CTDRB: Controllable Timed Data Release Using Blockchains

Jingzhe Wang(✉) and Balaji Palanisamy

University of Pittsburgh, Pittsburgh, PA, USA
{jiw148,bpalan}@pitt.edu

Abstract. The notion of Timed Data Release (TDR) supports time-based sensitive data protection in such a way that sensitive data can be accessed only after a prescribed amount of time has passed. With recent advancements in blockchain techniques, practical solutions to support decentralized TDR using blockchains (BTDR) is gaining importance. Briefly, such designs entrust blockchain decentralized networks to serve as a decentralized time agent to protect the data and release the data at a prescribed release time. However, as a variant of outsourced data management service, BTDR inherently incurs the tension between data confidentiality protection as well as data control. Unfortunately, the off-the-shelf arts only strive to protect the data without rigorous support for the control of data.

In this paper, we design a controllable framework for BTDR called *CTDRB*. At a high level, *CTDRB* realizes data access control as well as data lifetime control while protecting data confidentiality. The novel technical contributions of *CTDRB* are three-fold: first, we adopt a temporal CP-ABE cryptographic scheme, serving as a basis, to enable the data access control; second, on top of such a design, we enable data lifetime control by carefully designing a time token control service on Ethereum. We then design two representative data lifetime control primitives, namely *Data Revocation* and *Data Release Time Modification*. The former refers to revoking the data before its prescribed release time while the latter modifies the release of data at a time ahead of its prescribed release time; third but not the least, we perform security analysis of *CTDRB* and implement it using the *Ethereum* blockchain. Our results show that *CTDRB* incurs only a moderate on-chain *gas consumption* and demonstrates high efficiency.

Keywords: Timed data release · Blockchain · Smart contract

1 Introduction

Timed data release (TDR) is a time-based data protection primitive that aims at protecting sensitive data by making it accessible only after a prescribed amount of time has passed. Several real-world applications require TDR. For example, in secure voting mechanisms, votes are not permitted to be accessed until the close of the polling process. With recent advancements in blockchain techniques, practical solutions to support decentralized TDR using blockchains (BTDR) is gaining importance [4,10,16–18,20–22,24,32,33]. Briefly, such designs entrust blockchain decentralized networks to serve as a decentralized time agent to protect the data and release the data at a

© ICST Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2023
Published by Springer Nature Switzerland AG 2023. All Rights Reserved
F. Li et al. (Eds.): SecureComm 2022, LNICST 462, pp. 231–249, 2023.
https://doi.org/10.1007/978-3-031-25538-0_13

prescribed release time. Given the open nature of blockchain decentralized networks, in which a large number of mutually distrusted nodes exist, protecting data confidentiality against adversarial actions in such environments before the release time is inherently challenging. Various off-the-shelf arts have developed effective designs for data protection. Specifically, both strong cryptographic constructions [20–22] as well as practical realizations in the Ethereum blockchain [10, 16–18, 24, 32, 33] have been designed.

As a variant of outsourced data management service, however, BTDR inherently meets the tension between data confidentiality protection as well as data control. One can imagine a scenario when a data sender wants to reduce the data access scope after the release time in such a way that the data is only accessible to a selected list of recipients who are eligible to access it instead of making the data public. As another scenario, after the data is released, the data sender may become aware of some incorrect information in the data. Under this scenario, the sender would like to revoke the data before the prescribed release time. Thus, supporting BTDR to serve a broad range of real-world applications to perform dynamic data control is of significant importance. Current techniques, however, invariably fail to meet such necessity and supporting dynamic data control in BTDR while protecting data confidentiality is still largely open.

In this paper, we take the first step towards designing a controllable framework for BTDR, coined as *CTDRB*. At a high level, *CTDRB* realizes data access control as well as data lifetime control while protecting data confidentiality. Specifically, in CTDRB, we investigate the controllable primitive from the two aspects namely data access control and data lifetime control. The former refers to limiting the access scope of the data. The latter is an attractive feature supported by our framework, in which a data sender can flexibly tweak the data publishing time. Our work starts with enabling data access control. Specifically, we adopt TAFC [14], a cryptographic construction that employs ciphertext-policy attribute-based encryption (CP-ABE) [12] with timed release encryption [27], to enable data access control in CTDRB. Briefly, TAFC encrypts the data with an access structure (access policy) that embeds a time trapdoor, corresponding to a prescribed release time. Later, at the release time, a trusted authority generates and publishes a time token. Anyone holding satisfying attributes as well as the time token is able to decrypt the data.

While adopting TAFC realizes the data access control, such a construction fails to meet our need for providing data lifetime control. It is attributed to the following challenges: *first*, TAFC only provides theoretical constructions of basic encryption and decryption operations. Directly adopting such constructions limits the data sender's ability to perform lifetime control over the data such as revoking the data before the release time; *second*, TAFC heavily relies on a trusted central time agent to publish a time token. In CTDRB, however, it is impractical that such a trusted time agent exists in a fully decentralized blockchain network. To address this challenge, we design a novel time token control service on the Ethereum blockchain to support data lifetime control. Concretely, our design adopts a hierarchical key management scheme. The time token is encrypted using a public key and stored at a decentralized storage platform. The protection of the corresponding private key is delegated to a group of Ethereum network nodes, namely trustees, who jointly protect such a private key and release it when the release time arrives. With the help of such a design, a sender can manually

tweak the release time of the private key to impact the release time of the time token, which then realizes the data lifetime control. However, due to the inherent risk in BTDR design [17], trivially provisioning such a service inevitably exposes to the following two threats: drop attack and release-ahead attack. Drop attack may render the private key unavailable, which impacts the availability of the time token control service. Release-ahead poses a risk of pre-maturely releasing the data, in which adversaries may collude with the trustees as well as the recipients to illegally get access to the data at a time earlier than the prescribed one by getting the time token as well as a satisfying attribute set. Thus, to mitigate the aforementioned threats, we design the service with the following two countermeasures: (1) to lessen the impact of the drop attack, we adopt Shamir's (t, n) threshold secret share scheme [28] to split the private key into multiple shares, in which at least t shares of the private key can make it recoverable, (2) to set barriers for the adversaries seeking to prematurely release the data, we keep the identities of the trustees as well as the recipients private, which increases the efforts requires from the adversaries to launch such an attack. Grounded on the proposed time token delivery service, we then materialize two representative data lifetime control primitives, namely data revocation and data release time modification. Data revocation refers to revoking the data before its prescribed release time while data lifetime control modifies the release of data at a time ahead of its prescribed one. These primitives are practically supported by our carefully designed interactive protocol.

We provide rigorous security analysis by quantitatively measuring the attack resilience of our proposed framework. We also implement proof-of-concept smart contracts programmed in *Solidity* [7] and test it on the *Rinkeby* network to evaluate corresponding gas consumption. By performing case studies, we demonstrate that our proposed data revocation and data release time modification primitive only incur moderate gas cost under various service conditions.

