tlock: practical timelock encryption from threshold BLS

Nicolas Gailly[1], Kelsey Melissaris[2], Yolan Romailler[1]

[1] Protocol Labs
https://research.protocol.ai/
[2] Department of Computer Science
Aarhus University, Denmark

Abstract. We present a practical construction and implementation of timelock encryption, in which a ciphertext is guaranteed to be decryptable only after some specified time has passed. We employ an existing threshold network, the League of Entropy, implementing threshold BLS [BLS01, Bol03] in the context of Boneh and Franklin's identity-based encryption [BF01] (BF-IBE). At present this threshold network broadcasts BLS signatures over each round number, equivalent to the current time interval, and as such can be considered a decentralised key holder periodically publishing private keys for the BF-IBE where identities are the round numbers. A noticeable advantage of this scheme is that only the encryptors and decryptors are required to perform any additional cryptographic operations; the threshold network can remain unaware of these computations and does not have to change to support the scheme. We also release an open-source implementation of our scheme and a live web page that can be used in production now relying on the existing League of Entropy network acting as a distributed public randomness beacon service using threshold BLS signatures.

Keywords: Time-lock · Timelock · Timed-release · Time-lapse · Threshold · Pairing

1 Introduction

Timelock encryption (TLE) was first introduced on the Cypherpunks mailing list in 1993 by Tim May [May93], the founder of the crypto-anarchist movement, and subsequently received additional attention thanks to Rivest, Shamir and Wagner in 1996 [Riv96]. The notion of TLE can be expressed simply: ciphertexts are guaranteed to be decryptable after a specified point in time. As the name suggests TLE ciphertexts remain locked until the designated time, after which even the encryptor cannot prevent decryption. Timelock encryption has also been referred to as *Time-lapse encryption*, *Timed-release encryption* or *Timed encryption*.

Timelock encryption finds multiple applications:

- **Sealed-bid auctions** in which encrypted bids are decryptable only after the bidding period has elapsed;
- **Mitigation of preservation risk during embargo periods** e.g. for legal documents, confessions, or vulnerability reports with coordinated disclosure, by ensuring that a document can only be decrypted after a given time period;
- **Conditional transfers of assets** by encrypting private keys to a given future time, and relocating funds prior to release time should the wealth transfer be deemed unnecessary;
- **Miner extractable value (MEV) prevention mechanism** in which transactions in a blockchain are encrypted via a timelock scheme in order to prevent miners from performing MEV attacks.

1.1 Prior art

Initial approaches to practical timelock encryption were founded upon *proof-of-work* systems, with security guaranteed under the assumption that some puzzle requires a certain amount of sequential computation, and therefore time, to solve. The first to take this approach, introducing the notion of time-lock puzzles, were Rivest, Shamir and Wagner [Riv96], noting that:

> There are two natural approaches to implementing timed release crypto:
>
> – Use "time-lock puzzles"–computational problems that cannot be solved without running a computer continuously for at least a certain amount of time.
> – Use trusted agents who promise not to reveal certain information until a specified date.
>
> Using trusted agents has the obvious problem of ensuring that the agents are trustworthy. Secret sharing approaches can be used to alleviate this concern.

A theoretical formalisation of time-lock puzzles can be found in [BGJ+15]. Time-lock puzzles are inherently founded in proof of work and are therefore highly sensitive to unpredictable advances in both hardware and algorithms. For instance the LCS time-lock puzzle released in 1999 by Ron Rivest was anticipated to remain secure for 35 years, but was successfully solved *fifteen years early* in 2019 via two independent methods [csa19]: once by simply running sequential operations on a single core of a modern CPU for 3.5 years, and again by running the computation for only 2 months on dedicated FPGA hardware.

Rabin and Thorpe take the alternative approach and propose the idea of relying on Pedersen distributed key generation, Feldman verifiable threshold secret sharing, and ElGamal encryption [RT06]. Therein multiple parties implement a "Time-Lapse Cryptography Service" by publishing public keys and then releasing the related private keys at given times. To the best of our knowledge such a Time-Lapse Cryptography Service has never been implemented and deployed in practice. Moreover this setting requires explicit action by a threshold of members to compute the decryption operation. In other words, the work required by the network is linear in the number of ciphertexts to decrypt.

In [LJKW18] a timelock scheme is built on top of the concept of computational reference clocks using extractable Witness encryption, and a practical instantiation relying on Bitcoin is presented. However, this scheme relies on multilinear maps which have not yet been securely constructed.

In [CDK+21] IBE and Witness encryption are again used to achieve a novel notion of "Encryption to the Current Winner" (ECW) where the receiver of an encrypted message is determined by the current state of a blockchain and is not yet known at the time of encryption. Ultimately this scheme is then transformed to achieve a different notion of "Encryption to the Future" (EtF) towards parties selected at arbitrary points in the future, unlike classical timelock or Timelapse encryption where any party knowing the ciphertext can decrypt once the specified time has passed. In direct contrast to our timelock definition which permits anyone to decrypt after the designated time, in EtF only some party with certain rights (the future winner) not yet determined in the present can decrypt ciphertexts. Notice that the setting is also quite different between both works, assuming dynamic committees with the strong assumption that "YOSO"—You Only Speak Once. This has not yet been implemented nor deployed in practice to the extent of our knowledge.

Threshold networks have also been used as an encryption "recipient." For example, Ferveo [Ano21] and Shutter [shu22] are systems where users encrypt their transactions for threshold networks (which can also be the validator set of a blockchain). Once encrypted transactions are included in a block, they get decrypted by the network, thereby preventing front-running. In comparison, our scheme does not rely on the threshold network to perform an operation *per ciphertext*, rather simply to emit one value per round and the decryption happens publicly. Anyone can compute a valid decryption of a ciphertext encrypted for this round.

1.2 Our contributions

We present, implement and benchmark a secure timelock encryption (TLE) scheme from the League of Entropy (LoE) [LoE20], a production-ready randomness beacon currently deployed over drand nodes [dra]. Our timelock encryption scheme permits encryption under existing public parameters, previously generated and utilized by the existing threshold network, such that ciphertexts can be decrypted by any party given only certain time-related information already broadcast by that network. We provide the first implementation and benchmarks for this approach.

League of Entropy: The LoE is a threshold network in which each member holds a share of an unknown secret, that was generated via a Distributed Key Generation [DKG] procedure. Periodically the network computes and broadcasts a threshold BLS signature over the *round number* associated with the current timestamp. Verification simply involves the round number, the network public key and the signature. The LoE has been in production since 2020, at the time of writing is composed of 23 nodes operated by different companies, universities & foundations), and is already used by large networks, e.g. Filecoin. The LoE has, since deployment, had 100% uptime.

Identity Based Encryption: IBE is an encryption paradigm which replaces public keys with public identity strings, e.g. emails, names, addresses, etc. At any point some trusted party, holding a master secret key, can compute and distribute the corresponding secret key to enable decryption. We exploit an equivalence between IBE and TLE to construct our scheme [CHKO08].

