Applications of Timed-release Encryption with Implicit Authentication

Angelique Loe[1], Liam Medley[1][0009−0009−4533−7890], Christian O'Connell[2], and Elizabeth A. Quaglia[1][0000−0002−4010−773X]

[1] Royal Holloway, University of London
{angelique.loe.2016, liam.medley.2018, elizabeth.quaglia}@rhul.ac.uk
[2] Independent co362@cantab.ac.uk

Abstract. A whistleblower is a person who leaks sensitive information on a prominent individual or organisation engaging in an unlawful or immoral activity. Whistleblowing has the potential to mitigate corruption and fraud by identifying the misuse of capital. In extreme cases whistleblowing can also raise awareness about unethical practices to individuals by highlighting dangerous working conditions. Obtaining and sharing the sensitive information associated with whistleblowing can carry great risk to the individual or party revealing the data. In this paper we extend the notion of timed-release encryption to include a new security property which we term *implicit authentication*, with the goal of making the practice of whistleblowing safer.
We formally define the new primitive of timed-release encryption with implicit authentication (TRE-IA), providing rigorous game-base definitions. We then build a practical TRE-IA construction that satisfies the security requirements of this primitive, using repeated squaring in an RSA group, and the RSA-OAEP encryption scheme. We formally prove our construction secure and provide a performance analysis of our implementation in Python along with recommendations for practical deployment and integration with an existing whistleblowing tool SecureDrop.

Keywords: time-lock puzzle · timed-release encryption · applied cryptography

1 Introduction

In 2013, Edward Snowden leaked highly classified information from the National Security Agency [37, 38]. This information was leaked at great personal risk. Other recent cases of whistleblowing include the Panama papers [32], the Paradise papers [9], and the Pandora papers [27]. Leaking information subjected the whistleblowers to personal danger due to the power and influence of the organisations whose data was leaked. In the case of the Panama papers, the whistleblower claimed their 'life was in danger' [21].

In this work we construct a cryptographic tool based on timed-release encryption [17], which can augment existing tools for whistleblowers, such as SecureDrop [2]. Our goal is to provide an element of guaranteed delay in the release of

sensitive information which has potential to make the practice of whistleblowing safer. We model our solution on the Edward Snowden case, in which all classified material was destroyed before arriving in Russia, in order 'To protect himself from Russian leverage' [1].

We propose a construction which offers the concept of delay-based encryption for whistleblowers, to allow them to rely on cryptographic assurances rather than the trust of a journalist or ombudsman. The technique we introduce allows sensitive information to be encrypted in such a way that a) there is a predictable delay between the *receipt of the ciphertext* encapsulating the leaked information and the *release* of the information, and b) there is no way an adversary can forge a chosen document to insert alongside the genuine documents. This delay will afford the whistleblower time to destroy all classified material after encapsulating the material, and hence ensure their safety. In the Snowden case, the delay would have allowed passage to a safe harbour country without the sensitive information being decrypted until a specified time.

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

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

Paper structure and contributions. In the remainder of this section we detail our methodology in approaching this problem, outlining the security goals we desire and providing an overview of our construction, before discussing relevant related work. In Section 2 we formally define the primitive *TRE with implicit authentication* (TRE-IA), giving game-based definitions of the required security properties. In Section 3 we present our construction for a TRE-IA scheme, which is based upon the BBS-random number generator and RSA-OAEP encryption. In Section 4 we prove our construction is secure under the definitions given in Section 2. In Section 5 we provide a Python implementation of our construction, along with a performance analysis to demonstrate its practicality. We also provide a practical example of how to use TRE-IA with SecureDrop [2] to show how our construction would integrate with existing whistleblower tools.

1.1 Technical Overview

The goal of this work is to explore how a time-delay can be used by vulnerable parties such as whistleblowers in order to make the distribution of sensitive material safer. In order to do so, we introduce a novel construction based upon a clear set of properties, which we can implement in practice and which may augment existing whistleblower tools. Therefore, we define the following security goals that we believe may be helpful to a whistleblower based on the real life cases of the Panama papers leak [21] and the Edward Snowden leak [37, 38].

1. An *adjustable time delay*: this will allow the whistleblower to destroy all materials that can be used against them. It also allows the whistleblower to have a configurable amount of time to reach a place of safety.
2. *Maximum flexibility*: this will allow the whistleblower to determine when a) they can encrypt and distribute messages and b) they can 'start the clock' for evaluating the delay. This is achieved through the separation of the ciphertexts and the *challenge*. When the challenge is distributed this starts the clock.
3. *Implicit authentication*: this ensures that no other entity can generate a document of their choice to insert into the leak.

Property 1 can be useful for a whistleblower to protect themselves from the dangers associated with carrying sensitive material. Property 2 allows a whistleblower to gather various different pieces of evidence over time and encrypt and distribute this evidence to journalists. The whistleblower can also ensure that the journalists cannot yet leak the material until a time delay has passed, thus mitigating risks to the personal safety of the whistleblower. We believe it is crucial that the whistleblower remains in control of all aspects of the system, and by giving the whistleblower the freedom to distribute ciphertexts *without* 'starting the clock' on the time delay, we minimise the trust placed in journalists, and provide the whistleblower with fine-grained control of when the documents are leaked.

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

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

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

A time-lock puzzle [36] encrypts a message to the future, in such a way that once a solver spends a predictable amount of time evaluating the encrypted message, they obtain the plaintext message. One could think of using the naive approach of simply encrypting each message as a TLP and passing it to a journalist. This achieves property 1, however it limits the whistleblower to the condition that they encrypt all materials at once, and allows adversarial parties to impersonate the whistleblower. As we wish for the whistleblower to have a finer control

of the encryption process we see that the latter method has limitations. For example, the whistleblower may wish for multiple messages to be encrypted. This is a reasonable assumption as the Panama papers exposed over eleven million leaked documents [32] and the Paradise papers exposed over thirteen million leaked documents [9]. Using only a basic TLP results in the loss of property 2, that is, the whistleblower loses the element of maximum flexibility. A more appropriate approach is to use a symmetric key as the solution to a TLP, and then distribute ciphertexts separately. This gives us a solution which is close to ideal, as ciphertexts can be distributed to journalists, with guarantees both time delay and flexibility as to when the whistleblower starts the clock. In other words, we can achieve properties 1 and 2 using this approach, however we are missing the implicit authentication property described in 3.

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

As a starting point, we use TRE as defined by Chvojka et al., who show that one can generically construct TRE from a TLP and a public encryption scheme [17]. However, Chvojka et al.'s approach to constructing TRE does not give us the desired property of a secret encryption key, nor does it necessarily imply that it is impossible to derive the encryption key from the decryption key.

Therefore, to achieve these goals simultaneously we propose a variant of timed-release encryption, which we term TRE with implicit authentication (TRE-IA) and instantiate this with a construction based upon the BBS-CSPRNG random number generator [11], and the RSA-OAEP encryption scheme [8].

1.2 Related Work

Time, delay, and encryption There are several primitives in the literature that have considered the component of time and delay in the context of encryption. Time-lock puzzles (TLPs) and the associated Timed-Release Crypto schemes, first proposed by Rivest et al. in [36], encrypt a message to the future. TLPs create a puzzle π from which a message m can be decrypted after time t. Timed-Release Crypto is closely related to TRE-IA in that it achieves a very similar functionality. However, there are differences between the two primitives: In essence, TRE-IA uses distinct (asymmetric) keys for encryption and decryption, as opposed to one symmetric key. In both primitives decryption keys are recovered after the delay, in order to decrypt the ciphertext. However, in TRE-IA only the decryption key is recovered. This leads to a different functionality: the whistleblower has control over what information is leaked (i.e., encrypted) as the sole holder of the encryption key.

