Time-Lock Encrypted Storage for Blockchains

Amit Agarwal, Kushal Babel, Sourav Das, and Babak Poorebrahim Gilkalaye

Category Labs

Abstract. We introduce time-lock encrypted storage (tTLES), a storage service provided by blockchains. In tTLES, clients store encrypted values towards a future decryption time τ_{tgt} (measured in block height). The security of tTLES requires that a value is decrypted only if (i) the encrypted value is included in the blockchain, and (ii) the time τ_{tgt} has passed. This is crucially different from existing schemes, which only enforce either of these conditions but not both. We formalize tTLES, and present an efficient protocol that relies on (in a black-box manner) a threshold identity-based encryption scheme, and a recent batch threshold decryption scheme. Finally, we discuss various applications that will benefit from tTLES.

1 Introduction

Modern blockchains are increasingly hosting applications ranging from finance and on-chain governance to games and contests that require storing data confidentially for a period of time before it is decrypted and processed [14,19,10].

Consider the example of an on-chain Kaggle*-like Machine Learning model training contest. Here, participants (clients) submit their ML model weights to the on-chain application before a submission deadline, and the models are evaluated after the deadline to distribute rewards to the submitters in proportion to their model performance. The on-chain submissions to the contest need to remain confidential until the submission deadline so that participants do not gain an unfair advantage by seeing the model weights of other participants before the deadline. Moreover, submissions that participants send to the mempool but are not included in the blockchain (e.g., due to congestion or denial-of-service attacks) need to remain private indefinitely, even after the submission deadline, to protect the intellectual property of the clients.

In this work, we are interested in the following two key confidentiality properties of privacy-sensitive blockchain applications that can receive encrypted values (as transactions) as inputs:

- **Time-lock indistinguishability:** Encrypted values that are included in the blockchain need to remain confidential until a specified target time τ_{tgt} (e.g., submission deadline in the above example), after which they are decrypted publicly and processed.
- **Non-inclusion privacy:** Encrypted values that are *not* included in the blockchain (e.g., due to congestion or denial-of-service attacks) need to remain confidential indefinitely, even after the target time τ_{tgt} has passed.

One naive strategy to achieve these properties is to use secret sharing. The client can secret share its storage value among the blockchain validators, with one share given to each validator, who then reconstruct the storage value at the target time τ_{tgt} if it is included in the blockchain, and do not reconstruct it otherwise. This strategy, while achieving the desired properties, is not practical as it requires communication and computation costs that scale linearly with the number of validators and the number of encrypted values, and the computation cost is incurred in an "online manner" at the target time τ_{tgt}.

Existing efficient solutions for time-lock encryption [13] fail to provide non-inclusion privacy, i.e., they do not provide confidentiality for storage values that are not included in the blockchain. At a high level, these solutions use threshold identity-based encryption (tIBE) [2] where clients encrypt storage values with the identity τ_{tgt}, and the decryption key for time τ_{tgt} is derived (in a threshold fashion) from a public parameter that is made available at time τ_{tgt}. While this gives the time-lock indistinguishability property efficiently

*https://www.kaggle.com/

with the release of the key material at τ_{tgt}, it does not provide non-inclusion privacy since the decryption key for time τ_{tgt} can be used to decrypt all storage values encrypted with id τ_{tgt}, including those that are not included in the blockchain.

Recent solutions that threshold decrypt a batch of ciphertexts [1,6,9,5] provide non-inclusion privacy, as any ciphertexts that are not included in the batch remain confidential. However, such solutions do not inherently provide the time-lock indistinguishability property. They can be invoked at the target time τ_{tgt} with the input batch comprised of all submitted ciphertexts encrypted towards the target time τ_{tgt}, but these solutions incur a threshold decryption cost that necessarily depends on the size of the batch, and hence are not practical for decrypting at once a large number of values submitted towards τ_{tgt}. This form of decryption cost is incurred in an "online manner" when the threshold decryption is to be performed, and hence can be prohibitive.

1.1 Our contributions

In this work, we have the following contributions:

- We define the notion of (thresholdizable) time-lock encrypted storage (tTLES) (see Section 3), which achieves both the time-lock indistinguishability property and non-inclusion privacy, without incurring an "online" cost at τ_{tgt} that depends on the number of values encrypted towards τ_{tgt}. Instead, the (threshold) decryption cost of decrypting the stored values is amortized over time during the submission of the clients' encrypted values.
- We provide a concrete construction of time-lock encrypted storage (tTLES) primitive, that relies on a threshold identity-based encryption (tIBE) and batched threshold decryption (BTE) in a black-box manner (see Section 4).
- We discuss some interesting applications of time-lock encrypted storage (tTLES) primitive in the context of blockchains (see Section 1.4).

1.2 Our technique

At a high-level, we construct a threshold time-lock encrypted storage (tTLES) scheme by simply combining, in a black-box manner, a batch threshold decryption primitive (BTE) with a time-lock encryption primitive (tIBE). In a bit more detail, a client that wishes to encrypt a message m towards a target time τ_{tgt} samples a symmetric key s, and encrypts m using this key simply as $m \oplus \mathsf{PRG}(s)$, where PRG is a vanilla pseudorandom generator. The client then derives two secret shares, s_1 and s_2, of the symmetric key s, i.e., $s_1 \oplus s_2 = s$. It encrypts the first share s_1 using the batch decryption primitive towards time τ_{tgt} and the second share s_2 using the time-lock encryption primitive towards time τ_{tgt}. The final ciphertext ct consists of a tuple of four values $(\tau_{\mathsf{tgt}}, \mathsf{BTE.Enc}(s_1), \mathsf{tIBE.Enc}(s_2, \tau_{\mathsf{tgt}}), m \oplus \mathsf{PRG}(s))$.[†]

The batch threshold decryption component $\mathsf{BTE.Enc}(s_1)$ provides confidentiality for non-included values (since s_1 will be hidden unless ct is included in the blockchain), while the threshold time-lock encryption component $\mathsf{tIBE.Enc}(s_2, \tau_{\mathsf{tgt}})$ gives us the confidentiality for included values until time τ_{tgt} has reached (since s_2 will be hidden until time τ_{tgt}). For efficiency, the batch decryption component is invoked during the submission time (more precisely at the time of inclusion) so as to amortize the cost over the submission window and recover one secret share s_1 of the key s. At the target time τ_{tgt}, decryption is made efficient (independent of the number of values encrypted towards τ_{tgt}) by just releasing (in a threshold fashion) a succinct key $k_{\tau_{\mathsf{tgt}}}$ related to τ_{tgt} in order to recover the second share of the encryption key. More precisely, the tIBE interface provides an efficient threshold key issuance protocol $\Pi\text{-}\mathsf{KeyIss}$ which can be used to derive $k_{\tau_{\mathsf{tgt}}} \leftarrow \Pi\text{-}\mathsf{KeyIss}(\tau_{\mathsf{tgt}})$. Note that the running time and communication cost of this $\Pi\text{-}\mathsf{KeyIss}$ protocol is completely independent of the number of values encrypted towards τ_{tgt}. However, once $k_{\tau_{\mathsf{tgt}}}$ is made publicly

[†] An alternative approach would be to have nested encryption such that $\mathsf{ct} = (\tau_{\mathsf{tgt}}, \mathsf{BTE.Enc}(\mathsf{tIBE.Enc}(s, \tau_{\mathsf{tgt}})), m \oplus \mathsf{PRG}(s))$. We describe the secret-sharing based construction in this paper due to its simplicity.

available, anyone (e.g., a smart contract) can publicly decrypt $\mathsf{tIBE.Enc}(s_2, \tau_{\mathsf{tgt}})$ to recover the second share s_2.

Our technique maps cleanly to the operation of a blockchain. In every block, clients submit requests to store encrypted values. The blockchain validators can *partially decrypt* these values in every block (i.e. $\mathsf{BTE.Enc}(s_1)$), and store these intermediate decryption values in the blockchain to be decrypted with a small cost at the target time of decryption. Values that are not included in any block until the target block remain confidential.

1.3 Related Work

To the best of our knowledge, no prior work has studied time-lock encrypted storage that simultaneously guarantees both time-lock indistinguishability and non-inclusion privacy.

Prior works that only provide time-lock indistinguishability guarantees can be categorized into two groups based on their underlying design approach: (i) *time-release* schemes and (ii) *time-lock puzzles*. In time-release schemes, a committee holds secret-shared state and releases decryption material at a designated time, enabling delayed access to ciphertexts [7,12,13]. In contrast, in time-lock puzzles, decryption involves solving a sequential computational puzzle whose expected solution time matches the target delay [16,11,15]. Puzzle-based methods face two practical drawbacks in decentralized settings: they require significant computational resources and are sensitive to hardware assumptions.

Numerous recent papers have studied the batched threshold encryption (BTE) primitive that can be trivially extended to build encrypted storage that only provides the non-inclusion privacy property [8,1,6,9,5,3]. Clients can submit encrypted values using a BTE scheme, a subset of which will then be included in a block in the blockchain. The blockchain validators then decrypt the subset of encrypted values included using the BTE decryption algorithm. Clearly, this approach does not provide time-lock indistinguishability. Moreover, existing BTE constructions impose batch-size–dependent costs and require $\Omega(B)$ bilinear pairing operations to decrypt a batch of size B — which effectively caps the number of ciphertexts that can be decrypted in a single batch.

1.4 Other Applications

Besides the ML contest example we discussed above, the tTLES primitive can be useful for several other kinds of blockchain applications, such as:

On-chain puzzle solving. Game hosts post challenges with submission deadlines, and participants store their encrypted solutions using a tTLES. The time-lock indistinguishability property of tTLES prevents solutions from being copied before the deadline. The non-inclusion privacy property of tTLES ensures that solutions excluded from the blockchain (e.g., due to censorship) remain hidden, preventing adversaries from learning honest participants' work without allowing them to earn rewards.

On-chain governance. Voters submit their encrypted votes before a voting deadline using a tTLES. The time-lock indistinguishability property of tTLES prevents vote manipulation and bandwagon effects by keeping votes secret until the deadline. The non-inclusion privacy property protects the privacy of voters whose votes get censored, i.e., did not get included in the blockchain. Moreover, the non-inclusion privacy property prevents potential misinformation claims about election fraud from excluded votes.

Sealed-bid auctions. Bidders store their encrypted bids before an auction deadline using a tTLES. The time-lock indistinguishability property of the tTLES ensures the sealed nature of the auction until the deadline, while non-inclusion privacy protects bidders whose bids are not included in the blockchain, preventing sellers from learning excluded valuations that could lead to seller's remorse or strategic manipulation in future auctions.

2 Preliminaries

Notation. Let λ denote the security parameter. We assume that all algorithms get λ in unary as input. By $\mathsf{poly}(\lambda)$ and $\mathsf{negl}(\lambda)$, we mean the class $\lambda^{O(1)}$ and $\frac{1}{\lambda^{\omega(1)}}$ respectively. Given a security parameter λ, we use PPT to denote probabilistic $\mathsf{poly}(\lambda)$-time Turing Machines with $\mathsf{poly}(\lambda)$-sized advice. For a pair of distribution ensembles $(\{X_\lambda\}_{\lambda\in\mathbb{N}}, \{Y_\lambda\}_{\lambda\in\mathbb{N}})$, we say the ensembles are "computationally indistinguishable", denoted by $X_\lambda \approx_c Y_\lambda$, if $|\Pr[X_\lambda = 1] - \Pr[Y_\lambda = 1]| = \mathsf{negl}(\lambda)$. For a finite set S, we write $s \leftarrow\!\$\ S$ to denote that s is sampled uniformly at random from S, and we write $|S|$ to denote the size of S. For an integer $a \in \mathbb{N}$, we use $[a]$ to denote the ordered set $\{1, \ldots, a\}$. Furthermore, we write "$\leftarrow$" for probabilistic assignment and "$:=$" for deterministic assignment. Also, we use the notation $v \leftarrow \Pi\text{-}\star(u)$ for any algorithm to denote that it is a multi-party computation protocol among a set of parties, where each party has some private input, a common public input u and each honest party produces a common public output v (which can be $\perp$).

Threat model. We consider a network of n parties, denoted with the set $\{1, \ldots, n\}$. We consider an adversary $\mathcal{A}$ that can statically corrupt up to $t < n$ parties in the protocol. The corrupted parties are considered malicious and can deviate arbitrarily from the specified protocol. We also consider a trusted entity $\mathcal{E}$ to emulate an idealized blockchain protocol run by n parties. We elaborate on our assumptions on $\mathcal{E}$ in Section 3.

2.1 Threshold Identity-Based Encryption

Our protocol relies on the standard threshold identity-based encryption (tIBE) scheme [18], as defined below.