Timelock Encryption: The key insight to bind these two mechanisms is to view a BLS signature on an identity as the corresponding private key, as in Boneh and Franklin's IBE [BF01]. Timelock encryption is then achieved by employing their IBE to encrypt under the round number as the public identity string. As that round number is associated to a unique timestamp, this is equivalent to encrypting under the associated time. When the BLS signature on that round number is released by the LoE then anyone can use that signature to decrypt messages. This approach has several advantages:

- **Based on a proven, production-ready system:** The LoE has been consistently and successfully running for over two years. Every 30 seconds the LoE broadcasts randomness, without any downtime. New nodes are regularly onboarded to further increase the safety and liveness guarantees. The security of our scheme relies only on the existing, proven, production-ready system (beyond any computational assumptions, of course).
- **No cooperation is required from the network**: the LoE is unaware of any encryption/decryption based on the network. Only the clients will need to perform cryptographic work to encrypt and decrypt, and anyone can build a system satisfying their own specific needs (distribution of the ciphertexts, etc.)

– **Ciphertexts are publicly decryptable**: any party, internal or external to the existing system, can decrypt a message as soon as they access the corresponding signature. Decryption does not require any participation by the original author after the ciphertext has been released. In fact, we do not loose any functionality; to modify this publicly decryptable scheme into one which designates a specific decrypter, the encrypter needs only to encrypt a standard public key encryption ciphertext.

2 Preliminaries

2.1 Groups & Computational Assumptions

Bilinear Maps. A type-III bilinear group is described via $(\mathrm{e}, p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, G_1, G_2, G_T)$ where $\mathbb{G}_1, \mathbb{G}_2$ are additive groups with generators G_1, G_2 respectively and $\mathbb{G}_T$ is a multiplicative group with generator G_T, each of prime order p. The deterministic pairing map $\mathrm{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is *bilinear* meaning $\forall (a, b) \in \left(\mathbb{Z}_p^*\right)^2$: $\mathrm{e}(aG_1, bG_2) = \mathrm{e}(G_1, G_2)^{ab}$, and *non-degenerate* meaning that $\mathrm{e}(G, H) = 1$ implies either G or H are 0 in their respective groups. We also require *computability* meaning that membership testing, group operations, and the pairing are efficiently computable and all elements are represented in linear size. Additionally, type-III bilinear groups require that there is no efficiently computable mapping between $\mathbb{G}_1$ and $\mathbb{G}_2$.

Decisional, Computational and Gap Diffie-Hellman. Consider a finite cyclic group $\mathbb{G}$ of prime order $q \in \mathbb{P}$ with generator g. The Computational Diffie-Hellman (CDH) problem is to compute, for random $h, h' \in \mathbb{G}$ the element $\bar{h} = g^{\log_g(h) \cdot \log_g(h')}$. The Decisional Diffie-Hellman (DDH) problem is, given the h, h', to distinguish such an element $\bar{h}$ from a random group element.

The CDH (resp. DDH) assumption in $\mathbb{G}$ is that there does not exist a probabilistic polynomial time algorithm solving CDH (resp. DDH) in $\mathbb{G}$ that achieves a probability of success non-negligibly greater than an algorithm which merely outputs random guesses. A group $\mathbb{G}$ is called a Gap Diffie-Hellman (GDH) group if there exists a probabilistic polynomial time algorithm $\mathcal{V}_{\mathbb{G},g}$ solving the DDH problem, but the CDH assumption holds.[3]

(Co-)Bilinear Diffie-Hellman. Let $\mathsf{bg} = (\mathrm{e}, p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, G_1, G_2, G_T)$ be a description of a type-III bilinear group. For random $a, b, c \in \mathbb{Z}_q^*$ and $(P, Q) \in \mathbb{G}_1 \times \mathbb{G}_2$ the Co-Bilinear Diffie-Hellman (CBDH) problem is to compute the element $\mathrm{e}(P, Q)^{abc}$ given the elements (P, aP, bP, Q, aQ, cQ). The CBDH assumption in bg is that no probabilistic polynomial time algorithm $\mathcal{A}$ can solve the CBDH problem in bg.

2.2 Threshold BLS Signatures

Boneh-Lynn-Shacham (BLS) signatures [BLS01] are digital signatures provably secure under the GDH assumption on the underlying group. The BLS signature admits a (t, n) threshold construction [Bol03] in which the signing key is jointly generated by n nodes such that a signature can be interactively generated by at least t of those nodes. We recall the construction below.

Threshold Key Generation is the distributed key generation protocol (DKG) for discrete logarithm based systems by Gennaro et al. [GJKR99]. Each node i locally outputs their share s_i of

[3] We also refer to GDH as a computational assumption, corresponding to the CDH assumption in a group that provably admits an efficient algorithm for DDH.

the joint private key $s \in \mathbb{Z}_p^*$, and we assume that the public keys $P_i = s_i G_1 \in \mathbb{G}_1$ are public. The algorithm outputs a joint public key $P = sG_1 \in \mathbb{G}_1$. The sharing is such that the secret s can be constructed by any subset of nodes $I \subseteq N = \{1, \ldots, n\}$ such that $|I| \geq t$ via Lagrange interpolation [lag]:

$$P = sG_1 = \sum_{i \in I} \big(s_i L_i(0)\big) G_1 \in \mathbb{G}_1$$

where $L_i(x)$ is the i-th Lagrange polynomial defined over the set I.

Threshold Signing Recall that the BLS signature of a message M is $\pi = sH(M)$ for $H : \{0,1\}^* \to \mathbb{G}_2$ a secure hash to curve function onto $\mathbb{G}_2$. To generate this signature distributively, each participating node computes and broadcasts their share of the signature $\pi_\rho^{(i)} = s_i H(M)$ on the message M. Signatures can then be reconstructed by any node (optionally) after verifying the partial signatures under the public key of the associated node i. The final signature under the network public key P can then be computed from any t verifying partial signatures:

$$\pi = \sum_{i \in I} \big(\pi_\rho^{(i)} L_i(0)\big) = sH(M) \in \mathbb{G}_2$$

where $L_i(x)$ is the i-th Lagrange polynomial defined over the set I

Security of standard BLS is reducible to the gap diffie-hellman assumption on the underlying group. Threshold-BLS provably satisfies the following two desirable properties: (1) *existential unforgeability* meaning that given oracle access to both key generation and signatures the adversary cannot produce a verifying signature on a new message, and (2) *robustness* meaning that an adversary cannot prevent key generation or signing. While unforgeability of (t, n)-BLS can be obtained for any $t < n$, robustness (which is required in our application) is proven against malicious probabilistic polynomial time adversaries corrupting less than $t \leq \frac{n}{2}$ nodes.

2.3 Identity-Based Encryption

Identity-based encryption (IBE) [BF01] is a variant of public key encryption in which public "identity" strings function as public keys. With an IBE scheme one can encrypt a message M to an identity ID which can be decrypted with a related decryption key $\mathsf{sk}_{\mathsf{ID}}$ issued by a trusted party to the owner of that identity.

