PR-TDR: Privacy-preserving and Reliable Timed Data Release

Jingzhe Wang, Balaji Palanisamy
University of Pittsburgh, School of Computing and Information, Pittsburgh, PA, USA

***Abstract*—Timed Data Release(TDR) is a practical security mechanism that safeguards data until a prescribed time has elapsed. However, existing TDR frameworks do not focus on reliability guarantees and lack formal security analysis. To this end, we propose PR-TDR, a novel framework that supports privacy-preserving and reliable timed data release while providing provable security properties. PR-TDR includes two novel contributions: a formal privacy-preserving design for TDR, named P-TDR and a reliable lifetime secret key management built on top of P-TDR that systematically empowers P-TDR with reliability. P-TDR prevents adversaries from accessing the data prior to the prescribed release time. At the core of the design of P-TDR, a group of decentralized peers, which operates under an honest-majority assumption, collaboratively takes charge of managing the lifetime secret key. Each peer stores a key share of the secret key. The proposed reliability layer that empowers P-TDR with reliability guarantees incorporates two carefully designed protocols that operate before the prescribed release time, namely the lifetime secret key auditing protocol and the lifetime secret key share recovery protocol. The auditing protocol enables a *semi-honest auditor* to confirm the availability of the lifetime secret key with the peers while not gaining any knowledge about the secret key itself. The recovery protocol allows peers that have lost their respective shares of the lifetime secret key to recover them with the help of other peers, ensuring that the lifetime secret key remains private. We provide formal security proof to demonstrate that PR-TDR satisfies the desired security properties. We implement our framework using Ethereum and our performance evaluations confirm that PR-TDR not only embodies the desired security properties but also operates efficiently.**

***Index Terms*—Timed Release, Blockchain, Smart Contract, Timed Cryptography**

I. Introduction

Timed Data Release (TDR) is a practical mechanism designed to protect sensitive data by making it accessible only after a period of time has passed. Several real-world applications benefit from TDR. For example, in secure voting mechanisms, votes should not be accessible until the polling process has concluded. In recent years, there has been several efforts on developing practical and decentralized TDR constructions, focusing on various aspects including (1) safeguarding TDR against rational adversaries [23], [24] (2)incorporating reputation-based techniques to enhance resilience against attacks [35], [37] (3) enriching TDR with controllable functionalities. However, these techniques have notable limitations:

Lack of Reliable Constructions: Reliability in TDR encompasses two aspects: data availability checks and fault tolerance, both of which are crucial for real-world applications. Data availability checks ensure that the data is accessible at a time prior to a prescribed release time. For instance, in secure voting, a voter may delegate the protection of their ballot to the TDR service but might want to verify its availability before polling process begins. Additionally, in cases where ballots are missing due to failures, fault-tolerant services can facilitate the recovery of the lost ballots in TDR. Existing constructions have generally overlooked reliability.

Lack of Formal Security Guarantees: In demonstrating the privacy of TDR, existing constructions either show that rational adversaries [12] will not attack under the game theory assumptions or provide attack-resilient analysis that indicate how likely it is for data to preserve privacy. As a result, these constructions do not provide formal cryptographic privacy guarantees for TDR. In light of theses limitations, we naturally ask the following research question :

Can we construct an efficient, privacy-preserving, and reliable TDR while entailing formal security guarantees?

To this end, we introduce PR-TDR, a novel and practical TDR design that supports privacy-preserving timed-release data protection and reliability while offering formal security guarantees. We approach the design of PR-TDR by first designing a privacy-preserving version of TDR, coined P-TDR. A natural approach to enabling cryptographic privacy is to leverage the recent threshold timed-release encryption construction, Ti-TiRE, introduced in [3]. Although this straightforward adoption may yield cryptographic privacy guarantee that no probablistic polynomial time (PPT) adversary can get the data before prescribed release time, it still cannot meet TDR's requirement that the data be encrypted to a specific receiver. The primary challenge here is that the security definition in Ti-TiRE only considers one type of adversary, namely corrupted peers, while in P-TDR we have two type of adversaries, namely corrupted peers and the receiver. To address this, our P-TDR incorporates Ti-TiRE with IND-CPA-Secure public key encryption scheme, functioning as a hybrid encryption scheme, to offer formal privacy-preserving guarantee for TDR while maintaining efficiency.

At the core of P-TDR, we have a group of decentralized peers collaboratively managing the lifetime secret key, serving as a time reference. Each peer holds a secret share of the lifetime secret key. Such shares are derived from $(t+1, n)$ shamir's secret sharing scheme [30] during a trusted setup. The parameter t means that at least $t+1$ correct shares can recover the lifetime secret key. We also assume

that the group of peers operates under an *honest-majority* assumption ($t < \frac{n}{2}$) in which an adversary can corrupt no more than t peers.

Building on top of P-TDR, we design a reliable lifetime secret key management that systematically empowers P-TDR with reliability. The reliable secret key management involves two protocols: lifetime secret key auditing protocol PR-TDR.Audit and lifetime secret key share recovery protocol PR-TDR.Recover

PR-TDR.Audit allows individual peer to prove to an external *semi-honest* auditor that his/her held share of the lifetime secret key is indeed the one generated by the trusted setup, without disclosing any information about the share. While existing PoR/PDP [1], [18] techniques seem to work for our scheme, it does not preserve privacy when performing integrity check. Therefore, inspired by [3], we construct PR-TDR.Audit with the help of *non-interactive zero knowledge proof* (NIZK) [14]. Precisely, the *semi-honest* auditor broadcasts an audit request to the group of peers. Then, each peer then generates a proof and sends it back to the auditor. Once the auditor receives $t+1$ successful proofs, it is convinced of the lifetime secret key's availability.

PR-TDR.Recover is designed to address the situation where one peer loses its held lifetime secret key share due to failures. A natural approach to enable recovery would be to adopt the share recovery techniques from proactive secret sharing [17] and privacy-preserving byzantine fault-tolerant state machine replications (BFT-SMR) [4], [34]. However, such constructions either cater to proactive security or operates in asynchronous BFT-SMR scenario, which are not suitable for P-TDR. Therefore, we extract the underlying design philosophy and construct a new protocol, PR-TDR.Recover, tailored for our case. Our PR-TDR.Recover works in synchronous network model and assumes a *honest-majority* assumption. In essence, PR-TDR.Recover allows any peer requiring recovery service to broadcast requests to other peers. The other peers then collaboratively generate a new polynomial, which encodes the lost share of the requesting peer. They then send corresponding shares of this new polynomial to the recovering peer. With theses shares in hand, the recovering peer can interpolate the new polynomial and retrieve his lost share. To address malicious adversarial attacks where invalid shares might be sent, we employ a polynomial commitment scheme [19] to make the protocol verifiable.

We provide rigorous security proofs to demonstrate that PR-TDR satisfies the desired security properties. We implement PR-TDR and conduct extensive evaluations, with results showing that PR-TDR is efficient.

In summary, this paper makes the following key contributions:

- We present the formal investigation into mechanism designs for privacy-preserving and reliable time data release.
- We propose PR-TDR, an efficient privacy-preserving and reliable timed data release design while preserving formal security guarantee. PR-TDR consists of P-TDR, a privacy-preserving timed data release mechanism, and a reliable lifetime secret key management, including the lifetime secret key auditing protocol (PR-TDR.Audit) and the lifetime secret key share recovery protocol (PR-TDR.Recover).
- We define formal security models and present constructions for P-TDR, PR-TDR.Audit, and PR-TDR.Recover
- We rigorously prove that P-TDR, PR-TDR.Audit, and PR-TDR.Recover satisfy the desired security properties.
- Finally, we implement our framework and demonstrate its efficiency and effectiveness.

Roadmap The rest of the paper is organized as follows. Section II gives an overview of PR-TDR. In Section III, we provide preliminaries adopted in this paper. In Section IV, we formally define PR-TDR. We construct PR-TDR in Section V. In Section VI, we prove security properties of PR-TDR. Section VII demonstrates implementation and evaluations of PR-TDR. In Section VIII, we perform literature review. In Section X, we conclude this paper.

II. PR-TDR: A Bird's-eye View

In this section, we provide a high-level overview of PR-TDR.

A. Key Entities

PR-TDR consists of the following key entities:

- *Data Sender* : Data sender S prepares the data, specifies the prescribed release time, and the data receiver. S encrypts the data and sends it to receiver.
- *Data Receiver*: Data receiver R stores the encrypted data sent from S and aims to decrypt the data after prescribed release time.
- *Peers*: A set $\mathcal{P}$ of decentralized peers collaboratively manages the lifetime secret key and provides time reference.
- *Auditor*: External auditor V performs the lifetime secret key auditing at any time point before the release time.
- *Ethereum Blockchain*: In our framework, we use Ethereum for the smart contract service that stores critical service information, such as epoch-based decryption key discussed later.

