Attack-resilient Blockchain-based Decentralized Timed Data Release

Jingzhe Wang and Balaji Palanisamy

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

Abstract. Timed data release refers to protecting sensitive data that can be accessed only after a pre-determined amount of time has passed. While blockchain-based solutions for timed data release provide a promising approach for decentralizing the process, designing a reliable and attack-resilient timed-release service that is resilient to malicious adversaries in a blockchain network is inherently challenging. A timed-release service on a blockchain network is inevitably exposed to the risk of *post-facto attacks* where adversaries may launch attacks after the data is released in the blockchain network. Existing incentive-based solutions for timed data release in Ethereum blockchains guarantee protection under the assumption of a fully rational adversarial environment in which every peer acts rationally. However, these schemes fail invariably when even a single participating peer node in the protocol starts acting maliciously and deviates from the rational behavior.

In this paper, we propose an attack-resilient and practical blockchain-based solution for timed data release in a mixed adversarial environment, where both malicious adversaries and rational adversaries exist. The proposed mechanism incorporates an effective decentralized reputation model to evaluate the behaviors of the peer in the network. Based on the reputation model, we design a suite of novel reputation-aware timed-release protocols that effectively handles the mixed adversarial environment consisting of both malicious adversaries and rational adversaries. We implement a prototype of the proposed approach using Smart Contracts and deploy it on the Ethereum official test network, *Rinkeby*. For extensively evaluating the proposed techniques at scale, we perform simulation experiments to validate the effectiveness of the reputation-aware timed data release protocols. The results demonstrate the effectiveness and strong attack resilience of the proposed mechanisms and incurs only a modest gas cost.

Keywords: Timed Release, Blockchain, Smart Contract

1 Introduction

Timed data release refers to protecting sensitive data that can be accessed only after a pre-determined amount of time has passed. Examples of application using timed data release include secure auction systems where important bidding

information needs protection until arrivals of all bids and secure voting mechanisms where votes are not permitted to be accessed until the close of the polling process. Since the early research effort on timed information release [16], there has been several efforts focusing on providing effective protection of timed release of data. In the past few decades, a number of rigorous cryptographic constructions [5, 6, 8, 9, 21] have enriched the theoretic foundation of the timed-release paradigm to provide provable security guarantees. Even though the theoretic constructions in cryptography provide strong foundations for the development of the timed data release, designing a scalable and attack resilient infrastructure support for timed release of data is a practical necessity to support emerging real-world applications, especially decentralized applications that require timed data release. Recently, a category of network structures, namely self-emerging data infrastructures [19], have been proposed to provide a practical infrastructure support for supporting the timed data release paradigm. Such a self-emerging data infrastructure aims at protecting the data until a prescribed release time and automatically releasing it to the recipient. In such data infrastructures, participating entities of a decentralized peer-to-peer network (e.g., an Ethereum Blockchain network) take charge of protecting and transferring the data. This approach provides an alternate decentralized management of the timed release in contrast to traditional solutions (e.g., cloud storage platforms) that may provide a centralized view to support timed data release. A centralized construction completely relies on a single point of trust that becomes a key barrier to security and privacy, especially in emerging decentralized applications.

Decentralized design of self-emerging data infrastructures [13] has been gaining attention recently with the proliferation of blockchains and blockchain-based decentralized applications. A blockchain provides a public decentralized ledger system operated by global volunteers connected through a *peer-to-peer* network. Powerful consensus protocols such as *Proof-of-Work* guarantee the correctness of operations in a blockchain. Such attractive features of blockchains provide a flexible and reliable design platform for developing decentralized self-emerging infrastructures. While blockchains enable a promising platform for building decentralized infrastructures, several inherent risks have been exposed. In this paper, we particularly focus on two major risks of blockchain-based infrastructures: *First*, the open and public environment in blockchains, where a large number of mutually distrusted participants jointly engage in some services, is full of uncertainty. Such an environment may consist of peers with heterogeneous unpredictable behaviors. One can imagine a scenario where some participants with misbehavior always seek opportunities to sabotage a decentralized service while some other peers may perform actions for seeking maximum profit. *Second*, the blockchain-based infrastructure is inevitably under the threats of *post-facto attacks*, where adversaries may launch potential attacks after the data is released in the blockchain network and control some of the participating peer nodes. In this paper, we materialize such a type of attack with two representative examples, namely *drop attack* and *release-ahead attack*. In a drop attack, an adversary may successfully launch such an attack by destroying the data at any time before the