Contributions. Concretely, we make the following key contributions in this paper:

– We propose a controllable framework for blockchain-based timed data release to realize both data access control as well as data lifetime control while protecting the confidentiality of the data.
– We first adopt TAFC, a variant of CP-ABE scheme, to provide data access control. Atop such a scheme, we propose a novel time token control service on the Ethereum blockchain to enable data lifetime control. Two representative data lifetime control primitives namely data revocation and data release time modification are developed.
– We rigorously analyze the security guarantee of our framework.
– We implement our prototype in *Ethereum* and perform extensive case studies.

Organization. The rest of the paper is organized as follows. In Sect. 2, we provide preliminaries adopted in this paper. A high-level description of *CTDRB* as well as corresponding security assumptions are given in Sect. 3. In Sect. 4, we construct *CTDRB* with technical details. We analyze our proposed *CTDRB* in Sect. 5. In Sect. 6, we discuss the *proof-of-concept* implementations in smart contracts. Related work is discussed in Sect. 7. We conclude this paper in Sect. 8.

2 Preliminaries

In this section, we first introduce some background about Ethereum blockchains in Sect. 2.1. Then, in Sect. 2.2, we introduce the major cryptographic primitives adopted in our work.

2.1 A Primer on the Ethereum Blockchain

Ethereum [2] is a pioneering platform that integrates the abstraction of smart contracts [29] with blockchains. Informally, an *Ethereum* smart contract is composed of a piece of computer code executed and stored on the *Ethereum* blockchain. Such code contractually enforces predefined policies among users (accounts) in *Ethereum*. The cryptocurrency associated with *Ethereum* is called *Ether*.

The two types of accounts, namely Externally Owned Account (*EOA*) and Contract Owned Account (*CA*), perform activities in the *Ethereum* account network. An *EOA*, controlled by some users on Internet, is associated with a unique public-private key pair as well as a balance of *Ether*. A *CA*, without a private key, is responsible for storing the smart contract code and maintaining a balance of ether. All interactions between *EOAs* and *CAs* rely on the concept of transaction. A transaction is initialized and signed with a private key of *EOA*.

The low-level *peer-to-peer* network maintained by *Ethereum* workers (miner nodes) provide a decentralized context for the submission and execution of transactions. The *Gas* mechanism in Ethereum drives the flow of transactions from an incentive perspective. The *Gas* is measured by *Ether*. For example, to submit a new transaction, a user needs to pay some gas for Ethereum workers to execute the transaction. The powerful consensus algorithm, *Proof-of-Work (PoW)*, guarantees the confirmation and correctness of transactions, which brings Ethereum with the attractive characteristics of *tamper-resistance* and *immutability*.

2.2 Cryptographic Primitives

In our work, we use the following cryptographic primitives.

Cryptographic Hash Function: We denote $hash(\cdot)$ to indicate a cryptographic hash function guaranteeing a collision resistant property. Specifically, in *CTDRB*, we adopt the *Keccak256* implementation supported by the *Ethereum* blockchain, where $hash(\cdot) := Keccak256(\cdot)$.

Cryptographic Digital Signature: Our work heavily relies on the cryptographic digital signature primitive to realize verifications. Specifically, we denote $Sig(\cdot)$ as the *ECDSA* signature scheme adopted in the Ethereum blockchain.

TAFC-Time and Attribute Factors Combined Access Control: *TAFC* [14] is a cryptographic scheme that integrates *timed-release encryption* (TRE) [27] with *cipertext-policy attribute-based encryption* (CP-ABE) [12] to realize timed-control primitives in CP-ABE. We next briefly introduce the adopted *TAFC*, which consists of the following algorithms:

$TAFC.Setup \rightarrow (pk, mk)$: The **Setup** algorithm takes an implicit security parameter. It outputs a public parameter pk and a master key mk.

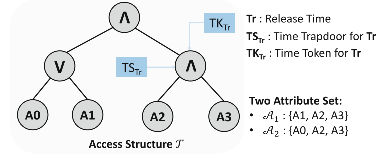


Fig. 1. TAFC sketch

$TAFC.KeyGen(mk, \mathcal{A}) \rightarrow sk_{\mathcal{A}}$: On input of a master key mk and an attribute set $\mathcal{A}$, the **Key Generation** algorithm outputs a secret key $sk_{\mathcal{A}}$ corresponding to $\mathcal{A}$.

$TAFC.Encryption(pk, D, \mathcal{T}) \rightarrow CD$: The **Encryption** algorithm takes as input a public key pk, a data plaintext D, and a specified access structure $\mathcal{T}$. It generates a ciphertext CD.

$TAFC.TokenGen(mk, t) \rightarrow TK_t$: On input of a master key mk and a time point t, the **Time Token Generation** algorithm generates a time token TK_t.

$TAFC.TrapdoorExp(TK, CD) \rightarrow CD'$: The **Trapdoor Exposure** algorithm takes as input a time token TK and a ciphertext CD. It generates a modified ciphertext CD'.

$TAFC.Decryption(CD', sk_{\mathcal{A}}) \rightarrow D$: The **Decryption** algorithm takes as input a modified ciphertext CD' and a secret key $sk_{\mathcal{A}}$ corresponding to an attribute set $\mathcal{A}$. It outputs a plaintext D.

As an example, in Fig. 1, assume that we expect that the data D can be accessible at the release time T_r. We embed a time trapdoor TS_{T_r} corresponding to T_r into the access structure $\mathcal{T}$ and encrypt D by adopting $TAFC.Encryption$ to get a ciphertext CD. A user, say user 1, holding the attribute set $\mathcal{A}_1 : \{A_1, A_2, A_3\}$ can adopt $TAFC.KeyGen$ to get his/her own secret key, namely $sk_{\mathcal{A}_1}$. At T_r, a time token TK_{T_r}, corresponding to T_r, is generated and released by $TAFC.TokenGen$. Then, user 1 can expose TS_{T_r} in $\mathcal{T}$ by adopting $TAFC.TrapdoorExp$, which gives a modified ciphertext CD'. After the exposure of TS_{T_r}, user 1 can decrypt CD' using his/her own secret key. Such a scheme guarantees that the user holding both TK_r and $sk_{\mathcal{A}_1}$ can successfully decrypt the ciphertext.

Shamir's (t, n) Threshold Secret Share Scheme: Shamir's secret share scheme [28] splits an original secret into n different shares such that any t shares are capable of reconstructing the original secret. We use $sss(t, n)$ to denote such a scheme.

3 *CTDRB:* In a Nutshell

In this section, we first introduce the overview of the CTDRB framework and discuss the security assumptions in *CTDRB*.

3.1 Framework Overview

We formally discuss the lifecycle of *CTDRB*. We begin by describing the role of each entity and we provide the formal definition of the data control primitives. We then present a high-level workflow formed among such components.