Definition 1 (Threshold Identity-Based Encryption). *A threshold identity-based encryption scheme* tIBE *has the following* PPT *algorithms and protocols:*

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$*: The randomized setup algorithm takes as input the security parameter* λ *and outputs public parameters* pp*, which are an implicit input to all subsequent algorithms and protocols.*
- $(\mathsf{mpk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{KeyGen}(n, t)$*: The randomized key-generation algorithm takes as input the total number of parties* n*, the corruption threshold* t*, and outputs a master public key* mpk *to all, and the* i*-th secret key* sk_i *to party* i*.*
- $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, m, \mathsf{id})$*: The randomized encryption algorithm takes the master public key* mpk*, a message* $m \in \mathcal{M}$*, and an identity* $\mathsf{id} \in \{0,1\}^*$*, and outputs a ciphertext* $\mathsf{ct} \in \mathcal{C}$*. Here,* $\mathcal{M}$ *and* $\mathcal{C}$ *are the message and ciphertext spaces, respectively.*
- $k_{\mathsf{id}}/\perp \leftarrow \Pi\text{-}\mathsf{KeyIss}(\mathsf{id})$*: This is an interactive protocol among all parties for key issuance, where each party* P_i *has a private input* sk_i*, and all parties have a common input* id*. The protocol outputs a decryption key* k_{id} *for the identity* id *or* $\perp$ *indicating failure.*
- $m/\perp := \mathsf{Dec}(\mathsf{ct}, k_{\mathsf{id}})$*: The deterministic decryption algorithm takes a ciphertext* ct *and the identity key* k_{id}*, and outputs a message* m *or* $\perp$ *on failure.*

We require the tIBE scheme to satisfy the standard correctness, indistinguishability under chosen-plaintext attack ($\mathsf{IND\text{-}CPA}$) security and the robustness properties. Intuitively, the correctness definition ensures messages encrypted under a identity id can be decrypted correctly with the corresponding issued key k_{id}. Similarly, the $\mathsf{IND\text{-}CPA}$ security guarantees privacy of messages encrypted under identities whose keys are unknown to the adversary. Note that this should hold even when the adversary statically corrupts up to t parties. Finally, the robustness requirement ensures that under-honest majority assumption, i.e., with $t < n/2$, an adversary can not prevent honest parties from successfully decrypting honestly generated ciphertexts. We note that in our robustness definition, we let the adversary to learn secret keys of all n parties. We formalize these properties next and refer the reader to [4] for more details of these properties.

Game $\mathsf{Expt}^{\mathsf{tIBE\text{-}cpa}}_{\mathcal{A},b}(n,t,\lambda)$:

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
$\mathcal{Q}_{\mathsf{KeyIss}} \leftarrow \emptyset$
$\mathcal{C} \leftarrow \mathcal{A}(\mathsf{pp})$
$(\mathsf{mpk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{KeyGen}(n,t)$
$(m_0, m_1, \mathsf{id}^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{KeyIss}}}(\mathsf{mpk}, \{\mathsf{sk}_i\}_{i\in\mathcal{C}})$
$\mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{mpk}, m_b, \mathsf{id}^*)$
$b' \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{KeyIss}}}(\mathsf{ct}^*)$
if $|\mathcal{C}| \leq t \ \wedge \ \mathsf{id}^* \notin \mathcal{Q}_{\mathsf{KeyIss}}$:
    return b'
return 0

Oracle $\mathcal{O}_{\mathsf{KeyIss}}(\mathsf{id})$:

$\mathcal{Q}_{\mathsf{KeyIss}} \leftarrow \mathcal{Q}_{\mathsf{KeyIss}} \cup \{\mathsf{id}\}$
// Π-KeyIss is run among all n parties where $\mathcal{A}$ controls the parties in $\mathcal{C}$
$k_{\mathsf{id}} \leftarrow \Pi\text{-}\mathsf{KeyIss}(\mathsf{id})$
return k_{id}

Game $\mathsf{Expt}^{\mathsf{tIBE\text{-}rob}}_{\mathcal{A}}(n,t,\lambda)$:

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
$\mathcal{C} \leftarrow \mathcal{A}(\mathsf{pp})$
$(\mathsf{mpk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{KeyGen}(n,t)$
$(m^*, \mathsf{id}^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{KeyIss}}}(\mathsf{mpk}, \{\mathsf{sk}_i\}_{i\in[n]})$
$\mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{mpk}, m^*, \mathsf{id}^*)$
$k_{\mathsf{id}^*} \leftarrow \mathcal{O}_{\mathsf{KeyIss}}(\mathsf{id}^*)$
if $|\mathcal{C}| \leq t \ \wedge \ \mathsf{Dec}(\mathsf{ct}^*, k_{\mathsf{id}^*}) \neq m^*$:
    return 1
return 0

Fig. 1: Security game $\mathsf{Expt}^{\mathsf{tIBE\text{-}cpa}}_{\mathcal{A},b}$ and $\mathsf{Expt}^{\mathsf{tIBE\text{-}rob}}_{\mathcal{A}}$ and for a tIBE scheme

Definition 2 (Correctness of tIBE). *For all $n, t < n$, all identity id, all message $m \in \mathcal{M}$, here $\mathcal{M}$ is the message space, the following holds.*

$$\Pr\left[\mathsf{Dec}(k_\tau, \mathsf{ct}) = m \ : \ \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{mpk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{KeyGen}(n,t) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, m, \mathsf{id}) \\ k_\tau \leftarrow \Pi\text{-}\mathsf{KeyIss}(\tau) \end{array}\right] = 1 - \mathsf{negl}(\lambda)$$

Definition 3 (IND-CPA security of tIBE). *Let $\mathsf{Expt}^{\mathsf{tIBE\text{-}cpa}}_{\mathcal{A},b}$ be the game as defined in Fig. 1. A tIBE scheme is IND-CPA secure if for all $n, t < n$, all PPT adversaries $\mathcal{A}$, we have*

$$\left|\Pr\left[\mathsf{Expt}^{\mathsf{tIBE\text{-}cpa}}_{\mathcal{A},0}(n,t,\lambda) \Rightarrow 1\right] - \Pr\left[\mathsf{Expt}^{\mathsf{tIBE\text{-}cpa}}_{\mathcal{A},1}(n,t,\lambda) \Rightarrow 1\right]\right| = \mathsf{negl}(\lambda)$$

Definition 4 (Robustness of tIBE). *Let $\mathsf{Expt}^{\mathsf{tIBE\text{-}rob}}_{\mathcal{A}}$ be the game as defined in Fig. 1. A tIBE scheme is robust if for all $n \in \mathbb{N}, t < n/2$, and all PPT adversaries $\mathcal{A}$, we have*

$$\Pr\left[\mathsf{Expt}^{\mathsf{tIBE\text{-}rob}}_{\mathcal{A}}(n,t,\lambda) \Rightarrow 1\right] = \mathsf{negl}(\lambda)$$

CPA vs chosen-ciphertext attack (CCA) secure tIBE. In our tTLES construction, we issue identity keys only publicly. More specifically, we will associate each identity id with a time (measured in block height) and publicly issue k_{id}, once the block at height id has been finalized. Therefore, all these keys are also available to the adversary after issuance. Hence, the IND-CPA security of tIBE suffices.

Concretely, we will instantiate the tIBE scheme using a threshold version of the IBE scheme from [2].

2.2 Batched Threshold Encryption

Our tTLES protocol also requires a chosen-ciphertext attack (IND-CCA) secure batched threshold encryption (BTE) primitive with *associated data* [6,9,1]. We will summarize the interface of BTE and the properties

Game $\mathsf{Expt}^{\mathsf{BTE\text{-}cca}}_{\mathcal{A},b}(n,t,\lambda,\mathsf{B}_{\mathsf{max}})$:

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
$\mathcal{Q}_{\mathsf{BDec}} \leftarrow \emptyset$
$\mathcal{C} \leftarrow \mathcal{A}(\mathsf{pp})$
$(\mathsf{mpk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{KeyGen}(n,t)$
$(m_0, m_1, \mathsf{ad}^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{BDec}}}(\mathsf{mpk}, \{\mathsf{sk}_i\}_{i\in\mathcal{C}})$
$\mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{mpk}, m_b, \mathsf{ad}^*)$
$b' \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{BDec}}}(\mathsf{ct}^*)$
if $|\mathcal{C}| \leq t \ \wedge \ \mathsf{ct}^* \notin \mathcal{Q}_{\mathsf{BDec}}$:
    return b'
return 0

Oracle $\mathcal{O}_{\mathsf{BDec}}(\{\mathsf{ct}_i\}_{i\in[B]})$:

// Π-BDec is run among all n parties where $\mathcal{A}$ controls the parties in $\mathcal{C}$
Let $\{\tilde{m}_i\}_{i\in[B]} \leftarrow \Pi\text{-}\mathsf{BDec}(\mathsf{mpk}, \{\mathsf{ct}_i\}_{i\in[B]})$
$\mathcal{Q}_{\mathsf{BDec}} \leftarrow \mathcal{Q}_{\mathsf{BDec}} \cup \{\mathsf{ct}_i\}_{i\in[B]}$
return $\{\tilde{m}_i\}_{i\in[B]}$

Game $\mathsf{Expt}^{\mathsf{BTE\text{-}rob}}_{\mathcal{A}}(n,t,\lambda,\mathsf{B}_{\mathsf{max}})$:

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
$\mathcal{C} \leftarrow \mathcal{A}(\mathsf{pp})$
$(\mathsf{mpk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{KeyGen}(n,t)$
Let $\mathsf{inp} := (\mathsf{mpk}, \{\mathsf{sk}_i\}_{i\in[n]})$
$(B, S \subseteq [B]) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{BDec}}}(\mathsf{inp})$
if $B > \mathsf{B}_{\mathsf{max}}$: **return** 0
$\{(m_i, \mathsf{ad}_i)\}_{i\in[B]\setminus S} \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{BDec}}}(\mathsf{inp})$
for all $i \in [B] \setminus S$:
    $\mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{mpk}, m_i, \mathsf{ad}_i)$
$\{\mathsf{ct}_i\}_{i\in S} \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{BDec}}}(\{\mathsf{ct}_i\}_{i\in[B]\setminus S})$
$\{\tilde{m}_i\}_{i\in[B]} \leftarrow \mathcal{O}_{\mathsf{BDec}}(\{\mathsf{ct}_i\}_{i\in[B]})$
if $|\mathcal{C}| \leq t \ \wedge \ (\exists\ i \in [B] \setminus S \text{ s.t. } \tilde{m}_i \neq m_i)$:
    return 1
return 0

Fig. 2: Security and robustness game $\mathsf{Expt}^{\mathsf{BTE\text{-}cca}}_{\mathcal{A},b}$ and $\mathsf{Expt}^{\mathsf{BTE\text{-}rob}}_{\mathcal{A}}$, respectively, for a BTE scheme

it provides. We note that existing BTE protocols do not guarantee IND-CCA security with associated data. However, using techniques from [4], we can generically transform those schemes to guarantee CCA security even with associated data.

Definition 5 (Batched Threshold Encryption with Associated Data). *A batched threshold encryption scheme* BTE *with associated data consists of the following* PPT *algorithms and protocols:*

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^{\mathsf{B}_{\mathsf{max}}})$*: The randomized setup algorithm takes as input the security parameter λ and the maximum batch size $\mathsf{B}_{\mathsf{max}}$ and outputs public parameters* pp *which is an implicit input to all subsequent interfaces.*
- $(\mathsf{mpk}, \{sk_i\}_{i\in[n]}) \leftarrow \mathsf{KeyGen}(n,t)$*: The randomized key-generation algorithm takes as input the total number of parties n, the corruption threshold t, and outputs a master public key* mpk *to all, and the i-th secret key* sk_i *to party i.*
- $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, \mathsf{msg}, \mathsf{ad})$*: The randomized encryption algorithm takes as input a public key* mpk*, and a message* msg*, and associated data* ad*, and outputs a ciphertext* ct*.*
- $\{\mathsf{msg}_\ell\}_{\ell\in[\mathsf{B}]}/\perp \leftarrow \Pi\text{-}\mathsf{BDec}(\mathsf{mpk}, \{\mathsf{ct}_\ell\}_{\ell\in[\mathsf{B}]})$*: This is an interactive protocol among all parties for batch decryption, where each party P_i has a private input* sk_i*, and all parties have common input* mpk *and a batch of ciphertexts* $\{\mathsf{ct}_\ell\}_{\ell\in[\mathsf{B}]}$ *where* $\mathsf{B} \leq \mathsf{B}_{\max}$*. The protocol outputs the original decrypted messages* $\{\mathsf{msg}_\ell\}_{\ell\in[\mathsf{B}]}$*, otherwise* $\perp$*.*

We require the BTE scheme to satisfy the *correctness*, IND-CCA *security* with associated data, and *robustness* properties. We formalize these properties in Definitions 6 to 8 and describe them next.