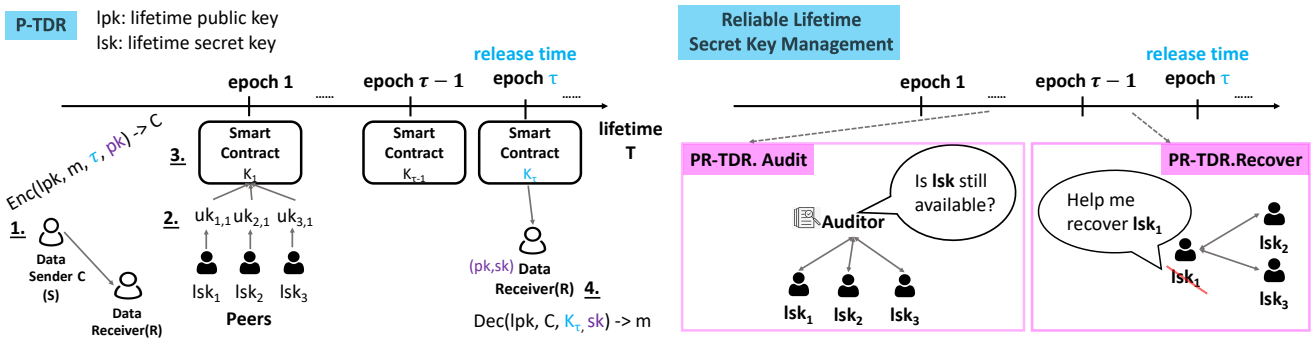
We next present the workflow of PR-TDR involving the entities described above.

B. PR-TDR: Workflow

PR-TDR initiates with P-TDR and, subsequently, reliable lifetime secret key management is activated upon request. The detailed workflow is as follows:

1) P-TDR: In P-TDR, time is quantified in terms of epochs. The maximum number of epochs that PR-TDR can accommodate is referred to as the *lifetime*, denoted as T.

We next give an example to show how P-TDR works. In Fig. 1a, suppose S would like to send R data m that becomes accessible after τ. To realize it, S specifies release time τ for m and adopts the encryption algorithm, provided by P-TDR.Enc, to get ciphertext c. S then hands R the ciphertext c and can subsequently go offline. To denote the passage of time, the group of time serves $\mathcal{P}$, each holding a secret share of lifetime secret key lsk, undertakes the following operations at each epoch:(1) generates an update key, and (2) uploads it



(a) P-TDR

(b) Reliable Lifetime Secret Key Management

Fig. 1: An Overview of PR-TDR

to the smart contract. For example, as illustrated in Fig. 1a, $P_1 \in \mathcal{P}$ generates update key $uk_{1,1}$ from lsk_1 corresponding to epoch 1 and uploads $uk_{1,1}$ to the smart contract.

Once getting enough update keys from the peers, the smart contract combines them and derives decryption key K_1. This procedure recurs at each epoch within T. Once the time reaches τ, the designated release time, R retrieves K_τ from the smart contract. By applying decryption algorithm of P-TDR.Dec with the ciphertext c, the decryption key K_τ, and private key sk, R decrypts and gets m. We provide a formal description of P-TDR in Section IV-A and the construction of P-TDR in Section V-A.

2) Reliable Lifetime Secret Key Management: On top of P-TDR, we design reliable lifetime secret key management (Fig. 1b) to provide the following two services: (1) ensure that lifetime secret key is available, and (2) any peer that has lost their share of the lifetime secret key can get it recovered. The availability service is handled by our auditing protocol PR-TDR.Audit where an external auditor can verify the availability of the lifetime secret key by verifying the availability of the shares of the secret key. For example, in Fig. 1b, when an auditor would like to verify whether the lifetime secret key lsk is available, they can verify the availability of each corresponding share lsk_1, lsk_2, and lsk_3. We formalize PR-TDR.Audit in Section IV-B1 and provide the construction in Section V-B1.

The recovery service is handled by our recovery protocol PR-TDR.Recover. PR-TDR.Recover allows any peer that has lost its share to request recovery. For instance, in Fig. 1b, a peer that has lost lsk_1 can request its recovery from the other two peers. Following the execution of PR-TDR.Recover, lsk_1 will be recovered. We present a formal description of PR-TDR.Recover in Section IV-B2 and its construction in Section V-B2.

III. Preliminaries

In this section, we discuss the preliminaries and provide a detailed background of the building blocks that we adopt in our construction.

A. Shamir's Secret Sharing Scheme

The notion of $(t+1, n)$ secret sharing proposed by [30], enables one to distribute shares of a secret to a group of n parties, while ensuring that an adversary controlling t or fewer of n shares learns no information on the secret. To recover the secret, at least $t+1$ correct shares are needed. We use $\mathsf{sss}(s, t+1, n)$ to denote that Shamir's construction has been adopted to generate shares by inputting the secret s, such that at least $t+1$ correct shares out of n can recover s. The shares are represented by a vector $(s_1, s_2, ...s_n)$. Additionally, s satisfies the following property: there exists a polynomial $f(\cdot) \in \mathbb{F}[X]$ with a degree of at most t where $f(0) = s$ and $f(i) = s_i$ for $i \in \{1, ..., n\}$.

B. Polynomial Commitment Scheme

Polynomial commitment scheme is a cryptographic primitive that allows a committer to commit to a polynomial with short string that can later be adopted by a verifier to confirm claimed evaluations of the committed polynomial. In our work, we utilize the following key algorithms:

- PC.Setup$(1^k, t) \rightarrow (cpk, csk)$: Given a security parameter k and polynomial degree at most t, this algorithm generates a public-private key pair (cpk, csk).
- PC.Commit$(cpk, f(\cdot)) \rightarrow com(f(\cdot))$: Given cpk and polynomial $f(\cdot)$ with at most degree of t, this algorithm generates the commitment $com(f(\cdot))$ to $f(\cdot)$.
- PC.CreateWitness$(cpk, f(\cdot), i) \rightarrow (i, f(i), w_i)$: Given $cpk, f(\cdot), i$ this algorithm outputs $(i, f(i), w_i)$, where w_i is a witness for the evaluation $f(i)$ of $f(\cdot)$ at i.
- PC.VerifyEval$(cpk, com, i, f(i), w_i) \rightarrow \{0, 1\}$: This algorithm verifies $f(i)$ is indeed the evaluation at i of the polynomial committed in com. If this holds true, the algorithm outputs 1, otherwise it outputs 0.

We also adopt the Discrete Logarithm-based construction proposed by Kate et.al [19], which possesses two key properties namely *binding* and *hiding*. The binding property asserts that the committed polynomial cannot be altered once it is committed, while the hiding property guarantees that the commitment does not reveal the polynomial itself.

C. Threshold Incremental Timed-Release Encryption

Threshold Incremental Timed-release Encryption (Ti-TiRE), proposed in [3], is an encryption scheme that supports timed-release encryption in general. Specifically, Ti-TiRE consists of the following algorithms:

- Ti-TiRE.Setup$(1^k, T, n, t) \rightarrow (pp, lpk, (lsk_1, ..., lsk_n))$: This algorithm takes as input the security parameter 1^k and the lifetime duration T in terms of the number of epochs and outputs a public parameter pp, a lifetime public key lpk and a list of n shares of life time secret key $(lsk_1, ..., lsk_n)$.
- Ti-TiRE.PartUKGen$(lsk_j, \tau) \rightarrow uk_{j,\tau}$: On input a share of lifetime secret key lsk_j and an epoch $\tau \in \{1, ..., T\}$, PartUKGen(lsk_j, τ) outputs a partial update key $uk_{j,\tau}$ corresponding to the epoch τ and user j.
- Ti-TiRE.UKCombine$(u_{1,\tau}, ..., u_{t+1,\tau}) \rightarrow u_\tau$: Given the list of partial update keys, the algorithm recovers the whole update key uk_τ corresponding to τ.
- Ti-TiRE.DKGen$(K_{\tau-1}, uk_\tau) \rightarrow K_\tau$: On input a decryption key for epoch $\tau - 1$ and an update key for τ, this algorithm outputs a decryption key K_τ for epoch τ.
- Ti-TiRE.Enc$(lpk, m, \tau) \rightarrow c$: This algorithm uses the lifetime public key lpk to encrypt a message m locked until epoch τ, and outputs ciphertext c
- Ti-TiRE.Dec$(lpk, K, c) \rightarrow m/\perp$: On input the lifetime public key lpk, the decryption key K, and the ciphertext c, this algorithm outputs m or failure.

Ti-TiRE satisfies correctness property and is IND-CPA secure [20] in the random oracle model. For the formal definition, we refer the interested readers to [3].

D. Public Key Encryption

We also rely on public key encryption schemes. A public key encryption scheme is denoted as PK = (Gen,Enc,Dec) with message space $\mathcal{M} = \{0,1\}^l$. Specifically, PK.Gen is a key generation algorithm, PK.Enc is an encryption algorithm, and PK.Dec is a decryption algorithm. We assume that PK satisfies correctness property and is IND-CPA secure [20]. We use ElGamal encryption [13] as an instantiation and we refer the interested readers to [13] for more details.