3.1.1 Key Components

At a high-level perspective, our proposed *CTDRB* consists of the following four key entities, namely *Data Sender*, *Data Recipient*, *Time Token Control Service*, and *Decentralized Storage Service*. We present the design as follows: (1) **Data Sender:** A data sender, denoted as S, is in possession of data D and needs a timed-release service. In *CTDRB*, S files a Ethereum smart contract, denoted as $\mathcal{SC}$, to support an incoming service. S then takes charge of strategically provisioning an incoming timed data release service to enable data access control as well as data lifetime control. (2) **Time Token Control Service:** The time token control service (T2CS) aims at providing data lifetime control support by designing a hierarchical key management scheme with the help of the Ethereum blockchain. Such a design is contractually enforced by $\mathcal{SC}$. Specifically, the sender S generates a time token TK_{T_r} related to a release time T_r. S then adopts a public key encryption approach to encrypt the TK_{T_r} with the public key. S then splits the corresponding private key into multiple secret shares by adopting Shamir's threshold secret share scheme. A group of trustees, denoted as TE will be recruited from the $\mathcal{SC}$ to jointly protect the shares before T_r and release the shares at T_r. At T_r, when getting the shares and reconstructing the private key, the encrypted time token can be decrypted. The data lifetime control is realized by temporally adjusting the secret share of the private key. (3) **Data Recipient:** Recipients receive the data at the release time from the timed data release service. Each recipient holds his/her own attributes as well as a published time token to decrypt the data. Without loss of generality, such recipients are formally captured by $\mathcal{R} = \{R_1, ..., R_m\}$. (4) **Decentralized Storage Service:** In our framework, we use IPFS [11] to provide a decentralized storage service. In particular, IPFS takes charge of storing the encrypted data and related information in T2CS.

3.1.2 Data Control Primitives

Keeping the key entities of *CTDRB* in mind, we formally introduce two types of controllable data primitives supported by our framework, namely *data access control* and *data lifetime control*. Formal descriptions are given as follows: (1) *data access control:* the data access control aims at limiting the access scope of the data in a timed release service. After getting the time token, only the recipients holding satisfying attribute set can decrypt the encrypted data. (2) *data lifetime control:* we design two representative data lifetime control primitives, namely *data revocation* and *data release time modification*. The data revocation primitive refers to revoking the data, by the data sender S, at a time, namely T', before T_r. The completeness of such a primitive renders the data inaccessible to anyone except for S; the data release time modification primitive is also issued by the sender S, which allows the encrypted data to be accessible at a time before T_r.

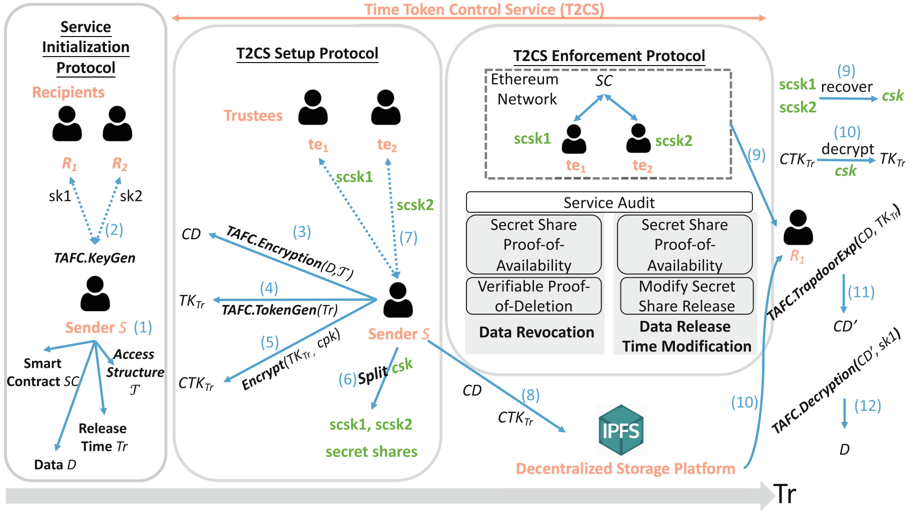


Fig. 2. CTDRB overview

3.1.3 Workflow Overview

To systematically support such primitives, the proposed approach tightly couples the four entities by carefully designing a suite of protocols, consisting of *Service Initialization Protocol*, *T2CS Setup Protocol*, and *T2CS Enforcement Protocol*. We first sketch the proposed protocols at a high-level here and we present detailed constructions in Sect. 4. As shown in Fig. 2, CTDRB starts with the service initialization protocol. In this protocol, the sender S aims at initializing the basic service information (step 1) and enabling the data access control by distributing attribute secret key to the recipients (step 2). After the initialization protocol, CTDRB moves to the T2CS phase, which enables the data lifetime control. Specifically, T2CS consists of two protocols, T2CS setup protocol and T2CS enforcement protocol. In the T2CS setup protocol, S encrypts the data D with the specified access structure $\mathcal{T}$ to generate the ciphertext CD (step 3) and generates a time token TK_{T_r} related to the release time T_r (step 4) and encrypts TK_{T_r} to get the ciphertext CTK_{T_r} with the public key cpk (step 5). The corresponding private key csk is then split to multiple shares by adopting $sss(t,n)$ (step 6). S then distributes the shares to a set of trustees (step 7). At the end of this protocol, S uploads CD and CTK_{T_r} to IPFS (step 8). After the setup, the T2CS enforcement protocol starts. It includes two suites of protocols to support the data revocation primitive and the data release time modification primitive, which are issued upon S's request. If no data lifetime control is needed, after the release time T_r, any recipients, say R_1, first retrieves the secret shares from the Ethereum network and recovers the csk (step 9). R_i then decrypts the encrypted time token CTK_{T_r} to get TK_{T_r}. Followed by exposing the time trapdoor (step 11), R_i can adopt his/her attribute key, sk_1, to get the original data D (step 12).

3.2 Adversarial Model and Assumptions

From an adversarial model standpoint, we make the following assumptions: (1) we assume that the data sender S is always honest when engaging in a timed release service, (2) by agreeing with *TAFC* [14], we assume that the set of recipients are prone to launch collusion attacks among $\mathcal{R}$ to access an unauthorized data. Moreover, given the $T2CS$ design, the recipients, if they know who are the trustees, are also able to collude with the trustees to pre-maturelly release the time token, which renders an illegal pre-mature release of the data, (3) the trustees in T2CS are modeled as rational adversaries who are driven by self-interest and only choose to violate timed-release service protocol when doing so let him/her earn a higher profit [17], (4) we assume that *IPFS* may act in a *honest-but-curious* manner and always provides a reliable storage service during the lifecycle of a timed release service.

4 *CTDRB:* A Holistic View

In this section, we illustrate our controllable framework by formally describing its concrete constructions.

4.1 Service Initialization Protocol

CTDRB starts with the Service Initialization Protocol, namely Π_{init}. Π_{init} aims at initializing the service parameters (Phase-1) and distributing the attribute secret key (Phase-2) to enable data access control. Π_{init} includes the following assumptions: (1) we assume that S and $\mathcal{R}$ know each other *apriori* and are in possession of the public key of each other; also, they adopt the symmetric key encryption approach in [17] to realize private communications. (3) only step-3 is an on-chain operation, and it is publicly known. We adopt the boxed convention to indicate on-chain operations in the rest of this paper. The details of Π_{init} are shown below:

Protocol: Service Initialization Π_{init}

Phase 1: Service Parameter Initialization

Sender S:

1. $(pk, mk) \leftarrow TAFC.Setup$

2. prepares D, specifies an expected release time T_r, and prepares an universal set of attribute $\mathcal{U}$.