Definition 1 (Identity-Based Encryption). *An identity-based encryption scheme $\mathcal{E}_{\mathsf{IBE}}$ is a tuple of four algorithms:*

$\mathsf{SetUp}(1^\lambda) \to (\mathsf{pp}, \mathsf{msk})$**:** *Setup is a randomized algorithm which takes as input the security parameter λ and outputs public parameters pp and a master secret key msk.*

$\mathsf{Extract}(\mathsf{pp}, \mathsf{msk}, \mathsf{ID}) \to \mathsf{sk}_{\mathsf{ID}}$**:** *extract is a randomized algorithm which takes as input the public parameters pp, the master secret key msk and an identity string $\mathsf{ID} \in 0, 1^*$, and output a private decryption key $\mathsf{sk}_{\mathsf{ID}}$.*

$\mathsf{Encrypt}(\mathsf{pp}, \mathsf{ID}, M) \to C$**:** *encryption is a randomized algorithm which takes as input the public parameters pp, an identity string ID and a message M and outputs a ciphertext C.*

$\mathsf{Decrypt}(\mathsf{pp}, \mathsf{sk}_{\mathsf{ID}}, C) \to \{M', \perp\}$**:** *decryption takes as input the public parameters pp, a private key $\mathsf{sk}_{\mathsf{ID}}$ and a ciphertext C and outputs either a message M' or an error $\perp$.*

Security for IBE is generally defined as either semantic security under chosen plaintext or chosen ciphertext attack, called CPA and CCA respectively. The corresponding security games closely resemble those for public key encryption; an adversary against CPA is asked to break indistinguishability given adaptive access to Extract, whereas an adversary against CCA is additionally provided adaptive access to Decrypt.

Boneh and Franklin provide schemes satisfying both security requirements, reducing security to the co-BDH assumption for type-III bilinear groups. We only note that the BF-IBE Extract algorithm essentially outputs a BLS signature on the hash of the public identity string, and therefore the master secret key is a BLS secret key. We omit the exact construction as our tTLB scheme presented in Section 5 is equivalent to the Boneh-Franklin CCA-secure IBE scheme (BF-IBE) under three modifications: (1) our scheme is instantiated over *type-III bilinear groups*, (2) we *flip the pairing* and switch membership of the public key to $\mathbb{G}_1$, and (3) our *master secret key is decentralized* via threshold-BLS.

3 Timelock Encryption

Agent-based TLE. As mentioned, Timelock Encryption (TLE) is a cryptographic primitive with which ciphertexts can only be decrypted after the specified time. At present TLE comes in two distinct flavors: agent-based, in which a trusted agent is employed to guarantee decryptability of ciphertexts, and puzzle-based, in which ciphertexts are decryptable with the solution of a puzzle which is assumed to take some amount of time to solve. We adopt the *agent-based* approach to TLE – Definition 2 assumes a trusted agent to periodically publish time-related decryption keys at each time interval.

Publicly Decryptable TLE. Traditionally TLE ciphertexts are encrypted to both a time ρ and the public key pk of a designated decryptor, such that decryption requires both the decryptor's secret key sk and the time-related key π_ρ. In contrast our TLE definition stipulates that ciphertexts are decryptable by any party given only π_ρ. We refer to the former as *designated-decryptor* TLE and the latter as *publicly decryptable* TLE. As this work focuses entirely on agent-based publicly-decryptable TLE these qualifiers are often dropped, except when necessary.

Agent-based Publicly Decryptable TLE Definition. Definition 2 specifies the algorithms which constitute a TLE scheme. First, setup is performed by the agent via the SetUp algorithm, which outputs an agent secret key and public parameters. At each time interval ρ the agent runs the RoundKey algorithm and publishes the output round-related decryption key π_ρ. Encryption and decryption can be run by any party, as outlined.

Definition 2 (Timelock Encryption (TLE) Scheme). *A Timelock Encryption scheme $\mathcal{T}$ is a tuple of polynomial-time algorithms:*

$\mathsf{SetUp}(1^\lambda) \to (\mathsf{pp}, \mathsf{ask})$**:** *the probabilistic setup algorithm takes as input the security parameter λ and computes the public parameters* pp *and master secret key* ask*.*

$\mathsf{RoundKey}(\mathsf{pp}, \mathsf{ask}, \rho) \to \pi_\rho$**:** *the round key algorithm takes as input the public parameters* pp*, the master secret key* ask *and the round number ρ and outputs the round key π_ρ.*

$\mathsf{Encrypt}(\mathsf{pp}, \rho, M) \to \mathsf{ct}_\rho$**:** *the probabilistic encryption algorithm takes as input the public parameters* pp*, the round number ρ and a message M and outputs a ciphertext ct_ρ.*

$\mathsf{Decrypt}(\mathsf{pp}, \pi_\rho, \mathsf{ct}_\rho) \to M'$**:** *the deterministic decryption algorithm takes as input the public parameters* pp*, the round number ρ and a ciphertext ct_ρ and outputs either a message M'.*

Correctness. Perfect TLE correctness requires that for every pair $(\mathsf{pp}, \mathsf{ask})$ output by SetUp:

$$\mathsf{Decrypt}(\mathsf{RoundKey}(\mathsf{ask}, \rho), \mathsf{Encrypt}(\rho, M)) = M$$

Indistinguishability. A variety of security models for TLE were analyzed in [CV19], the most relevant being semantic security against adaptive chosen-ciphertext and chosen-plaintext attacks, denoted TLE-CCA and TLE-CPA respectively. In both attacks an adversary is given a ciphertext encrypted under an adversarially chosen round and must determine which of two adversarially chosen messages was encrypted. Conventionally the distinction between CPA and CCA security hinges upon oracle access, as is the case for TLE; a TLE-CPA adversary is permitted adaptive access to an oracle outputting round keys for arbitrary times RK, whereas an adversary against TLE-CCA is additionally granted adaptive access to a decryption oracle Dec.

Time. One primary obstacle to modeling time-related cryptographic primitives is the formalization of time. We manage to circumvent this problem entirely. In our TLE definition each time interval corresponds to an incremental round number ρ, and in our indistinguishability security experiments this abstract representation of time is decoupled from "real" time by neglecting to enforce that the oracle $\mathcal{O}_{\mathsf{RK}}$ be queried sequentially. Essentially by providing an adversary with keys corresponding to arbitrary rounds we simultaneously eliminate the structure that correlates with time and strengthen our security guarantees. We use the terms round and time interchangeably throughout the text.