prescribed release time, which results in the failure of the release of the data. For instance, in a secure biding system, such an attack may destruct the protected biding information before the arrival of all bids. A release-ahead attack may be launched by an adversary who covertly interacts with some participating nodes to intercept the data and perform premature release of such data before the prescribed release time. For example, adversaries may maliciously disclose the protected biding information before the prescribed release time. With such concerns in mind, designing a reliable and attack-resilient timed release service is significantly challenging. Existing blockchain-based protection for timed data release focus on two aspects. *First*, grounded on the game theory, incentive-based solutions [12–14] protect the timed data release from peers with a fully rational context in the Ethereum network. *Second*, a cryptography-based solution [18] is proposed to handle the malicious adversaries who launch incentive-based attacks. However, existing solutions either disregard the heterogeneous marketplace in blockchains or ignore the damage of *post-facto* attacks, which makes blockchain-based timed-release service less practical and secure.

In this paper, we carefully consider a mixed adversarial environment in blockchains where both rational peers and malicious peers exist. Specifically, a rational peer only performs attacks when s/he receives higher profit. A malicious peer always deviates from the timed-release protocol without being concerned of any monetary loss. Designing a strong timed-release protocol that survives in such contexts consisting of heterogeneous unpredictable behaviors incurs multiple challenges. First, the incentive-only mechanism in blockchain-based timed release is not sufficient for evaluating peers with malicious behaviors and it is important to design a metric that is able to effectively capture the behaviors of each peer. Second, it is crucial to measure and quantify attack resilience and designing an attack-resilient timed-release scheme to mitigate the impact of mixed adversarial environment is inherently challenging. Finally, evaluating peers' dynamic behaviors in a decentralized environment is essential to identifying and rewarding honest peer behavior in the system. For addressing these challenges, we first propose an uncertainty-aware reputation measure to evaluate the behavior of each peer. Such a measure captures how likely a peer may perform honest actions or malicious actions in an incoming timed-release request. Based on the uncertainty-based reputation model, we propose a suite of reputation-aware timed-release protocol consisting of two key ingredients: *First*, a reputation-time-aware peer recruitment policy is designed to achieve better drop attack resilience and release-ahead attack resilience while the selected peers' working time windows cover the entire lifecycle of the timed-release service. *Second,* a suite of decentralized on-chain protocols are proposed to guarantee the normal operations of the timed-release protocol. For extensively evaluating our proposed timed-release protocol, we first perform simulation studies using a synthetic dataset to evaluate the techniques at scale. We then implement a proof-of-concept prototype through real-world smart contracts using *Solidity* programming language, and deploy the smart contracts on the *Ethereum* official test net, *Rinkeby*. The results demonstrate that, compared with the existing solutions, the proposed

techniques achieve significantly higher attack resilience while incurring only a modest on-chain gas cost.

In summary, our key contributions of this paper are as follows:

(1) We carefully design an effective uncertainty-based reputation mechanism for blockchain-based timed-release services.

(2) We propose a suite of novel reputation-aware timed-release protocol to construct the expected timed-release scheme that achieves better attack resilience.

(3) We perform extensive evaluations for our proposed protocol in terms of scalable simulations as well as proof-of-concept prototype implementation on official Ethereum test network *Rinkeby*.

The rest of this paper is organized as follows. In Section 2, we provide an overview of the framework as well as the adversary model. Then, we highlight the limitations in the existing works with motivating examples. In Section 3, we first build our reputation model with formal descriptions. Then, using the proposed reputation model, the full view of the construction of our reputation-aware timed-release protocol is unfolded. In Section 4, we perform our simulation studies and show the numerical results, and we implement our proposed techniques through real-world smart contracts in *Solidity*, and we deploy them on the *Ethereum* official testnet *Rinkeby*. The detailed on-chain gas evaluations are also shown in this section. In Section 5, we show the current positions of the timed-release research as well as the related work. In Section 6, we conclude the paper.

2 Background & Motivation

In this section, we first introduce blockchain-based self-emerging data infrastructures. Then, the adversary models, analysis of the limitations as well as motivations are demonstrated with examples.