$\boxed{3}$. deploys a smart contract $\mathcal{SC}$ on *Ethereum* and publish $T_r, \mathcal{U}$ in $\mathcal{SC}$.

Phase 2: Attribute Secret Key Distribution:

Sender S:

4. sends a request, req_i, to each recipient R_i through a secure off-chain channel. $req_i := \{'\text{init}', \mathcal{U}, T_r, Addr(\mathcal{SC}), Sig_{privk_S}(hash('\text{init}', \mathcal{U}, T_r, Addr(\mathcal{SC}))\}$

Recipient $R_i \in \mathcal{R}$:

5. upon getting req_i, verifies the service information $T_r, \mathcal{U}$.

6. prepares an attribute set $\mathcal{A}_i$ further adopted to perform decryption in *TAFC*.

7. issues a response, $resp_i$, to S for his/her secret key, $sk_{\mathcal{A}_i}$. Concretely, we have $resp_i := \{'\text{request secret key}', \mathcal{A}_i, Sig_{privk_{R_i}}(hash('\text{request secret key}', \mathcal{A}_i))\}$

Sender S:

8. upon receiving $resp_i$, generates the data decryption key $sk_{\mathcal{A}_i} \leftarrow TAFC.KeyGen(mk, \mathcal{A}_i)$
9. issues a new message, namely $dist_i$ to R_i to distribute the generated $sk_{\mathcal{A}_i}$, where $dist_i := \{'\text{secret key}', sk_{\mathcal{A}_i}, Sig_{privk_S}(hash('\text{secret key}', sk_{\mathcal{A}_i}))$

4.2 T2CS Setup Protocol

After the provisioning in Π_{init}, *CTDRB* moves to the T2CS Setup protocol, namely Π_{setup}. The objectives of Π_{setup} are two-fold: (1) in Phase-1, S individually encrypts D and generates a time token, (2) in Phase-2, S interactively communicates with the trustees to provision T2CS to enable the data lifetime control. Several key designs are mentioned as follows: (1) In step-6, we assume that there are already N trustees registered in $\mathcal{SC}$, where $N > n$. Specifically, we provide an interface in $\mathcal{SC}$ to let anyone who wants to participate in T2CS by submitting his/her EOA address, public key, and a security deposit d to $\mathcal{SC}$ at any time point, (2) As mentioned in Sect. 3.2, *CTDRB* is under the risk of the collusion between the trustees as well as the recipients. As a countermeasure, we henceforth privately keep the recruitment evidence of the selected trustees, which keeps the recipients and the trustees double-blind. Moreover, to further help sender to retrieve the shares, in step-10, S generates a cryptographic nonce for each selected trustee te_j, namely rn_j, to build a map to index the hash of each share without revealing the real identity of each selected trustee in $\mathcal{SC}$. The details of Π_{setup} are illustrated below. After the execution of Π_{setup}, the time travel of D normally starts.

Protocol: T2CS Setup Π_{setup}

Phase 1: Data Encryption and Time Token Generation

Sender S:

1. specifies an access structure $\mathcal{T}$ for D and gets $CD \leftarrow TAFC.Encryption(pk, D, \mathcal{T})$
2. generates a time token TK_{T_r} for T_r,where $TK_{T_r} \leftarrow TAFC.TokenGen(mk, T_r)$

Phase 2: T2CS Provisioning

Sender S:

3. generates a pair of keys to encrypt TK_{T_r}, namely *time token control key pair*, involving a control public key cpk, and a control private key csk.
4. S encrypts TK_{T_r} with cpk to get CTK_{T_r}.
5. S splits csk into a list of shares by parameterizing $sss(t, n)$ with a pre-determined t and n. We denote $SS = \{scsk_1, ..., scsk_n\}$ to represent list of shares of csk.
6. S randomly selects n trustees from $\mathcal{SC}$ to form a set of trustees, denoted as $TE = \{te_1, ..., te_n\}$, to further jointly take charge of protecting SS until T_r.
7. S issues a request rec_req_j to te_j through an off-chain channel, specifically $rec_req_j := \{'\text{recruit}', T_r, Addr(\mathcal{SC}), Sig_{privk_S}('\text{recruit}', hash(T_r, Addr(\mathcal{SC}))\}$.

Each Selected Trustee: $te_j \in TE$:

8. first performs verification when receiving rec_req_j, then in case of agreeing with participation, te_j sends S a response as follows:
$rec_res_j := \{'\text{agree}', Sig_{privk_{te_j}}(hash('\text{agree}', T_r, Addr(\mathcal{SC}))\}$.

Sender S:

9. Upon getting rec_res_j from te_j, S first generates a cryptographic nonce for te_j, denoted as rn_j.
10. hashes $scsk_j$ to get $hash(scsk_j)$ and constructs a tuple M_j to index rn_j through the address of te_j, where $M_j := \{Addr(te_j), rn_j\}$; also, S constructs another tuple, $MH_j := \{rn_j, hash(scsk_j)\}$ to index the share hash through rn_j.

11. distributes $scsk_j$ to te_j by sending a share distribution message: $dist_share_j := \{'\text{share}', scsk_j, Sig_{privk_S}(hash('\text{share}', scsk_j))\}$.
12. After receiving all responses from TE, S ends the protocol as follows:
12.a uploads CD as well as CTK_{T_r} to IPFS, which gives S two pointers, namely $pt(CD)$ and $pr(CTK_{T_r})$ respectively.
12.b builds two maps, one is $\mathcal{M}$,the other is $\mathcal{MH}$. Specifically, $\mathcal{M} := \bigcup_{j=1}^{n} M_j$, and $\mathcal{MH} := \bigcup_{j=1}^{n} MH_j$
$\boxed{12.c}$ keeps $\mathcal{M}$ private and publishes $\mathcal{MH}$, $pt(CD)$, as well as $pt(CTK_{T_r})$ on $\mathcal{SC}$.
$\boxed{12.d}$ updates the state in $\mathcal{SC}$

4.3 T2CS Enforcement Protocol

The T2CS Enforcement Protocol forms a key part of *CTDRB*. It aims at systematically supporting the data revocation as well as the data release time modification primitive. Concretely, the expected data control primitives will be realized in terms of controlling the publishing lifecycle of the time token TK_{T_r}. Naturally, it is equivalent to manually controlling the lifecycle of the csk in terms of temporally adjusting the publishing of the secret shares of csk. Thus, in this protocol, we temporally control the shares held by the set of trustees. The T2CS enforcement protocol consists of four sub-protocols: (1) Secret Share Proof-of-Availability, (2) Data Revocation, (3) Data Release Time Modification and (4) Service Audit, which are detailed as follows:

(1) Secret Share Proof-of-Availability (Π_{poa}): Π_{poa} aims at checking the availability of the secret shares SS of csk before performing the data lifetime control primitives, which is shown as follows:

Protocol: Secret Share Proof-of-Availability Π_{poa}

At a time point T', where $T' < T_r$
Phase 1 Challenging the Availability of the Secret Share
Sender S**:**
1. locally indexes $\mathcal{M}$ to check the set of responsible trustees.
2. sends a challenge request, $avail_challenge_j$ for each te_j in $\mathcal{M}$ one by one, specifically $avail_challenge_j := \{'\text{check share}', Sig_{privk_S}(hash('\text{check share}'))\}$
Phase 2 Availability Proof Generation:
Challenged Trustee te_j**:**
3. After verifying $avail_challenge_j$, te_j attaches $scsk_j$ to a response message to S, denoted as $avail_resp_j := \{'\text{share}', scsk_j, Sig_{privk_{te_j}}(hash('\text{share}', scsk_j))\}$.
Phase 3 Sender Verification:
Sender S**:**
4. After getting $scsk_j$, S hashes $scsk_j$ in terms of adopting $h' \leftarrow hash(scsk_j)$ to get h'.
5. S checks h' with the original one in $\mathcal{MH}$ in $\mathcal{SC}$.
5.a If they match, S continues to challenge next trustee;
$\boxed{5.b}$ otherwise, S publishes the misbehavior on $\mathcal{SC}$
6. S ends the availability check as follows:
$\boxed{6.a}$ Case 1: S is confirmed that at least t shares are available, which successfully ends Π_{poa}. Π_{poa} outputs $TRUE$, and S updates the state in $\mathcal{SC}$

$\boxed{6.b}$ Case 2: S is confirmed that the available shares cannot recover csk. Π_{poa} outputs $FALSE$. This aborts the current timed data release service.

With the help of Π_{poa}, next, we show the detailed design of the data revocation and the data release time modification primitive.

(2) Data Revocation (Π_{rvk}): S performs the data revocation primitive by executing Π_{rvk}. Specifically, inspired by [13], Π_{rvk} incorporates Π_{poa} with our proposed *Verifiable Proof-of-Deletion* (VPoD) mechanism to realize the data revocation primitive. The proposed VPoD aims at deleting the shares of csk held by the set of trustees. This operation then makes the decryption of CTK_{T_r} impossible. In case TK_{T_r} is not available by decrypting CTK_{T_r} at T_r, even the recipients that hold satisfying attributes, cannot decrypt CD. The details of Π_{rvk} are shown as follows:

Protocol: Data Revocation Π_{rvk}

At a time point T', where $T' < T_r$, the following steps capture the data revocation primitive:

Phase 1: Check Availability

Sender S:

1. updates the state of $\mathcal{SC}$ to RVK_P (revocation pending)
2. interacts with each te_j in TE to check availability in terms of adopting Π_{poa}. If Π_{poa} gives $TRUE$, S moves to step 3; otherwise, S aborts the current service.

Phase 2: Verifiable Proof-of-Deletion

3. S sends each te_j a deletion request one by one, denoted as del_req_j, where $del_req_j := \{'\text{share deletion}', Sig_{privk_S}(hash('\text{share deletion}'))\}$

Each Trustee te_j who have passed Π_{poa} :

4. Upon receiving del_req_j, te_j deletes the share $scsk_j$.
5. After the deletion, te_j generates a deletion response with an evidence, denoted as del_resp_j, where $del_resp_j := \{'\text{deleted}', Sig_{privk_{te_j}}(hash('\text{deleted}', scsk_j))\}$

Sender S:

$\boxed{6.}$ After getting the corresponding del_resp_j, S validates it and publishes it on-chain in $\mathcal{SC}$. Specifically, S only needs to guarantee that $(n - t + 1)$ shares are deleted.

$\boxed{7.}$ Once S already gets $(n - t + 1)$ share deletion evidences, S ensures that the csk is unrecoverable. S then transfers the state from RVK_P to RVK_E and ends the service.

In Π_{rvk}, the design of step-5 and step-6 enables verifiable capability. Specifically, since S publishes the recruitment relationship as well as the evidence in step-6, anyone discovering the deleted share can perform verification. We will introduce the detailed verification design, namely *Cheating Misbehavior in VPoD*, in the service audit protocol.

(3) Data Release Time Modification (Π_{mod}): We next discuss the realization of the data release time modification primitive. Specifically, after checking the availability, we modify the release time by informing each te_i to release the share of csk at T'_r, a new release time before T_r. The detailed protocol Π_{mod} is illustrated as follows:

Protocol: Data Release Time Modification Π_{mod}

At a time point T', **Sender S** would like to modify the originally prescribed release time T_r to T'_r, where $T' < T'_r < T_r$. S performs the following steps:

1. S updates the prescribed time T_r in $\mathcal{SC}$ to T_r' and the state of $\mathcal{SC}$ to a new state MOD_P (modification pending).
2. S notifies the recipients $\mathcal{R}$ the modification.
3. S interacts with each te_j in TE to check availability by adopting Π_{poa}. If Π_{poa} gives $TRUE$, S moves to step 4; otherwise, S aborts the current service.

Each Trustee te_j:

4. S sends a release time modification proposal, denoted as mod_req_j to each te_j, where $mod_req_j := \{'\text{modification}', T_r', Sig_{privk_S}(hash('\text{modification}', T_r'))\}$.
5. After validating mod_req_j, te_j constructs an agreement, denoted as $mod_resp_j := \{'\text{agree}', Sig_{privk_{te_j}}(hash('\text{agree}', scsk_j, T_r'))$ and sends to S.

Sender S:

6. S updates the state of $\mathcal{SC}$ to MOD_A (modification approved)

When the time hits T_r', **Each Trustee te_j:**

7. te_j publishes his/her share $scsk_j$ with a signature $Sig_{privk_{te_j}}(hash(scsk_j))$ on $\mathcal{SC}$.

After T_r', **Each Recipient $R_i \in \mathcal{R}$:**

8. Retrieves all the submitted shares in terms of interacting with $\mathcal{SC}$.
9. Reconstructs csk from the secret sharing scheme.
10. Verifies csk with the original one in $\mathcal{SC}$ in terms of checking hash. In case verification successes, R_i then gets $pt(CTK_{T_r})$ as well as $pt(CD)$ from $\mathcal{SC}$ to retrieve IPFS and decrypt CTK_{T_r} with csk to get TK_{T_r}.
11. By holding TK_{T_r}, R_i adopts $TAFC.TrapdoorExp(TK_{T_r}, CD)$ to expose the time trapdoor, which gives R_i a modified ciphertext of CD, namely CD'
12. R_i adopts $TAFC.Decryption(CD', sk_{\mathcal{A}_i})$ to get D at T_r'.

(4) Service Audit (Π_{aud}): Since our controllable primitives highly depend on the decentralized $T2CS$ design, guaranteeing the completeness of the execution of the primitives is quite important. In this part, we provide the Service Audit Protocol, which captures the corresponding misbehavior as well as makes the execution of the controllable primitives transparent. We outline the detailed design below.

Protocol: Service Audit Π_{aud}

Cheating Misbehavior in *VPoD*: Any trustee or recipient who discovers $scsk_j$ after T' can adopt the following steps to submit a misbehavior report:

1. Anyone jointly holding the state of $\mathcal{SC}$ is RVK_E, a discovered share $scsk_j$, the public key of the corresponding trustee $pubk_{te_j}$, as well as the evidence submitted by te_j, $Sig_{privk_{te_j}}(hash('\text{deleted}', scsk_j))$, can locally verify and report them with $\mathcal{SC}$, then the reporter can get rewards.

