomorphically which is the winning puzzle (and the second highest bid in the case of Vickrey auctions) and then solve only the relevant puzzles. The problem with this approach is that it relies on fully homomorphic encryption, or on indistinguishability obfuscation (IO), both of which are currently lacking a practical construction.

Burdges et al. propose using long chains of isogenies to achieve delay encryption, which is a more practical approach than using homomorphic time-lock puzzles. However, their candidate construction has implementation issues in the form of a very large evaluator storage requirement and a setup that grows proportionally to the time delay. Whilst promising in its approach, these challenges make this candidate problematic in practice.

TIDE overcomes these issues with a setup that takes on average 1 to 2 seconds on a consumer-grade desktop PC for a 2048-bit RSA modulus. Once the public key parameters are output the delay is completely and predictably adjustable with the time parameter t. Furthermore, the maximum storage size of any party is bounded by the size of the RSA modulus.

Finally, we support these arguments with an implementation study, including code that can be run quickly on consumer-grade hardware. In the next section we provide the timings and results of our implementation analysis.

A.1 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. Furthermore, we have ensured that the authors of this code are anonymized.

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

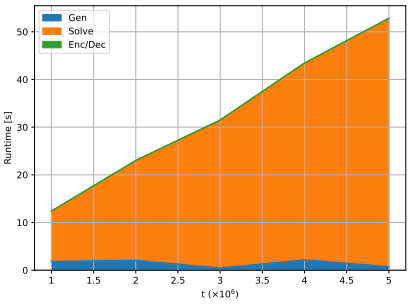

Fig. 1. The impact of adjusting parameter t on the run time of Gen, Solve, Encrypt, and Decrypt algorithms when run on a desktop PC with a 2048 bit modulus. The primary effect is on Solve, which displays a linear increase.

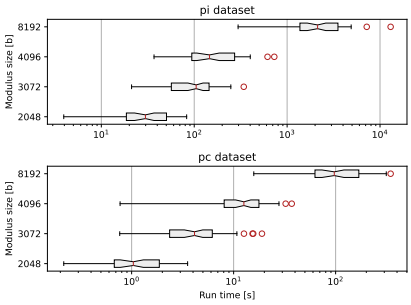

Fig. 2. The spread of setup time across modulus sizes and machines. Setup time increases in response to an increase in modulus size. The dispersion of run times is similar across different devices.

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 1 demonstrates our first experiment.

This experiment shows how the run time of Gen, Solve, Encrypt and Decrypt is impacted by the time parameter t for a 2048 bit modulus 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 3.2 Solve as t varies supports the RSW time-lock assumption in Definition 1. Furthermore, we see that Algorithm 3.1 Gen 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 12. We also observe that Gen has minor variations in the run time when compared to Solve and Encrypt and Decrypt. This is a result of the randomised nature of Gen in comparison to the deterministic behaviour of Solve, Encrypt, and Decrypt.