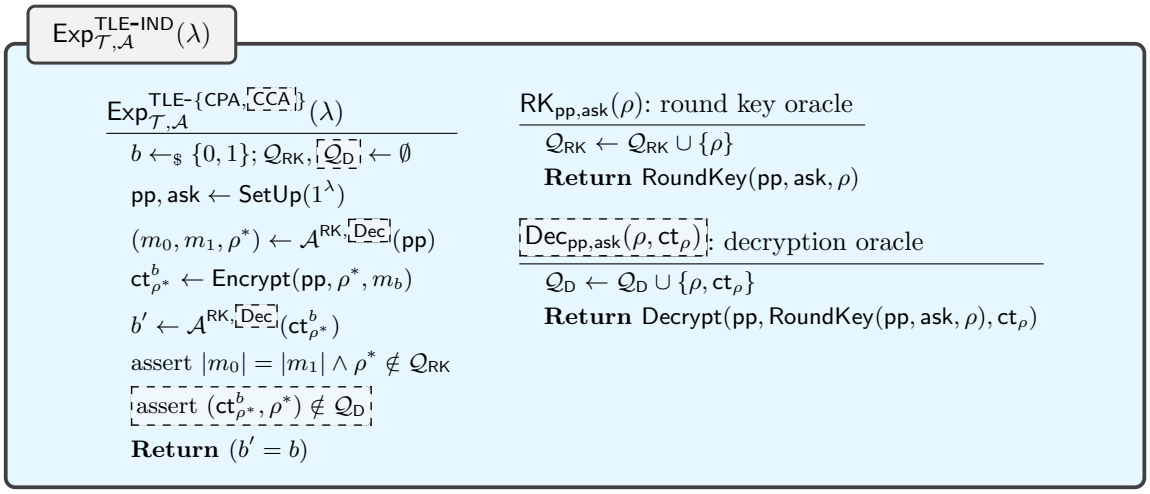

Fig. 1: The timelock encryption indistinguishability security experiments.

Definition 3 (TLE Indistinguishability (TLE-$\star$-security)). *Let $\mathcal{T}$ be a timelock encryption scheme satisfying Definition 2 and let the security experiment $\mathsf{Exp}^{\mathsf{TLE\text{-}IND}}_{\mathcal{T},\mathcal{A}}(\lambda)$ be as in Figure 1. For $\star \in \{\mathsf{CPA}, \mathsf{CCA}\}$ we say that $\mathcal{T}$ is TLE-$\star$ secure if for all PPT adversaries $\mathcal{A}$ the advantage of $\mathcal{A}$ as defined below is negligible in λ:*

$$\mathbf{Adv}^{\mathsf{TLE}\text{-}\star}_{\mathcal{T},\mathcal{A}}(\lambda) := \left| \frac{1}{2} - \mathsf{Exp}^{\mathsf{TLE}\text{-}\star}_{\mathcal{T},\mathcal{A}}(\lambda) \right|$$

4 Threshold Timelock Encryption: Definition & Model

Threshold-TLE. We adapt Definition 2 of TLE to threshold timelock encryption (tTLE), in which a threshold network acts as the trusted agent. Definition 4 employs threshold cryptography to decentralize the algorithms of SetUp and RoundKey into interactive protocols tSetUp and tRoundKey to be run by a set of parties.

Model. We assume a set of n parties $\mathcal{P} = \{P_1, \ldots, P_n\}$ functioning as the decentralized agent, equipped with a complete network of peer-to-peer channels. We assume a partially synchronous setting, in which messages for each protocol round are received within a specified time but the adversary is permitted to speak last.

Notation. A set of parties $P_1, \ldots, P_n$ with private inputs $x_1, \ldots, x_n$ and common input x executing protocol Π with transcript trans, and terminating with both private local honest party outputs $y_1, \ldots, y_n$ and global public output Y is written:

$$(\mathsf{trans}, Y, (y_1, \ldots, y_n)) \leftarrow \Pi\langle P_1(x_1), \ldots P_n(x_n), \mathcal{A}\rangle(x)$$

Threshold Timelock Encryption Definition. Definition 4 specifies the protocols and algorithms which constitute a tTLE scheme. Here, both setup and round key are interactive protocols between stateful parties which essentially share the TLE agent secret key. Encryption and decryption can be run by any party, independent of their participation in setup.

Definition 4 (Threshold Timelock Encryption (tTLE) Scheme). *A Threshold Timelock Encryption scheme $\mathcal{T}$ is a pair of interactive algorithms and a pair of polynomial-time algorithms:*

- Π-$\mathsf{tSetUp}(1^\lambda, \mathsf{th}) \to (\mathsf{pp}, \mathsf{mpk})$**:** *the interactive setup protocol is executed by a set of n parties $\mathcal{P}$ which as input the security parameter and the threshold* th *and publicly outputs parameters* pp *and a master public key* mpk*.*
- Π-$\mathsf{tRoundKey}(\mathsf{pp}, \rho) \to \pi_\rho$**:** *the interactive round key protocol is executed by a subset of parties $\mathcal{P}$ which takes as input the public parameters* pp*, the round number ρ and publicly outputs the round key π_ρ.*
- $\mathsf{Encrypt}(\mathsf{pp}, \rho, M) \to \mathsf{ct}_\rho$**:** *the non-interactive encryption algorithm takes as input the public parameters* pp*, the round number ρ and a message M and outputs a ciphertext ct_ρ.*
- $\mathsf{Decrypt}(\mathsf{pp}, \pi_\rho, \mathsf{ct}_\rho) \to M'$**:** *the non-interactive decryption algorithm takes as input the public parameters* pp*, the round key π_ρ and a ciphertext ct_ρ and outputs a message M'.*

Correctness & Robustness. Definition 5 outlines the robustness requirement for tTLE, essentially stating that an adversary corrupting at most t parties can neither prevent setup nor round key computation. This requirement is especially useful for the tTLE primitive as a threshold agent neglecting to compute and output round keys can prevent any and all decryption arbitrarily.

Definition 5 (tTLE Robustness). *Let $\mathcal{T}$ be a threshold timelock encryption scheme as in Definition 4, where the agent is realized by a set of n parties $\mathcal{P}$. We say that $\mathcal{T}$ is robust if for all $n = n(\lambda)$, for all thresholds $t < \lfloor \frac{n}{2} \rfloor$ and for all malicious polynomial time adversaries $\mathcal{A}$ statically corrupting at most t parties the protocols* tSetUp *and* tRoundKey *complete successfully.*

Then, perfect correctness for tTLE requires that for all $\mathsf{pp}, \mathsf{mpk}$ output by the setup protocol and all decryption keys π_ρ output by the round key protocol the following holds:

$$\mathsf{Decrypt}(\mathsf{pp}, \pi_\rho, \mathsf{Encrypt}(\mathsf{pp}, \rho, M)) = M$$

Indistinguishability. The tTLE indistinguishability experiments are similar to those for TLE. The adversary is first permitted to corrupt a subset of the parties $C \subseteq \mathcal{P}$, then plays these parties during an execution of setup thereby jointly generating the public parameters and the master public key. The modifications to the round key and decryption oracles are further described below.

Interactive Round Key Oracle. Security is defined against an adversary granted access to the stateful threshold round key oracle tRK. This oracle engages in polynomially many, potentially interleaved, executions of the round key protocol with the adversary, playing the queried subset of honest parties. The protocol executions can exist in parallel and are identified via an adversarially chosen session identifier. An initial query of the form (sid, T, ρ) merely begins the protocol execution. The adversary can behave arbitrarily during the executions, sending messages on behalf of any corrupted party.