Naturally, the *correctness* property ensures that correctly encrypted ciphertexts successfully decrypt to the original encrypted messages. Formally,

Definition 6 (Correctness of BTE**).** *A batched threshold encryption scheme* BTE *is correct for all $n, t < n$, all $\mathsf{B}_{\mathsf{max}}$, all $B < \mathsf{B}_{\mathsf{max}}$, all messages $(m_1, m_2, \ldots, m_B) \in \mathcal{M}^\ell$, and all associated data $(\mathsf{ad}_1, \mathsf{ad}_2, \ldots, \mathsf{ad}_B)$, here $\mathcal{M}$ is the message space, the following holds.*

$$\Pr\left[\begin{array}{l|c} & \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^{\mathsf{B}_{\mathsf{max}}}) \\ \forall\ i \in [B] & (\mathsf{mpk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{KeyGen}(n,t) \\ \tilde{m}_i = m_i & \{\mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{mpk}, m_i, \mathsf{ad}_i)\}_{i\in[B]} \\ & \{\tilde{m}_j\}_{j\in[B]} \leftarrow \Pi\text{-}\mathsf{BDec}(\mathsf{mpk}, \{\mathsf{ct}_i\}_{i\in[B]}) \end{array}\right] = 1 - \mathsf{negl}(\lambda)$$

The IND-CCA security with associated data for BTE ensures that an adversary cannot tell apart between encryption of two equal-length messages, even if it can invoke the batch decryption protocol (Π-BDec) on arbitrary batches of its choice not containing the challenge ciphertext. We include the *associated data* property in our IND-CCA definition to prevent any adversary from copying honestly generated ciphertexts for any time τ and submitting them to the batch decryption protocol by claiming them to be for a different time $\tau' \neq \tau$. As we discuss later, preventing such malleability attacks is crucial to guarantee the non-inclusion privacy guarantee we discuss in Section 1. We note that this should hold, even when the adversary statically corrupts up to t parties during the protocol. More formally,

Definition 7 (IND-CCA security of BTE with associated data). *Let* $\mathsf{Expt}^{\mathsf{BTE\text{-}cca}}_{\mathcal{A},b}$ *be the game as defined in Fig. 2. A batched threshold encryption scheme* BTE *is* IND-CCA *secure with associated data if for all* $n, t < n$*, and all PPT adversaries* $\mathcal{A}$*, we have*

$$\left| \Pr\left[\mathsf{Expt}^{\mathsf{BTE\text{-}cca}}_{\mathcal{A},0}(n,t,\lambda,\mathsf{B_{max}}) \Rightarrow 1\right] - \Pr\left[\mathsf{Expt}^{\mathsf{BTE\text{-}cca}}_{\mathcal{A},1}(n,t,\lambda,\mathsf{B_{max}}) \Rightarrow 1\right] \right| = \mathsf{negl}(\lambda)$$

Finally, the *robustness* property prevents an adversary $\mathcal{A}$, that can corrupt up to t parties, from invoking the batch decryption protocol (Π-BDec) on malformed ciphertexts to disrupt decryption of correctly generated ciphertexts. We note that this is a generalization of the rouge-ciphertext security property defined in [6], where $\mathcal{A}$ can not corrupt any of the decrypting parties. Moreover, we want the robustness property to hold even when the adversary learns the secret keys of all n parties. More formally,

Definition 8 (Robustness of BTE). *Let* $\mathsf{Expt}^{\mathsf{BTE\text{-}rob}}_{\mathcal{A}}$ *be the game as defined in Fig. 2. A batched threshold encryption scheme* BTE *is robust if for all* $n \in \mathbb{N}, t < n/2, \mathsf{B_{max}} \in \mathbb{N}$*, we have the following:*

$$\Pr[\mathsf{Expt}^{\mathsf{BTE\text{-}rob}}_{\mathcal{A}}(n,t,\lambda) \Rightarrow 1] = \mathsf{negl}(\lambda)$$

2.3 Pseudorandom Generator

Finally, our tTLES protocol relies on a pseudorandom generator (PRG) scheme $\mathsf{PRG} : \{0,1\} \to \{0,1\}^{\ell(\lambda)}$ for some function $\ell(\lambda) > \lambda$. A PRG is secure, if the following holds:

Definition 9 (Pseudorandom Generator Security). *Consider the games* $\mathsf{Expt}_{\mathsf{PRG}}$ *and* $\mathsf{Expt}_{\mathsf{Rand}}$ *defined in Fig. 3. A pseudorandom generator scheme* PRG *is secure, if for all* PPT *adversary* $\mathcal{A}$*, we have the following:*

$$\left| \Pr\left[\mathsf{Expt}_{\mathsf{PRG}}(1^\lambda) \Rightarrow 1\right] - \Pr\left[\mathsf{Expt}_{\mathsf{Rand}}(1^\lambda) \Rightarrow 1\right] \right| = \mathsf{negl}(\lambda).$$

Game $\mathsf{Expt}_{\mathsf{PRG}}(1^\lambda)$:	**Game** $\mathsf{Expt}_{\mathsf{Rand}}(1^\lambda)$:
1: Sample $x \leftarrow \{0,1\}^\lambda$	1: Sample $y \leftarrow \{0,1\}^{\ell(\lambda)}$
2: **return** $\mathcal{A}(\mathsf{PRG}(x))$	2: **return** $\mathcal{A}(y)$

Fig. 3: Security games for pseudorandom generator

3 Defining Threshold Time-lock Encrypted Storage

A threshold time-lock encrypted storage (tTLES) is a storage service provided by an entity $\mathcal{E}$ (which can be a blockchain) and a set of n servers, among which up to t servers can be malicious. tTLES has clients, who can store encrypted data in tTLES and specify a future decryption time $\tau \in \mathbb{N}$ (time is just a monotonic counter here). We assume that $\mathcal{E}$ provides the following guarantees, which can, for instance, be easily achieved using a blockchain.

1. $\mathcal{E}$ maintains a monotonic current time τ_{curr} as time.
2. At any time τ_{curr}, $\mathcal{E}$ receives a batch of encrypted storage requests $C_{\tau_{\mathsf{curr}}} := \{\mathsf{ct}_i\}_{i\in[B_{\tau_{\mathsf{curr}}}]}$ where $B_{\tau_{\mathsf{curr}}}$ is an integer upper bounded by $\mathsf{B}_{\mathsf{max}}$. Each encrypted value $\mathsf{ct}_i \in C_{\tau_{\mathsf{curr}}}$ is encrypted towards some future time $\tau_i > \tau_{\mathsf{curr}}$. We note that each time τ_i can be distinct.
3. At any time τ_{curr}, upon receiving the batch of encrypted values $C_{\tau_{\mathsf{curr}}}$, $\mathcal{E}$ can run arbitrary public validation checks on $C_{\tau_{\mathsf{curr}}}$. Looking ahead, $\mathcal{E}$ can enforce inclusion policies based on the associated data in clients' requests.
4. At any time τ_{curr}, $\mathcal{E}$ maintains the encrypted storage values in storage $\mathsf{St}_{\tau'}$, for all future time $\tau' > \tau_{\mathsf{curr}}$ for which any client has submitted an encrypted storage request.

With access to an entity $\mathcal{E}$ that provides the guarantees we describe above, we define a time-lock encrypted storage protocol tTLES as follows.

Definition 10. *A threshold time-lock encrypted storage (tTLES) protocol* tTLES *consists of the following PPT algorithms:*

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^{\mathsf{B}_{\mathsf{max}}})$. *The setup algorithm takes as input the security parameter* λ*, an upper bound* $\mathsf{B}_{\mathsf{max}}$ *on the batch size, and outputs the public parameters for the scheme. We assume that all the interfaces below take* pp *as input.*
- $(\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{KeyGen}(n, t)$. *The key generation algorithm takes as input the total number of parties* n*, a threshold* $t < n$*, and outputs a public key* mpk *and* n *secret keys* $\{\mathsf{sk}_i\}_{i\in[n]}$*, where the* i*-th party receives* sk_i*.*
- $k_\tau \leftarrow \Pi\text{-}\mathsf{KeyIss}(\tau)$. *This is an interactive protocol that is executed among all parties, where each party* P_i *has a private input* sk_i*, and all parties have a common input* τ*. The protocol outputs the decryption key* k_τ *for time* τ*.*
- $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, m, \tau)$. *A client uses the randomized encryption algorithm* Enc *to encrypt a message* m*, under the public key* mpk*, and a future time* τ*.*
- $\{\tilde{m}_\ell\}_{\ell\in[\mathsf{B}]}/\perp \leftarrow \Pi\text{-}\mathsf{BDec}(\mathsf{mpk}, \{\mathsf{ct}_j\}_{j\in[B]})$. *This is an interactive protocol among all parties for batch decryption, where each party* P_i *has a private input* sk_i*, and all parties have common input* mpk *and a batch of ciphertexts* $\{\mathsf{ct}_\ell\}_{\ell\in[\mathsf{B}]}$*, possibly encrypted towards different times, where* $\mathsf{B} \leq \mathsf{B}_{\max}$*. The protocol outputs partially decrypted messages* $\{\tilde{m}_\ell\}_{\ell\in[\mathsf{B}]}$*, otherwise* $\perp$*.*
- $m := \mathsf{Dec}(k_\tau, \tilde{m})$. *The deterministic decryption algorithm takes a decryption key* k_τ *and a partially decrypted message* $\tilde{m}$*, and outputs the message* m*.*

Intuition behind tTLES **formulation.** We formalize the tTLES primitive envisioning the following usage scenarios. In a blockchain, consider a future target time (or block height) τ_{tgt}, that a client wants to encrypt its data towards. Then, any time before τ_{tgt}, the client can encrypt a value towards time τ_{tgt} using the Enc algorithm and submit it to get included in the blockchain (modeled by $\mathcal{E}$ in our definition) before the block at height τ_{tgt} gets finalized.

For decrypting a ciphertext ct (created w.r.t time τ_{tgt}), instead of defining a single decryption protocol, say $\Pi\text{-}\mathsf{SDec}(\tau_{\mathsf{tgt}}, \mathsf{ct})$ which outputs the message m iff ct has been included at time $\tau_{\mathsf{incl}} \leq \tau_{\mathsf{tgt}}$, we split the decryption process across three protocols and algorithms $(\Pi\text{-}\mathsf{KeyIss}, \Pi\text{-}\mathsf{BDec}, \mathsf{Dec})$. This is motivated by the following efficiency reason: In the former approach, the $\Pi\text{-}\mathsf{SDec}$ protocol will need to be executed on the entire set of encrypted storage values in $\mathsf{St}_{\tau_{\mathsf{tgt}}}$ which can be of unbounded size, and hence will be infeasible to execute in a timely manner in practice. Our approach avoids this pitfall by splitting decryption across three protocols, where the resources needed for $\Pi\text{-}\mathsf{KeyIss}$ and Dec are independent of the storage size $\mathsf{St}_{\tau_{\mathsf{tgt}}}$. Although the cost of running $\Pi\text{-}\mathsf{BDec}$ depends on the size of $\mathsf{St}_{\tau_{\mathsf{tgt}}}$, in our design, parties run $\Pi\text{-}\mathsf{BDec}$ in batches, once for every time leading up to the target time τ_{tgt}. As a result, the cost of running $\Pi\text{-}\mathsf{BDec}$ gets amortized over a long time interval, and does not result in a bottleneck in practice.

Properties of tTLES**.** Naturally, we want any tTLES protocol to satisfy correctness, which ensures that honestly generated ciphertexts get decrypted correctly.

Definition 11 (Correctness). *For all $n, t < n$, all $\lambda, \mathsf{B}_{\mathsf{max}}$, all $B < \mathsf{B}_{\mathsf{max}}$, all messages $(m_1, m_2, \ldots, m_B) \in \mathcal{M}^\ell$, and all time $(\tau_1, \tau_2, \ldots, \tau_B)$, here $\mathcal{M}$ is the message space, the following holds.*

$$\Pr\left[\begin{array}{c} \forall\ i \in [B] \\ \mathsf{Dec}(k_{\tau_i}, \tilde{m}_i) = m_i \end{array} \ : \ \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^{\mathsf{B}_{\mathsf{max}}}) \\ (\mathsf{mpk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{KeyGen}(n, t) \\ \{\mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{mpk}, m_i, \tau_i)\}_{i\in[B]} \\ \{k_{\tau_i} \leftarrow \Pi\text{-}\mathsf{KeyIss}(\tau_i)\}_{i\in[B]} \\ \{\tilde{m}_j\}_{j\in[B]} \leftarrow \Pi\text{-}\mathsf{BDec}(\mathsf{mpk}, \{\mathsf{ct}_i\}_{i\in[B]}) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda)$$

In terms of security, we intuitively require a tTLES to reveal the encrypted storage values when both: (i) the encrypted values are included in the blockchain, as per some inclusion condition c (say before the target decryption time), and (ii) the decryption time for the encrypted value has passed. Informally, we refer to these properties as *non-inclusion privacy* and *time-lock indistinguishability* property. The non-inclusion privacy ensures privacy of all the encrypted storage values that are not included in the chain, and the time-lock indistinguishability ensures that the standard timed delay privacy property. We formalize these requirements using the security game $\mathsf{Expt}^{\mathsf{tTLES\text{-}sec}}_{\mathcal{A},b}$ in Figure 4.

In the security game $\mathsf{Expt}^{\mathsf{tTLES\text{-}sec}}_{\mathcal{A},b}$, the game maintains the current time τ_{curr} and a set $\mathcal{Q}_{\mathsf{BDec}}$ of partially decrypted values. Additionally, a PPT adversary $\mathcal{A}$ gets access to the time increment oracle $\mathcal{O}_{\mathsf{TInc}}$, which upon each invocation with a batch of ciphertexts, (i) increments the current time (line 12), (ii) runs the partial batch-decryption protocol on the input batch of ciphertexts after removing the invalid ciphertexts based on the inclusion condition (line 13-16), and (iii) issues a key for the current time τ_{curr} (line 17). The winning condition of $\mathcal{A}$ is similar to the $\mathsf{IND\text{-}CCA}$ security game of public key encryption schemes. Specifically, for the challenge ciphertext $\mathcal{A}$ chooses two equal length messages (m_0, m_1) along with a decryption time τ^* (see line 5), and receives a challenge ciphertext ct^*. To win the game, $\mathcal{A}$ can then either request a key issuance for time τ^* or the batch decryption of a batch containing ct^*, but not both. We check this constraint in line 9. Intuitively, here the check $\tau_{\mathsf{curr}} \geq \tau^*$ ensures the time-lock indistinguishability, and the check $\mathsf{ct}^* \in \mathcal{Q}_{\mathsf{BDec}}$ ensures the non-inclusion privacy. $\mathcal{A}$ wins the game $\mathsf{Expt}^{\mathsf{tTLES\text{-}sec}}_{\mathcal{A},b}$, if it can successfully guess the challenge bit b, without violating this constraint. We formalize this in Definition 12.