Previous literature also discusses timed-release encryption (TRE) in two main forms. The classical definition of a TRE scheme is to use a third-party time-server to release messages after a given amount of time [29, 16]. The most recent paper by Chvojka et al. defines TRE as a combination of a time-lock puzzle and

public-key encryption [17]. This is in order to introduce the notion of a *sequential* TRE scheme, with the goal of building a public sequential-squaring service which acts as the time server. This allows many parties to time-lock messages, whilst only one party performs the lengthy computation. Conceptually, the classical definition of timed-release encryption differs from the Chvojka et al. scheme because the former relies on a trusted time-server. For the remainder of this work, when we write TRE we will be referring to the method of Chvojka et al.

The main difference between TRE and TRE-IA is the former requires encryption to be public rather than only a prerogative of the whistleblower: In TRE-IA the encryption key cannot be derived by the solver even when the decryption key is recovered. In a TRE-IA scheme assurance is provided through the property of implicit authentication that only the whistleblower can encrypt to the classic notion of a *public key* in a standard PKE scheme.

In the delay-based cryptography literature there are also Verifiable Delay Functions (VDFs), first proposed by Boneh et al. [12] and a new, closely related primitive called Delay Encryption (DE) [14], derived from VDFs. VDFs require a prover to compute a slow sequential computation of length t, also known as a challenge, and then efficiently prove to a verifier that they have done so. VDFs do not generate a ciphertext, or indeed have any method for encrypting data, and so are clearly distinct from TRE.

In DE, the concept of a *session* is introduced. A session consists of a session ID, which any party may encrypt a message to, and an extraction key. Any party may extract the session key, which allows for decryption of all messages encrypted to the session ID, and this extraction takes t time to run. This primitive is distinct from TRE-IA, in that DE allows anyone to encrypt to the session ID, compared to just the whistleblower being in control of encryption in TRE-IA.

Signcryption A cryptographic primitive offering a similar property to implicit authentication (IA) is Signcryption [41–43]. IA provides the receiving party with assurance that the encrypted documents were sent by the whistleblower. The IA property states that only the holder of the private encryption key can generate a legitimate ciphertext that can be correctly decrypted to a chosen message. In a similar fashion the concept of Signcryption was introduced to provide a single computation that would simultaneously provide the authenticity from a digital signature scheme (DSS), and the confidentiality from a public-key encryption (PKE) scheme. However, Signcryption does not consider the property of delay which is crucial to our TRE-IA scheme.

Tools for whistleblowers A variety of tools exist which can provide protection to whistleblowers. The Tor [4] browser can be used to navigate the Internet anonymously, PGP [44] keys and encrypted email services can support the secure communication between a whistleblower and the investigative journalist, as well as end-to-end encrypted messaging services such as Signal [3]. Closely aligned to our intended end-goal is the SecureDrop [2] submission system, which enables whistleblowers to securely deliver documents containing leaked information. The novelty of our proposed solution in this space is the introduction of a delay, which provides, when combined with encryption, a time 'bubble' within

which the whistleblower can reach safety. Indeed, we see TRE-IA as an addition to a whistleblower's toolbox featuring the property of a time-lock delay. In this context, we recognise that no tool is perfect and solutions which guarantee the safety of the whistleblower should be grounded in reality. Our proposed scheme represents a first, technological step towards introducing delay as a form of protection to whistleblowers. Accordingly, to provide an example of integrating with existing whistleblowing tooling, in Section 5.2 we show how the TRE-IA construction can be augmented to work with the SecureDrop tool.

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

2 Timed-Release Encryption with Implicit Authentication

We now provide the definition and properties of TRE-IA, which can be seen as an extension of the TRE primitive as defined by Chovjka et al. [17]. We deviate from the security model of Chvojka et al. in the following way: (i) to fit in with our model of the encryptor alone knowing the encryption key, we require that the encryptor runs setup, (ii) we introduce the new notion of implicit authentication. We provide an intuition of this security property, along with a detailed description of the game in the following section.

In the definition and security games that follow, $\mathcal{E}$ is the encryptor, $\mathcal{D}$ is the decryptor, $\mathcal{A}$ is the PPT adversary, $\leftarrow_R$ represents a probabilistic algorithm, and $\leftarrow$ represents a deterministic algorithm.

Definition of TRE-IA. A TRE-IA scheme consists of the following algorithms: ($\mathsf{TRE.Setup}$, $\mathsf{TRE.Gen}$, $\mathsf{TRE.Enc}$, $\mathsf{TRE.Solve}$, $\mathsf{TRE.Dec}$), defined as follows.

- $pp, td \leftarrow_R \mathsf{TRE.Setup}(1^\lambda)$. $\mathsf{TRE.Setup}$ is an algorithm run by $\mathcal{E}$ that takes as input security parameter 1^λ and outputs the public parameter pp and trapdoor td. $\mathcal{E}$ must keep td private. The parameter pp can be given to $\mathcal{D}$ after $\mathsf{TRE.Setup}$ completes. $\mathsf{TRE.Setup}$ runs in time $\mathsf{poly}(\lambda)$.
- $e, d, C, t \leftarrow_R \mathsf{TRE.Gen}(pp, td, t)$. $\mathsf{TRE.Gen}$ is an algorithm run by $\mathcal{E}$ that takes as input the public parameter pp, the trapdoor td, and time parameter t and computes an encryption key e, decryption key d, and a public challenge C. The term t indicates the number of sequential steps required to evaluate C to recover the decryption key d when the trapdoor td is not known. The parameter t can be given to $\mathcal{D}$ after $\mathsf{TRE.Gen}$ completes. However, care must be taken when the challenge C is given to the decryptor $\mathcal{D}$. $\mathcal{E}$ must only provide challenge C when they wish to 'start the clock' on recovering the decryption key d. $\mathsf{TRE.Gen}$ runs in time $\mathsf{poly}(\lambda)$.
- $c \leftarrow_R \mathsf{TRE.Enc}(m, e, pp)$. $\mathsf{TRE.Enc}$ is an algorithm run by $\mathcal{E}$ that takes as input a message m, encryption key e, and public parameter pp and outputs ciphertext c. The ciphertext c can be given to $\mathcal{D}$ after $\mathsf{TRE.Enc}$ completes. $\mathsf{TRE.Enc}$ runs in time $\mathsf{poly}(\lambda)$.

- $d \leftarrow \mathsf{TRE.Solve}(pp, C, t)$. $\mathsf{TRE.Solve}$ is an algorithm run by $\mathcal{D}$ that takes as input the parameters pp, C, t and outputs the decryption key d. $\mathsf{TRE.Solve}$ requires t sequential steps to recover d with a run time of $(t)\mathsf{poly}(\lambda)$.
- $\{m', \perp\} \leftarrow \mathsf{TRE.Dec}(c, d, pp)$. $\mathsf{TRE.Dec}$ is an algorithm run by $\mathcal{D}$ that takes as input the ciphertext c, decryption key d, and public parameter pp and outputs plaintext m' or error $\perp$. $\mathsf{TRE.Dec}$ runs in time $\mathsf{poly}(\lambda)$.

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

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

Correctness Game

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

Security In TRE-IA, similarly to the TRE definition presented by Chvokja et al. [17], *security* is defined as an indistinguishability game as follows:

Security Game

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

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

Implicit Authentication Game

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

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

Implicit authentication is the new property we introduce in TRE, which ensures that an adversary is unable to *forge* a ciphertext for a message of their choice, hence providing an implicit guarantee that ciphertexts are authentic. In the context of our application, this property ensures that a malicious party cannot insert a document of their choice into the leak provided by a genuine whistleblower.