Interactive Setup and the Decryption Oracle. Robustness guarantees that an adversary can neither prevent setup nor round key computation. In light of the robustness requirement we observe that such an adversary cannot prevent setup in the indistinguishability game from terminating successfully. Additionally, with access to the local outputs of the honest parties during setup the decryption oracle can exclusively play honest parties in a successful execution of the round key protocol to facilitate decryption.[4] Therefore, we can expect that these protocols terminate successfully during the experiment in Figure 2, assuming that the adversary has corrupted no more than the threshold number of parties.

Definition 6 (tTLE Indistinguishability (tTLE-$\star$-security)). *Let $\mathcal{T}$ be a threshold timelock encryption scheme as in Definition 4 and let the security experiment $\mathsf{Exp}^{\mathsf{tTLE\text{-}IND}}_{\mathcal{T},\mathcal{P},\mathcal{A}}(\lambda, t)$ be as in Figure 2. For $\star \in \{\mathsf{CPA}, \mathsf{CCA}\}$ we say that $\mathcal{T}$ is tTLE-$\star$ secure if $\mathcal{T}$ for all $|\mathcal{P}| = n(\lambda)$, for all thresholds $t < \lfloor \frac{n}{2} \rfloor$ and for all polynomial-time static malicious adversaries corrupting at most t parties the advantage of the adversary as defined below negligible in λ:*

$$\mathbf{Adv}^{\mathsf{tTLE}\text{-}\star}_{\mathcal{T},\mathcal{P},\mathcal{A}}(\lambda, t) := \left| \frac{1}{2} - \mathsf{Exp}^{\mathsf{tTLE}\text{-}\star}_{\mathcal{T},\mathcal{P},\mathcal{A}}(\lambda, t) \right|$$

Definition 7 (tTLE Security). *Let $\mathcal{T}$ be a threshold timelock encryption scheme as in Definition 4. We say that $\mathcal{T}$ is secure if $\mathcal{T}$ satisfies correctness, robustness and indistinguishability.*

5 Our Threshold Timelock Encryption Scheme

Our threshold timelock encryption scheme is a direct application of Boneh and Franklin's IBE [BF01] with the following key differences:

- We instantiate BF-IBE over type-III bilinear groups, as described in section 6 of [BF01].
- The trusted third party, or the agent, is replaced with a threshold network implementing BLS [Bol03].
- The network public key is an element of $\mathbb{G}_1$ (as opposed to $\mathbb{G}_2$), with membership of every other element flipped correspondingly.

[4] We implicitly assume that the oracles have access to the state of every honest party simulated in the experiment.

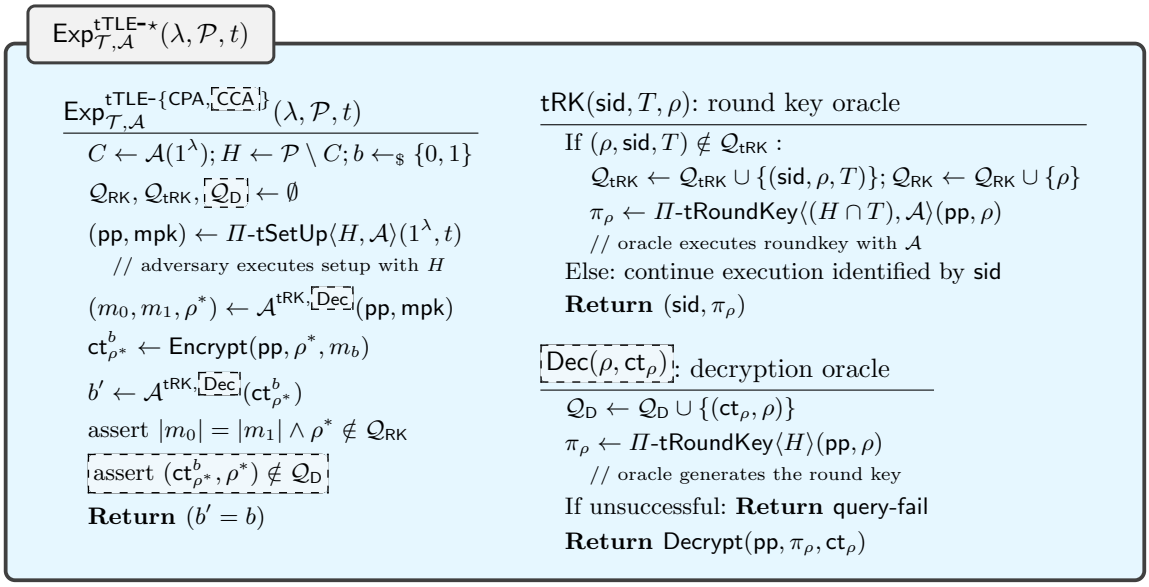

Fig. 2: The threshold timelock encryption indistinguishability security experiments.

5.1 The tTLE Construction

The Set Up & Round Key Protocols. We assume access to a type-III bilinear group $(\mathrm{e},\mathbb{G}_1,\mathbb{G}_2,p)$ with security parameter λ, along with four cryptographically secure hash functions:[5]

$$H_1 : \{0,1\}^* \to \mathbb{G}_2$$
$$H_2 : \mathbb{G}_T \to \{0,1\}^\ell$$
$$H_3 : \{0,1\}^\ell \times \{0,1\}^\ell \to \mathbb{Z}_p^*$$
$$H_4 : \{0,1\}^\ell \to \{0,1\}^\ell$$

Generators for $\mathbb{G}_1$ and $\mathbb{G}_2$ are sampled as G_1, G_2 respectively.

Π-$\mathsf{tSetUp}(1^\lambda,t)$: As opposed to sampling an agent secret key at random from $\mathbb{Z}_p^*$ and setting the public key to $P = sG_1$ as in BF-IBE the n parties $\mathcal{P}$ run the (t,n)-threshold distributed key generation protocol of Gennaro et al. [GJKR99]. After setup each of the n parties locally output their private share s_i of the agent secret key s. The public output is the network public key $P = sG_1 \in \mathbb{G}_1$ and each party's public key $P_i = s_iG_1$. The public parameters are set to the following:

$$\mathsf{pp} = ((\mathrm{e},\mathbb{G}_1,\mathbb{G}_2,p),(G_1,G_2,H,P),(t,n),\ell,(H_1,H_2,H_3,H_4))$$

Π-$\mathsf{tRoundKey}(\mathsf{pp},\rho)$: the round key protocol is then the threshold BLS signing algorithm, using the keys generated at setup. Each party in any qualified subset $I \subseteq N = \{1,\ldots,n\}$ with combined set of shares $\mathcal{S}' = \{s_i | i \in I\}$ locally computes a signature on the round number ρ as $\pi_\rho^{(i)} = s_iH_1(\rho)$. Each

[5] "Cryptographically secure" informally denotes collision, preimage and second-preimage resistant hash functions.

partial signature can be verified against the public key of the party and aggregated via Lagrange interpolation, as explained in 2.2.