Definition 12 (TLES security). *Let $\mathsf{Expt}^{\mathsf{tTLES\text{-}sec}}_{\mathcal{A},b}$ be the game as defined in Fig. 4. A tTLES scheme is secure if for all $n, t < n$, all security parameter λ, all $\mathsf{B}_{\mathsf{max}} \in \mathbb{N}$ and all PPT adversaries $\mathcal{A}$, we have*

$$\left|\Pr\left[\mathsf{Expt}^{\mathsf{tTLES\text{-}sec}}_{\mathcal{A},0}(n, t, \lambda, \mathsf{B}_{\mathsf{max}}) \Rightarrow 1\right] - \Pr[\mathsf{Expt}^{\mathsf{tTLES\text{-}sec}}_{\mathcal{A},1}(n, t, \lambda, \mathsf{B}_{\mathsf{max}}) \Rightarrow 1]\right| = \mathsf{negl}(\lambda)$$

Finally, from the usability perspective, we also require the tTLES protocol tTLES to be *robust* in the following sense. Under the honest-majority assumption (i.e., with $t < n/2$), an adversary cannot prevent honest parties from successfully decrypting the encrypted values, even when the adversary includes these honestly encrypted ciphertexts in batches containing arbitrarily crafted adversarial ciphertexts. We formalize this property using the game $\mathsf{Expt}^{\mathsf{tTLES\text{-}rob}}_{\mathcal{A}}$ in Fig. 4. Note that we want robustness to hold, even when the adversary learns the secret keys of all honest parties.

Definition 13 (TLES robustness). *Let $\mathsf{Expt}^{\mathsf{tTLES\text{-}rob}}_{\mathcal{A}}$ be the game as defined in Fig. 4. A tTLES scheme is secure if for all $n, t < n/2$, all security parameter λ, all $\mathsf{B}_{\mathsf{max}} \in \mathbb{N}$ and all PPT adversaries $\mathcal{A}$, we have*

$$\Pr\left[\mathsf{Expt}^{\mathsf{tTLES\text{-}rob}}_{\mathcal{A}}(n, t, \lambda, \mathsf{B}_{\mathsf{max}}) \Rightarrow 1\right] = \mathsf{negl}(\lambda)$$

Efficiency requirements. To eliminate expensive solutions, we require a tTLES protocol to satisfy the following efficiency properties:

(E1) Let $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, m, \tau)$, then we want that size of the ciphertext is proportional to the length of the underlying message, i.e., $|\mathsf{ct}| = O(|m|)$.

Game $\mathsf{Expt}^{\mathsf{tTLES\text{-}sec}}_{\mathcal{A},b}(n,t,\lambda,\mathsf{B}_{\mathsf{max}})$:

$\tau_{\mathsf{curr}} := 0$
$\mathcal{Q}_{\mathsf{BDec}} \leftarrow \emptyset$
$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^{\mathsf{B}_{\mathsf{max}}})$
$\mathcal{C} \leftarrow \mathcal{A}(\mathsf{pp})$ *// we assume $\mathcal{A}$ to be stateful*
$(\mathsf{mpk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{KeyGen}(n,t)$
$(m_0, m_1, \tau^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{TInc}}}(\mathsf{mpk}, \{\mathsf{sk}_i\}_{i\in\mathcal{C}})$
$\mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{mpk}, m_b, \tau^*)$
$b' \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{TInc}}}(\mathsf{ct}^*)$
if $\tau_{\mathsf{curr}} \geq \tau^* \ \wedge \ \mathsf{ct}^* \in \mathcal{Q}_{\mathsf{BDec}}$:
  return 0
return b'

Oracle $\mathcal{O}_{\mathsf{TInc}}(\{\mathsf{ct}_j\}_{j\in[B]})$:

$\tau_{\mathsf{curr}} := \tau_{\mathsf{curr}} + 1$
parse each ct_j as (τ_j, ct'_j)
// checking inclusion condition
$S := \left\{\mathsf{ct}_j \mid j \in [B] \wedge \tau_j > \tau_{\mathsf{curr}}\right\}$
$\mathcal{Q}_{\mathsf{BDec}} \leftarrow \mathcal{Q}_{\mathsf{BDec}} \cup S$
// Π-BDec and Π-KeyIss is run among all n parties where $\mathcal{A}$ controls the parties in $\mathcal{C}$
$\{\tilde{m}_i\}_{i\in[|S|]} \leftarrow \Pi\text{-}\mathsf{BDec}(\mathsf{mpk}, S)$
$k_{\mathsf{curr}} \leftarrow \Pi\text{-}\mathsf{KeyIss}(\tau_{\mathsf{curr}})$
return $(k_{\mathsf{curr}}, \{\tilde{m}_i\}_{i\in[|S|]})$

Game $\mathsf{Expt}^{\mathsf{tTLES\text{-}rob}}_{\mathcal{A}}(n,t,\lambda,\mathsf{B}_{\mathsf{max}})$:

$\tau_{\mathsf{curr}} := 0$
$\mathcal{Q}_{\mathsf{BDec}} \leftarrow \emptyset$
$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^{\mathsf{B}_{\mathsf{max}}})$
$\mathcal{C} \leftarrow \mathcal{A}(\mathsf{pp})$ *// we assume $\mathcal{A}$ to be stateful*
$(\mathsf{mpk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{KeyGen}(n,t)$
$(B, U \subseteq [B]) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{TInc}}}(\mathsf{mpk}, \{\mathsf{sk}_i\}_{i\in[n]})$
if $B > \mathsf{B}_{\mathsf{max}}$: **return** 0
$\{(m_i, \tau_i)\}_{i\in[B]\setminus U} \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{TInc}}}(\mathsf{mpk}, \{\mathsf{sk}_i\}_{i\in\mathcal{C}})$
// Π-BDec and Π-KeyIss is run among all n parties where $\mathcal{A}$ controls the parties in $\mathcal{C}$
for all $i \in [B] \setminus U$:
  $\mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{mpk}, m_i, \tau_i)$
  $k_{\tau_i} \leftarrow \Pi\text{-}\mathsf{KeyIss}(\tau_i)$
$\{\mathsf{ct}_i\}_{i\in U} \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{TInc}}}(\{\mathsf{ct}_i\}_{i\in[B]\setminus U})$
$\{\tilde{m}_i\}_{i\in[B]} \leftarrow \Pi\text{-}\mathsf{BDec}(\mathsf{mpk}, \{\mathsf{ct}_i\}_{i\in[B]})$
if $|\mathcal{C}| \leq t \ \wedge \ (\exists\, i \in [B] \setminus U$ s.t. $\mathsf{Dec}(k_{\tau_i}, \tilde{m}_i) \neq m_i)$:
  return 1
return 0

Fig. 4: Security and robustness game $\mathsf{Expt}^{\mathsf{tTLES\text{-}sec}}_{\mathcal{A},b}$ and $\mathsf{Expt}^{\mathsf{tTLES\text{-}rob}}_{\mathcal{A}}$, respectively, for a tTLES protocol

(E2) For any time τ, let St_τ be the encrypted storage maintained by $\mathcal{E}$, then we want $|\mathsf{St}_\tau|$ to be unbounded, i.e., $\mathcal{E}$ should be able to store arbitrary number of values in the encrypted storage. Note that this also implies that $|\mathsf{St}_\tau| \gg \mathsf{B}_{\mathsf{max}}$.

(E3) Let $m := \mathsf{Dec}(k_\tau, \tilde{m})$ for some message $\tilde{m}$, then the running time of Dec is $O(|\tilde{m}|, \lambda)$.

(E4) For any invocation of Π-BDec, we require the per party communication cost to be $O(\lambda)$.

(E5) For any time τ, the running time of the entity $\mathcal{E}$ is bounded by $\tilde{O}(\mathsf{B}_{\mathsf{max}})$.

4 Design and Analysis of Time-lock Encrypted Storage

In this section, we present our tTLES protocol and its security analysis. Our protocol is extremely simple and relies on threshold identity-based encryption (tIBE) and batched threshold encryption (BTE) in a black-box manner. Before we describe our approach, we want to note issues with designing tTLES using just tIBE or BTE. As discussed in Section 1, protocols such as tlock [13] that rely only on tIBE do not provide the non-inclusion privacy property. Below, we describe an approach that only relies on BTE, and discuss its limitations.

An inefficient solution using only BTE. Using just a BTE scheme, we can design a tTLES protocol as follows. Clients can submit their encrypted storage values towards a target time τ_{tgt}, anytime before τ_{tgt}. Let $\mathsf{St}_{\tau_{\mathsf{tgt}}}$ be the set of encrypted values to be decrypted at time τ_{tgt}. To decrypt these values, at time τ_{tgt}, parties run the $\mathsf{BTE}.\Pi\text{-}\mathsf{BDec}$ protocol to decrypt all the values in $\mathsf{St}_{\tau_{\mathsf{tgt}}}$ simultaneously.

This approach is correct and ensures both time-lock indistinguishability and non-inclusion privacy. However, it is not very efficient, as parties need to run the $\mathsf{BTE}.\Pi\text{-}\mathsf{BDec}$ on a large batch size $|\mathsf{St}_{\tau_{\mathsf{tgt}}}|$ at once, which can be very expensive [1,6,9]. Moreover, if $|\mathsf{St}_{\tau_{\mathsf{tgt}}}|$ can be unbounded, as required by the efficiency property (E2), this approach would be infeasible.

$\underline{\mathsf{Setup}(1^\lambda, 1^{\mathsf{B}_{\max}})}$:

1: Let $\mathsf{pp}_{\mathsf{BTE}} \leftarrow \mathsf{BTE.Setup}(1^\lambda, 1^{\mathsf{B}_{\max}})$
2: Let $\mathsf{pp}_{\mathsf{tIBE}} \leftarrow \mathsf{tIBE.Setup}(1^\lambda)$
3: **return** $\mathsf{pp} \leftarrow (\mathsf{pp}_{\mathsf{BTE}}, \mathsf{pp}_{\mathsf{tIBE}})$.

// All algorithms implicitly take pp *as input*

$\underline{\mathsf{KeyGen}(n, t)}$:

4: Let $(\mathsf{mpk}_{\mathsf{BTE}}, \{\mathsf{sk}_{\mathsf{BTE},i}\}_{i\in[n]}) \leftarrow \mathsf{BTE.KeyGen}(n, t)$
5: Let $(\mathsf{mpk}_{\mathsf{tIBE}}, \{\mathsf{sk}_{\mathsf{tIBE},i}\}_{i\in[n]}) \leftarrow \mathsf{tIBE.KeyGen}(n, t)$
6: **return** $\mathsf{mpk} := (\mathsf{mpk}_{\mathsf{BTE}}, \mathsf{mpk}_{\mathsf{tIBE}}), \quad \{\mathsf{sk}_i := \mathsf{sk}_{\mathsf{BTE},i}, \mathsf{sk}_{\mathsf{tIBE},i}\}_{i\in[n]}$

$\underline{\mathsf{Enc}(\mathsf{mpk}, m, \tau)}$:

7: Sample $s_1, s_2 \leftarrow\!\!\$\ \{0,1\}^\lambda$
8: Let $\mathsf{ct}_2 \leftarrow \mathsf{BTE.Enc}(\mathsf{mpk}_{\mathsf{BTE}}, s_1, \mathsf{ad} := \tau)$
9: Let $\mathsf{ct}_3 \leftarrow \mathsf{tIBE.Enc}(\mathsf{mpk}_{\mathsf{tIBE}}, s_2, \mathsf{id} := \tau)$
10: Let $\mathsf{ct}_4 := m \oplus \mathsf{PRG}(s_1 \oplus s_2, |m|)$ *// PRG expands the seed* $(s_1 \oplus s_2)$ *to* $|m|$ *bit pseudorandom string*
11: **return** $\mathsf{ct} := (\mathsf{ct}_1 := \tau, \mathsf{ct}_2, \mathsf{ct}_3, \mathsf{ct}_4)$

$\underline{\Pi\text{-}\mathsf{BDec}(\mathsf{mpk}, \{\mathsf{ct}_j\}_{j\in[B]})}$:

12: $\{s_{j,1}\}_{j\in[B]} \leftarrow \mathsf{BTE.}\Pi\text{-}\mathsf{BDec}(\mathsf{mpk}_{\mathsf{BTE}}, \{\mathsf{ct}_{j,2}\}_{j\in[B]})$
13: **return** $\{\tilde{m}_j\}_{j\in[B]} := \{(\mathsf{ct}_{j,1}, s_{j,1}, \mathsf{ct}_{j,3}, \mathsf{ct}_{j,4})\}_{j\in[B]}$

$\underline{\Pi\text{-}\mathsf{KeyIss}(\tau)}$:

14: **return** $\mathsf{tIBE.}\Pi\text{-}\mathsf{KeyIss}(\mathsf{id} = \tau)$

$\underline{\mathsf{Dec}(k_\tau, \tilde{m})}$:

15: Parse $(\_, s_1, \mathsf{ct}_3, \mathsf{ct}_4) := \tilde{m}$
16: let $s_2 := \mathsf{tIBE.Dec}(\mathsf{ct}_3, k_\tau)$
17: **return** $m := \mathsf{ct}_4 \oplus \mathsf{PRG}(s_1 \oplus s_2)$

Fig. 5: Description of our tTLES protocol.