E. Non-interactive Zero Knowledge

Non-interactive zero-knowledge proofs NIZK allow one (prover) to prove validity of some statement to another (verifier) in such a way that nothing except the validity of the statement is disclosed and no interaction between the prover and verifier is required. In general, NIZK consists of two algorithms NIZK.Prove and NIZK.Verify. We use the construction proposed in [14] as our instantiation and assume that it satisfies two key properties: perfect completeness and zero-knowledge.

F. Blockchains

In our design, we employ *Ethereum* [38] as our blockchain environment. Ethereum incorporates the concept of smart contracts [32]. Roughly, an Ethereum smart contract is a piece of computer code executed and stored on the Ethereum blockchain network. We note that function invocations on smart contracts typically incur cost, namely gas cost [8]. These costs are represented in Ether [8], the cryptocurrency used within Ethereum.

IV. PR-TDR : Formal Definition

In this section, we present the formal definition of the various components involved in the construction of PR-TDR.

A. P-TDR: Formal Definition & Security Model

We first formally define P-TDR as follows:

Definition 1. *A P-TDR scheme is defined by seven algorithms associated with message space $\mathcal{M} = \{0,1\}^l$ and ciphertext space $\mathcal{C}$. P-TDR involves the following parties: a set $\mathcal{P}$ of peers , the sender S and the receiver R. The seven algorithms are as follows:*

- P-TDR.Setup$(1^k, T, n, t) \rightarrow (pp, lpk, (lsk_1, ..., lsk_n))$: Run by a trusted party, this algorithm takes as input key parameters, involving security parameter k, the lifetime duration T in terms of number of time epochs, total number of peers n, and the threshold parameters t. This algorithm outputs public parameters pp, a lifetime public key lpk, and a set $(lsk_1, ..., lsk_n)$ of shares of a lifetime secret key lsk and distribute the set to $\mathcal{P}$.
- P-TDR.KeyGen$(1^k) \rightarrow (pk, sk)$: Run by receiver R, this algorithm takes as input the security parameter k and outputs a key pair (pk, sk).
- P-TDR.PartUKGen$(lsk_i, \tau) \rightarrow uk_{i,\tau}$: Run by a peer $P_i \in \mathcal{P}$. On input of their share of lifetime secret key lsk_i and current time epoch τ, this algorithm outputs partial update key $uk_{i,\tau}$ of P_i corresponding to epoch τ.
- P-TDR.UKCombine$(uk_{1,\tau}, ..., uk_{t+1,\tau}) \rightarrow uk_\tau$: On input of a list of partial update keys $(uk_{1,\tau}, ..., uk_{t+1,\tau})$, this algorithm combines them to derive uk_τ.
- P-TDR.DKGen$(K_{\tau-1}, uk_\tau) \rightarrow K_\tau$: On input of a decryption key $K_{\tau-1}$ for the epoch $(\tau - 1)$ and update key u_τ for the epoch τ, this algorithm generates decryption key K_τ for current epoch τ.
- P-TDR.Enc$(lpk, m, \tau, pk) \rightarrow c$: Run by sender S, this algorithm takes as input the lifetime pulbic key lpk, a message $m \in \mathcal{M}$, a pre-scribed release time epoch τ, the public key pk of R , and output a ciphertext $c \in \mathcal{C}$.
- P-TDR.Dec$(lpk, K, c, sk) \rightarrow m/\perp$: Run by receiver R, this algorithm takes as input lpk, a ciphertext c, a decryption key K, and a private key sk, and outputs either a message m or a failure symbol $\perp$.

Given the above formal definition, we next define the desired properties of P-TDR. We first capture the correctness property.

Definition 2. (Correctness) P-TDR satisfies correctness if $c =$ P-TDR.Enc(lpk, m, τ, pk), then $m =$ P-TDR.Dec(lpk, c, K_τ, sk). Here, lpk is generated by PR-TDR.Setup, (pk, sk) is generated by P-TDR.KeyGen, and K_τ is generated by P-TDR.DKGen on input τ.

Next, we introduce security models of P-TDR. In P-TDR, we consider two kinds of adversaries: (1) a malicious adversary who corrupts at most t out of n peers and would like to get original data before prescribed release time, and (2) a curious receiver who intends to perform decryption before the release time τ. We formally define security models of P-TDR

in form of security games. In our definition, we consider the two types of adversaries separately.

Definition 3. (IND-TS-CPA Game - $\mathsf{Game}^{TS}_{\text{P-TDR}}$)
- **Setup**: The challenger $\mathcal{C}$ runs P-TDR.Setup to generate lpk, pp, and the list of $(lsk_1, ..., lsk_n)$. $\mathcal{C}$ runs P-TDR.KeyGen to get a key pair (pk, sk). Then, $\mathcal{C}$ gives $\mathcal{A}$ pp, lpk, and pk.
- **Corruption**: $\mathcal{A}$ outputs the identities I of the set of corrupted peers, where $I \subset \{1, ..., n\}$ such that $|I| \leq t$. Then, $\mathcal{C}$ gives $(lsk_i)_{i \in I}$ to $\mathcal{A}$.
- **Challenge**: $\mathcal{A}$ selects two messages m_0 and m_1 where $m_0, m_1 \in \mathcal{M}$ and a prescribed release time τ and passes m_0, m_1, τ to $\mathcal{C}$. $\mathcal{C}$ chooses a random bit b and computes $c = \text{P-TDR.Enc}(lpk, m_b, \tau, pk)$ and passes c to $\mathcal{A}$
- **Guess**: $\mathcal{A}$ outputs his guess b' on b and outputs b'

Then, we define $\mathcal{A}'$s advantage in $\mathsf{Game}^{TS}_{\text{P-TDR}}$ as $\mathsf{Adv}_{\mathcal{A}}(k) = \left|Pr[b' = b] - \frac{1}{2}\right|$

Definition 4. P-TDR scheme is IND-TS-CPA secure if for all polynomial time adversaries $\mathcal{A}$ the function $\mathsf{Adv}_{\mathcal{A}}(k)$ in $\mathsf{Game}^{TS}_{\text{P-TDR}}$ is negligible.

Definition 5. (IND-R-CPA Game - $\mathsf{Game}^{R}_{\text{P-TDR}}$)
- **Setup**: The challenger $\mathcal{C}$ runs P-TDR.Setup to generate lpk, pp, $(lsk_1, ..., lsk_n)$. $\mathcal{C}$ runs P-TDR.KenGen(1^k) to generate (pk, sk). $\mathcal{C}$ gives (pp, lpk, pk, sk) to $\mathcal{A}$.
- **Phase 1**: $\mathcal{A}$ can adaptively issue polynomial number of derivation key queries for any epoch $\tau \in T$ (Query, τ, i) where $i \in \{1, ..., n\}$. $\mathcal{C}$ adopts P-TDR.PartUKGen(lsk_i,τ) to respond to each query.
- **Challenge**: $\mathcal{A}$ selects two message m_0 and m_1 where $m_0, m_1 \in \mathcal{M}$ and a prescribed release time τ' such that $\tau \leq \tau'$ where τ is such queries in **Phase 1**. $\mathcal{A}$ passes m_0, m_1, τ' to $\mathcal{C}$. $\mathcal{C}$ chooses a random bit b and computes $c =$P-TDR.Enc(lpk, m_b, τ', pk) and c is passed to $\mathcal{A}$.
- **Phase 2:** $\mathcal{A}$ can continue to make the key query with the requirement in **Challenge**.
- **Guess**: $\mathcal{A}$ outputs his guess b' for b and outputs b'

Then, we define $\mathcal{A}'$s advantage in $\mathsf{Game}^{R}_{\text{P-TDR}}$ as $\mathsf{Adv}_{\mathcal{A}}(k) = \left|Pr[b' = b] - \frac{1}{2}\right|$

Definition 6. P-TDR scheme is IND-R-CPA secure in the random oracle model if for all polynomial time adversaries $\mathcal{A}$ the function $\mathsf{Adv}_{\mathcal{A}}(k)$ in $\mathsf{Game}^{R}_{\text{P-TDR}}$ is negligible.

B. Reliable Lifetime Secret Key Management: Formal Definition & Security Model

In this subsection, we formally define the auditing protocol and the recovery protocol of PR-TDR.