Missing Share Report in Π_{mod}: Any recipients failing to get the original secret recovered can adopt the following steps to submit a missing share report:

2. In case R_i fails to verify the reconstructed R_i, R_i submits a request to S.
3. S then publishes the recruitment relationship.
4. R_i verifies the shares one by one by checking the original hash of shares published by S.
5. R_i reports the misbehavior trustees and aborts the current service ahead of time.

5 Security Analysis

In this section, we analyze the security of *CTDRB* from the following aspects:

Collusion Attacks Among the Recipients: As we mention in Sect. 3.2, after getting the published time token, the set of recipients may perform collusion misbehavior in such a way that the colluding recipients pool their secret keys together to forge a new secret key associated with a satisfying attribute set to decrypt the data. Due to the construction in $TAFC.KeyGen$, each user's $sk_{\mathcal{A}_i}$ corresponding to the attribute set $\mathcal{A}_i$ is associated with a secure random number. Such a number will be adopted in the $TAFC.Decryption$ phase. In case an adversary tries to combine different secret keys coming from different sets of attributes to forge a new secret key, the forged one will render a different random number. Even though the collusion behavior may acquire a satisfying attribute set, the wrong random number will lead to a failed decryption. Therefore, by seamless adopting $TAFC$, *CTDRB* is resistant to the collusion attacks among the recipients.

Data Confidentiality: In *CTDRB*, we prevent the time token release-ahead attack by anonymizing the recruitment relationship of the selected trustees. Though such a countermeasure sets a barrier to the adversary, is it important to quantitatively measure the protection effectiveness of the scheme. We provide the details of our analysis, we highlight the key settings first. We assume that there are totally N registered trustees in the smart contract $\mathcal{SC}$. The secret share scheme $sss(t, n)$, determined by the sender, has two fixed parameters, t and n. We also have the ordered relationship $t \leq n < N$. With our rational adversarial setting of the trustees, we assume that the deposit of each trustee is denoted as d. By following the common attack strategy in [18], we consider the attack approach, namely bribery attack [18]. Such an attack is launched by the adversary by deploying a bribery contract to tempt each trustee to disclose his/her share.

Resilience to Bribery Attack: We analyze the resilience in terms of capturing the success probability of such an attack. Given a determined secret share scheme $sss(t, n)$, the notion of authorized sets refers to the set of parties that are able to reconstruct the original secret. In $sss(t, n)$, any set of trustees consisting of at least t members is an authorized set. Without loss of generality, in TE, we capture the collection, namely $\mathcal{TE}_t$, of all the subsets of TE consisting of at least t trustees, as follows:$\mathcal{TE}_t = \{TE' \subset \{te_1, te_2, ..., te_n\} : |TE'| \geq t\}$. We also denote $n(TE')$ as the number of the subsets of TE that consists of at least t trustees. Hence, $n(\mathcal{TE}_t) = \sum_{i=t}^{n} \binom{n}{i} \leq 2^n$. In our scheme, $\mathcal{TE}_t$ is the target collection which the adversary aims for. Bribing any one set involved in $\mathcal{TE}_t$ can render the bribery attack successful. However, due to our anonymization design, from the list in $\mathcal{SC}$ consisting of all registered trustees, the adversary cannot distinguish which trustees are selected. We assume that there are totally N registered trustees. Thus, the number of the subset of N registered trustees is 2^N. The attack success probability is then defined as a function in N, where $p(N) = \frac{n(\mathcal{TE}_t)}{2^N}$. Since $f(N) = 2^{-N}$ is negligible and $n(\mathcal{TE}_t)$ is a constant, $p(N)$ is also negligible. Therefore, our anonymization design renders the success probability of the bribery attack very small and negligible.

6 Evaluations

We discuss the details of our implementation and evaluation in this section. We start by describing the key implementations adopted in our framework in Sect. 6.1. We first

provide detailed smart contract implementations with gas cost in Sect. 6.2.1. Then, in Sect. 6.2.2, we discuss the extensive case studies we performed.

6.1 Implementations and Environment

We implement the cryptographic constructions of *TAFC* in *Charm* [8], a cryptographic library implemented in *Python* [5]. For the secret sharing operation, we adopt the shamir's secret sharing implemented in *Charm*. We adopt the *Keccak256* hash implementation in *Web3.py* [6]. For the digital signature, we use the *ECDSA* implementation supported by *Web3.py*.

We use *Brownie* [1] to provide an interactive environment to work with *Ethereum*. Specifically, in *Brownie*, we implement the design of $\mathcal{SC}$ in *Solidity* [7], a smart contract oriented programming language supported by *Ethereum*. We create multiple accounts in *Brownie* acting as the sender S, the set of recipients $\mathcal{R}$, as well as the set of trustees TE. For IPFS, we employ Infura [3] to use the service.

6.2 Evaluations

6.2.1 Smart Contract Implementation

Table 1. Key functions & gas cost

Protocol	Step	Function	Gas cost
Service Initialization	$\Pi_{init}.3$	*deployment*	665964
	$\Pi_{init}.3$	*newService*	132024
T2CS Setup	$\Pi_{setup}.12.c$	*setOriHash*	106864
	$\Pi_{setup}.12.c$	*submitCDPointer*	42707
	$\Pi_{setup}.12.c$	*submitCTKPointer*	42706
T2CS Enforcement	$\Pi_{mod}.1$	*modifyReleaseTime*	28149
	$\Pi_{rvk}.6$	*submitDeletionEvidence*	63117
	$\Pi_{mod}.7$	*submitShare*	41757
	$\Pi_{mod}.7$	*submitShareSignature*	24781
	$\Pi_{poa}.5.b$	*reportUnavailableShare*	41021
Service Audit	$\Pi_{aud}.1$	*cheatingDeletionReport*	43594
	$\Pi_{aud}.3$	*publishRelationship*	63684
	$\Pi_{aud}.5$	*missingShareReport*	42860
Others	-	*trusteeRegistration*	78121
	-	*reward*	41213
	-	*stateTransition*	21042

In Table 1, we show our implemented functions with corresponding gas cost in the smart contract. Gas cost is a key metric to economically measure the cost-effectiveness of a deployed smart contract. By analyzing the table, we observe that

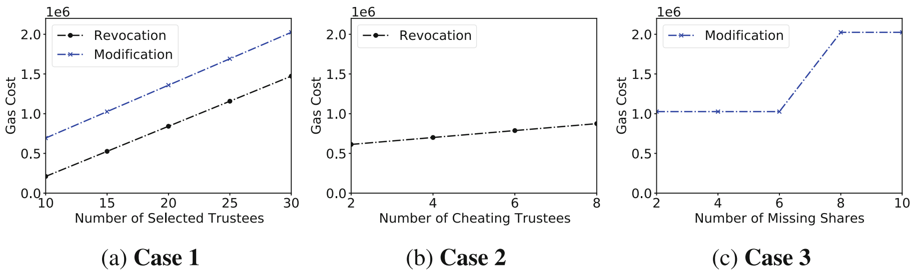
there are three functions, $deployment$ (665964), $newService$ (132024), as well as $setOriHash$ (106864), incurring a relatively higher gas cost in comparison with the others. Since such functions will only be invoked once for each timed data release service, their gas cost is a one-time cost.