The idea of modelling security against a ciphertext forgery is inspired by the notions of plaintext integrity and ciphertext integrity [7, 10] in the symmetric encryption setting. More specifically, plaintext integrity states that it should be infeasible to produce a ciphertext decrypting to any message which the sender has not encrypted, and ciphertext integrity requires that it be infeasible to produce a ciphertext not previously produced by the sender, regardless of whether or not the underlying plaintext is 'new' [7].

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

To overcome this, we took the approach of modelling our implicit authentication game as an encryption analogue of *selective* forgery [26, 30], a property used in digital signature schemes where an adversary first commits to a target message m^* and is later challenged to forge a signature for this target message. The key difference in our implicit authentication game is that the adversary is instead asked to output an encryption of the target message, rather than a digital signature.

At a high-level, the TRE-IA game proceeds as follows. The adversary is first asked to output a message m^* that they wish to encrypt. The adversary is

then given the decryption key, and access to an encryption oracle. Finally, the adversary is asked to output a ciphertext c to the challenger. The adversary wins the game if c decrypts to the message m^*. We make this precise in the Implicit Authentication game.

3 Construction of a TRE-IA scheme

In this section we provide the implementation details of a concrete TRE with implicit authentication. The TLP element of our TRE-IA is derived from the construction of the Blum Blum Shub CSPRNG [11], and as such we name it the BBS-TRE. Our TRE-IA construction also uses the RSA-OAEP PKE scheme.

Recall that implicit authentication states that without access to the encryption key an adversary should not be able to forge a ciphertext for a message of their choice. When RSA is used in practice it is standard procedure to use $e = 65537$ as the public encryption exponent [6]. This does not allow for implicit authentication, as an adversary can guess this. Using any 'standard' fixed encryption key, or a key from a fixed small set will allow an adversary to guess this key, and hence encrypt a message as a ciphertext with more than negligible probability. As such, we design our construction to choose e at random, to ensure that we obtain the implicit authentication property whilst still conforming to the NIST SP-800-56B standard for random public exponent key pair generation [6], Section 6.3.2. Using the BBS-CSPRNG provides an elegant solution to integrating random keys in a TRE-IA setup, as seen in [28].

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

Algorithm 4: Square and Multiply Binary Fast Modular Exponentiation Algorithm [18].

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

A notable selection of delay-based cryptographic schemes such as time-lock puzzles, verifiable delay functions, delay encryption, and timed-release encryption rely on the Rivest Shamir Wagner (RSW) time-lock assumption [13, 29, 33, 36, 39]. Like other RSW-based delay-based cryptographic schemes our BBS-TRE also relies on the RSW time-lock assumption.

Definition 1. ***RSW time-lock assumption:***
Let $N = pq$ be an RSA modulus and uniformly select $x_0 \in \mathbb{Z}_N^*$. If adversary $\mathcal{A}$ does not know the factorisation or group order of N then calculating $x_t \equiv x_0^{2^t} \bmod N$ is a non-parallelisable calculation that will require t sequential modular exponentiations calculated with Algorithm 4 Square and Multiply [36].

Our BBS-TRE is summarised as follows:

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

In our BBS-TRE $\mathsf{TRE.Setup}$ outputs the public parameters N, k_0, k_1, G, H. The first parameter is the RSA modulus N which is a Blum integer. A Blum integer is the product of two Gaussian primes i.e. $N = pq$, where $p \equiv q \equiv 3 \bmod 4$ [11]. The modulus being a Blum integer is key requirement for the correctness of our scheme. The parameters k_0, k_1, G, H are the RSA-OAEP parameters which can be seen in detail in Algorithm 7. $\mathsf{TRE.Setup}$ also outputs the trapdoor $\phi(N) \coloneqq (p-1)(q-1)$ and keeps this parameter private.

Next, the $\mathsf{TRE.Gen}$ algorithm outputs the encryption and decryption keys, the challenge (x_0, x_t) and the time parameter t. In the challenge (x_0, x_t) the term x_0 is a randomly sampled quadratic residue of N, denoted $x_0 \in \mathcal{QR}_N$. The decryption key d is calculated with the trapdoor using Algorithm 4 with the parameters $(x_0, 2^{t-1} \bmod \phi(N), N)$. If $\mathsf{gcd}(d, \phi(N)) = 1$, then in the challenge (x_0, x_t), the term x_t is set to $d^2 \bmod N$ and the encryption key e is set to $d^{-1} \bmod \phi(N)$. Next the $\mathsf{TRE.Enc}$ algorithm inputs a message m and the encryption key e and the public encryption parameters pp to output the ciphertext c using the RSA-OEAP PKE scheme. This scheme deviates from a traditional RSA-OAEP PKE scheme as the encryption key e remains private to ensure the property of implicit authentication.

Next, the $\mathsf{TRE.Solve}$ algorithm sequentially calculates the decryption key d using Algorithm 4 with the parameters $(x_0, 2^{t-1}, N)$. The RSW time-lock assumption tells us that finding the term d will require $t-1$ sequential modular exponentiations to calculate if the trapdoor $\phi(N)$ is not known. Finally, the $\mathsf{TRE.Dec}$ algorithm takes as input the ciphertext c and the decryption key d and the public parameters pp, and outputs either the message m or an error $\perp$ using the RSA-OEAP PKE scheme. We now provide the full details of our BBS-TRE algorithms.

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

Algorithm 5: $\mathcal{E}$ runs $\mathsf{TRE.Setup}$ on security parameter 1^λ to output public parameter pp and trapdoor td.

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

2) $\mathcal{E}$ runs $e, d, C, t \leftarrow_R \mathsf{TRE.Gen}(pp, td, t)$ to generate the encryption and decryption keys and the challenge, as seen on Algorithm 6. First, $\mathsf{TRE.Gen}$ sets the variable gcd to 0. Next, $\mathsf{TRE.Gen}$ enters a while loop to generate an appropriate encryption and decryption exponent e, d for RSA-OAEP. This is done by first uniformly selecting $x \in \mathbb{Z}_N^*$ and computing $x_0 \equiv x^2 \bmod N$. Then $\mathsf{TRE.Gen}$ evaluates $d \equiv x_0^{2^{t-1} \bmod \phi(N)} \bmod N$. The decryption key d is calculated using Algorithm 4 with the parameters $(x_0, 2^{t-1} \bmod \phi(N), N)$. Note that $\mathcal{E}$ is able to reduce the exponent $2^{t-1} \bmod \phi(N)$ using the trapdoor. The while loop runs until the decryption key d computed on line 6 is coprime to $\phi(N)$. That is, until $\mathsf{gcd} \coloneqq \mathtt{gcd}(d, \phi(N)) = 1$. Once this is found the while loop exits and the term $x_t \coloneqq d^2 \bmod N$ is calculated and the encryption key e is calculated using the extended Euclidean Algorithm. In Theorem 1 we prove that the while loop will terminate. Furthermore, we prove that in expectation the number of iterations the while loop will require to generate a challenge such that $\mathsf{gcd} \coloneqq \mathtt{gcd}(d, \phi(N)) = 1$ is $\frac{\pi^2}{3} \approx 3.3$. This finding and the associated proof is a key contribution of our concrete TRE-IA construction. Finally, the challenge C is set to the tuple (x_0, x_t). $\mathsf{TRE.Gen}$ then outputs the encryption and decryption keys e, d and the challenge and time parameter C, t. The challenge that $\mathcal{D}$ must solve to recover the decryption key is: for seed x_0, find d such that $d \equiv \sqrt{x_t} \bmod N$. The encryption key e must remain private, and C and t must only be released to $\mathcal{D}$ once $\mathcal{E}$ would like

the decryption key d to be extracted under the RSW time-lock assumption by using the TRE.Solve algorithm.