2.1 Decentralized Timed-release of Self-Emerging Data using Ethereum Blockchain

There are four key components for supporting a timed-release service, namely *Data Sender*, *Data Recipient*, *Cloud*, and *Blockchain Infrastructure* respectively. Without loss of generality, we denote multiple timed-release service requests as $Req = \{req_1, ..., req_m\}$, where $m \in \mathbb{Z}^+$. A pair of data sender and data recipient as well as a group of peers over the Ethereum network are responsible for each request. We formally describe the four key components below.

Data Sender: A new timed-release service is initialized by a data sender S_i, S_i encrypts the private data that needs to be transferred with a secret key, and sends the encrypted private data to a trusted cloud storage platform. Then, at the start time T_s^i of the service req_i, S_i sends the encrypted secret key to the blockchain network and such a key will be released at a prescribed release time T_r^i. By taking into consideration multiple service requests, we denote $\mathcal{S} = \{S_1, ..., S_m\}$ as the data sender set for different timed-release service requests.

Data Recipient: For each timed-release service req_i, the corresponding data recipient R_i is responsible for receiving and decrypting the encrypted secret key sent by the data sender S_i at the prescribed release time T_r^i, and decrypting the encrypted private data from the cloud storage to obtain the original private data. We assume that S_i and R_i can perform negotiation before a new req_i in terms of off-chain interactions. We denote $\mathcal{R} = \{R_1, \ldots, R_m\}$ as the data recipient set for different timed-release service requests.

Cloud: A cloud storage infrastructure acts as a trusted third-party storage platform between a pair of data sender and data recipient to store the encrypted private data.

Blockchain Infrastructure: In our framework, we use the Ethereum network as the core infrastructure for supporting a decentralized service. Specifically, the smart contract is owned by a *Contract Account (CA)*, and each registered peer as well as data senders and data recipients are represented by multiple *Externally Owned Accounts(EOAs)* in the Ethereum account network. Without loss of generality, we denote $\mathcal{SC}$ as the smart contract and $\mathcal{P} = \{P_1, ..., P_j\}$ as the registered peers in $\mathcal{SC}$ for participating the timed-release service.

2.2 Adversary Model

2.2.1 Mixed Adversarial Environment

We consider three different types of peer accounts [1], namely *honest peer*, *rational peer*, and *malicious peer*, existing in the *Ethereum* account network. Specifically, every *honest peer* always participates in timed-release service protocols with absolutely honest actions. This type of peer never performs any malicious actions.

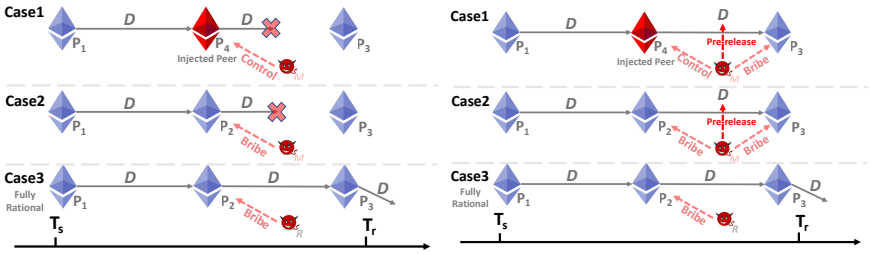
Every *rational peer* acts with economic rationality. Such a type of peer is driven by self-interest and only chooses to violate timed-release service protocol when doing so let him earn a higher profit. Every *malicious peer* always maliciously launches attacks and deviates arbitrarily from the prescribed timed-release service in an attempt to violate security.

To concretely capture such a mixed adversarial environment, we assume that there always exists a malicious adversary M holding an *EOA* as well as a global view of our protocol to aggressively break normal operations of our timed-release service. Such an adversary may adopt two potential approaches to corrupt heterogeneous peers. One is bribery [13], where the rational peers are the chief victims. The other one is malicious peer injection, where M intentionally creates a set of peer accounts acting as the malicious peers and controls them to register themselves with the timed-release smart contract, $\mathcal{SC}$, at any time. Once such malicious peers are selected as participants for a timed-release service, potential attacks will occur. We formally define the attacks next.

2.2.2 Post-facto Attacks

We consider two concrete *post-facto attacks* in our framework. One is *drop attack*,

[1] In this paper, we use the terms peer and peer account interchangeably. We also highlight that a peer may represent an individual holding *Ethereum* account and not a miner node.