1) Auditing Protocol:

Definition 7. (PR-TDR.Audit) PR-TDR.Audit enables a semi-honest auditor to verify the availability of the lifetime secret key with the peers prior to a prescribed release time. A PR-TDR.Audit protocol is secure if the following properties hold:
- ***Completeness:*** *If at least $t + 1$ peers honestly follow the protocol, the semi-honest auditor convinces that the life time secret key lsk is available.*
- ***Privacy:*** *The semi-honest auditor learns nothing regarding the life time secret key lsk except the auditing result.*

2) Recovery Protocol:

Definition 8. (PR-TDR.Recover) PR-TDR.Recover allows peers that has lost their shares of the lifetime secret key to obtain new shares, encoding the same lifetime secret key, with the assistance of other lifepeers. A PR-TDR.Recover protocol is secure if the following properties hold for any PPT adversary $\mathcal{A}$.
- ***Correctness:*** *If $\mathcal{A}$ can corrupt no more than t peers, the recovering party will correctly recover the share.*
- ***Termination:*** *In a synchronous network, if $\mathcal{A}$ can corrupt no more than t peers, then all honest peers successfully complete PR-TDR.Recover.*
- ***Secrecy:*** *If $\mathcal{A}$ cannot corrupt more than t peers during the lifetime T, then $\mathcal{A}$ obtains no information regarding the shared lifetime secret key lsk.*

C. Other Assumptions

We also assume that the identities of the group of peers $\mathcal{P}$ are known before S initiates a service. Additionally, we assume that secure channels have been pre-built between each peer in $\mathcal{P}$.

V. PR-TDR:The Construction

A. P-TDR

We construct a new encryption scheme by integrating Ti-TiRE with IND-CPA secure public key encryption scheme PK. P-TDR allows sender S to encrypt a message m to a receiver R. The encrypted message is associated with a prescribed release time τ specified by S. R can decrypt the encrypted message if R has their private key sk and a decryption key corresponding to τ. We next detail our P-TDR construction: Given the definition of Ti-TiRE and PK, we instantiate P-TDR as follows:
- P-TDR.Setup$(1^k, T, n, t) \rightarrow (pp, lpk, (lsk_1, ..., lsk_n))$: Run Ti-TiRE.Setup$(1^k, T, n, t)$ to get the following ingredients: (1) lifetime secret key lsk and lifetime public key lpk, and (2) a list $(lsk_1, ..., lsk_n)$ of shares of lsk, where $(lsk_1, ..., lsk_n)$ is derived from shamir's secret sharing $\mathsf{sss}(lsk, t + 1, n)$; (3) a list of commitments $(\gamma_1, ..., \gamma_n)$, each of which commits on lsk_i respectively. Such commitments are derived from *pederson commitment* [3]. At the end of P-TDR.Setup, the trusted party distributes $(lsk_1, ..., lsk_n)$ to $\mathcal{P}$ and broadcasts $(\gamma_1, ..., \gamma_n)$.
- P-TDR.KeyGen$(1^k) \rightarrow (pk, sk)$: P-TDR runs PK.Gen$(1^k)$ to generate a key pair (pk, sk) for R.
- P-TDR.PartUKGen$(lsk_i, \tau) \rightarrow uk_{i,\tau}$: P-TDR straightforwardly inherits Ti-TiRE.PartUKGen(lsk_i, τ), enabling each P_i derive partial update key given lsk_i and τ. It will yield updated key $u_{i,\tau}$. This operation will be performed at each time epoch.

- P-TDR.UKCombine$(uk_{1,\tau}, uk_{2,\tau}, ..., uk_{t+1,\tau}) \rightarrow uk_\tau$: P-TDR directly adopts Ti-TiRE.UKCombine $(uk_{1,\tau}, uk_{2,\tau}, ..., uk_{t+1,\tau})$ to output uk_τ.
- P-TDR.DKGen$(K_{\tau-1}, uk_\tau) \rightarrow K_\tau$: P-TDR adopts Ti-TiRE.UKCombine$(K_{\tau-1}, uk_\tau)$ to get K_τ.
- P-TDR.Enc$(lpk, m, \tau, pk) \rightarrow c$: S randomly picks $r \leftarrow \{0,1\}^l$ and sets $m' = m \oplus r$. Then, we run Ti-TiRE.Enc(lpk, r, τ) to get c_0 and PK.Enc(pk, m') to get c_1. The ciphertext will be $c = (c_0, c_1)$.
- P-TDR.Dec$(lpk, K, c, sk) \rightarrow m/\perp$: Parse c_0 and c_1 from c. First run Ti-TiRE.Dec(lpk, K, c_0). If decryption succeeds, it will yield r ; otherwise, it will yield $\perp$. Then, run PK.Dec(sk, c_1), which will output m' if the decryption succeeds, otherwise, outputs $\perp$. If the former gives r and the latter gives m', we can derive $m = r \oplus m'$;otherwise, output $\perp$.

B. Reliable Lifetime Secret Key Management

1) Auditing Protocol: Our auditing protocol (PR-TDR.Audit) aims at providing a systematic approach to verify the availability of lsk without leaking any information regarding lsk beyond the verification result. From a high-level perspective, PR-TDR.Audit allows the group of peers $\mathcal{P}$ to run NIZK.Proof individually and submit the corresponding proofs to V. Subsequently, V adopts NIZK.Verify to verify each proof. After having $(t+1)$ correct proofs, V ensures that the lifetime secret key lsk is still available. Detailed Protocol is presented as follows:

PR-TDR.Audit

1. Auditor V broadcasts a verification request to $\mathcal{P}$

2. Upon receiving the request, each $P_i \in \mathcal{P}$ generates a proof π_i by adopting NIZK.Proof. Here, π_i proves that $P_i's$ held lifetime secret key share lsk_i is indeed the one generated in P-TDR.Setup. Then, P_i sends π_i to V.

3. After receiving the list of proofs $\pi := \{\pi_1, ..., \pi_n\}$, and with the commitments $(\gamma_1, ..., \gamma_n)$ in hand, V starts checking each proof π_i by adopting NIZK.Verify to each π_i. Once V accumulates $t+1$ correct proofs, the verification procedure stops, and V outputs 1, indicating a successful verification. If the required number of proofs is not attained, V outputs 0, meaning that the verification has failed.

2) Recovery Protocol: To support PR-TDR.Recover, we assume that the trusted setup runs PC.Setup to get (cpk, csk) and com_f :=PC.Commit$(cpk, f(\cdot))$ to generate commitment on polynomial $f(\cdot)$, where $f(\cdot)$ is the polynomial encoding lsk. The high-level idea behind our recovery protocol PR-TDR.Recover is to enable a peer, who lost their held shares of lsk, to recover the lost shares by interacting with other peers in $\mathcal{P}$. To ease our exposition, we first present our design philosophy in semi-honest adversary setting, where the corrupted peers do not deviate from the protocol. In case a peer P_{rc} has lost their share, P_{rc} can broadcast their recovery request. After receiving the request, the remaining peers in $\mathcal{P} \setminus P_{rc}$ jointly generate a random polynomial $r(\cdot)$ encoding $r(\alpha_{rc}) = 0$. Each peer P_r derives a new share $z(\alpha_r) = r(\alpha_r) + f(\alpha_r)$ and sends $z(\alpha_r)$ to P_{rc} through

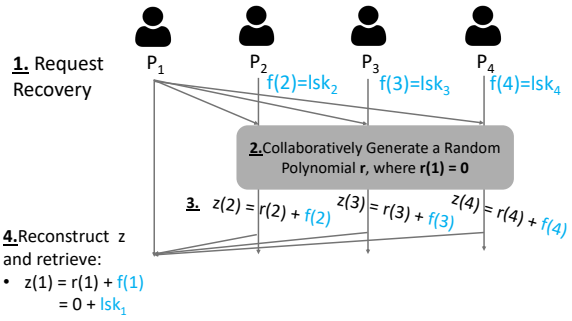


Fig. 2: PR-TDR.Recover Example

secure channels. Once P_{rc} has received enough shares, they then recover a new polynomial $z(\cdot) = (f+r)(\cdot)$ and evaluate it at α_{rc}, which yields $z(\alpha_{rc}) = f(\alpha_{rc}) + r(\alpha_{rc}) = f(\alpha_{rc})$ where $f(\alpha_{rc})$ is the lost share of P_{rc}.

We next show a working example under the semi-honest adversary setting. In Fig. 2, P_1 has lost their life time secret key share, lsk_1. P_1 starts PR-TDR.Recover by broadcasting a recovery request to P_2, P_3, and P_4. Upon receiving the request, P_2, P_3, P_4 will collaboratively generate a random polynomial $r(\cdot)$ with a specific encoding where $r(1) = 0$. Each peer from $\{P_2, P_3, P_4\}$ now holds two shares: one from $r(\cdot)$, the other being the lifetime secret key share $f(1) = lsk_1$. Subsequently, each peer in $\{P_2, P_3, P_4\}$ undertakes the following operations : (1) derives a new share by computing the sum of the two shares it holds. For example, P_2 derives $z(2)$ as $z(2) = f(2) + r(2)$, where $z(\cdot)$ a new polynomial ; (2) sends the new share to P_1. On P_1's side, after receiving 3 shares $z(1), z(2)$, and $z(3)$, P_1 interpolates the polynomial $z(\cdot)$ from the three shares. Upon interpolating $z(\cdot)$, P_1 evaluates $z(1)$, where $z(1) = f(1) + r(1) = lsk_1 + 0 = lsk_1$. This computation restores P_1's original lifetime secret key share, lsk_1.