Algorithm 6: $\mathcal{E}$ runs TRE.Gen on public parameter, trapdoor, and time parameter pp, td, t to create the encryption and decryption exponents and the challenge e, d, C, t.

input : $pp, \phi(N), t$ // $t \in \mathbb{N}$
$\mathsf{gcd} \coloneqq 0$
while $gcd \neq 1$ **do**
  $x \coloneqq \mathcal{U}(2, 2^{\frac{\lambda}{2}})$ // $x \in \mathbb{Z}_N^*$
  $x_0 \coloneqq x^2 \bmod N$
  $d \coloneqq x_0^{2^{t-1} \bmod \phi(N)} \bmod N$
  $\mathsf{gcd} \coloneqq \mathsf{gcd}(d, \phi(N))$
end
$x_t \coloneqq d^2 \bmod N$
$e \coloneqq d^{-1} \bmod \phi(N)$ // EEA
$C \coloneqq (x_0, x_t)$
output: e, d, C, t

3) $\mathcal{E}$ runs $c \leftarrow_R \mathsf{TRE.Enc}(m, e, pp)$ to output ciphertext c, as seen on Algorithm 7. TRE.Enc is the encryption algorithm of the RSA-OAEP PKE scheme. Each step of the algorithm is described on the comments of each line. The final step is to output the ciphertext c. In a TRE scheme the ciphertext c can be released to the decryptor independently of the challenge and time parameter output by TRE.Gen.

Algorithm 7: $\mathcal{E}$ runs TRE.Enc on (m, e, pp) to output ciphertext c.

input : m, e, pp
// $pp \coloneqq (N, k_0, k_1, G, H)$
$m' \coloneqq m \,||\, 0^{k_1}$ // Zero pad to $n - k_0$ bits
$r \coloneqq \mathsf{rand}(k_0)$ // 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
$m'' \coloneqq X \,||\, Y$ // Create message object
$c \coloneqq m''^e \bmod N$ // RSA encrypt
output: c

4) $\mathcal{D}$ runs $d \leftarrow \mathsf{TRE.Solve}(pp, C, t)$ to evaluate the challenge and output the decryption key as seen on Algorithm 8. First TRE.Solve calculates the term $\sqrt{x_t}$ by entering the parameters $(x_0, 2^{t-1}, N)$ into Algorithm 4. By the RSW time-lock assumption it will take $t-1$ sequential steps to calculate d because the trapdoor is not known by the decryptor $\mathcal{D}$. Next, TRE.Solve checks if $\sqrt{x_t}^2 \bmod N = x_t$ is true. If the condition is true, then d is set to $\sqrt{x_t}$ and output and the algorithm terminates.

5) $\mathcal{D}$ runs $\{m, \perp\} \leftarrow \mathsf{TRE.Dec}(c, d, pp)$ to output the plaintext message m or $\perp$, as seen on Algorithm 9. TRE.Dec is the decryption algorithm of the RSA-OEAP PKE scheme. Each step of the algorithm is described on the comments of

each line. The final step is to output the message m or $\perp$. By the correctness of the RSA-OAEP PKE scheme, if the parameter d extracted by TRE.Solve under the RSW time-lock assumption has the property $ed \equiv 1 \bmod \phi(N)$ (line 9 of TRE.Gen), then the message m will be recovered. Else, TRE.Dec will output $\perp$.

Algorithm 8: $\mathcal{D}$ runs TRE.Solve to evaluate pp, C, t and output the decryption key d.

input : pp, C, t
// $pp \coloneqq (N, k_0, k_1, G, H)$, $C \coloneqq (x_0, x_t)$
1 $\sqrt{x_t} \coloneqq x_0^{2^{t-1}} \bmod N$
2 **if** $\sqrt{x_t}^2 \bmod N = x_t$ **then**
3 $\quad d \coloneqq \sqrt{x_t}$
4 **end**
output: d

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

Algorithm 9: $\mathcal{D}$ runs TRE.Dec on (c, d, pp) to recover message m or output $\perp$.

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

4 Security Analysis

In this section we provide the security analysis of our concrete BBS-TRE scheme. First we prove that our BBS-TRE is correct and secure. We then prove that our scheme holds the property of implicit authentication.

We first provide proof of the correctness of our scheme. The outline of our proof will be as follows: first we will prove that TRE.Gen will terminate and

generate a suitable RSW time-lock puzzle challenge, second we will prove that TRE.Solve will correctly output the decryption key d, third we will prove that the decryption key d is unique because N is a Blum integer, and finally we will prove that the decryption key will correctly return the original message m when it is used to decrypt the ciphertext c generated with the encryption exponent e.

First we must prove that the while loop in TRE.Gen will terminate and generate a suitable challenge and decryption key.

Theorem 1. *The while loop in TRE.Gen will in expectation take $\frac{\pi^2}{3}$ trials to generate a suitable challenge and decryption key d.*

Proof. The probability of two randomly selected integers being coprime is $\frac{6}{\pi^2}$ [24], Theorem 33. The Blum integer $N = pq$ generated with TRE.Setup is randomly selected using the Miller Rabin Monte Carlo algorithm [31]. Next, the $\phi(N)$ is calculated as $(p-1)(q-1)$. Therefore, $\phi(N)$ is always even. Each iteration of the while loop in TRE.Gen is a Bernoulli trial. In our Bernoulli trial N and hence $\phi(N)$ are randomly selected and the integer d on line 5 of TRE.Gen is also randomly selected. We model d as a random integer as it is an output of the BBS CSPRNG. In each trial, if $\mathsf{gcd}(d, \phi(N)) = 1$ the outcome is a success, otherwise if $\mathsf{gcd}(d, \phi(N)) \neq 1$ then the outcome is a failure. Therefore, in each trial, the probability of selecting two random integers which are coprime when one integer is even is $\frac{6}{2\pi^2} = \frac{3}{\pi^2}$.

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

Second we must prove that the TRE.Solve algorithm correctly calculates the decryption key d. To prove this we must show that the Square and Multiply algorithm correctly calculates any term correctly in a BBS sequence. We first provide a brief summary of the BBS CSPRNG and subsequently a security analysis of our BBS-TRE for each of the required properties. The BBS CSPRNG [11] starts by selecting $x \in \mathbb{Z}_N^*$ and calculates the seed value $x_0 \equiv x^2 \bmod N$. To produce a string of t bits, the least significant bit is extracted from each term $x_i \equiv x_{i-1}^2 \bmod N$ for $i \in (1, \ldots, t)$. The equivalent representation of the first t terms of the sequence can be seen in Table 1.

Theorem 2. *Algorithm 4 Square and Multiply correctly calculates the x_i term of the BBS CSPRNG.*

Proof. The input to calculate the term x_i of the BBS CSPRNG takes as input $(x_0, 2^i, N)$, where x_0 is the seed term, and N is a Blum integer. Consider the base case when $i := 1$. The algorithm proceeds as follows: d is set to 1 and the

[3] Numerical analysis also indicated that over thousands of trials, independent of the size of $\phi(N)$, the average number of iterations the while loop must run until a suitable challenge was found was 3.3.