The Encryption & Decryption Algorithms. The encryption of a message $M \in \{0,1\}^\ell$ to round number ρ is done via Algorithm 1, and decryption is done via Algorithm 2 Given a ciphertext ct_ρ and the round key π_ρ anyone, even parties not in $\mathcal{P}$, can decrypt via Algorithm 2. Encryption and decryption are exactly the CCA-secure variant of BF-IBE. Informally, a ciphertext ct_ρ is composed of U the ephemeral public key for the encrypter, W a one time pad encryption of the message M, and V a commitment to (the preimage of) the one time pad with decommitment information r.

Towards intuition, one can consider the computation of element $\mathsf{PK}_\rho = \mathrm{e}(P, H_1(\rho))^r$ in encryption step 3 of Algorithm 1 and $\mathrm{e}(U, \pi_\rho)$ in decryption step 3 of Algorithm 2 as two distinct ways to compute the same *shared secret* $\mathrm{e}(G_1, H_1(\rho))^{rs}$ which can be used to unpad the random σ.

Algorithm 1 Encryption

1: **procedure** $\mathsf{Enc}(\mathsf{pp}, \rho, M)$
2: Parse $\mathsf{pp} \rightarrow \big(\mathsf{bg}, P, H = (H_1, H_2, H_3, H_4)\big)$
3: $\mathsf{PK}_\rho \leftarrow \mathrm{e}(P, H_1(\rho))$ ▷ round public key
4: $\sigma \leftarrow_\$ \{0,1\}^\ell$ ▷ nonce
5: $r \leftarrow H_3(\sigma, M)$
6: $U \leftarrow rG_1$ ▷ ephemeral public key
7: $V \leftarrow \sigma \oplus H_2\big((\mathsf{PK}_\rho)^r\big)$ ▷ hiding commitment to nonce σ
8: $W \leftarrow M \oplus H_4(\sigma)$ ▷ one-time-pad
9: **return** $\big(\mathsf{ct} = (U, V, W), \tau = r\big)$
10: **end procedure**

Algorithm 2 Decryption

1: **procedure** $\mathsf{Dec}(\mathsf{pp}, \rho, \pi_\rho, \mathsf{ct}_\rho)$
2: Parse $\mathsf{ct}_\rho \rightarrow (U, V, W)$
3: $\sigma' \leftarrow V \oplus H_2(\mathrm{e}(U, \pi_\rho))$
4: $M' \leftarrow W \oplus H_4(\sigma')$
5: $r \leftarrow H_3(\sigma', M)$
6: **if** $U = rG_1$ **then**
7: **return** M'
8: **else**
9: **return** $\perp$
10: **end if**
11: **end procedure**

Correctness & Security. We demonstrate the correctness of the scheme and claim to satisfy security. The robustness of our scheme is inherited from that of threshold BLS and CCA security is inherited from the underlying IBE. A full proof will appear in the final version of the paper.

Computation of σ:

$$\begin{aligned}\sigma &= V \oplus H_2(\mathrm{e}(U, \pi_\rho)) \\ &= \sigma \oplus H_2((\mathsf{PK}_\rho)^r) \oplus H_2(\mathrm{e}(rG_1, sH_1(\rho))) \\ &= \sigma \oplus H_2(\mathrm{e}(G_1, H_1(\rho))^{rs}) \oplus H_2(\mathrm{e}(G_1, H_1(\rho))^{rs}) \\ &= \sigma\end{aligned}$$

Computation of M:

$$M = W \oplus H_4(\sigma) = M \oplus H_4(\sigma) \oplus H_4(\sigma) = M$$

CCA Validity check:

$$U = rG_1 = H_3(\sigma, M)G_1 = U$$

Theorem 1 (tTLE-CCA Security). *The tTLE scheme* $\mathcal{T} = \{\Pi\text{-}\mathsf{tSetUp}, \Pi\text{-}\mathsf{RoundKey}, \mathsf{Enc}, \mathsf{Dec}\}$ *specified above achieves tTLE security in the random oracle model for all polynomial-sized sets of parties* $\mathcal{P}$ *against static malicious polynomial-time adversaries corrupting at most* $t < \left\lfloor \frac{|\mathcal{P}|}{2} \right\rfloor$ *parties under the Gap Diffie-Hellman and co-Bilinear Diffie-Hellman assumptions.*

5.2 Optimizations

Precomputations for encryption Note that the "round public key" for round ρ, PK_ρ, is the same for all ciphertexts encrypted towards ρ, and requires to compute a pairing operation $\mathrm{e}(P, H_1(\rho))$. This is quite costly and can hurt using this scheme at scale. On constrained devices, or on blockchain systems, this step can be optimised by precomputing the round public keys of epochs.

Preprocessing Note that the first step of the decryption neither needs to be done online nor verified. We take steps towards preventing *malleability attacks*; the following modifications prevent an attacker from manipulating a ciphertext to change the decrypted message. In this section we do not modify encryption, only decryption.

We first introduce explicit roles to distinguish between the relevant actors:

1. encrypter: the party running the encryption algorithm
2. helper: a party running precomputation, i.e. expensive operations to aid with decryption
3. decrypter: the party running the decryption algorithm, potentially on-chain

Signature Embedding In this model the *encrypter* will prefix the message M with a signature $\pi_M = Sig(M)$ from a secure signature scheme satisfying integrity and authenticity, and instead encrypts the resulting M':

$$M' = \pi_M \mid\mid M$$

The *encrypter* will also *sign* the outer ciphertext. In the context of an on-chain transaction, the outer signature is done by signing the transaction.

Precomputations for decryption Once the *helper* has access to the information π_ρ associated to the round ρ it precomputes the following for each encrypted transactions related to ρ:

$$\sigma_i = V \oplus H_2(\mathrm{e}(U_i, \pi_\rho))$$

and submits these precomputations to the *decrypter*. In the context of blockchain, the helper basically submits the preprocessed elements in batch.

Decryption and fault handling The *decrypter* (the block verification/execution layer) decrypts all the messages with a valid *outer* signature:

$$M_i' = W_i \oplus \sigma_i$$

and *verifies* if the embedded signature π_M is valid. Note that here only the computationally cheap XOR operation is necessary.

If this signature verification passes, the decrypter outputs the message M_i. If signature verification fails, the decrypter must determine whether the *encrypter* or the *helper* is dishonest, i.e. whether the signature or the σ_i given is incorrect. First, the decrypter regenerates the random mask value r_i and performs the standard final decryption check:

$$U_i = H_4(\sigma_i, M_i')G_1$$

If the check passes: the encryption steps are correct but the *encrypter* has inserted an invalid signature into the message and, in this case, the decryption should be discarded. If the check fails: the ciphertext has been tampered with, i.e. the *helper* has been misbehaving. In this case the *decrypter* has to run the full decryption algorithm alone from the original ciphertext. Note that in the context of blockchain good behaviour can be incentivised by, for example, slashing any block producer that includes invalid σ_i.

5.3 Generalisation to public witness encryption

While our system relies on the notion of "rounds" serving as identities in an IBE scheme, a generalisation in which any other message is signed by the threshold network is possible.