4.1 Our Approach

We summarize our tTLES protocol tTLES in Fig. 5, and describe it next. tTLES is simple, efficient, and relies on threshold identity based encryption and batched threshold encryption in a black-box manner.

Setup (line 1-3). The setup algorithm runs the setup algorithm of BTE and tIBE, and outputs $(\mathsf{pp}_{\mathsf{BTE}}, \mathsf{pp}_{\mathsf{tIBE}})$ as its public parameters.

Key generation (line 4-6). Similar to the setup algorithm, our key generation algorithm runs the key generation algorithm of both BTE and tIBE. It then outputs the combined public key $(\mathsf{mpk}_{\mathsf{BTE}}, \mathsf{mpk}_{\mathsf{tIBE}})$ as its public key. Similarly, the secret key of the i-th party is the combined secret key $(\mathsf{sk}_{\mathsf{BTE},i}, \mathsf{sk}_{\mathsf{tIBE},i})$.

Encryption (line 7-11). To encrypt a message msg towards a target time τ_{tgt}, a client samples two secrets $s_1, s_2 \in \{0,1\}^\lambda$. The client then encrypts s_1 using the BTE encryption algorithm with τ_{tgt} as the associated data, and encrypts s_2 using tIBE with τ_{tgt} as the identity. The client then expands $s_1 \oplus s_2$ to $|m|$ bit pseudorandom string using a pseudorandom generator, and uses it as a one-time pad to encrypt the message m. Here, the secrets (s_1, s_2) can be seen as 2-out-of-2 secret shares of some secret $s := s_1 \oplus s_2$ [17]. Note that by using τ_{tgt} as the associated data in the BTE encryption, we ensure that the generated ciphertext ct is non-malleable even with respect to time τ_{tgt}.

Batch decryption at inclusion time (line 12-13). Let us first describe how the inputs to the tTLES.Π-BDec protocol are created. At any given time τ_{curr}, the bulletin-board entity $\mathcal{E}$ receives batch $\{\mathsf{ct}_j\}_{j\in[B]}$ of B ciphertexts. $\mathcal{E}$ parses each ciphertext ct_j as $\mathsf{ct}_j = (\tau_j, \mathsf{ct}_{j,2}, \star)$, where τ_j is the target decryption time of the ciphertext ct_j. Next, $\mathcal{E}$ checks that $\tau_j > \tau_{\mathsf{curr}}$, and creates a batch containing only such valid ciphertexts.

Without loss of generality, let us assume that all the ciphertexts in $\{\mathsf{ct}_j\}_{j\in[B]}$ satisfy the abovementioned inclusion condition. Then, our batch decryption protocol, invokes the BTE.Π-BDec protocol on input

$(\mathsf{mpk}_{\mathsf{BTE}}, \{\mathsf{ct}_{j,2}\}_{j\in[B]})$. Let $\{s_{j,1}\}$ be the output of the $\mathsf{BTE.\Pi\text{-}BDec}$ protocol. Our protocol, then outputs $\{\tilde{m}_j\}_{j\in[B]} := \{(\mathsf{ct}_{j,1}, s_{j,1}, \mathsf{ct}_{j,3}, \mathsf{ct}_{j,4})\}_{j\in[B]}$ as the partially decrypted values.

Key issuance (line 14). To issue a key for any time τ, our protocol runs the tIBE key issuance algorithm with $\mathsf{id} = \tau$, and outputs its output.

Decryption (line 15-17). Given the tIBE key k_τ for any time τ, and a partially decrypted ciphertext $\tilde{m} = (\tau, s_1, \mathsf{ct}_3, \mathsf{ct}_4)$, the decryption algorithm first recovers the share s_2 by running the $\mathsf{tIBE.Dec}$ with k_τ as the key. The algorithm outputs as $\mathsf{ct}_4 \oplus \mathsf{PRG}(s_1 \oplus s_2)$ as the message.

4.2 Analysis

The correctness of our protocol is clear. Next, we will argue security, assuming that the underlying BTE and the tIBE scheme are secure as per Definitions 3 and 7, respectively. Formally, we prove the following:

Theorem 1 (tTLES Security). *Assuming the existence of an IND-CPA secure threshold Identity Based Encryption (IBE) scheme (Definition 1) and IND-CCA secure Batched Threshold Encryption (BTE) scheme with associated data (Definition 5), and a pseudorandom generator (*PRG*), our tTLES protocol* tTLES *(Fig. 5) is secure as per Definition 12.*

Proof. The security of our scheme follows from the $\mathsf{IND\text{-}CCA}$ security of the BTE scheme and $\mathsf{IND\text{-}CPA}$ security of the tIBE scheme. To see this, consider the following set of experiments:

- Expt_0: This is the experiment $\mathsf{Expt}^{\mathsf{tTLES\text{-}sec}}_{\mathcal{A},b}(n, t, \lambda, \mathsf{B}_{\mathsf{max}})$ as defined in Fig. 4 with $b = 0$ and TLES scheme instantiated with our construction in Fig. 5. Let $\ell = |m_0| = |m_1|$, be the length of the challenge messages.
- Hyb_0: This is the same as Expt_0 except that when forming the challenge ciphertext $\mathsf{ct}^* = (\mathsf{ct}^*_1, \mathsf{ct}^*_2, \mathsf{ct}^*_3, \mathsf{ct}^*_4)$, instead of setting ct^*_4 as $m_0 \oplus \mathsf{PRG}(s_1 \oplus s_2, \ell)$, we sample a random string $r \leftarrow\$ \{0,1\}^\lambda$ and set ct^*_4 as $m_0 \oplus \mathsf{PRG}(r, \ell)$
- Hyb': This is the same as Hyb_0 except that when forming the challenge ciphertext $\mathsf{ct}^* = (\mathsf{ct}^*_1, \mathsf{ct}^*_2, \mathsf{ct}^*_3, \mathsf{ct}^*_4)$, we set ct^*_4 as $r \leftarrow\$ \{0,1\}^\ell$.
- Hyb_1: This is the same as Hyb' except that when forming the challenge ciphertext $\mathsf{ct}^* = (\mathsf{ct}^*_1, \mathsf{ct}^*_2, \mathsf{ct}^*_3, \mathsf{ct}^*_4)$, we sample a random string $r \leftarrow\$ \{0,1\}^\lambda$ and set ct^*_4 as $m_1 \oplus \mathsf{PRG}(r, \ell)$
- Expt_1: This is the experiment $\mathsf{Expt}^{\mathsf{tTLES\text{-}sec}}_{\mathcal{A},b}(n, t, \lambda, \mathsf{B}_{\mathsf{max}})$ as defined in Fig. 4 with $b = 1$ and TLES scheme instantiated with our construction in Fig. 5.

We now prove that $\mathsf{Expt}_0 \approx_c \mathsf{Expt}_1$. To do this, we argue that each intermediate pairs of hybrids are computationally indistinguishable. At a high-level, the proof structure looks as follows:

- $\mathsf{Expt}_0 \approx_c \mathsf{Hyb}_0$: This follows from the $\mathsf{IND\text{-}CCA}$ security of the BTE scheme and $\mathsf{IND\text{-}CPA}$ security of the threshold IBE scheme.
- $\mathsf{Hyb}_0 \approx_c \mathsf{Hyb}'$: This follows from the security of the pseudorandom generator.
- $\mathsf{Hyb}' \approx_c \mathsf{Hyb}_1$: This again follows from the security of the pseudorandom generator.
- $\mathsf{Hyb}_1 \approx_c \mathsf{Expt}_1$: This follows from the $\mathsf{IND\text{-}CCA}$ security of the BTE scheme and $\mathsf{IND\text{-}CPA}$ security of the threshold IBE scheme in a manner identical to the indistinguishability argument between Expt_0 and Hyb_0.

We will now rigorously prove each of the four aforementioned indistinguishability claims.

Lemma 1. $\mathsf{Expt}_0 \approx_c \mathsf{Hyb}_0$.

Proof. Consider an interaction of $\mathcal{A}$ with a game (either Expt_0 or Hyb_0). In this interaction, let E be the event that $\mathcal{A}$ makes oracle query on the ciphertext ct^*_2 with associated data τ^*. Similarly, let F be the event that $\mathcal{A}$ makes $\geq \tau^*$ oracle queries.

Intuitively, we define these events to separately analyze the cases where the indistinguishability of Expt_0 and Hyb_0 is reliant on the BTE scheme and tIBE scheme. Specifically, we will now establish the following three lemmas, and then combine them to prove that $\mathsf{Expt}_0 \approx_c \mathsf{Hyb}_0$.

Reduction algorithm $\mathcal{A}_{\mathsf{BTE}}$:

1. Initialize $\tau_{\mathsf{curr}} := 0$ and $\mathcal{Q}_{\mathsf{BDec}} \leftarrow \emptyset$
2. Receive $\mathsf{pp}_{\mathsf{BTE}}$ from $\mathcal{C}_{\mathsf{BTE}}$, and sample $\mathsf{pp}_{\mathsf{tIBE}} \leftarrow \mathsf{tIBE.Setup}(1^\lambda)$
3. Set $\mathsf{pp} := (\mathsf{pp}_{\mathsf{BTE}}, \mathsf{pp}_{\mathsf{tIBE}})$ and send pp to $\mathcal{A}$
4. Receive the corruption set $\mathcal{C}$ from $\mathcal{A}$ and forward it to the challenger $\mathcal{C}_{\mathsf{BTE}}$
5. Receive $(\mathsf{mpk}_{\mathsf{BTE}}, \{\mathsf{sk}_{\mathsf{BTE},i}\}_{i\in\mathcal{C}})$ from $\mathcal{C}_{\mathsf{BTE}}$, and sample $(\mathsf{mpk}_{\mathsf{tIBE}}, \{\mathsf{sk}_{\mathsf{tIBE},i}\}_{i\in[n]}) \leftarrow \mathsf{tIBE.KeyGen}(n, t)$
6. Send $\mathsf{mpk} := (\mathsf{mpk}_{\mathsf{BTE}}, \mathsf{mpk}_{\mathsf{tIBE}})$, $\quad \{\mathsf{sk}_i := \mathsf{sk}_{\mathsf{BTE},i}, \mathsf{sk}_{\mathsf{tIBE},i}\}_{i\in[\mathcal{C}]}$
7. Simulate the oracle $\mathcal{O}_{\mathsf{TInc}}$ for adversary $\mathcal{A}$ as follows:
   (a) Update $\tau_{\mathsf{curr}} := \tau_{\mathsf{curr}} + 1$
   (b) For every query $Q = \{\mathsf{ct}_j\}_{j\in[B]}$ from $\mathcal{A}$, compute S as in Line 14 in Figure 4, and forward S to $\mathcal{C}_{\mathsf{BTE}}$.
   (c) Once $\mathcal{C}_{\mathsf{BTE}}$ initiates an execution of the protocol $\Pi\text{-}\mathsf{BDec}(\mathsf{mpk}_{\mathsf{BTE}}, S)$ on behalf of the set of honest parties $[n] \setminus \mathcal{C}$, forward the protocol messages back and forth between $\mathcal{C}_{\mathsf{BTE}}$ and $\mathcal{A}$ (which is executing $\Pi\text{-}\mathsf{BDec}(\mathsf{mpk}_{\mathsf{BTE}}, S)$ on behalf of the set of corrupt parties $\mathcal{C}$). Let $\{\tilde{m}_i\}_{i\in[|S|]}$ be the output of the execution of $\Pi\text{-}\mathsf{BDec}(\mathsf{mpk}_{\mathsf{BTE}}, S)$ computed by the $\mathcal{C}_{\mathsf{BTE}}$ on behalf of honest parties. Receive $\{\tilde{m}_i\}_{i\in[|S|]}$ from $\mathcal{C}_{\mathsf{BTE}}$.
   (d) Update $\mathcal{Q}_{\mathsf{BDec}} \leftarrow \mathcal{Q}_{\mathsf{BDec}} \cup S$
   (e) Initiate an execution of the protocol $\Pi\text{-}\mathsf{KeyIss}(\tau_{\mathsf{curr}})$ by simulating the set of honest parties $[n] \setminus \mathcal{C}$ using the secret keys $\{\mathsf{sk}_{\mathsf{tIBE},i}\}_{i\in[n]\setminus\mathcal{C}}$, forward the protocol messages back and forth between the simulated set of honest parties and $\mathcal{A}$ (which is executing $\Pi\text{-}\mathsf{KeyIss}(\tau_{\mathsf{curr}})$ on behalf of the set of corrupt parties $\mathcal{C}$). Let k_{curr} be the output of the execution of $\Pi\text{-}\mathsf{KeyIss}(\tau_{\mathsf{curr}})$ computed by the simulated honest parties.
   (f) Send $(k_{\mathsf{curr}}, \{\tilde{m}_i\}_{i\in[|S|]})$ to $\mathcal{A}$.
8. Simulate the challenge phase for adversary $\mathcal{A}$ in the following way:
   (a) Receive the challenge (m_0, m_1, τ^*) from $\mathcal{A}$
   (b) Sample $s_{1,0}, s_{1,1} \leftarrow\$ \{0,1\}^\lambda$ and forward $(s_{1,0}, s_{1,1}, \mathsf{ad} := \tau^*)$ to $\mathcal{C}_{\mathsf{BTE}}$.
   (c) Receive the response ciphertext $\mathsf{ct}^*_{\mathsf{BTE}}$ from $\mathcal{C}_{\mathsf{BTE}}$ where $\mathsf{ct}^*_{\mathsf{BTE}} \leftarrow \mathsf{BTE.Enc}(\mathsf{mpk}_{\mathsf{BTE}}, s_{1,0}, \mathsf{ad} := \tau^*)$ or $\mathsf{ct}^*_{\mathsf{BTE}} \leftarrow \mathsf{BTE.Enc}(\mathsf{mpk}_{\mathsf{BTE}}, s_{1,1}, \mathsf{ad} := \tau^*)$
   (d) Sample $s_2 \leftarrow\$ \{0,1\}^\lambda$ and compute $\mathsf{ct}^*_{\mathsf{tIBE}} \leftarrow \mathsf{tIBE.Enc}(\mathsf{mpk}_{\mathsf{tIBE}}, s_2, \mathsf{id} := \tau^*)$
   (e) Compute $\mathsf{ct}^*_{\mathsf{PRG}} := m_0 \oplus \mathsf{PRG}(s_{1,0} \oplus s_2, \ell)$
   (f) Send $\mathsf{ct}^* := (\tau^*, \mathsf{ct}^*_{\mathsf{BTE}}, \mathsf{ct}^*_{\mathsf{tIBE}}, \mathsf{ct}^*_{\mathsf{PRG}})$ to $\mathcal{A}$
9. Let b' be the bit output by $\mathcal{A}$. Define, $\tilde{b}$ as:

$$\tilde{b} = \begin{cases} 0 & : \text{if } \tau_{\mathsf{curr}} \geq \tau^* \ \wedge \ \mathsf{ct}^* \in \mathcal{Q}_{\mathsf{BDec}} \\ b' & : \text{otherwise} \end{cases}$$

10. If $\tau_{\mathsf{curr}} \geq \tau^* \ \wedge \mathsf{ct}^* \notin \mathcal{Q}_{\mathsf{BDec}}$, output $\tilde{b}$, otherwise output 0.

Fig. 6: Reduction algorithm $\mathcal{A}_{\mathsf{BTE}}$ to break the IND-CCA security of BTE.

Claim 2 $\left|\Pr[\,\mathsf{Expt}_0 \Rightarrow 1 \wedge \neg\mathsf{E} \wedge \mathsf{F}\,] - \Pr[\,\mathsf{Hyb}_0 \Rightarrow 1 \wedge \neg\mathsf{E} \wedge \mathsf{F}\,]\right| = \mathsf{negl}(\lambda)$.

Proof. Suppose the lemma is false w.r.t. an adversary $\mathcal{A}$ that participates in either Expt_0 and Hyb_0. We can use $\mathcal{A}$ to construct an adversary $\mathcal{A}_{\mathsf{BTE}}$ that contradicts the IND-CCA security of the BTE scheme. Let $\mathcal{C}_{\mathsf{BTE}}$ be the challenger for BTE scheme which executes the steps outlined in BTE security game $\mathsf{Expt}^{\mathsf{BTE\text{-}cca}}_{\mathcal{A},}$ (see Fig. 2) while interacting with the reduction adversary $\mathcal{B}$ and providing it access to the oracle $\mathcal{O}_{\mathsf{BDec}}$.

Let $\mathsf{Expt}^{\mathsf{BTE\text{-}cca}}_{\mathcal{A},0}$ and $\mathsf{Expt}^{\mathsf{BTE\text{-}cca}}_{\mathcal{A},1}$ denote the experiment where $\mathcal{C}_{\mathsf{BTE}}$ encrypts $s_{1,0}$ and $s_{1,1}$, respectively. The output of these experiments, as defined in BTE security game $\mathsf{Expt}^{\mathsf{BTE\text{-}cca}}_{\mathcal{A},b}$ (see Fig. 2), is either 0 if the ciphertext $\mathsf{ct}^*_{\mathsf{BTE}}$ with associated data $\mathsf{ad} := \tau^*$ is queried to $\mathcal{C}_{\mathsf{BTE}}$ or is equal to the bit received from the reduction adversary $\mathcal{A}_{\mathsf{BTE}}$, which we describe in Fig. 6.

By the construction of $\mathcal{A}_{\mathsf{BTE}}$, we note the following:

– When $\mathcal{C}_{\mathsf{BTE}}$ encrypts $s_{1,0}$, the view of $\mathcal{A}$ is identical to its view in Expt_0 and the bit $\tilde{b}$ (defined in Line 13 of the adversary $\mathcal{A}_{\mathsf{BTE}}$) is identical to the output of Expt_0 (as defined in Fig. 4).

– Similarly, when $\mathcal{C}_{\mathsf{BTE}}$ encrypts $s_{1,1}$, the view of $\mathcal{A}$ is identical to its view in Hyb_0 and the bit $\tilde{b}$ is identical to the output of Hyb_0.

Moreover, the adversary $\mathcal{A}_{\mathsf{BTE}}$ outputs 1 when all the following conditions are true:

– $\tilde{b} = 1$ (equivalent to Expt_0 or Hyb_0 outputting 1)
– $\mathsf{ct}^* \notin \mathcal{Q}_{\mathsf{BDec}}$ (equivalent to the event $\neg\mathsf{E}$ occurring)
– $\tau_{\mathsf{curr}} \geq \tau^*$ (equivalent to the event F occurring)

Hence, it follows that,

$$\Pr[\mathsf{Expt}^{\mathsf{BTE\text{-}cca}}_{\mathcal{A},0} \Rightarrow 1] = \Pr[\mathsf{Expt}_0 \Rightarrow 1 \wedge \neg\mathsf{E} \wedge \mathsf{F}] \tag{1}$$

$$\Pr[\mathsf{Expt}^{\mathsf{BTE\text{-}cca}}_{\mathcal{A},1} \Rightarrow 1] = \Pr[\mathsf{Hyb}_0 \Rightarrow 1 \wedge \neg\mathsf{E} \wedge \mathsf{F}] \tag{2}$$

Subtracting Eq. (2) from Eq. (1), we get

$$\left|\Pr[\mathsf{Expt}^{\mathsf{BTE\text{-}cca}}_{\mathcal{A},0} \Rightarrow 1] - \Pr[\mathsf{Expt}^{\mathsf{BTE\text{-}cca}}_{\mathcal{A},1} \Rightarrow 1]\right| = \left|\Pr[\mathsf{Expt}_0 \Rightarrow 1 \wedge \neg\mathsf{E} \wedge \mathsf{F}] - \Pr[\mathsf{Hyb}_0 \Rightarrow 1 \wedge \neg\mathsf{E} \wedge \mathsf{F}]\right| \neq \mathsf{negl}(\lambda)$$

This contradicts the IND-CCA security of the BTE scheme. □

Claim 3 $\left|\Pr[\mathsf{Expt}_0 \Rightarrow 1 \wedge \neg\mathsf{F}] - \Pr[\mathsf{Hyb}_0 \Rightarrow 1 \wedge \neg\mathsf{F}]\right| = \mathsf{negl}(\lambda)$.

Proof. Suppose the lemma is false w.r.t. an adversary $\mathcal{A}$ that participates in Expt_0 and Hyb_0. We can use $\mathcal{A}$ to construct an adversary $\mathcal{A}_{\mathsf{tIBE}}$ that contradicts the IND-CPA security of the tIBE scheme. Let $\mathcal{C}_{\mathsf{tIBE}}$ be the challenger for tIBE scheme which executes the steps outlined in tIBE security game (Fig. 1) while interacting with the reduction adversary $\mathcal{A}_{\mathsf{tIBE}}$ and providing it access to the oracle $\mathcal{O}_{\mathsf{Keylss}}$.

Let $\mathsf{Expt}^{\mathsf{tIBE\text{-}cpa}}_{\mathcal{A},0}$ and $\mathsf{Expt}^{\mathsf{tIBE\text{-}cpa}}_{\mathcal{A},1}$ denote the experiment where $\mathcal{C}_{\mathsf{tIBE}}$ encrypts $s_{2,0}$ and $s_{2,1}$, respectively. The output of these experiments, as defined in tIBE security game $\mathsf{Expt}^{\mathsf{tIBE\text{-}cpa}}_{\mathcal{A},b}$ (see Fig. 1), is 0 if $\mathsf{id} \coloneqq \tau^*$ is queried to $\mathcal{C}_{\mathsf{tIBE}}$, else the output is equal to the bit received from the reduction adversary $\mathcal{A}_{\mathsf{tIBE}}$, which we describe in Fig. 7.

By the construction of $\mathcal{A}_{\mathsf{tIBE}}$, we note the following:

– When $\mathcal{C}_{\mathsf{tIBE}}$ encrypts $s_{2,0}$, the view of $\mathcal{A}$ is identical to its view in Expt_0 and the bit $\tilde{b}$ (defined in Line 13 of the adversary $\mathcal{A}_{\mathsf{tIBE}}$) is identical to the output of Expt_0 (as defined in Fig. 4).
– Similarly, when $\mathcal{C}_{\mathsf{tIBE}}$ encrypts $s_{2,1}$, the view of $\mathcal{A}$ is identical to its view in Hyb_0 and the bit $\tilde{b}$ is identical to the output of Hyb_0.

Note that $\mathcal{A}_{\mathsf{tIBE}}$ outputs 1 when both of the following conditions are true:

– $\tilde{b} = 1$ (equivalent to Expt_0 or Hyb_0 outputting 1); and
– $\tau_{\mathsf{curr}} < \tau^*$ (equivalent to the event $\neg\mathsf{F}$ occurring)

Hence, it follows that,

$$\Pr[\mathsf{Expt}^{\mathsf{tIBE\text{-}cpa}}_{\mathcal{A},0} \Rightarrow 1] = \Pr[\mathsf{Expt}_0 \Rightarrow 1 \wedge \neg\mathsf{F}] \tag{3}$$

$$\Pr[\mathsf{Expt}^{\mathsf{tIBE\text{-}cpa}}_{\mathcal{A},1} \Rightarrow 1] = \Pr[\mathsf{Hyb}_0 \Rightarrow 1 \wedge \neg\mathsf{F}] \tag{4}$$

Subtracting Eq. (4) from Eq. (3), we get

$$\left|\Pr[\mathsf{Expt}^{\mathsf{tIBE\text{-}cpa}}_{\mathcal{A},0} \Rightarrow 1] - \Pr[\mathsf{Expt}^{\mathsf{tIBE\text{-}cpa}}_{\mathcal{A},1} \Rightarrow 1]\right| = \left|\Pr[\mathsf{Expt}_0 \Rightarrow 1 \wedge \neg\mathsf{F}] - \Pr[\mathsf{Hyb}_0 \Rightarrow 1 \wedge \neg\mathsf{F}]\right| \neq \mathsf{negl}(\lambda)$$

This contradicts the IND-CPA security of the tIBE scheme. □

Reduction algorithm $\mathcal{A}_{\mathsf{tIBE}}$:

1. Initialize $\tau_{\mathsf{curr}} := 0$ $\mathcal{Q}_{\mathsf{BDec}} \leftarrow \emptyset$
2. Receive $\mathsf{pp}_{\mathsf{tIBE}}$ from $\mathcal{C}_{\mathsf{tIBE}}$ and sample $\mathsf{pp}_{\mathsf{BTE}} \leftarrow \mathsf{BTE.Setup}(1^\lambda)$
3. Set $\mathsf{pp} := (\mathsf{pp}_{\mathsf{BTE}}, \mathsf{pp}_{\mathsf{tIBE}})$ and send pp to $\mathcal{A}$
4. Receive the corruption set $\mathcal{C}$ from $\mathcal{A}$ and forward it to the challenger $\mathcal{C}_{\mathsf{tIBE}}$
5. Receive $(\mathsf{mpk}_{\mathsf{tIBE}}, \{\mathsf{sk}_{\mathsf{tIBE},i}\}_{i\in\mathcal{C}})$ from $\mathcal{C}_{\mathsf{tIBE}}$ and sample $(\mathsf{mpk}_{\mathsf{BTE}}, \{\mathsf{sk}_{\mathsf{BTE},i}\}_{i\in[n]}) \leftarrow \mathsf{BTE.KeyGen}(n, t)$
6. Send $\mathsf{mpk} := (\mathsf{mpk}_{\mathsf{BTE}}, \mathsf{mpk}_{\mathsf{tIBE}})$, $\{\mathsf{sk}_i := \mathsf{sk}_{\mathsf{BTE},i}, \mathsf{sk}_{\mathsf{tIBE},i}\}_{i\in[\mathcal{C}]}$ to $\mathcal{A}$
7. Simulate the oracle $\mathcal{O}_{\mathsf{TInc}}$ for adversary $\mathcal{A}$ in the following way:
   (a) Update $\tau_{\mathsf{curr}} := \tau_{\mathsf{curr}} + 1$
   (b) For every query $Q = \{\mathsf{ct}_j\}_{j\in[B]}$ from $\mathcal{A}$, compute the set S as defined in Line 14 in Figure 4. Initiate an execution of the protocol $\Pi\text{-}\mathsf{BDec}(\mathsf{mpk}_{\mathsf{BTE}}, S)$ by simulating the set of honest parties $[n] \setminus \mathcal{C}$ using the secret keys $\{\mathsf{sk}_{\mathsf{BTE},i}\}_{i\in[n]\setminus\mathcal{C}}$, forward the protocol messages back and forth between the simulated set of honest parties and $\mathcal{A}$ (which is executing $\Pi\text{-}\mathsf{BDec}(\mathsf{mpk}_{\mathsf{BTE}}, S)$ on behalf of the set of corrupt parties $\mathcal{C}$). Let $\{\tilde{m}_i\}_{i\in[|S|]}$ be the output of the execution of $\Pi\text{-}\mathsf{BDec}(\mathsf{mpk}_{\mathsf{BTE}}, S)$ computed by the simulated honest parties.
   (c) Update $\mathcal{Q}_{\mathsf{BDec}} \leftarrow \mathcal{Q}_{\mathsf{BDec}} \cup S$
   (d) Send $\mathsf{id} := \tau_{\mathsf{curr}}$ to $\mathcal{C}_{\mathsf{tIBE}}$.
   (e) Once $\mathcal{C}_{\mathsf{tIBE}}$ initiates an execution of the protocol $\Pi\text{-}\mathsf{KeyIss}(\mathsf{id} := \tau_{\mathsf{curr}})$ on behalf of the set of honest parties $[n] \setminus \mathcal{C}$, forward the protocol messages back and forth between $\mathcal{C}_{\mathsf{tIBE}}$ and $\mathcal{A}$ (which is executing $\Pi\text{-}\mathsf{KeyIss}(\mathsf{id} := \tau_{\mathsf{curr}})$ on behalf of the set of corrupt parties $\mathcal{C}$). Let k_{curr} be the output of the execution of $\Pi\text{-}\mathsf{KeyIss}(\mathsf{id} := \tau_{\mathsf{curr}})$ computed by the $\mathcal{C}_{\mathsf{tIBE}}$ on behalf of honest parties. Receive k_{curr} from $\mathcal{C}_{\mathsf{tIBE}}$.
   (f) Send $(k_{\mathsf{curr}}, \{\tilde{m}_i\}_{i\in[|S|]})$ to $\mathcal{A}$.
8. Simulate the challenge phase for adversary $\mathcal{A}$ in the following way:
   (a) Receive the challenge (m_0, m_1, τ^*) from $\mathcal{A}$
   (b) Sample $s_{2,0}, s_{2,1} \leftarrow\$ \{0,1\}^\lambda$ and forward $(s_{2,0}, s_{2,1}, \mathsf{id} := \tau^*)$ to $\mathcal{C}_{\mathsf{tIBE}}$.
   (c) Receive the response ciphertext $\mathsf{ct}^*_{\mathsf{tIBE}}$ from $\mathcal{C}_{\mathsf{tIBE}}$ where $\mathsf{ct}^*_{\mathsf{tIBE}} \leftarrow \mathsf{tIBE.Enc}(\mathsf{mpk}_{\mathsf{tIBE}}, s_{2,0}, \mathsf{id} := \tau^*)$ or $\mathsf{ct}^*_{\mathsf{tIBE}} \leftarrow \mathsf{tIBE.Enc}(\mathsf{mpk}_{\mathsf{tIBE}}, s_{2,1}, \mathsf{id} := \tau^*)$
   (d) Sample $s_1 \leftarrow\$ \{0,1\}^\lambda$ and compute $\mathsf{ct}^*_{\mathsf{BTE}} \leftarrow \mathsf{BTE.Enc}(\mathsf{mpk}_{\mathsf{BTE}}, s_1, \mathsf{ad} := \tau^*)$
   (e) Compute $\mathsf{ct}^*_{\mathsf{PRG}} := m_0 \oplus \mathsf{PRG}(s_1 \oplus s_{2,0}, \ell)$
   (f) Set $\mathsf{ct}^* := (\tau^*, \mathsf{ct}^*_{\mathsf{BTE}}, \mathsf{ct}^*_{\mathsf{tIBE}}, \mathsf{ct}^*_{\mathsf{PRG}})$ and send ct^* to $\mathcal{A}$
9. Let b' be the bit output by $\mathcal{A}$. Define, $\tilde{b}$ as:

$$\tilde{b} = \begin{cases} 0 & : \text{if } \tau_{\mathsf{curr}} \geq \tau^* \ \wedge \ \mathsf{ct}^* \in \mathcal{Q}_{\mathsf{BDec}} \\ b' & : \text{otherwise} \end{cases}$$

10. If $\tau_{\mathsf{curr}} < \tau^*$, output $\tilde{b}$, otherwise output 0.

Fig. 7: Reduction algorithm $\mathcal{A}_{\mathsf{tIBE}}$ to break the IND-CPA security of tIBE.

Claim 4 $\left|\Pr[\mathsf{Expt}_0 \Rightarrow 1 \wedge \mathsf{E} \wedge \mathsf{F}] - \Pr[\mathsf{Hyb}_0 \Rightarrow 1 \wedge \mathsf{E} \wedge \mathsf{F}]\right| = 0$

Proof. The proof is trivial. When events E and F both occur, then Expt_0 always outputs 0 (see line 8 in Fig. 4), and so does Hyb_0. Therefore, $\Pr[\mathsf{Expt}_0 \Rightarrow 1 \wedge \mathsf{E} \wedge \mathsf{F}] = \Pr[\mathsf{Hyb}_0 \Rightarrow 1 \wedge \mathsf{E} \wedge \mathsf{F}] = 0$. □

Now that we have proved all the three intermediate lemmas, we move on to proving the claim. Note that the following holds.

$$\Pr[\mathsf{Expt}_0 \Rightarrow 1] = \Pr[\mathsf{Expt}_0 \Rightarrow 1 \wedge \neg\mathsf{F}] + \Pr[\mathsf{Expt}_0 \Rightarrow 1 \wedge \neg\mathsf{E} \wedge \mathsf{F}] + \Pr[\mathsf{Expt}_0 \Rightarrow 1 \wedge \mathsf{E} \wedge \mathsf{F}] \tag{5}$$

$$\Pr[\mathsf{Hyb}_0 \Rightarrow 1] = \Pr[\mathsf{Hyb}_0 \Rightarrow 1 \wedge \neg\mathsf{F}] + \Pr[\mathsf{Hyb}_0 \Rightarrow 1 \wedge \neg\mathsf{E} \wedge \mathsf{F}] + \Pr[\mathsf{Hyb}_0 \Rightarrow 1 \wedge \mathsf{E} \wedge \mathsf{F}] \tag{6}$$

Subtracting Eq. (6) from Eq. (5), we get

Reduction algorithm $\mathcal{A}_{\mathsf{PRG}}$:

1. Initialize $\tau_{\mathsf{curr}} \coloneqq 0$ and $\mathcal{Q}_{\mathsf{BDec}} \leftarrow \emptyset$
2. Sample $\mathsf{pp}_{\mathsf{BTE}} \leftarrow \mathsf{BTE.Setup}(1^\lambda, 1^{\mathsf{B}_{\mathsf{max}}})$ and $\mathsf{pp}_{\mathsf{tIBE}} \leftarrow \mathsf{tIBE.Setup}(1^\lambda)$
3. Set $\mathsf{pp} \coloneqq (\mathsf{pp}_{\mathsf{BTE}}, \mathsf{pp}_{\mathsf{tIBE}})$ and send pp to $\mathcal{A}$.
4. Receive the corruption set $\mathcal{C}$ from $\mathcal{A}$.
5. Sample $(\mathsf{mpk}_{\mathsf{BTE}}, \{\mathsf{sk}_{\mathsf{BTE},i}\}_{i\in[n]}) \leftarrow \mathsf{BTE.KeyGen}(n,t)$ and $(\mathsf{mpk}_{\mathsf{tIBE}}, \{\mathsf{sk}_{\mathsf{tIBE},i}\}_{i\in[n]}) \leftarrow \mathsf{tIBE.KeyGen}(n,t)$
6. Send $\mathsf{mpk}, \{\mathsf{sk}_i\}_{i\in\mathcal{C}}$ to $\mathcal{A}$.
7. Simulate the oracle $\mathcal{O}_{\mathsf{TInc}}$ for adversary $\mathcal{A}$ in the following way:
   (a) Update $\tau_{\mathsf{curr}} \coloneqq \tau_{\mathsf{curr}} + 1$
   (b) For every query $Q = \{\mathsf{ct}_j\}_{j\in[B]}$ from $\mathcal{A}$, compute the set S as defined in Line 14 in Figure 4. Initiate an execution of the protocol Π-$\mathsf{BDec}(\mathsf{mpk}_{\mathsf{BTE}}, S)$ by simulating the set of honest parties $[n] \setminus \mathcal{C}$ using the secret keys $\{\mathsf{sk}_{\mathsf{BTE},i}\}_{i\in[n]\setminus\mathcal{C}}$, forward the protocol messages back and forth between the simulated set of honest parties and $\mathcal{A}$ (which is executing Π-$\mathsf{BDec}(\mathsf{mpk}_{\mathsf{BTE}}, S)$ on behalf of the set of corrupt parties $\mathcal{C}$). Let $\{\tilde{m}_i\}_{i\in[|S|]}$ be the output of the execution of Π-$\mathsf{BDec}(\mathsf{mpk}_{\mathsf{BTE}}, S)$ computed by the simulated honest parties.
   (c) Update $\mathcal{Q}_{\mathsf{BDec}} \leftarrow \mathcal{Q}_{\mathsf{BDec}} \cup S$
   (d) Initiate an execution of the protocol Π-$\mathsf{KeyIss}(\tau_{\mathsf{curr}})$ by simulating the set of honest parties $[n] \setminus \mathcal{C}$ using the secret keys $\{\mathsf{sk}_{\mathsf{tIBE},i}\}_{i\in[n]\setminus\mathcal{C}}$, forward the protocol messages back and forth between the simulated set of honest parties and $\mathcal{A}$ (which is executing Π-$\mathsf{KeyIss}(\tau_{\mathsf{curr}})$ on behalf of the set of corrupt parties $\mathcal{C}$). Let k_{curr} be the output of the execution of Π-$\mathsf{KeyIss}(\tau_{\mathsf{curr}})$ computed by the simulated honest parties.
   (e) Send $(k_{\mathsf{curr}}, \{\tilde{m}_i\}_{i\in[|S|]})$ to $\mathcal{A}$.
8. Simulate the challenge phase for adversary $\mathcal{A}$ in the following way:
   (a) Receive the challenge (m_0, m_1, τ^*) from $\mathcal{A}$
   (b) Sample $s_1, s_2 \leftarrow\$ \{0,1\}^\lambda$
   (c) Compute $\mathsf{ct}^*_{\mathsf{BTE}} \leftarrow \mathsf{BTE.Enc}(\mathsf{mpk}_{\mathsf{BTE}}, s_1, \mathsf{ad} \coloneqq \tau^*)$ and $\mathsf{ct}^*_{\mathsf{tIBE}} \leftarrow \mathsf{tIBE.Enc}(\mathsf{mpk}_{\mathsf{tIBE}}, s_2, \mathsf{id} \coloneqq \tau^*)$
   (d) Receive a challenge string u of length ℓ from $\mathcal{C}_{\mathsf{PRG}}$ where $u \coloneqq \mathsf{PRG}(s)$ for $s \leftarrow\$ \{0,1\}^\lambda$ or $u \leftarrow\$ \{0,1\}^\ell$
   (e) Send $\mathsf{ct}^* \coloneqq (\tau^*, \mathsf{ct}^*_{\mathsf{BTE}}, \mathsf{ct}^*_{\mathsf{tIBE}}, \mathsf{ct}^*_{\mathsf{PRG}} \coloneqq m_0 \oplus u)$ to $\mathcal{A}$
9. Let b' be the bit output by $\mathcal{A}$. If $\tau_{\mathsf{curr}} \geq \tau^* \ \wedge \ \mathsf{ct}^* \in \mathcal{Q}_{\mathsf{BDec}}$, output 0. Otherwise, output b'.