Table 1. The first t terms of the BBS CSPRNG. The first row identifies which $i \in (1, \ldots, t)$ is being calculated. The second, third, and fourth rows are equivalent representations of the same term, i.e. in the penultimate column, for $i := t-1$, the terms x_{t-1}, x_{t-2}^2, and $x^{2^{t-1}}$ are equivalent.

i	0 seed	1	2	$\ldots$	$t-2$	$t-1$	t
x_i	x_0	x_1	x_2	$\ldots$	x_{t-2}	x_{t-1}	x_t
x_{i-1}^2	x_0	x_0^2	x_1^2	$\ldots$	x_{t-3}^2	x_{t-2}^2	x_{t-1}^2
$x_0^{2^i}$	x_0	$x_0^{2^1}$	$x_0^{2^2}$	$\ldots$	$x_0^{2^{t-2}}$	$x_0^{2^{t-1}}$	$x_0^{2^t}$

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 4 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 4 will have $d := x_0^{2^k} \bmod N$, and j will be the final digit of $B := 10\ldots0$. For any k, the variable B will be a binary string starting with the digit 1 followed by a trail of k digits equal to 0. 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 4 will terminate at the $k+1$ iteration as the final digit of B was processed, and the algorithm will return $d := x_0^{2^{k+1}} \bmod N$.

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

Proof. Suppose encryptor $\mathcal{E}$ honestly generates a random public parameter, challenge and time parameter $pp := (N, k_0, k_1, G, H), (C := (x_0, x_t), t)$ and presents these to an honest $\mathcal{D}$. Next, suppose $\mathcal{D}$ selects the legitimate evaluation algorithm TRE.Solve to evaluate (C, t). The TRE.Solve algorithm will calculate the decryption key d by entering the following parameters $(x_0, 2^{t-1}, N)$ into the Algorithm 4, which will output $d := x_0^{2^{t-1}} = x_{t-1} \bmod N$. TRE.Solve will correctly output the BBS term x_{t-1} with overwhelming probability due to the correctness of Algorithm 4 noted in Theorem 2. Therefore, the TRE.Solve algorithm will correctly output $d := x_{t-1}$.

Next we must prove that the decryption key $d := x_{t-1} = \sqrt{x_t} \bmod N$ output by TRE.Solve is unique. First we must recall that d by definition of being a

term in a BBS CSPRNG sequence is a quadratic residue of the modulus N and provide a brief definition.

Definition 2. ***Quadratic residues****. Let $N = pq$, where p, q are λ bit primes. If $r \equiv x^2 \bmod N$, for some $x \in \mathbb{Z}_N^*$, we say that r is a quadratic residue of N. This is denoted as $r \in \mathcal{QR}_N$.*

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

$$d \coloneqq \sqrt{x_t} \bmod N \tag{1}$$

This challenge arises because the Chinese Remainder Theorem isomorphism indicates that when $N = pq$, where p, q are distinct odd primes, that Equation 1 has four distinct solutions [25]. That is, $\pm a \equiv \pm b \equiv \sqrt{x_t} \bmod N$, where $a \neq b$.

Theorem 3. *If $N = pq$ is a Blum integer, then the decryption key d in our BBS-TRE extracted by* TRE.Solve *is unique.*

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

Without loss of generality, eliminate the two roots with $\mathcal{J}_N$ equal to -1, say $\mathcal{J}_N(b) = \mathcal{J}_N(-b) = -1$. This leaves $\mathcal{J}_N(a) = \mathcal{J}_N(-a) = 1$. It is the case that only one of $-a$ or a has $\mathcal{J}_p = \mathcal{J}_q = 1$ as $p \equiv 3 \bmod 4$. Therefore, it is this one that is the only quadratic residue modulo N [11].

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

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

Corollary 1. *The BBS-TRE is correct.*

Proof. By Theorem 1 we know that TRE.Gen will terminate and output a suitable RSW TLP challenge (C, t). By Theorem 2 and Lemma 1 we know that the decryption key d will be recovered by TRE.Solve. By Theorem 3 we know that the decryption key d against the modulus N is unique. From the Fermat-Euler Theorem [22] we know that the decryption key d calculated on line 5 of TRE.Gen

is the same as the decryption key recovered by TRE.Solve on line 1. From the correctness of the Extended Euclidean Algorithm [25] we know that e calculated on line 9 of TRE.Gen is the multiplicative inverse of d. Finally, from the correctness of the RSA-OEAP scheme [8] we know that TRE.Dec will correctly recover the message m using decryption key d from the ciphertext c output by TRE.Enc using encryption key e with overwhelming probability.

Next we prove the security of our scheme. To prove the security of our scheme a set of arguments will need to be addressed. The outline of our proof will be as follows: First we prove that finding a square root $\mod N$ when N is an RSA modulus is equivalent to factoring N. Second we prove that given the public parameters and the RSW challenge an adversary cannot derive the decryption key d in less than t sequential steps. Finally we prove that if $\mathcal{E}$ selects one of two equal length messages to encrypt using TRE.Enc and outputs ciphertext c, then the only way an adversary can guess with greater than $\frac{1}{2} + \mathsf{negl}(\lambda)$ probability which message was encrypted is to honestly run TRE.Solve and recover the decryption key d.

Theorem 4. *Let $N = pq$, where p and q are λ bit primes. Then given any $r \in \mathcal{QR}_N$, finding x such that $x^2 \equiv r \bmod N$ is equivalent to factoring N.*

Proof. Proof can be found in the paper by Rabin [34].

The next part of our proof of the security property is to show that the adversary cannot recover the decryption key in less than t sequential steps. If the adversary can recover d in less than t sequential steps then they can output b' equal to b with probability greater than $\frac{1}{2} + \mathsf{negl}(\lambda)$ by decrypting the ciphertext. Our proof is split into two parts: i) when $\mathcal{A}$ attempts to compute d in less than t sequential steps, and ii) when $\mathcal{A}$ attempts to recover $\sqrt{x_t} \bmod N$ using a method that does not use C, t. Therefore, our next step is to prove that the only way d can be recovered without knowing the factors (or trapdoor) of N is to honestly evaluate the challenge in t sequential steps.

Theorem 5. *Our BBS-TRE requires t sequential steps to recover the decryption key d.*

Proof. Suppose $\mathcal{E}$ honestly generates a random public parameter pp and generates the encryption key, decryption key, challenge, and time parameter e, d, C, t. Next $\mathcal{A}$ selects two messages of the same length m_0 and m_1 for $\mathcal{E}$ to encrypt. $\mathcal{E}$ uniformly selects $b \in \{0, 1\}$ and encrypts m_b. $\mathcal{A}$ produces a PPT algorithm $\mathcal{A}_0$ which pre-processes pp and c and outputs a state $\mathsf{st} \leftarrow \mathcal{A}_0(pp, c)$. $\mathcal{E}$ sends the challenge and time parameter to $\mathcal{A}$. $\mathcal{A}$ produces a PPT algorithm $\mathcal{A}_1$ to output $b' \leftarrow \mathcal{A}_1(\mathsf{st}, C, t)$.

We start by proving part i), that computing d reduces to the RSW time-lock assumption. First we recall from Lemma 1 that TRE.Solve correctly outputs the decryption key d in t sequential steps, and we know from Theorem 3 that the decryption key d is unique. Next we note that the pre-processing is carried out before the $\mathcal{A}$ is given the challenge and time parameter C, t. Therefore, the

probability that $\mathcal{A}$ can compute d in less than t steps is negligible. Specifically, if TRE.Solve is honestly run, then d is calculated using Algorithm 4 with input $(x_0, 2^{t-1}, N)$. Therefore, by the RSW time-lock assumption, calculating d with Algorithm 4 requires $t-1$ sequential steps.

Next, suppose $\mathcal{A}$ selects PPT algorithm $\mathcal{A}_{<t}$ to evaluate d in less than $t-1$ sequential steps. However, using such an algorithm $\mathcal{A}_{<t}$ contradicts the RSW time-lock assumption.