Public Decryption Service: A threshold network can follow commands from a central authority (e.g. a smart contract) where each ciphertext is submitted with explicitly associated conditions. We associate a string to each potential set of conditions that encrypters may use. For example, the conditions "currentRound $>$ 100 && poolTokens $>$ 999" are a mixture of both time and financial conditions. These conditions are associated to a unique string S which is *to be used as the identity/public key* in the cryptosystem. Then, when these conditions are met, and only then, the threshold network releases the BLS signature over S, effectively releasing the IBE private key associated with the identity S. At this point, any third party can decrypt any ciphertext associated with S. Note here again that the advantage compared to regular public key encryption is that the work of the threshold network, for a given condition / identity, is *constant* with respect to the number of ciphertexts associated with any given condition, since any party can perform decryption once provided with the relevant IBE decryption key (or signature).

6 Implementation

In this section, we cover the implementation of our scheme in practice.

6.1 The League of Entropy

Since our solution leverages a threshold network of mutually untrusting parties, in which we only trust that there is never a threshold t of malicious nodes, any practical instantiation requires such a network.

Thankfully, in 2019, the League of Entropy was launched and participants, including a diverse set of companies, individuals and universities, are running drand nodes to provide a distributed, public, verifiable randomness beacon service based on threshold BLS signatures. The League of Entropy group key is generated using Pedersen Distributed Key Generation, which means it was never in memory on any given device. Since then, their network has grown to 23 nodes, spanning multiple disjoint organisations, data centres (ensuring that no more than 30% of the nodes are running on AWS or any cloud provider), jurisdiction and geographical localisations (LoE contains at least one node in each of the US, South America, Europe and Asia). The current threshold is 13 and the network has been providing signed random beacons with 100% uptime since the launch of their mainnet network in August 2020. It powers large networks such as the Filecoin network where it is used as a randomness source for the leader election protocol. It is also known to have been used for covid-related random sampling, and other use cases where public, verifiable randomness is desirable, including gambling systems and sortitions.

Previously, all drand beacons formed a "chain" by being linked to each other through their signatures: the signature of beacon b_{i+1} was produced by signing the message $m||b_i$. More recently, drand has set up a new network in which new random beacons are not linked to previous beacons, allowing anyone to know in advance the message being signed: it is simply the round number of that beacon. We can leverage this fact and treat the round number as the identity in our scheme to achieve timelock encryption in practice.

6.2 Implementation

We have implemented our scheme by implementing the CCA version `FullIdent` of the IBE scheme from [BF01] and adding it to the Go cryptography library named Kyber [tt22], as well as to our own timelock library in Typescript [MR22]. Being able to rely on existing libraries to perform the BLS 12-381 curve operations in both cases, as detailed below in 6.3, has allowed our implementations of IBE to be very concise, in less than 150 lines of code.

Relying on these IBE implementations, we provide an open-source Go timelock library, tlock, as well as a CLI-tool, tle, to perform timelock encryption [RG22], and a Typescript library, tlock-js [MR22], along with a demo website [MAGR22] that enables you to test timelock encryption and decryption locally in your browser.

Our timelock libraries are using the age framework [Val19] to encrypt the data itself and use our tTLE scheme to "wrap" the corresponding symmetric key, effectively allowing us to encrypt arbitrarily-sized data while only time-locking a symmetric key, as is usually done with public-key encryption schemes and known as "hybrid encryption". Doing so, and relying on our IBE implementation, has allowed our timelock logic to have a minimal code footprint again, in less than 200 lines of code.

We are relying on the existing drand HTTP API endpoints to retrieve the necessary BLS signatures of random beacons produced by the League of Entropy.

6.3 Benchmarks

Since we have produced two different, interoperable implementations: a Go one, and a Typescript (TS) one, we can compare them to get a sense of the performances one can expect when doing timelock encryption. Since our schemes relies on pairing-based cryptography, and we are relying on we chose to instantiate it on the pairing-friendly curve BLS 12-381.

For our Go version, we are relying on the kilic [kil] implementation of BLS 12-381, and for our TS version, we are relying on the noble framework implementation [Mil].

After publicly releasing our timelock libraries, we were made aware of a third-party implementation in Rust of our scheme [Lui]. We are not including it in the below table, since we did not test interoperability, but its performance surprisingly seemed in line with that of our Typescript implementation.

The following table shows the speed of the different operations we perform, for each library:

	Go	TS
Pairing	1133 op/s	73 op/s
Key encryption	539 op/s	26 op/s
Key decryption	1014 op/s	56 op/s

Table 1: Benchmarks of the main operations in our timelock implementations, done on commodity hardware (AMD Ryzen 9 5900X 12-Core processor)

Typescript slower by nature compared to a compiled languages such as Go renders the two implementation hardly comparable, however both are fast enough to make timelock encryption usable in practice without a user noticing, since the slowest operation is currently taking 38ms, which is roughly the threshold at which users start to be able to notice latency [EYAE99].

6.4 Tradeoffs: $\mathbb{G}_1$ vs $\mathbb{G}_2$

The BLS signature scheme can be instantiated in two different ways, depending on which group we want to instantiate public keys and signatures:

$$(a)\ \pi \in \mathbb{G}_1,\ P \in \mathbb{G}_2$$
$$(b)\ \pi \in \mathbb{G}_2,\ P \in \mathbb{G}_1$$

This can have significant performance and size consequences, since e.g. when using the BLS 12-381 curve, the group $\mathbb{G}_2$ is defined over an extension field of degree 2 compared to the group $\mathbb{G}_1$, meaning coordinates in $\mathbb{G}_2$ are twice the size of coordinates in $\mathbb{G}_1$.

Signature Tradeoffs: The option (a) gives a *shorter* signature size, as well as an easier 'hash-to-curve' method (which is important in the context of onchain verification), for a larger public key. The option (b) gives a *larger* signature size, and a more computationally intensive 'hash-to-curve' method but uses a smaller public key. Only considering signature consideration, the option (a) is usually the most optimal one.

Encryption Tradeoffs: Our encryption scheme lies in *the same group* as where the public key lies, hence it is strictly better to use option (b) given its group operations are strictly cheaper than $\mathbb{G}_2$.

Tension between encryption vs signature: For signature, it's usually best to use option (a) (in the context of onchain verification) but for encryption it's best to use option (b). Currently, LoE is set on the (b) method but is planning to switch to (a) to alleviate the cost of verifying randomness onchain. The performance of the scheme will be impacted but there are further optimizations we can do on the decryption part. For example, the decryption check $U = rG_2$ can be optimized given it's a fixed based scalar multiplication. Note there is no 'hash-to-curve' required for decryption so decryption should still remain relatively performant.

7 Future Work

Batch decryption: Decryption requires one pre-computed pairing and one individual pairing per decryption. A pairing can be costly and thus a method for batch decryption would be beneficial to using this method at scale for every round and multiple transactions.

One advantage that we have over the "generic" identity-based setting is that every ciphertext is encrypted under the same "ID". The main problem is that each encryption uses an individual $r = H_3(\sigma, M)$ where both σ and M are different per user.