Since our final objective is to survive our PR-TDR.Recover protocol in malicious adversary settings, we next make the above semi-honest version verifiable. We note that when up to t peers are becoming malicious, the semi-honest version will fail due to the following potential attacks:(1) A malicious peer, when generating random polynomials, may distribute invalid share to other peers; (2) A malicious peer may generate an invalid polynomial without correctly encoding $r(\alpha_{rc}) = 0$. To mitigate the above risk, we equip our semi-honest design with polynomial commitment scheme. Our detailed design is presented below.

PR-TDR.Recover

1. Assume that P_{rc} is a peer within the group $\mathcal{P}$ and requires recovery for its lost share, represented by the evaluation of $f(\cdot)$ at α_{rc}, where $f(\cdot)$ is a polynomial of degree t. P_{rc} initiates PR-TDR.Recover by broadcasting a recovery request.

2. Upon receiving the recovery request from P_{rc}, each peer $P_i \in \mathcal{P} \setminus P_{rc}$ will perform the following operations:

PR-TDR.Recover(Cont'd)

- (i) Each P_i generates a random polynomial $r^{(i)}(\cdot)$ of degree t with a specific evaluation where $r^{(i)}(\alpha_{rc}) = 0$. Additionally, P_i adopts PC.Commit$(cpk, r^{(i)})$ to create a commitment $c_i = com(r^{(i)}(\cdot))$ to $r^{(i)}(\cdot)$.
- (ii) P_i prepares $r^{(i)}(\alpha_r)$ and $r^{(i)}(\alpha_{rc})$ for each receiving peer P_r, where $P_r \in \mathcal{P} \setminus \{P_i, P_{\alpha_{rc}}\}$. In addition, P_i adopts PC.CreateWitness to create two witnesses $w^{(i)}_{\alpha_r}$ and $w^{(i)}_{\alpha_{rc}}$, corresponding to $r^{(i)}(\alpha_r)$ and $r^{(i)}(\alpha_{rc})$, respectively. Such witnesses enable P_r to perform the following verification: (1) the correctness of $r^{(i)}(\alpha_r)$ and a correct evaluation of $r^{(i)}(\alpha_{rc})$. P_i then sends c_i, $r^{(i)}(\alpha_r)$, $w^{(i)}_{\alpha_r}$, $r^{(i)}(\alpha_{rc})$, and $w^{(i)}_{\alpha_{rc}}$ to P_r.

3. Each receiving party $P_r \in \mathcal{P} \setminus P_{rc}$ then verifies the following:

- Verify whether $r^{(i)}(\alpha_r)$ is a valid evaluation of α_r on $r^{(i)}(\cdot)$ by adopting PC.VerifyEval$(cpk, c_i, \alpha_r, r^{(i)}(\alpha_r), w^{(i)}_{\alpha_r})$
- Verify whether $r^{(i)}(\alpha_{rc}) = 0$ by adopting PC.VerifyEval$(cpk, c_i, \alpha_{rc}, r^{(i)}(\alpha_{rc}), w^{(i)}_{\alpha_{rc}})$

If either verification fails, the protocol moves to **Accusation Phase**, otherwise, the protocol continues in Step.4.

Accusation Phase: If any inconsistency is observed from P_r, P_r then sends an accusation request (accuse, P_i) to P_{rc}. P_{rc} broadcasts (accuse, P_i) to $P_r \in \mathcal{P} \setminus \{P_{rc}, P_i\}$ and goes back to step.1. Each party in $\mathcal{P} \setminus \{P_{rc}, P_i\}$ checks the corresponding share and the witnesses. If the shares and witnesses are invalid, then P_i is added to a blacklist B. Otherwise, P_r uses the share sent from P_i.

4. Each P_r then derives a new share $z^{(r)}(\alpha_r)$ by performing the following summation: $z^{(r)}(\alpha_r) = f(\alpha_r) + \sum_{k=1}^{|C|} r^{(k)}(\alpha_r)$, where C means the shares from the peers who are not in B.

5. Each P_r sends $z(\alpha_r)$ to P_{rc}. After receiving such shares, now, P_{rc} holds a set Z of shares, where $Z = \{z(\alpha_i)\}$. Before performing interpolation, P_{rc} first verifies that each share is indeed a valid share on $z(\cdot)$ by following a similar procedure as above (Step.3). If getting $t+1$ verification pass, P_{rc} then interpolates $z(\cdot)$ from the $t+1$ valid shares on $z(\cdot)$. P_{rc} finally evaluates $z(\cdot)$ at α_{rc}, where $z(\alpha_{rc}) = f(\alpha_{rc}) + \sum_{k=1}^{|C|} r^{(k)}(\alpha_{rc}) = f(\alpha_{rc}) + 0 = f(\alpha_{rc})$. $z(\alpha_{rc})$ gives P_{rc} the the lost share on $f(\cdot)$.

Communication Complexity For $O(n)$ recovering peers, we need $O(n^3)$ communication compelxity.

VI. Security Proof

A. P-TDR

We first give the correctness result for which the proof is straightforward if Ti-TiRE and PK are correct. Due to space considerations. we omit the details of the proof here.

Lemma 1. *If Ti-TiRE and PK are correct, then P-TDR satisfies correctness.*

The following two lemmas establish formal security guarantee for P-TDR.

Lemma 2. *If PK is an IND-CPA secure public key scheme, then P-TDR is IND-TS-CPA secure.*

Proof. Suppose that $\mathcal{A}$ has advantage $\epsilon(k)$ to successfully attack P-TDR. We prove the security by constructing another adversary $\mathcal{B}$ who attacks PK by adopting $\mathcal{A}$'s attack on P-TDR. We next show our reduction:

- **Setup:**The challenger $\mathcal{C}$ runs P-TDR.KeyGen to generate key pair (pk, sk). It gives pk to $\mathcal{B}$. Then, $\mathcal{B}$ runs P-TDR.Setup to get (pp, lpk). $\mathcal{B}$ then hands (pp, lpk, pk) to $\mathcal{A}$. $\mathcal{C}$ derives n shares $(lsk_1, ..., lsk_n)$ where $lsk_i \in \mathbb{Z}_q$.
- **Corruption Query:** Upon receiving the corruption query from $\mathcal{A}$ where $C \subset \{1, ..., n\}$ denotes the corruption indices, $\mathcal{B}$ gives C to the challenger $\mathcal{C}$. $\mathcal{C}$ then sends (lsk_i) where $i \in C$ to $\mathcal{B}$ and $\mathcal{B}$ hands $(lsk_i)_{i \in C}$ to $\mathcal{A}$.
- **Challenge:** $\mathcal{A}$ generates two messages m_0 and m_1 and selects a target release time τ. $\mathcal{A}$ gives $\mathcal{B}$ (m_0, m_1, τ). Upon receiving such information, $\mathcal{B}$ randomly picks $r \in \{0,1\}^l$ and gives $\mathcal{C}$ $m_0 \oplus r$ and $m_1 \oplus r$. $\mathcal{C}$ randomly chooses $b \in \{0,1\}$ and runs PK.Enc$(pk, m_b \oplus r)$ to get c_1 and gives $\mathcal{B}$ c_1. $\mathcal{B}$ runs Ti-TiRE.Enc(lpk, r, τ) to get c_2. Finally, $\mathcal{B}$ gives $c = (c_1, c_2)$ to $\mathcal{A}$
- **Guess:** $\mathcal{A}$ output his guess b' on b. $\mathcal{B}$ outputs what $\mathcal{A}$ guesses.