What remains is to show that giving $\mathcal{A}$ the challenge and time parameter $C := (x_0, x_t), t$ does not allow them to take square roots mod N faster than sequential squaring. To see this, note that by construction x_0 and x_t are the seed term and t^{th} term in a BBS CSPRNG sequence [11]. Under the Generalised BBS assumption [13], knowledge of these terms does not allow finding $d = \sqrt{x_t}$ faster than sequential squaring unless $x_0^{2^{\lambda(\lambda(N))}}$ mod N is calculated efficiently, where $\lambda(N)$ is the Carmichael function [15]. Finding $x_0^{2^{\lambda(\lambda(N))}}$ efficiently is an open problem given by Theorem 9 of Blum et al. [11, 19, 23].

Next, we prove part ii). Observe that finding $d = \sqrt{x_t} \bmod N$ i.e. taking square roots mod N without challenge and time parameter C, t reduces to the open problem of integer factorisation, Theorem 4. Therefore, as N is an RSA modulus we assume it cannot be factored by any PPT algorithm with more than negligible probability. Therefore, the only way a PPT algorithm could recover the unique decryption key d when the factorisation (or trapdoor) of N is not known is to honestly run TRE.Solve

Theorem 5 proves that the adversary cannot recover d in less than t sequential steps to win the Security game. Therefore, to conclude our proof of the security property we must demonstrate that the adversary cannot guess b in less than t sequential steps without knowledge of the decryption key.

Theorem 6. *Our BBS-TRE scheme is secure.*

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

Let IND-CPA^{RSA} be the standard IND-CPA game for RSA-OAEP [8]. We now recall the well-known result that RSA with OAEP padding is IND-CPA secure under the RSA assumption [20]. We will show that any adversary who can break the Security game can also break the RSA assumption.

When choosing the two messages in the Security game the adversary $\mathcal{A}$ has the same available information as they do in the IND-CPA^{RSA} game. We now analyse the difference between the two games when $\mathcal{A}$ receives the challenge and makes a guess. In the Security game, $\mathcal{A}$ is provided with an additional piece of information, the challenge C, but $\mathcal{A}$ is bounded by computational time t.

If an adversary wins the Security game with a non-negligible advantage without evaluating the challenge using algorithm $\mathcal{A}_0$, then they can also break the IND-CPA^{RSA} game with the same algorithm, as C is not provided to $\mathcal{A}_0$.

Next, given the challenge C, t and the state st the sequentiality property of Theorem 5 proves that if an adversary gains a non-negligible advantage by evaluating the challenge in time less than t with $\mathcal{A}_1$ then they are contradicting the

RSW-time lock assumption. Therefore, for the adversary to gain a non-negligible advantage in the Security game, they must break either the RSW time-lock assumption, or the IND-CPA^{RSA} security game, and hence the RSA assumption. The adversary will therefore guess the correct message with probability at most $\frac{1}{2} + \mathsf{negl}(\lambda)$, and hence our TRE-IA scheme is secure according to the Security Game presented in Section 2.

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

Theorem 7. *Our TRE scheme provides the implicit authentication property.*

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

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

In our construction d is chosen at random on lines 3 - 5 of $\mathsf{TRE.Gen}$. As we are working in $\mathbb{Z}_N^*$ there is only one multiplicative inverse of d, which is the encryption key e calculated on line 9 of $\mathsf{TRE.Gen}$. To derive e from d requires knowledge of $\phi(N)$, as $e := d^{-1} \bmod \phi(N)$. In order to learn $\phi(N)$, the adversary would need to factor the RSA modulus N, which is a well-known hard problem [34, 35]. Therefore, unless the adversary can factor N, they cannot guess e with more than negligible probability. Therefore with overwhelming probability the adversary will not learn the trapdoor $\phi(N)$, and hence will not be able to derive e from d.

Using the encryption oracle $\mathcal{O}^{\mathsf{enc}}$ $\mathcal{A}$ can obtain polynomially many ciphertexts.

Recall from [8] that RSA-OAEP has ciphertext indistinguishability under chosen-plaintext attack, which guarantees indistinguishability between encryptions of messages. This property guarantees in particular that the adversary has no advantage in identifying a ciphertext that will allow them to win the IA game.

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

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

5 Performance and Integration with SecureDrop

In this section we present an evaluation of how our TRE-IA scheme performs and provide a sample of how it can integrate with SecureDrop. We evaluate the run times of the algorithms executed by the encryptor (who is the whistleblower)

and those executed by the decryptor. Our analysis compares the dispersion of run times of TRE-IA algorithms in our construction when different parameters are used for the modulus size. Based on the results of our analysis we present a number of recommendations for TRE implementations.

5.1 Performance Analysis

Our test setup consists of a virtual machine running on an Intel i7-6700K 4 GHz CPU and 24 GB of RAM. The guest OS is Ubuntu 18.04 running Python 3.8.10. The `Crypto` library 2.6.1 is used to generate random Blum integers. The code can be found in `https://github.com/nxd0main/TRE-IA`.

Our performance analysis consists of running six different trials. Each trial tests how different parameters impact the encryptor and decryptor algorithms run times on our TRE-IA scheme implementation. Trial 1 is tested against a 2048 bit modulus, with the time parameter t set to 10 million. Trial 1 is run 200 times. Each time Trial 1 is executed we record the individual run times of the TRE.Setup, TRE.Gen, TRE.Enc, TRE.Solve, and TRE.Dec algorithms. Trial $2, 3, 4, 5$ and 6 are similar to Trial 1 except they are tested against a $2560, 3072$, 3584, 4096, and 4608 bit modulus, respectively. Table 2 summarises the trial parameters and a number of key descriptive statistics based on our analysis.

Table 2. Performance analysis trial parameters and descriptive statistics for TRE scheme run time tests. $\mu^*_{\mathcal{E}}$, $\mu_{\mathcal{E}}$, $\sigma_{\mathcal{E}}$, and $\mathrm{c}_{\mathcal{E}}$ are the aggregated encryptor algorithms run time mean without the TRE.Setup algorithm, run time mean with the TRE.Setup algorithm standard deviation, and coefficient of variance, respectively. $\mu_{\mathcal{D}}$, $\sigma_{\mathcal{D}}$, $\mathrm{c}_{\mathcal{D}}$, and $\beta_{\mathcal{D}}$ are the aggregated decryptor algorithms run time mean, standard deviation, coefficient of variance, and excess kurtosis, respectively.

Trial	**Modulus [b]**	**runs**	t	ℓ	$\mu^*_{\mathcal{E}}$ **[s]**	$\mu_{\mathcal{E}}$ **[s]**	$\sigma_{\mathcal{E}}$ **[s]**	$\mathbf{c}_{\mathcal{E}}$	$\mu_{\mathcal{D}}$ **[s]**	$\sigma_{\mathcal{D}}$ **[s]**	$\mathbf{c}_{\mathcal{D}}$	$\beta_{\mathcal{D}}$
1	2048	200	110^7	50	1.14	1.33	0.13	0.095	121.08	2.01	0.017	−0.18
2	2560	200	110^7	50	2.13	4.74	1.71	0.360	166.37	2.27	0.014	0.12
3	3072	200	110^7	50	3.57	4.07	0.33	0.081	221.41	2.25	0.010	−0.36
4	3584	200	110^7	50	5.54	12.08	4.56	0.378	284.07	2.93	0.010	0.15
5	4096	200	110^7	50	8.13	9.41	0.85	0.090	354.77	3.41	0.010	1.33
6	4608	200	110^7	50	11.31	25.02	8.38	0.335	432.73	4.08	0.009	4.98

Encryptor Analysis. We first provide an analysis of the distribution of the aggregated run time of the algorithms run by the encryptor $\mathcal{E}$. For each of the 200 iterations of the six trials in Table 2, the aggregated mean run time of the encryptor algorithms is noted under column $\mu_{\mathcal{E}}$ [s]. This column is the sum of the individual run times for TRE.Setup, TRE.Gen, and TRE.Enc algorithms. In the column noted $\mu^*_{\mathcal{E}}$ [s], the aggregated mean run time of the encryptor algorithms without TRE.Setup is recorded to illustrate the overhead of this algorithm.