Next, we highlight the key functions associated with the data revocation primitive as well as the data modification primitive: **Data Revocation:** The function invocations of the data revocation primitive consist of the following scenarios: (1) If no unavailable share and no cheating proof-of-deletion behavior exists, the revocation primitive involves $submitDeletionEvidence$ and $stateTransition$; (2) In case of the existence of unavailable shares as well as cheating proof-of-deletion behaviors, the revocation primitive totally involves $submitDeletionEvidence$, $stateTransition$, as well as $reportUnavailableShare$. **Data Release Time Modification:** The function invocations of the data release time modification primitive consist of the following scenarios: (1) If no unavailable share and no missing share, the modification primitive involves $modifyReleaseTime$, $submitShare$, as well as $submitShareSignature$; In case of the existence of unavailable shares as well as missing shares, the primitive totally involves $modifyReleaseTime$, $submitShare$, $submitShareSignature$, $publishRelationship$, as well as $missingShareReport$.

6.2.2 Case Study

We perform case studies to investigate gas consumption for our data revocation as well as data release time modification primitives.

Case 1-No Misbehavior: In this case study, we assume that all the trustees honestly follow the T2CS service. The objective of this study is to study the gas consumption of the two primitives when selecting different number of trustees. We set the threshold t of $sss(t, n)$ as $t = 8$ in this case study. In Fig. 3a, we observe that the gas cost incurred by the two control primitives linearly increases with the growing number of selected trustees. Such an observation follows our design, since in the data revocation protocol Π_{rvk}- step 6, the sender S must guarantee that $(n - t + 1)$ trustees have performed proof-of-deletion and publish them on-chain; also, for the data modification protocol Π_{mod}-step-7, we require all the selected trustees to submit his/her share.



(a) **Case 1** (b) **Case 2** (c) **Case 3**

Fig. 3. Case study results

Case 2-Cheating Proof-of-Deletion Existence: In this case study, we study the impact of number of cheating trustees on the gas consumption of the data revocation primitive. We assume that all trustees have passed the proof-of-availability check. Also, we adopt $sss(8, 15)$ as the secret sharing scheme in this case. Figure 3b gives us the gas evaluation results. Specifically, we observe that the gas cost incurred by the data revocation primitive linearly grows with the increasing number of cheating trustees. Such an observation empirically validates our design in Π_{aud} step-1, since the cheating behavior will trigger the auditing protocol when performing a data revocation primitive, in which anyone captures such a misbehavior prefers to report with $\mathcal{SC}$ to earn rewards. The rising number of cheating trustees renders an increasing number of invocations of *cheatingDeletionReport*, which inevitably incurs more gas cost.

Case 3-Missing Share Existence: In this case study, we study the impact of number of missing shares on the gas cost when performing a data release time primitive. We assume that all the selected trustees have passed the proof-of-availability check. Also, we adopt sss $(8, 15)$ as the secret sharing scheme in this case. Figure 3c shows the evaluation results. The results show two interesting observations, which are described as follows: *first*, when the number of missing shares is less than the threshold t, say 2, 4, and 6, the gas cost of the data release time modification primitive does not grow with the increasing number of missing shares. Here also, this observation follows our design as the recipient can recover the csk from the remaining shares, which will not trigger the auditing protocol; *second*, when we miss 8 shares, the gas cost dramatically increases. This empirical result validates our design in Π_{aud} step-3 and step-5. Since the remaining shares cannot recover csk, Π_{aud} will be triggered. The sender S will publish the relationship on $\mathcal{SC}$ in terms of issuing transactions, which causes more gas cost. Moreover, when we miss 10 shares, the gas cost remains the same. This observation is attributed to the number of the invocations of $publishRelationship$ that only relies on the total number of selected trustees.

7 Related Work

7.1 Timed Data Release Using Blockchains

With the recent advancements in blockchain techniques, we are witnessing a rising research trend in timed data release using blockchains. Specifically, one line of work aims at enriching theoretic foundations in terms of constructing elegant cryptographic solutions for timed release of time. The techniques outlined in [20–22] incorporate witness encryption with cryptographic puzzles constructed from blockchains to securely enclose the timed-release data. The other line of work strives to decentralize timed data release using real-world blockchains such as *Ethereum*. Based on adversarial contexts, this line of work can be categorized into the following two categories. The first category seeks to protect data under a rational adversary setting. Specifically, Li et al. [17] adopted a game theory model to contractually enforce the Ethereum blockchain networks to behave honestly in a timed data release service. As a concurrent work, Ning et al. [24] proposed a provable cryptographic construction to guarantee the protection of the data with the help of Shamir's threshold secret share scheme and smart contracts.

By considering a potential attack launched by a data sender, the techniques developed by Bacis et al. [10] incorporated secure multiparty computation with the *Ethereum* smart contracts to jointly perform data protection. Recently, SilentDelivery [18] suggested a novel design to realize anonymity and scalability in blockchain-based timed data release. Recent work also has investigated a mixed adversarial environment, where both fully malicious adversaries as well as fully rational adversaries exist. Specifically, Wang et al. [32,33] proposed a reputation-aware timed data release protocol on top of Ethereum to achieve high attack resilience in a mixed adversarial environment. All the current techniques, however, only focus on the issue of data protection. In contrast, our proposed work in this paper augments the existing blockchain-based timed data release techniques with practically controllable primitives.

7.2 Temporal-Aware Data Control in Public Outsourced Environments

There have been several efforts on supporting temporal data controls in outsourced environments. The first line of work focuses on realizing data lifetime control in untrusted storage environment, namely *assured file deletion*. Such a notion starts from the design [26] proposed by Perlman, which theoretically constructs a system that manually specifies the expiry time of a file and permanently deletes it after the expiry time with the help of public key infrastructure [25]. FADE [30], proposed by Tang et al., further generalizes the primitive in [26] as *policy-based file assured deletion* in real-world storage platforms. Specifically, such a design specifies policy to regulate file lifetime and assured file deletion request. The work presented in [31] enrich this design with access control by using CP-ABE [12]. Our work differs from this line of work along the following two aspects: (i) unlike prior work that focused on the assured deletion primitive, the proposed work focuses on the timed data release scheme and (ii) while prior work on data lifetime control primitive assumed a centralized key management service, we note that it is impractical in a decentralized blockchain environment. Therefore, directly adopting such prior techniques to perform data lifetime control in our blockchain-based timed data release framework is infeasible. (2) The second line of work enhances cloud-based applications with temporal-aware access control primitives. Specifically, Zhu et al. [36] developed temporal access control by adopting cryptographic integer comparisons and proxy re-encryption encryption scheme. LoTAC [9] enables location and time-based access control on cloud-stored data with the help of ElGamal encryption and tag-based encryption, which specifically grants access to data by considering the location as well as time information of a user. Adopting CP-ABE [12] to realize temporal-aware access control is proposed in [14,15,19,23,34,35], where time is treated as an attribute in CP-ABE. This line of work, however, only focused on enabling data access control with temporal-related constraints. Such techniques fail to meet our joint objective of data lifetime control and data access control.