It is easy to see that $\mathcal{B}$ perfectly simulates $\mathsf{Game}^{TS}_{\text{P-TDR}}$ for $\mathcal{A}$, since the view of $\mathcal{A}$, $(pp, lpk, (lsk_i)_{i \in C}, c)$, in the simulator is identical to the ones received in $\mathsf{Game}^{TS}_{\text{P-TDR}}$. Thus, $\mathcal{A}$'s advantage in the simulator is equal to the one defined in $\mathsf{Game}^{TS}_{\text{P-TDR}}$. In addition, in the simulator, $\mathcal{B}$ wins whenever $\mathcal{A}$ wins. By contradiction, if $\mathcal{A}$ wins with non-negligible advantage, $\mathcal{B}$ wins also, contradicting to the fact that PK is IND-CPA secure. This completes the proof. □

Lemma 3. *If Ti-TiRE is an IND-CPA encryption scheme in the random oracle model [5], then P-TDR is IND-R-CPA secure in the random oracle model.*

Proof. Suppose that $\mathcal{A}$ has advantage $\epsilon(k)$ to successfully attack P-TDR. We prove the security by constructing another adversary $\mathcal{B}$ who attacks Ti-TiRE by adopting $\mathcal{A}$'s attack on P-TDR. We next show our reduction:

- **Setup:** The challenger $\mathcal{C}$ runs P-TDR.Setup to generate key pair (pp, lpk, lsk). $\mathcal{C}$ derives $(lsk_1, ..., lsk_n)$ as shares of lsk. It gives (pp, lpk) to $\mathcal{B}$. Then, $\mathcal{B}$ runs P-TDR.KeyGen(1^k) to get (pk, sk) and gives (pp, lpk, pk, sk) to $\mathcal{A}$.
- **Update Key Query:** Upon receiving the update key query $(\mathsf{Query}, i, \tau)$ from $\mathcal{A}$, $\mathcal{B}$ randomly picks $lsk'_i \in \mathbb{Z}_q$, runs P-TDR.PartUKGen(lpk, lsk'_i, τ) to get $uk'_{i,\tau}$, and sends it to $\mathcal{A}$.
- **Challenge:** $\mathcal{A}$ generates two messages m_0 and m_1 and selects a target release time τ. $\mathcal{A}$ gives $\mathcal{B}$ (m_0, m_1, τ). Upon receiving such information, $\mathcal{B}$ gives (m_0, m_1, τ) to $\mathcal{C}$. Then $\mathcal{C}$ randomly picks $r \in \{0,1\}^l$ and randomly picks $b \in \{0,1\}$, calculates $c_0 =$Ti-TiRE.Enc(lpk, r, τ) and $m_b \oplus r$. Subsequently, $\mathcal{C}$ gives $\mathcal{B}$ c_0 and $m_b \oplus r$. $\mathcal{B}$ adopts PK.Enc$(pk, m_b \oplus r)$ to get c_1. Then, $\mathcal{B}$ gives $c = (c_0, c_1)$ to $\mathcal{A}$
- **Guess:** $\mathcal{A}$ output his guess b' on b. $\mathcal{B}$ outputs what $\mathcal{A}$ guesses.

It is easy to see that $\mathcal{B}$ perfectly simulates $\mathsf{Game}^{R}_{\text{P-TDR}}$ for $\mathcal{A}$, since the view of $\mathcal{A}$, $(pp, lpk, sk, uk_{i,\tau}, c)$, in the simulator

is identical to the ones received in $\mathsf{Game}^R_{\mathsf{P\text{-}TDR}}$. Thus, $\mathcal{A}$'s advantage in the simulator is equal to the one defined in $\mathsf{Game}^R_{\mathsf{P\text{-}TDR}}$. In addition, in the simulator, $\mathcal{B}$ wins whenever $\mathcal{A}$ wins. By contradiction, if $\mathcal{A}$ wins with non-negligible advantage, $\mathcal{B}$ wins also, contradicting to the fact that Ti-TiRE is IND-CPA secure in the random oracle model. This completes the proof. □

B. PR-TDR.Audit

The following two lemmas show that PR-TDR.Audit is secure as defined in Definition IV-B1.

Lemma 4. ***(Completeness)*** *If $\mathcal{A}$ corrupts no more than t times peer during lifetime T and the adopted NIZK is zero-knowledge, then the semi-honest auditor learns nothing regarding the original lifetime secret key.*

Lemma 5. ***(Privacy)*** *If $\mathcal{A}$ corrupts no more than t peers during the lifetime T and the adopted NIZK is perfect-completeness, then the semi-honest auditor is convinced.*

C. Recovery Protocol

In this subsection, we offer formal proof demonstrating that our proposed PR-TDR.Recover satisfies the security properties as defined in Section 5 Definition IV-B2. Before proving the properties, we first show that the polynomial phase yields a polynomial that is randomly generated

Lemma 6. *If $\mathcal{A}$ corrupts no more than t peers and the hiding property of the polynomial commitment holds, the polynomial $r(\cdot)$ generated in PR-TDR.Recover is random.*

Proof. See Appendix. □

Lemma 7. ***(Correctness)*** *If $\mathcal{A}$ corrupts no more than t peers, the binding property of the polynomial commitment scheme holds, P_{rc} will receive a new share $z(\alpha_{rc})$,such that $z(\alpha_{rc}) = f(\alpha_{rc})$, where $f(\cdot)$ is the original polynomial corresponding to the lifetime secret key lsk.*

Proof. Since $z(\cdot) = f(\cdot) + r(\cdot)$, we will first show that, if the given assumptions are true, the generated random polynomial $r(\cdot)$ is correct. This is because there will be at least $t+1$ honest peers and the shares held by such peers are consistent, thus well-defining a correct $r(\cdot)$ such that $r(\alpha_{rc}) = 0$. We next show that the set of corrupted peers cannot force P_{rc} accept an invalid polynomial $z'(\cdot)$. This is due to two aspects: (1) P_{rc} will eventually get at least $t+1$ correct shares on $z(\cdot)$ and (2) the t corrupted peers cannot cheat P_{rc} because of the binding property of the polynomial commitment scheme. Based on the above facts, P_{rc} will finally get valid $z(\alpha_{rc})$ such that $z(\alpha_{rc}) = f(\alpha_{rc})$, completing this proof. □

Lemma 8. ***(Termination)*** *If $\mathcal{A}$ corrupts no more than t peers, then PR-TDR.Recover always terminates.*

Proof. Given Lemma 6, the random polynomial generation phase concludes with at least $t+1$ honest peers observing consistent and valid shares corresponding to the random polynomial. In addition, at least $t+1$ honest peers hold valid share on original polynomial $f(\cdot)$. Thus, they can derive their shares on $z(\cdot)$ and send them to P_{rc}. Once P_{rc} gets enough valid shares of $z(\cdot)$ that pass verification, they will derive $f(\alpha_{rc})$ and terminate the protocol. This completes the proof. □

Lemma 9. ***(Secrecy)*** *If $\mathcal{A}$ corrupts no more than t peers during lifetime T, then the information obtained by $\mathcal{A}$ in PR-TDR.Recover is random and independent of the lifetime secret key lsk.*

Proof. Our proof consists of the following two parts: first, we shall show $\mathcal{A}$ cannot get any information regrading the original polynomial $f(\cdot)$ for which $f(0) = lsk$. Then, we shall show the accusation-response phase does not leak any information regarding lsk to $\mathcal{A}$. In proving the first part, we note that $r(\cdot)$ is randomly generated due to Lemma 6, and we denote $Cor(\mathcal{P})$ as the set of corrupted peers. Then, we have the following two scenarios: $P_{rc} \in Cor(\mathcal{P})$ and $P_{rc} \notin Cor(\mathcal{P})$. Specifically,

(1) $P_{rc} \notin Cor(\mathcal{P})$. In this case, $\mathcal{A}$'s view consists of t shares corresponding to $r(\cdot)$ from the corrupted peers and one special share $(r(\alpha_{rc}), 0)$. Thus, such $(t+1)$ shares can make $\mathcal{A}$ recover $r(\cdot)$. However, to recover the original polynomial $f(\cdot)$, $\mathcal{A}$ still needs to know $z(\alpha_{rc})$ to recover $z(\cdot)$ and then perform $f(\cdot) = z(\cdot) - r(\cdot)$ to recovery $f(\cdot)$. Since $P_{rc} \notin Cor(\mathcal{P})$, $\mathcal{A}$ cannot know $z(\alpha_{rc})$. The recovery is thus impossible.

(2) $P_{rc} \in Cor(\mathcal{P})$. In this case, $\mathcal{A}$ totally corrupts t peers including P_{rc}. To recover $f(\cdot)$, $\mathcal{A}$ will need to get $r(\cdot)$ and perform $f(\cdot) = z(\cdot) - r(\cdot)$. $\mathcal{A}$ only owns t shares corresponding to $r(\cdot)$ and thus cannot recover $r(\cdot)$, therefore fails to recover $f(\cdot)$. Additionally, because of the hiding property of our adopted polynomial commitment, all the broadcasted commitments, including the ones on $f(\cdot)$ from the trusted setup phase and the ones shown in PR-TDR.Recover, reveal no additional information to $\mathcal{A}$.

For the second part, it is obvious to see that the accusation-response phase reveals nothing regarding lsk since it only discloses invalid share. □

VII. Implementation & Evaluation

In this section, we discuss the implementation and performance evaluation of PR-TDR.

A. Implementations

We have implemented PR-TDR in Go [28] and the implementation leverages the open-source Ti-TiRE [3] framework. For the IND-CPA secure public key encryption scheme, we utilize ElGamal Encryption [13], which is implemented in Go. The polynomial commitment scheme is initialized using the discrete logarithm-based construction from KZG [19], following its open-source implementation [26]. NIZK is instantiated according to the construction in [14], and is based on the implementation open-sourced in [3]. Our Ethereum [38] smart contract is programmed in *Solidity* [33] and we deployed it in the local Ethereum testing environment provided by Ganache [31]. We use *Geth* [15] to perform smart contract interactions. The communication across different components is managed by the gRPC [16] library in Go. Evaluations were run on a MacBook Pro with an Apple M1 Max CPU and 32 GB RAM.