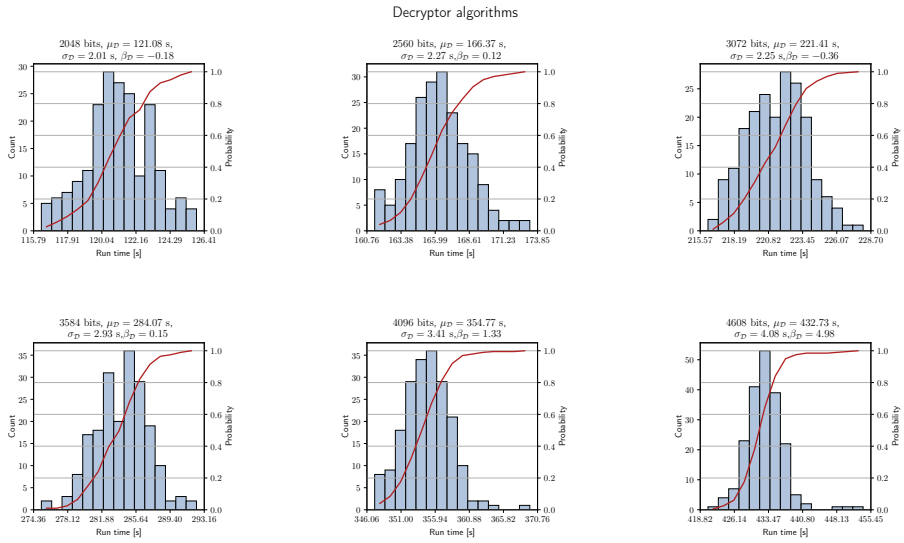

Fig. 1. Histogram with EDF for our BBS-TRE scheme Decryptor TRE.Solve and TRE.Dec algorithm run times. Heavy tailed distributions with high excess kurtosis can be seen on the 4096 bit and on the 4608 bit trial data. These trials show the probability of outliers is higher than a normal distribution. This can give a decryptor an inconsistent run time when extracting the decryption key and decrypting the ciphertext. If the selection of time parameter t is chosen to provide a specific time for the decryption of data for maximum impact, using parameters with a high probability of outlier run times is undesirable.

Table 2 also shows the standard deviation $\sigma_{\mathcal{E}}$ [s], and the coefficient of variance $c_{\mathcal{E}}$ for each trial. $c_{\mathcal{E}}$ allows us to normalise the dispersion of the run times around the mean for the different trials [5]. We see that the coefficient of variance is smaller for the $2048, 3072$, and 4096 bit modulus indicating a tigher run time around the mean. Tigher dispersion is ideal to give a more consistent experience. As the encryptor may be under duress, knowledge of run times for operation would be essential.

Recommendations: Based on the analysis of encryptor algorithm performance, we recommend choosing the 2048 bit and 3072 bit modulus sizes to ensure fast and consistent run times demonstrated by their low mean run times and low coefficients of variance respectively.

Decryptor Analysis. Next, we provide analysis of the distribution of the aggregated run times of the algorithms run by the decryptor $\mathcal{D}$. The aggregated decryptor algorithm run time equals the sum of the TRE.Solve and TRE.Dec algorithms. The mean $\mu_{\mathcal{D}}$, standard deviation $\sigma_{\mathcal{D}}$, coefficient of variation $c_{\mathcal{D}}$, and excess kurtosis $\beta_{\mathcal{D}}$ for each trial can be seen in Table 2. The coefficient of variation remains consistently low between all trials in Table 2. This is ideal

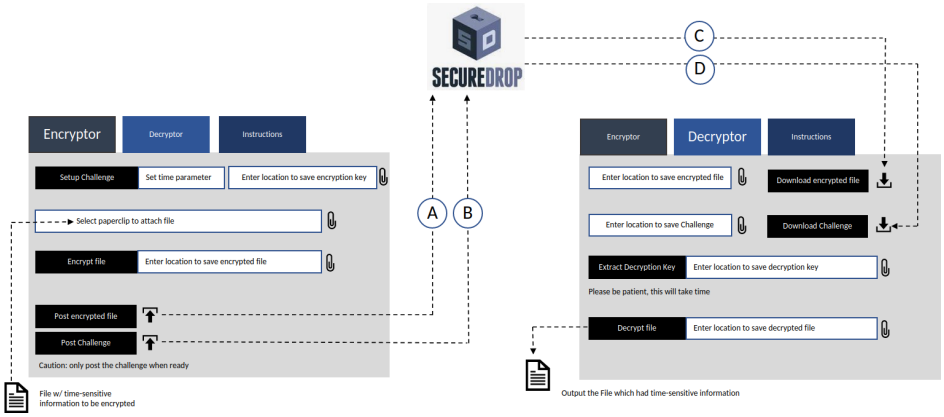

Fig. 2. SecureDrop integration with TRE-IA.

because having a low coefficient of variance for the decryptor algorithms indicates a tighter dispersion around the mean. By this measure, the decryptor algorithm run times appear to be consistent.

We also measure the dispersion properties of the decryptor run times using the excess kurtosis, denoted $\beta_{\mathcal{D}}$. Excess kurtosis provides information about the weight of the tails of a distribution [40]. Excess kurtosis also provides an indicator of the probability of outliers in the distribution. Therefore, lower excess kurtosis is ideal to provide assurance of consistent run times.

Figure 1 plots the histograms of the decryptor run times for each of the six trial types in Table 2. Each histogram also plots an empirical distribution function (EDF). Figure 1 shows us that the 2048 and 3072 bit BBS-TRE distributions show the lowest probability of outlier run times and that the 4096 bit and 4608 bit distributions show the highest probability of outlier run times.

The presence of heavy tailed distributions in our trials was unanticipated as the virtual machine resources and test parameters remained consistent. Having heavy tailed distributions with a high probability of outliers for decryptor algorithm run times when the same computational resource is provided is undesirable. For example, a decryptor extracting and decrypting the ciphertext on a 4608 bit construction would know from performance analysis that there is a 1 in 25 chance that they could take far longer than the expected mean time to extract the time-locked decryption key and decrypt the ciphertext.

Consider a concrete example extrapolated from the final histogram in Figure 1 for the 4608 bit modulus. If $\mathcal{E}$ selected the parameter t such that the expected extraction and decryption time was 7 days (168 hours), then a decryptor may take 8 hours longer than the expected run time to recover the plaintext.

Recommendations: Based on the analysis of decryptor algorithm performance, we recommend using the 2048 and 3072 bit sizes which demonstrate the lowest probability of outlier run times.

5.2 Using TRE-IA with SecureDrop

Our TRE-IA construction could augment an existing whistleblowing tool known as SecureDrop by adding the property of guaranteed delay with implicit authentication. SecureDrop is 'an open source whistleblower submission system that can securely access documents from anonymous sources' [2].