Fig. 8: Reduction algorithm $\mathcal{A}_{\mathsf{PRG}}$ to break the pseudorandom property of PRG.

$$\begin{aligned}\Pr[\mathsf{Expt}_0 \Rightarrow 1] - \Pr[\mathsf{Hyb}_0 \Rightarrow 1] = &\Big(\Pr[\mathsf{Expt}_0 \Rightarrow 1 \wedge \neg\mathsf{F}] - \Pr[\mathsf{Hyb}_0 \Rightarrow 1 \wedge \neg\mathsf{F}]\Big) \\ &+ \Big(\Pr[\mathsf{Expt}_0 \Rightarrow 1 \wedge \neg\mathsf{E} \wedge \mathsf{F}] - \Pr[\mathsf{Hyb}_0 \Rightarrow 1 \wedge \neg\mathsf{E} \wedge \mathsf{F}]\Big) \\ &+ \Big(\Pr[\mathsf{Expt}_0 \Rightarrow 1 \wedge \mathsf{E} \wedge \mathsf{F}] - \Pr[\mathsf{Hyb}_0 \Rightarrow 1 \wedge \mathsf{E} \wedge \mathsf{F}]\Big) \qquad (7)\end{aligned}$$

Now, by invoking Theorem 2, Theorem 3 and Theorem 4 in Eq. (7), it follows that:

$$\Big|\Pr[\mathsf{Expt}_0 \Rightarrow 1] - \Pr[\mathsf{Hyb}_0 \Rightarrow 1]\Big| = \mathsf{negl}(\lambda)$$

This completes the proof of our claim. □

Lemma 2. $\mathsf{Hyb}_0 \approx_c \mathsf{Hyb}'$.

Proof. Suppose that the claim is false w.r.t an adversary $\mathcal{A}$ that participates in Hyb_0 and Hyb'. We can use $\mathcal{A}$ to construct an adversary $\mathcal{A}_{\mathsf{PRG}}$ that contradicts the security of the PRG scheme. Let $\mathcal{C}_{\mathsf{PRG}}$ be the challenger for PRG scheme.

Let $\mathsf{Expt}_{\mathsf{PRG}}$ and $\mathsf{Expt}_{\mathsf{Rand}}$ denote the experiment where $\mathcal{C}_{\mathsf{PRG}}$ sets $u \coloneqq \mathsf{PRG}(s)$ and $u \leftarrow\$ \{0,1\}^\ell$ respectively. The output of these experiments, as defined in PRG security game, is equal to the bit received by the $\mathcal{C}_{\mathsf{PRG}}$ from the reduction adversary $\mathcal{B}$. By the construction of $\mathcal{B}$, we note the following:

– When $\mathcal{C}_{\mathsf{PRG}}$ sets $u \coloneqq \mathsf{PRG}(s)$ for $s \leftarrow\$ \{0,1\}^\lambda$, the view of $\mathcal{A}$ is identical to its view in Hyb_0 and the output of $\mathcal{A}_{\mathsf{PRG}}$ (see Line 12 of Fig. 8) is identical to the output of Hyb_0.
– Similarly, when $\mathcal{C}_{\mathsf{PRG}}$ samples $u \leftarrow\$ \{0,1\}^\ell$, the view of $\mathcal{A}$ is identical to its view in Hyb' and the output of $\mathcal{A}_{\mathsf{PRG}}$ is identical to the output of Hyb'.

Therefore, we get:

$$\Big|\Pr[\mathsf{Expt}_{\mathsf{PRG}} \Rightarrow 1] - \Pr[\mathsf{Expt}_{\mathsf{Rand}} \Rightarrow 1]\Big| = \Big|\Pr[\mathsf{Hyb}_0 \Rightarrow 1] - \Pr[\mathsf{Hyb}' \Rightarrow 1]\Big| \neq \mathsf{negl}(\lambda).$$

This contradicts the security of the PRG scheme and concludes the proof of our claim. □

Lemma 3. $\mathsf{Hyb}' \approx \mathsf{Hyb}_1$.

Proof. The proof for this claim follows from the security of the PRG scheme. Formally, the proof is identical to the proof of Lemma 2 ($\mathsf{Hyb}_0 \approx_c \mathsf{Hyb}'$) with the exception that m_0 is replaced with m_1 when the reduction adversary $\mathcal{B}$ creates the ciphertext $\mathsf{ct}^*_{\mathsf{PRG}}$. □

Lemma 4. $\mathsf{Hyb}_1 \approx_c \mathsf{Expt}_1$.

Proof. The proof for this claim follows from the IND-CCA security of the BTE scheme and IND-CPA security of the threshold IBE scheme. Formally, the proof is identical to the proof of Lemma 1 ($\mathsf{Expt}_0 \approx_c \mathsf{Hyb}_0$) with the exception that m_0 is replaced with m_1 when the reduction adversary $\mathcal{B}$ creates the ciphertext $\mathsf{ct}^*_{\mathsf{PRG}}$. □

Next, we will proof robustness of our scheme, assuming that the underlying schemes BTE and tIBE is robust as per Definitions 4 and 8, respectively. Formally,

Theorem 5 (tTLES Robustness). *Assuming the existence of an robust threshold Identity Based Encryption (IBE) scheme (Definition 4) and a robust Batched Threshold Encryption (BTE) scheme (Definition 8), our tTLES protocol* tTLES *(Fig. 5) is robust as per Definition 13.*

Proof. Any ciphertext ct in our tTLES is a tuple of four elements, i.e., $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{ct}_3, \mathsf{ct}_4)$ where ct_2 is a BTE ciphertext of random $s_1 \in \{0,1\}^\lambda$ with associated data τ, ct_3 is a tIBE ciphertext encrypting another random value s_2 under identity τ, and $\mathsf{ct}_4 = m \oplus \mathsf{PRG}(s_1 \oplus s_2, |m|)$.

Consequently, the tTLES decryption has two main steps: (i) decrypting ct_2 to recover s_1, and (ii), decrypting ct_3 to recover s_2. Clearly, if s_1 and s_2 are correct, since ct_4 is a deterministic function of s_1, s_2 and the underlying message m, we will correctly recover original m.

Next, we will show that except with negligible probability, both s_1 and s_2 are correctly recovered for all honestly encrypted messages in a batch. Let $\mathsf{F}_{\mathsf{BTE}}$ be the event that BTE decryption of any honestly encrypted message in the batch output by the adversary fails, and $\mathsf{F}_{\mathsf{tIBE}}$ be the event that tIBE decryption of any honestly encrypted message in the batch fails. Then, from Definitions 4 and 8, we have that there exist negligible functions $\mathsf{negl}_1(\cdot), \mathsf{negl}_2(\cdot)$ such that:

$$\Pr[\mathsf{F}_{\mathsf{BTE}}] = \Pr[\mathsf{Expt}^{\mathsf{BTE\text{-}rob}}_{\mathcal{A}}(1^\lambda) \Rightarrow 1] \leq \mathsf{negl}_1(\lambda), \quad \text{and} \quad \Pr[\mathsf{F}_{\mathsf{tIBE}}] = \Pr[\mathsf{Expt}^{\mathsf{tIBE\text{-}rob}}_{\mathcal{A}}(1^\lambda) \Rightarrow 1] \leq \mathsf{negl}_2(\lambda)$$

Now, using union bound, we have that:

$$\Pr[\mathsf{F}_{\mathsf{BTE}} \vee \mathsf{F}_{\mathsf{tIBE}}] \leq \mathsf{negl}_1(\lambda) + \mathsf{negl}_2(\lambda)$$

Finally, if neither $\mathsf{F}_{\mathsf{BTE}}$ nor $\mathsf{F}_{\mathsf{tIBE}}$ occurs, then all honestly encrypted messages in the batch are correctly decrypted, i.e.,

$$\Pr[\mathsf{Expt}^{\mathsf{tTLES\text{-}rob}}_{\mathcal{A}} \Rightarrow 1] \leq \Pr[\mathsf{F}_{\mathsf{BTE}} \vee \mathsf{F}_{\mathsf{tIBE}}] \leq \mathsf{negl}(\lambda)$$

for some negligible function $\mathsf{negl}(\cdot)$. This completes the proof that our tTLES protocol is robust. □

5 Discussion and Conclusion

In this paper, we presented time-lock encrypted storage (tTLES), a storage service provided by blockchains where clients can submit (using transactions) encrypted values to be decrypted at a future time τ_{tgt} (measured in block height). The security guarantees of tTLES require that the submitted values can only be decrypted after the time τ_{tgt} has elapsed (the time-lock indistinguishability property) and the transactions submitting the encrypted values have been included in the blockchain (the non-inclusion privacy property). We have provided a formal definition of tTLES and proposed an efficient protocol that relies on (in a black-box manner) a threshold identity-based encryption (tIBE) scheme and a recent batch threshold decryption (BTE) scheme. Our proposed solution allows for an unbounded number of ciphertexts to be encrypted towards any decryption time τ_{tgt} while keeping the decryption cost (both communication and computation) at time τ_{tgt} independent of the number of such ciphertexts.

We suggest two natural extensions of our tTLES design.

Generalizing to other decryption policies. Our tTLES protocols guarantee time-lock indistinguishability and non-inclusion privacy by secret sharing the ephemeral secret key $(s_1 \oplus s_2)$ with 2-out-of-2 secret sharing, encrypting the share s_1 with tIBE and the other share s_2 with BTE. This ensures that the decryption requires both shares s_1 and s_2. A natural extension of this approach is to support more expressive decryption policies, such as t-out-of-n decryption policies by secret sharing the ephemeral key using threshold secret sharing [17]. We note that these decryption conditions can be either internal to the blockchain protocols or can be fed in externally using external oracles.

Relaxing the encrypted storage inclusion condition. Our tTLES construction works with the inclusion condition that each encrypted storage encrypted towards a target decryption time τ_{tgt} gets included in a block before the time τ_{tgt} has elapsed. Although this inclusion condition is sufficient for the applications we discussed, some applications can benefit from a more flexible inclusion condition. For example, we may allow the client to specify two timestamps: the target decryption time τ_{tgt}, and an expiry time τ_{exp}, with possibly $\tau_{\mathsf{exp}} \geq \tau_{\mathsf{tgt}}$. Under this relaxed condition, encrypted values may still be included after τ_{tgt}, enabling use cases where users want ciphertexts to remain valid candidates for inclusion even after τ_{tgt} has passed.

Acknowledgements

We thank John Bergschneider, Dan Boneh, Mike Setrin and Ayush Yadav for their feedback and helpful discussions related to this paper. We specially thank Michael Setrin for discussions that motivated this work.

References

1. Agarwal, A., Fernando, R., Pinkas, B.: Efficiently-thresholdizable batched identity based encryption, with applications. In: Annual International Cryptology Conference. pp. 69–100. Springer (2025)
2. Boneh, D., Franklin, M.: Identity-based encryption from the weil pairing. In: Annual international cryptology conference. pp. 213–229. Springer (2001)
3. Boneh, D., Laufer, E., Tas, E.N.: Batch decryption without epochs and its application to encrypted mempools. Cryptology ePrint Archive (2025)
4. Boneh, D., Shoup, V.: A graduate course in applied cryptography. Draft 0.6 (2023)
5. Bormet, J., Choudhuri, A.R., Faust, S., Garg, S., Othman, H., Policharla, G.V., Qu, Z., Wang, M.: Beast-mev: Batched threshold encryption with silent setup for mev prevention. Cryptology ePrint Archive (2025)
6. Bormet, J., Faust, S., Othman, H., Qu, Z.: {BEAT-MEV}: Epochless approach to batched threshold encryption for {MEV} prevention. In: 34th USENIX Security Symposium (USENIX Security 25). pp. 3457–3476 (2025)
7. Cerulli, A., Connolly, A., Neven, G., Preiss, F.S., Shoup, V.: vetkeys: How a blockchain can keep many secrets. Cryptology ePrint Archive (2023)
8. Choudhuri, A.R., Garg, S., Piet, J., Policharla, G.V.: Mempool privacy via batched threshold encryption: Attacks and defenses. In: 33rd USENIX Security Symposium (USENIX Security 24). pp. 3513–3529 (2024)

9. Choudhuri, A.R., Garg, S., Policharla, G.V., Wang, M.: Practical mempool privacy via one-time setup batched threshold encryption. In: 34th USENIX Security Symposium (USENIX Security 25). pp. 3477–3495 (2025)
10. Ding, Q., Liebau, D., Wang, Z., Xu, W.: A survey on decentralized autonomous organizations (daos) and their governance. World Scientific Annual Review of Fintech **1**, 2350001 (2023)
11. Dujmovic, J., Garg, R., Malavolta, G.: Time-lock puzzles with efficient batch solving. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 311–341. Springer (2024)
12. Dziembowski, S., Faust, S., Luhn, J.: Shutter network: Private transactions from threshold cryptography. Cryptology ePrint Archive (2024)
13. Gailly, N., Melissaris, K., Romailler, Y.: tlock: Practical timelock encryption from threshold bls. Cryptology ePrint Archive (2023)
14. Galal, H.S., Youssef, A.M.: Succinctly verifiable sealed-bid auction smart contract. In: International Workshop on Data Privacy Management. pp. 3–19. Springer (2018)
15. Mahmoody, M., Moran, T., Vadhan, S.: Time-lock puzzles in the random oracle model. In: Annual Cryptology Conference. pp. 39–50. Springer (2011)
16. Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock puzzles and timed-release crypto (1996)
17. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (Nov 1979). `https://doi.org/10.1145/359168.359176`, `https://doi.org/10.1145/359168.359176`
18. Shamir, A.: Identity-based cryptosystems and signature schemes. In: Workshop on the theory and application of cryptographic techniques. pp. 47–53. Springer (1984)
19. Xiong, J., Wang, Q.: Anonymous auction protocol based on time-released encryption atop consortium blockchain. arXiv preprint arXiv:1903.03285 (2019)