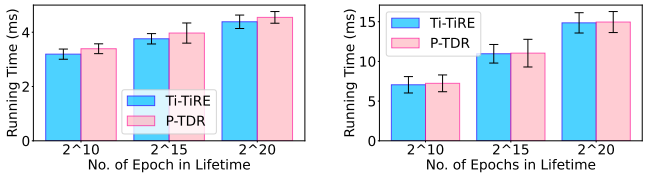
		Ti-TiRE	P-TDR
2^{10}	max	1.088	1.216
	avg	1.024	1.141
2^{15}	max	1.408	1.536
	avg	1.344	1.431
2^{20}	max	1.728	1.856
	avg	1.662	1.702

TABLE I: Ciphertext Size (KBs)

B. Evaluation Results

P-TDR Evaluation Results: In evaluating the performance of P-TDR, we consider three different metrics: the running time of P-TDR.Enc and P-TDR.Dec, ciphertext size, and on-chain gas cost incurred by update key uploading. The results of the evaluations are presented as follows:

Running time of P-TDR.Enc and P-TDR.Dec: We analyzed the running time of P-TDR.Enc and P-TDR.Dec across three different lifetimes, measured in number of epochs: 2^{10}, 2^{15}, and 2^{20}. For each epoch configuration, we executed 10000 times to obtain the average results. In our experiments, we use Ti-TiRE as a baseline for comparison. For the running time of P-TDR.Enc, in Fig 3a, we observe that P-TDR only incurs marginally higher computational overhead compared to Ti-TiRE across three different lifetime settings. The additional overhead is primarily due to Elgamal encryption. Moreover, the running time of P-TDR.Enc increases with the extension of lifetime, attributed to the inherent feature of Ti-TiRE. In Fig 3b, we report the running time of P-TDR.Dec. Across varying lifetimes, P-TDR.Dec's running time aligns closely with Ti-TiRE. The trends observed in the P-TDR.Enc experiments are reflected similarly in this experiment.

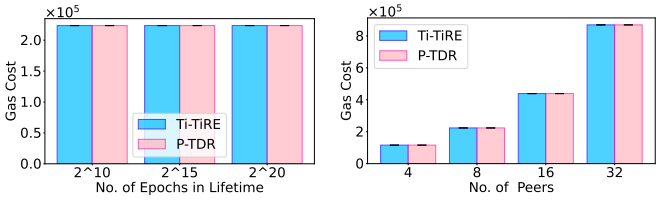


(a) Encryption Time (b) Decryption Time

Fig. 3: Encryption and Decryption Time of P-TDR

Ciphertext Size: Table I presents our findings on the ciphertext sizes under three distinct lifetime settings $(2^{10}, 2^{15}, 2^{20})$. These results are derived from averages over 10000 runs. We note that P-TDR produces ciphertext that is slightly longer than those generated by Ti-TiRE. Specifically, the size of P-TDR's ciphertext is 0.39 KBs longer in the 2^{10} setting, 0.087 KBs longer in the 2^{15} setting, and 0.04 KBs longer in the 2^{20} setting.

Per-epoch Gas Cost: Fig. 4 illustrates the gas cost incurred by the update key uploading operation. We conducted this experiment 100 times to get averaged results across two different settings: varying lifetime lengths and varying number of peers. In Fig. 4a, we fixed the number of peers as $n = 8$. Across three different lifetime settings, the gas cost of P-TDR remains nearly constant. This consistency arises because the update key only consists of single group element, irrespective the lifetime length. P-TDR benefits from this advantage, a salient feature supported by Ti-TiRE. In Fig. 4b, we fixed the length of lifetime as $T = 2^{15}$. We evaluate the impact of varying the group size $(4, 8, 16, 32)$. Here, we observe that the gas cost increases proportionally with the group size.



(a) Gas Cost $(n = 8)$ (b) Gas Cost $(T = 2^{15})$

Fig. 4: Gas Cost of P-TDR

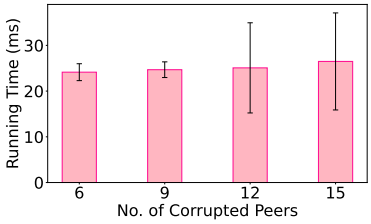
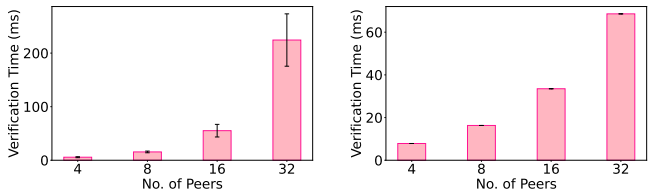


Fig. 5: PR-TDR.Audit Time Overhead

PR-TDR.Audit Evaluation Results: We assessed the auditing overhead in terms of verification time until we reach success on the side of auditor, as shown in Fig. 5. We fixed the group size of peers as $n = 32$, and varied the number of corrupted peers across four settings: $6, 9, 12, 15$. Our observations indicate that as the number of corrupted peers increases, the verification time overhead also rises. In the worst case where we have 15 peers corrupted, the averaged time overhead reaches approximately 25.217 ms.

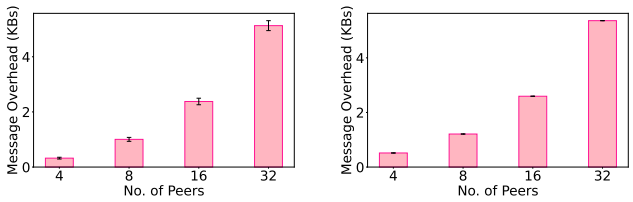
PR-TDR.Recover Evaluation Results: We evaluated PR-TDR.Recover from the two distinct perspectives: verification time cost and message overhead. The verification time cost pertains to the phase during which each peer performs secret share verification by adopting polynomial commitment scheme. The message overhead quantifies the communication complexity during secret share verification phase. We report the results separately for the recovering peer P_1 and other peers $P_i \in \mathcal{P} \setminus P_1$.



(a) Verification Time for P_i (b) Verification Time for P_1

Fig. 6: PR-TDR.Recover Verification Time

In Fig. 6a, we observe that as the group size of peers



(a) Message Overhead for P_i (b) Message overhead for P_1

Fig. 7: PR-TDR.Recover Message Overhead

increases, the verification time also grows. This observation supports our design, wherein each peer needs to verify received random shares from other peers. PC.VeriEval operation dominates the verification time. Regarding message overhead, we observed similar trends in Fig. 6a. This is because during verification, each peer receives share, polynomial commitment, and corresponding witness from other peers.

VIII. Related Work

A. Timed Data Release

There has been several efforts on constructing a distributed and decentralized approach for TDR [2], [21]–[24], [35]–[37] while ensuring data privacy until the release time. Li et.al [21], [22] performed an attack-resilience analysis and investigated the security strengths of TDR. Subsequently, Li et.al [23], [24] and Bacis et.al [2] integrated game theory with smart contracts and developed techniques to prevent TDR from rational adversaries. To safeguard TDR in decentralized environments from both rational and malicious adversaries, Wang et.al [35], [37] design a reputation-aware timed data release framework on the Ethereum blockchain. [36] focuses on enriching TDR with controllable primitives. Despite these advancements, current techniques suffer from the following limitations: (1) they lack provable security guarantees, and (2) they neglect the reliability of delivery services. In contrast, our proposed PR-TDR addresses theses deficiencies by supporting both of these features.

B. Timed-release Cryptography

In the cryptography community, encrypting data to the future is formally addressed through time-release cryptography. Starting from [29], the research in this field has developed along two primary approaches: the time-lock puzzle based and trusted peer based approach. The time-lock puzzle-based approach, represented by [6], [25], [29], encrypts the data within a puzzle that must be solved through sequential steps to enable decryption. Alternatively, the trusted-peer based approach relies on a trusted-peer to provide time reference. Data is encrypted to a future point of time, and upon the peer's announcement of the prescribed moment, the data is decrypted with peer's assistance. Notable contributions to this approach, include but not limited to the works in [3], [9], [10], [29].

Despite these theoretically elegant constructions, existing research often overlooks the aspect of reliability. Our work leverages the theoretical insights from [3] and advances the state of the art of TDR by introducing a novel reliable design. This enhancement not only fills a gap in literature but also increases the practicality of the timed-release cryptography in supporting real-world applications.