8 Conclusion

In this paper, we propose *CTDRB*, a controllable timed data release framework using blockchains, which offers both data access control as well as data lifetime control while

protecting the confidentiality of the data. With the help of TAFC, a variant ciphertext-policy attribute-based encryption construction, we realize the data access control in *CTDRB*. We implement data lifetime control by designing a time token control service on Ethereum. On top of such a design, two representative data lifetime control primitives namely, data revocation and data release time modification are carefully designed. Our security analysis demonstrates the attack resilience of *CTDRB*. We prototype *CTDRB* by implementing smart contracts on Ethereum and perform gas cost evaluations. By performing case studies under various service conditions, we demonstrate that our data lifetime control primitives incur only a modest gas cost.

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.

References

1. Brownie. https://eth-brownie.readthedocs.io/en/stable/
2. Ethereum. https://ethereum.org/en/
3. Infura. https://infura.io/
4. Kimono: trustless secret sharing using time-locks on ethereum. https://github.com/hillstreetlabs/kimono
5. Python. https://www.python.org/
6. A python interface for interacting with the ethereum blockchain and ecosystem. https://github.com/ethereum/web3.py
7. Solidity. https://docs.soliditylang.org/en/v0.8.10/
8. Akinyele, J.A., et al.: Charm: a framework for rapidly prototyping cryptosystems. J. Cryptographic Eng. **3**(2), 111–128 (2013)
9. Androulaki, E., Soriente, C., Malisa, L., Capkun, S.: Enforcing location and time-based access control on cloud-stored data. In: 2014 IEEE 34th International Conference on Distributed Computing Systems, pp. 637–648. IEEE (2014)
10. Bacis, E., Facchinetti, D., Guarnieri, M., Rosa, M., Rossi, M., Paraboschi, S.: I told you tomorrow: practical time-locked secrets using smart contracts. In: The 16th International Conference on Availability, Reliability and Security, ARES 2021, New York, NY, USA. Association for Computing Machinery (2021)
11. Benet, J.: IPFs-content addressed, versioned, p2p file system. arXiv preprint arXiv:1407.3561 (2014)
12. Bethencourt, J., Sahai, A., Waters, B.: Ciphertext-policy attribute-based encryption. In: 2007 IEEE Symposium on Security and Privacy (SP 2007), pp. 321–334. IEEE (2007)
13. Hao, F., Clarke, D., Zorzo, A.F.: Deleting secret data with public verifiability. IEEE Trans. Dependable Secure Comput. **13**(6), 617–629 (2016)
14. Hong, J., et al.: TAFC: time and attribute factors combined access control for time-sensitive data in public cloud. IEEE Trans. Serv. Comput. **13**(1), 158–171 (2017)
15. Huang, Q., Yang, Y., Fu, J.: Secure data group sharing and dissemination with attribute and time conditions in public cloud. IEEE Trans. Serv. Comput. **14**, 1013–1025 (2018)
16. Jiang, P., Qiu, B., Zhu, L.: Toward reliable and confidential release for smart contract via id-based TRE. IEEE Internet Things J. **9**(13), 11422–11433 (2022)

17. Li, C., Palanisamy, B.: Decentralized release of self-emerging data using smart contracts. In: 2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS), pp. 213–220. IEEE (2018)
18. Li, C., Palanisamy, B.: SilentDelivery: practical timed-delivery of private information using smart contracts. IEEE Trans. Serv. Comput. (to appear)
19. Li, Y., Dong, Z., Sha, K., Jiang, C., Wan, J., Wang, Y.: TMO: time domain outsourcing attribute-based encryption scheme for data acquisition in edge computing. IEEE Access **7**, 40240–40257 (2019)
20. Liu, J., Garcia, F., Ryan, M.: Time-release protocol from bitcoin and witness encryption for sat. Korean Circulation J. **40**(10), 530–535 (2015)
21. Liu, J., Jager, T., Kakvi, S.A., Warinschi, B.: How to build time-lock encryption. Des. Codes Crypt. **86**(11), 2549–2586 (2018)
22. Liu, J., Kakvi, S.A., Warinschi, B.: Extractable witness encryption and timed-release encryption from bitcoin. IACR Cryptology ePrint Archive, 2015:482 (2015)
23. Liu, Z., Jiang, Z.L., Wang, X., Yiu, S.-M., Zhang, R., Wu, Y.: A temporal and spatial constrained attribute-based access control scheme for cloud storage. In: 2018 17th IEEE International Conference on Trust, Security And Privacy In Computing and Communications/12th IEEE International Conference on Big Data Science and Engineering (TrustCom/BigDataSE), pp. 614–623. IEEE (2018)
24. Ning, J., Dang, H., Hou, R., Chang, E.-C.: Keeping time-release secrets through smart contracts. IACR Cryptology ePrint Archive, p. 1166 (2018)
25. Perlman, R.: The ephemerizer: Making data disappear (2005)
26. Perlman, R.: File system design with assured delete. In: Third IEEE International Security in Storage Workshop (SISW 2005), p. 6. IEEE (2005)
27. Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock puzzles and timed-release crypto (1996)
28. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (1979)
29. Szabo, N.: Formalizing and securing relationships on public networks. First Monday (1997)
30. Tang, Y., Lee, P.P.C., Lui, J.C.S., Perlman, R.: FADE: secure overlay cloud storage with file assured deletion. In: Jajodia, S., Zhou, J. (eds.) SecureComm 2010. LNICST, vol. 50, pp. 380–397. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16161-2_22
31. Tang, Y., Lee, P.P.C., Lui, J.C.S., Perlman, R.: Secure overlay cloud storage with access control and assured deletion. IEEE Trans. Dependable Secure Comput. **9**(6), 903–916 (2012)
32. Wang, J., Palanisamy, B.: Attack-resilient blockchain-based decentralized timed data release. In: 36th Annual IFIP WG 11.3 Conference on Data and Applications Security and Privacy (DBSec2022) (2022, to appear)
33. Wang, J., Palanisamy, B.: Protecting blockchain-based decentralized timed release of data from malicious adversaries. In: 2022 IEEE International Conference on Blockchain and Cryptocurrency (2022)
34. Xia, Q., et al.: TSLS: time sensitive, lightweight and secure access control for information centric networking. In: 2019 IEEE Global Communications Conference (GLOBECOM), pp. 1–6. IEEE (2019)
35. Yang, K., Liu, Z., Jia, X., Shen, X.S.: Time-domain attribute-based access control for cloud-based video content sharing: a cryptographic approach. IEEE Trans. Multimedia **18**(5), 940–950 (2016)
36. Zhu, Y., Hu, H., Ahn, G.-J., Huang, D., Wang, S.: Towards temporal access control in cloud computing. In: 2012 Proceedings IEEE INFOCOM, pp. 2576–2580. IEEE (2012)