Ideally we could use unified "randomness" that is exclusively dependent upon round number but this is unlikely to achieve CCA security. A more realistic goal is to settle for a linear number of operations which are *less* expensive than n pairings, i.e. $O(n)$ field operations would drastically improve decryption time.

Random Linear Combinations: We could trade off n pairings for n $\mathbb{G}_T$ computations. The prover already performs the pairing operations but randomises them and only gives the result. The random component can be computed publicly from the set of ciphertexts and thus the verifier merely aggregates individual results and checks if the output is consistent with what the prover sent.

Batch scalar multiplication: We could offer a "BatchVerify" API call that batches the scalar multiplication checks, using Multiple Scalar Multiplication (MSM) operations, such as the Pippenger algorithm [Boo17].

Chosen-Plaintext Secure Implementation: The current scheme and implementation are analyzed with respect to the CCA-secure variant of BF-IBE. We intend to leverage the fact that TLE-CCA security is unnecessary for publicly decryptable TLE, especially under our threshold assumptions. In which case, it is possible to implement a CPA variant of BF-IBE to achieve significant improvements in efficiency.

Quantum-resistant threshold network: To date there are not many threshold signature [CS19] or identity-based encryption algorithms which are quantum-resistant. A quantum-computer would currently be able to defeat both the BLS and IBE schemes underlying our timelock solution, and therefore completely compromise all encrypted data. It would be interesting to look into possible alternatives that would enable a practical and quantum resistant threshold network to provide timelock capabilities.

8 Acknowledgements

We want to thank Bryan Ford for the initial insight of binding the threshold BLS with IBE in this fashion. We are also grateful to Justin Drake for his insightful comments regarding open research problems, on-chain consumption and precomputations. Finally, we thank Rosario Gennaro, Anca Nitulescu and Matteo Companelli for their comments and help on the paper, as well as Patrick McClurg for his comments, help, and work on the implementations and benchmarks.

References

Ano21. Anoma. Ferveo: A synchronous distributed key generation protocol for front-running protection on public blockchains. https://github.com/anoma/ferveo, 2021. 3

BF01. Dan Boneh and Matt Franklin. Identity-based encryption from the weil pairing. In *Annual international cryptology conference*, pages 213–229. Springer, 2001. 1, 3, 5, 9, 14

BGJ+15. Nir Bitansky, Shafi Goldwasser, Abhishek Jain, Omer Paneth, Vinod Vaikuntanathan, and Brent Waters. Time-lock puzzles from randomized encodings. Cryptology ePrint Archive, Paper 2015/514, 2015. https://eprint.iacr.org/2015/514. 2

BLS01. Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 514–532. Springer, Heidelberg, December 2001. 1, 4

Bol03. Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In Yvo Desmedt, editor, *PKC 2003*, volume 2567 of *LNCS*, pages 31–46. Springer, Heidelberg, January 2003. 1, 4, 9

Boo17. Jonathan Bootle. Efficient Multi Exponentiation. https://jbootle.github.io/Misc/pippenger.pdf, 2017. 16

CDK+21. Matteo Campanelli, Bernardo David, Hamidreza Khoshakhlagh, Anders Konring, and Jesper Buus Nielsen. Encryption to the Future: A Paradigm for Sending Secret Messages to Future (Anonymous) Committees. Cryptology ePrint Archive, Paper 2021/1423, 2021. https://eprint.iacr.org/2021/1423. 2

CHKO08. Jung Hee Cheon, Nicholas Hopper, Yongdae Kim, and Ivan Osipkov. Provably secure timed-release public key encryption. *ACM Transactions on Information and System Security (TISSEC)*, 11(2):1–44, 2008. 3

CS19. Daniele Cozzo and Nigel P. Smart. Sharing the LUOV: threshold post-quantum signatures. In *IMA International Conference on Cryptography and Coding*, pages 128–153. Springer, 2019. 16

csa19. Programmers solve MIT's 20-year-old cryptographic puzzle. https://www.csail.mit.edu/news/programmers-solve-mits-20-year-old-cryptographic-puzzle, 2019. 2

CV19. Gwangbae Choi and Serge Vaudenay. Timed-release encryption with master time bound key. In Ilsun You, editor, *WISA 19*, volume 11897 of *LNCS*, pages 167–179. Springer, Heidelberg, August 2019. 7

DKG. Distributed key generation. https://en.wikipedia.org/wiki/Distributed_key_generation. 3

dra. drand: Distributed publicly verifiable randomness. https://drand.love. 3

EYAE99. Stephen R Ellis, Mark J Young, Bernard D Adelstein, and Sheryl M Ehrlich. Discrimination of changes of latency during voluntary hand movement of virtual objects. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, volume 43, pages 1182–1186. SAGE Publications Sage CA: Los Angeles, CA, 1999. 15

GJKR99. Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 295–310. Springer, Heidelberg, May 1999. 4, 10

kil. kilic. High-speed BLS12-381 implementation in Go. https://github.com/kilic/bls12-381. 15

lag. Lagrange polynomials. https://en.wikipedia.org/wiki/Lagrange_polynomial. 5

LJKW18. Jia Liu, Tibor Jager, Saqib A Kakvi, and Bogdan Warinschi. How to build time-lock encryption. *Designs, Codes and Cryptography*, 86(11):2549–2586, 2018. 2

LoE20. League of entropy. https://en.wikipedia.org/wiki/League_of_entropy, 2020. 3

Lui. Timofey Luin. Rust library for practical time-lock encryption using 'drand' threshold network. https://github.com/timoth-y/tlock-rs. 15

MAGR22. Patrick McClurg, Julia Armbrust, Nicolas Gailly, and Yolan Romailler. Timevault: web-demo of timelock encryption. https://timevault.drand.love/, 2022. 14

May93. Timothy C. May. Timed-release crypto. https://cypherpunks.venona.com/date/1993/02/msg00129.html, 1993. 1

Mil. Paul Miller. bls12-381 implementation in typescript. https://github.com/paulmillr/noble-bls12-381. 15

MR22. Patrick McClurg and Yolan Romailler. Timelock encryption using drand. https://github.com/drand/tlock-js/, 2022. 14

RG22. Yolan Romailler and Nicolas Gailly. Timelock encryption using drand. https://github.com/drand/tlock/, 2022. 14

Riv96. Wagner Rivest, Shamir. Timelock puzzles and timed release crypto. https://people.csail.mit.edu/rivest/RivestShamirWagner-timelock.ps, 1996. 1, 2

RT06. Michael O. Rabin and Christopher Thorpe. Time-lapse cryptography. 2006. 2

shu22. Shutter, a threshold encryption based frontrunning protection system for ethereum smart contracts. https://github.com/shutter-network/shutter, 2022. 3

tt22. Drand team and DEDIS team. Advanced crypto library for the Go language. https://github.com/drand/kyber/tree/master/encrypt/ibe, 2022. 14

Val19. Filippo Valsorda. A simple, modern and secure encryption tool (and Go library) with small explicit keys, no config options, and UNIX-style composability. https://github.com/FiloSottile/age/, 2019. 14