In Figure 2 the left hand box is the 'Encryptor' tab for the Whistleblower. The Setup Challenge button will run the TRE.Setup and TRE.Gen algorithms and allow the time parameter to be selected. Next there is dialogue box to securely save the encryption key e to maintain the property of IA. The dialogue box 'Encrypt file' will select the files that are required to be time-release encrypted by TRE.Enc. The whistleblower can then upload the public parameters and ciphertext at a time of their choosing, which is denoted by dashed-line A. The final button will upload the challenge to SecureDrop denoted by dashed-line B.

The right hand box is the 'Decryptor' tab for the receiving party. The decryptor can download and save the ciphertext as soon as it becomes available on SecureDrop denoted by dashed-line C. They will not be able to decrypt the ciphertext until the challenge becomes available and is downloaded, which is denoted by dashed-line D. Once the challenge is received the 'Extract Decryption Key' button will run TRE.Solve to recover d. Finally, the 'Decrypt file' will take the ciphertext, public parameters, and decryption key and run TRE.Dec to recover the original leaked file.

6 Conclusion

In this paper we introduced a variant of a delay-based primitive known as timed-release encryption with implicit authentication (TRE-IA). Implicit authentication is formally introduced with a game-based definition and we provide a concrete implementation with this property. Our implementation of a TRE-IA which we name the BBS-TRE uses the BBS CSPRNG and RSA-OAEP PKE as the building blocks. Our construction is implemented in Python and a performance evaluation against six common modulus sizes was provided. Our performance analysis allowed us to observe how the modulus size affected the run time consistency of the whistleblower and decryptor algorithms. Therefore, we were able to provide concrete recommendations for parameter selection for practical implementations of our TRE scheme. We also provided an example of how our TRE-IA scheme could be used in conjunction with the existing whistleblowing tool SecureDrop.

References

1. Edward Snowden's Motive Revealed: He Can 'Sleep at Night', https://www.nbcnews.com/feature/edward-snowden-interview/edward-snowdens-motive-revealed-he-can-sleep-night-n116851, 2014.

2. SecureDrop Whistleblower Submission System, https://securedrop.org, 2021.
3. Signal Messaging, https://signal.org/en, 2021.
4. The Tor Project, https://www.torproject.org, 2021.
5. H. Abdi. Coefficient of Variation. In *Encyclopedia of research design*, 2010.
6. E. Barker, L. Chen, A. Roginsky, A. Vassilev, R. Davis, and S. Simon. Sp 800-56b rev. 2, recommendation for pair-wise key-establishment using integer factorization cryptography. *ITL Computer Security Resource Center*, 2019.
7. M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In *Advances in Cryptology—ASIACRYPT 2000: 6th International Conference on the Theory and Application of Cryptology and Information Security Kyoto, Japan, December 3–7, 2000 Proceedings 6*, pages 531–545. Springer, 2000.
8. M. Bellare and P. Rogaway. Optimal Asymmetric Encryption. In *Advances in Cryptology EUROCRYPT '94*, 1994.
9. P. Berglez and A. Gearing. The panama and paradise papers. the rise of a global fourth estate. *International Journal of Communication*, 2018.
10. F. Berti, F. Koeune, O. Pereira, T. Peters, and F. Standaert. Ciphertext integrity with misuse and leakage: definition and efficient constructions with symmetric primitives. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pages 37–50, 2018.
11. L. Blum, M. Blum, and M. Shub. A Simple Unpredictable Pseudo-Random number Generator. In *Journal on Computing*, 1986.
12. D. Boneh, J. Bonneau, B. Bünz, and B. Fisch. Verifiable Delay Functions. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference*, 2018.
13. D. Boneh and M. Naor. Timed commitments. In *Annual international cryptology conference*. Springer, 2000.
14. J. Burdges and J. DeFeo. Delay Encryption. In *40th Annual International Conference on the Theory and Applications of Cryptographic Techniques. EUROCRYPT 2021*, 2021.
15. R. Carmichael. Note on a new number theory function. In *Bulletin of the American Mathematical Society*, 1910.
16. J. Cathalo, B. Libert, and J. Quisquater. Efficient and non-interactive timed-release encryption. In *International Conference on Information and Communications Security*. Springer, 2005.
17. 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*. Springer, 2021.
18. T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2009.
19. 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.
20. E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern. Rsa-oaep is secure under the rsa assumption. In *Annual International Cryptology Conference*. Springer, 2001.
21. J. Garside. Panama Papers: inside the Guardian's investigation into offshore secrets, https://www.theguardian.com/news/2016/apr/16/panama-papers-inside-the-guardians-investigation-into-offshore-secrets, 2016.
22. C. Gauss. *Disquisitiones Arithmeticae*. Yale University Press, 2009.
23. F. Griffin and I. Shparlinski. On the linear complexity profile of the power generator. In *IEEE Transactions on Information Theory*, 2000.

24. G. Hardy and E. Wright. *An introduction to the theory of numbers.* Oxford university press, 1979.
25. J. Katz and Y. Lindell. *Introduction to Modern Cryptography, Second Edition.* CRC Press, 2014.
26. A. Lenstra and I. Shparlinski. Selective forgery of rsa signatures with fixed-pattern padding. In *Public Key Cryptography: 5th International Workshop on Practice and Theory in Public Key Cryptosystems*, 2002.
27. M. Liedtke and J. Mattise. Leaked "pandora papers" expose how billionaires and corrupt leaders hide wealth. *Guardian (Sydney)*, 2021.
28. A. Loe, L. Medley, C. O'Connell, and E. Quaglia. Tide: A novel approach to constructing timed-release encryption. *ACISP22*, 2022.
29. W. Mao. Timed-release cryptography. In *International Workshop on Selected Areas in Cryptography.* Springer, 2001.
30. M.Girault and J. Misarsky. Selective forgery of rsa signatures using redundancy. In *Advances in Cryptology—EUROCRYPT'97: International Conference on the Theory and Application of Cryptographic Techniques*, 1997.
31. G. Miller. Riemann's Hypothesis and Tests for Primality. In *Journal of Computer and System Sciences, 13(3)*, 1976.
32. J. O'Donovan, H. Wagner, and S. Zeume. The value of offshore secrets: Evidence from the panama papers. *The Review of Financial Studies*, 2019.
33. K. Pietrzak. Simple verifiable delay functions. In *10th Innovations in Theoretical Computer Science Conference, ITCS 201*, 2019.
34. M. Rabin. Digitalized signatures and public-key functions as intractable as factorization. In *MIT/LCS/TR-212, MIT Laboratory for Computer Science*, 1979.
35. R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 1983.
36. 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.
37. W. Scheuerman. Whistleblowing as civil disobedience: The case of edward snowden. *Philosophy & Social Criticism*, 2014.
38. J. Verble. The nsa and edward snowden: surveillance in the 21st century. *Acm Sigcas Computers and Society*, 2014.
39. B. Wesolowski. Efficient Verifiable Delay Functions. In *Advances in Cryptology – EUROCRYPT 2019*, 2019.
40. P. Westfall. Kurtosis as Peakedness. In *The American Statistician*, 2014.
41. Y. Zheng. Digital signcryption or how to achieve cost(signature and encryption) cost(signature) + cost(encryption). In *International Conference of Theory Applied Cryptography Technology (CRYPTO) 97*, pages 165–179. Springer, 1997.
42. Y. Zheng. A new efficient signcryption scheme in the standard model. In *Security and Communication Networks vol. 8, no 5*, pages 703 – 878, 2015.
43. Y. Zheng and H. Imai. How to construct efficient signcryption schemes on elliptic curves. In *Proc. of IFIP SEC98 vol. 68, no. 5*, pages 227 – 233, 1998.
44. P. Zimmerman. Why I Wrote PGP, Essays on PGP. Phil Zimmermann and Associates LLC, 2013.