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

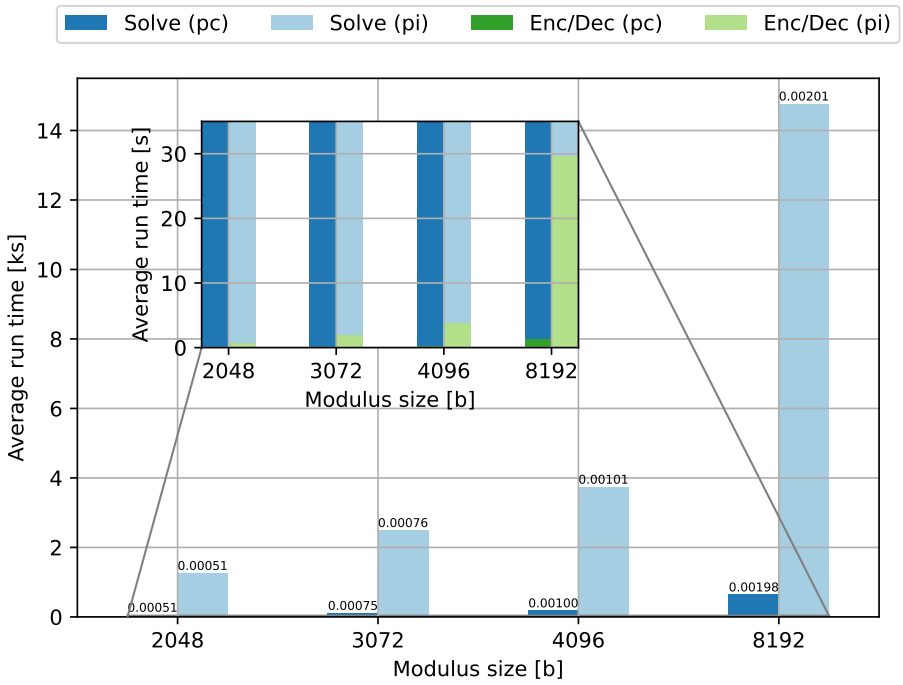

Fig. 3. Encrypt and Decrypt take significantly less time to run than Solve across modulus sizes. The inset shows a zoomed in view of the bar chart which is necessary for the effect of Encrypt and Decrypt to be observed. Above each bar is the ratio Solve to Encrypt and Decrypt run time.

modulus is included as an edge case to demonstrate performance at the upper bound present in real world applications.

In Figure 2, we plot the spread of run times for the Gen 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 [34]. 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 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 Gen 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 in relation to the modulus size, 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 auction. However, as we have seen in our experiments, this property is not an obstacle for practical use. Even in more computationally constrained environments such as the Pi, the Solve and Encrypt and Decrypt algorithms do not require an impractical time cost. Although 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 1, the value for t can be set with reasonable accuracy to introduce a desired delay for the target hardware.

B Number Theory

B.1 The Chinese Remainder Theorem

The Chinese Remainder Theorem (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.

Theorem 11. *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:

$$y = \sum_{i=1}^{k} \alpha_i N_i N_i^{-1} \mod N \tag{4}$$

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 [29].

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 paper we are concerned in applying the results of the CRT in the case where $N = pq$, where p and q are distinct odd primes. Therefore, we limit now limit our application of the CRT Isomorphism to this case.

Definition 7. *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

$$\mathbb{Z}_N^* \simeq \mathbb{Z}_p^* \times \mathbb{Z}_q^* \tag{5}$$

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

B.2 Fermat-Euler Theorem

Next, we present the Fermat-Euler Theorem.

Theorem 12. *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 [24].

Corollary 3. *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 12 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 2.1.

Next, we provide proof of Theorem 5.

Proof. (Theorem 5) If N is a Blum integer, then $N \equiv 1 \bmod 4$. By Theorem 4 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 $x' \in \mathcal{QR}_N$ [6].

B.3 Jacobi Symbol Algorithm

Finally, we provide the details of Algorithm B.2 Samplex which can efficiently sample the parameter x in the puzzle P without knowledge of the factors p and q. Algorithm B.2 can be used in lieu of the while loop on lines 7–12 in Algorithm 3.1 Gen. The proof of correctness of Algorithm B.2 can be found in Bach et al [2]. Therefore, the proof of Corollary 2 is also relevant in the case of Algorithm B.2. The original Python code for the function Jacobi(x, pk) can be found in [15].

Algorithm B.1: Jacobi is run on random input x and public key pk to calculate the Jacobi symbol of x .

```
   input : x, pk
 1 j = 1
 2 while x ≠ 0 do
 3 |   while x mod 2 = 0 do
 4 |   |   x ≔ x/2
 5 |   |   if pk mod 8 = 3 ∨ pk mod 8 = 5 then
 6 |   |   |   j ≔ −j
 7 |   |   end
 8 |   end
 9 |   if x mod 4 = 3 ∧ pk mod 4 = 3 then
10 |   |   j ≔ −j
11 |   end
12 |   x, pk ≔ pk, x
13 |   x ≔ x mod pk
14 end
15 if pk ≠ 1 then
16 |   j ≔ 0
17 end
   output: j
```

Algorithm B.2: Samplex is run on public key pk to efficiently sample parameter x without knowledge of p and q.

```
   input : pk
 1 do
 2 while j ≠ −1
 3 x ≔ U(2, pp)
 4 j ≔ Jacobi(x, pk)
   output: x
```