C. Data Integrity in Outsourcing Environment

The auditing protocol proposed in our work is related to research focusing on data integrity checks in remote storage. Proofs of Retrievability (PoR) [18] and Provable Data Possession (PDP) [1] are two pioneering works that have established a formal foundation for ensuring data integrity. Subsequently, several variants tailored for distributed storage have been proposed, including HAIL [7], MR-PDP [11], and MP-PoR [27]. However, existing constructions do not support data privacy while ensuring data integrity checks. On the other hand, the auditing protocol we propose in PR-TDR is designed to support privacy-preserving integrity check for TDR.

IX. Discussion

In this paper, we primarily focus on protecting PR-TDR under the critical assumption of an *honest-majority* group of peers. However, with a straightforward extension, PR-TDR can also function under a weaker assumption where the honest peers are rational peers [12] who may launch an attack if it is profitable. This can be achieved by adopting the techniques from [24] and carefully setting up an incentive policy that penalizes rational peers, thereby encouraging them to behave honestly.

X. Conclusion

This paper presents the first research effort into the formal and efficient design of privacy-preserving and reliable TDR. We introduce a novel framework called PR-TDR, which not only ensures privacy and reliability in TDR but also provides a formal security guarantee. We approach our design by first constructing P-TDR, a privacy-preserving TDR design that entails cryptographic privacy. Building on P-TDR, we then carefully design a reliable lifetime secret key management mechanism. It consists of the lifetime secret key auditing protocol PR-TDR.Audit and the lifetime secret key share recovery protocol PR-TDR.Recover. We formally define the secure properties of various components within PR-TDR and provide rigorous security proof. PR-TDR is implemented using Ethereum and our performance evaluations confirm that PR-TDR not only embodies the desired security properties but also operates efficiently.

Acknowledgement

This material is based upon work supported by the National Science Foundation under Grant #2020071. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

APPENDIX
PROOF OF LEMMA 6

To show $r(\cdot)$ is random, it suffices to show:(1) each honest peer generates polynomial randomly $r^{(i)}(\cdot)$ and the group of honest peers gets consistent views on the generated polynomial $r(\cdot)$, and (2) the adversary $\mathcal{A}$ cannot bias the generated $r(\cdot)$. The first one is easy to get since we have the *honest-majority* assumption. To prove (2), we see that, due to the hiding property of our adopted polynomial commitment, $\mathcal{A}$ gets no information regarding the $r^{(i)}(\cdot)$ and $r(\cdot)$, thus has no way of biasing the generation of $r(\cdot)$. This completes the proof.

REFERENCES

[1] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary Peterson, and Dawn Song. Provable data possession at untrusted stores. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 598–609, 2007.

[2] Enrico Bacis, Dario Facchinetti, Marco Guarnieri, Marco Rosa, Matthew Rossi, and Stefano Paraboschi. I told you tomorrow: Practical time-locked secrets using smart contracts. In *Proceedings of the 16th International Conference on Availability, Reliability and Security*, pages 1–10, 2021.

[3] Leemon Baird, Pratyay Mukherjee, and Rohit Sinha. i-tire: Incremental timed-release encryption or how to use timed-release encryption on blockchains? In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 235–248, 2022.

[4] Soumya Basu, Alin Tomescu, Ittai Abraham, Dahlia Malkhi, Michael K Reiter, and Emin Gün Sirer. Efficient verifiable secret sharing with share recovery in bft protocols. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 2387–2402, 2019.

[5] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[6] Dan Boneh and Moni Naor. Timed commitments. In *Annual international cryptology conference*, pages 236–254. Springer, 2000.

[7] Kevin D Bowers, Ari Juels, and Alina Oprea. Hail: A high-availability and integrity layer for cloud storage. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 187–198, 2009.

[8] Vitalik Buterin. Ethereum: A next generation smart contract decentralized application platform. https://ethereum.org/en/whitepaper/, 2014. Accessed: 2024-05-10.

[9] AC-F Chan and Ian F Blake. Scalable, server-passive, user-anonymous timed release cryptography. In *25th IEEE International Conference on Distributed Computing Systems (ICDCS'05)*, pages 504–513. IEEE, 2005.

[10] Jung Hee Cheon, Nicholas Hopper, Yongdae Kim, and Ivan Osipkov. Timed-release and key-insulated public key encryption. In *Financial Cryptography and Data Security: 10th International Conference, FC 2006 Anguilla, British West Indies, February 27-March 2, 2006 Revised Selected Papers 10*, pages 191–205. Springer, 2006.

[11] Reza Curtmola, Osama Khan, Randal Burns, and Giuseppe Ateniese. Mr-pdp: Multiple-replica provable data possession. In *2008 the 28th international conference on distributed computing systems*, pages 411–420. IEEE, 2008.

[12] Changyu Dong, Yilei Wang, Amjad Aldweesh, Patrick McCorry, and Aad Van Moorsel. Betrayal, distrust, and rationality: Smart counter-collusion contracts for verifiable cloud computing. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 211–227, 2017.

[13] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.

[14] Sebastian Faust, Markulf Kohlweiss, Giorgia Azzurra Marson, and Daniele Venturi. On the non-malleability of the fiat-shamir transform. In *Progress in Cryptology-INDOCRYPT 2012: 13th International Conference on Cryptology in India, Kolkata, India, December 9-12, 2012. Proceedings 13*, pages 60–79. Springer, 2012.

[15] Ethereum Foundation. Geth: Go ethereum. https://geth.ethereum.org, 2023. Accessed: 2024-05-01.

[16] The gRPC Authors. grpc for go. https://grpc.io/docs/languages/go/, 2023. Accessed: 2024-05-01.

[17] Amir Herzberg, Stanisław Jarecki, Hugo Krawczyk, and Moti Yung. Proactive secret sharing or: How to cope with perpetual leakage. In *Advances in Cryptology—CRYPT0'95: 15th Annual International Cryptology Conference Santa Barbara, California, USA, August 27–31, 1995 Proceedings 15*, pages 339–352. Springer, 1995.

[18] Ari Juels and Burton S Kaliski Jr. Pors: Proofs of retrievability for large files. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 584–597, 2007.

[19] Aniket Kate, Gregory M Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *Advances in Cryptology-ASIACRYPT 2010: 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings 16*, pages 177–194. Springer, 2010.

[20] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography: principles and protocols*. Chapman and hall/CRC, 2007.

[21] Chao Li and Balaji Palanisamy. Emerge: Self-emerging data release using cloud data storage. In *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, pages 26–33. IEEE, 2017.

[22] Chao Li and Balaji Palanisamy. Timed-release of self-emerging data using distributed hash tables. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 2344–2351. IEEE, 2017.

[23] Chao Li and Balaji Palanisamy. Decentralized privacy-preserving timed execution in blockchain-based smart contract platforms. In *2018 IEEE 25th International Conference on High Performance Computing (HiPC)*, pages 265–274. IEEE, 2018.

[24] Chao Li and Balaji Palanisamy. Decentralized release of self-emerging data using smart contracts. In *2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS)*, pages 213–220. IEEE, 2018.

[25] Giulio Malavolta and Sri Aravinda Krishnan Thyagarajan. Homomorphic time-lock puzzles and applications. In *Annual International Cryptology Conference*, pages 620–649. Springer, 2019.

[26] Sai Krishna Deepak Maram, Fan Zhang, Lun Wang, Andrew Low, Yupeng Zhang, Ari Juels, and Dawn Song. Churp: dynamic-committee proactive secret sharing. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 2369–2386, 2019.

[27] Maura B Paterson, Douglas R Stinson, and Jalaj Upadhyay. Multi-prover proof of retrievability. *Journal of Mathematical Cryptology*, 12(4):203–220, 2018.

[28] Rob Pike, Ken Thompson, and Robert Griesemer. Go programming language. https://golang.org, 2023. Accessed: 2024-05-01.

[29] Ronald L Rivest, Adi Shamir, and David A Wagner. Time-lock puzzles and timed-release crypto.

[30] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[31] Truffle Suite. Ganache. https://trufflesuite.com/ganache/, 2023. Accessed: 2024-05-01.

[32] Nick Szabo. Formalizing and securing relationships on public networks. *First monday*, 1997.

[33] Solidity Team. Solidity programming language. https://soliditylang.org, 2023. Accessed: 2024-05-01.

[34] Robin Vassantlal, Eduardo Alchieri, Bernardo Ferreira, and Alysson Bessani. Cobra: Dynamic proactive secret sharing for confidential bft services. In *2022 IEEE symposium on security and privacy (SP)*, pages 1335–1353. IEEE, 2022.

[35] Jingzhe Wang and Balaji Palanisamy. Attack-resilient blockchain-based decentralized timed data release. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 123–140. Springer, 2022.

[36] Jingzhe Wang and Balaji Palanisamy. Ctdrb: Controllable timed data release using blockchains. In *International Conference on Security and Privacy in Communication Systems*, pages 231–249. Springer, 2022.

[37] Jingzhe Wang and Balaji Palanisamy. Securing blockchain-based timed data release against adversarial attacks. *Journal of Computer Security*, (Preprint):1–29, 2023.

[38] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.