(a) Drop Attack (b) Release-ahead Attack

Fig. 1: *Post-facto* Attack Examples

which aims at destroying the data before the prescribed release time and results in a failing data release at the prescribed release time. Such an attack may be launched by M who controls one or more injected malicious peers engaging in the timed-release service. For example, in Case 1 of Fig. 1a, M controls the injected malicious peer P_4 to drop the data D[2], and after T_r, D is missing. In addition, M also can adopt bribery to corrupt rational peers. As an example, in Case 2, in another service, M could let the rational peer P_2 drop D by bribing P_2 through off-chain interactions to make P_2 earn more profit. As a consequence, after T_r, nothing will be released.

The other form of attack is the *release-ahead attack*. A successful release-ahead attack results in a premature release of the data D. It can be launched by M by corrupting a fraction of peers engaging in the timed-release service to get the data before the prescribed release time and disclose it. For example, in Fig. 1b Case 1, M may control P_4 and bribe P_3 to successfully launch a release-ahead attack. Specifically, we note that in our protocol, the data D will be encrypted using the public keys of P_1, P_4, and P_3. If M wants to pre-release D, he/she must control P_4 to acquire the encrypted data and bribe P_3 to get the private key of P_3 to decrypt the encrypted data and pre-release at that time point which is earlier than T_r. Additionally, by adopting bribery, in Case 2, M must bribe P_2 and P_3 to successfully launch the *release-ahead attack*.

2.3 Limitations of Existing Solutions

We present an example to illustrate the fully rational environment [13], described in Case 3 in Fig. 1a and Fig. 1b, in which the global-view adversary M also acts rationally. The incentive-only solution [13] can regulate each rational peer's behavior based on the existence of *Nash Equilibrium* [17], and D will be normally released at T_r as expected. However, by comparing Case 3 with Case 2 as well as Case 1 in Fig. 1a and Fig. 1b, it is easy to find that if we still apply the incentive-only solution in Case 1 and Case 2, the protection of D will inevitably suffer from a striking degradation even if there are only a few malicious peers. This

[2] We generalize D as any data transmitted over the *Ethereum* account network. In this paper, specifically, D refers to the secret key generated by the sender

limitation motivates us to design an attack-resilient protocol to protect the data during a timed-release service in our practical mixed adversarial environment.

3 Reputation-aware Timed-release Protocol

In this section, we demystify the detail of our reputation-aware timed-release service protocol. We first build an uncertainty-aware reputation mechanism to evaluate the behavior of each peer. Then, incorporated with such a mechanism, four tightly coupled components, namely ***Peer Registration***, ***Service Setup***, ***Service Enforcement*** , and ***Service Summary*** are elaborately described with examples.

3.1 Uncertainty-aware Reputation Measure

In our protocol, after each service, $\mathcal{SC}$ judges peers' on-chain behaviors with two distinct types. One is honestly following the protocol, denoted by FL, and the other is dishonestly deviating the protocol, denoted by DV. Such evaluations of behaviors are governed by our *Service Enforcement* and *Service Summary* protocols, which we will describe in detail in Section 3.2.3 and Section 3.2.4 respectively. Since there are multiple request demands of timed-release services, each registered peer may be recruited by different senders to engage in more than one service. Inspired by the binary assessments of behaviors as well as the engagement within multiple requests, we borrow the ideas from *Beta* distribution [7] to measure the reputation of each engaged peer from an uncertainty perspective. Next, we detail the establishment of our reputation measure. We start with the sketch of the *Beta* distribution, which is a two-parameter family of functions represented by α and β, defined as $f(Pr|\alpha,\beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} Pr^{\alpha-1}(1-Pr)^{\beta-1}$, where $0 \leq Pr \leq 1, \alpha > 0, \beta > 0$ with the constraint that the probability variable $Pr \neq 0$ if $\alpha < 1$, and $Pr \neq 1$ if $\beta < 1$. Given α and β, the expectation of Pr is described as $E[Pr] = \frac{\alpha}{\alpha+\beta}$ .Our reputation measure can be expressed as follows: For each peer P_j who participates in a timed-release service, an on-chain post-service evaluation for P_j within the i-th timed-release service $e(j,i)$ falls in $\mathcal{S} = \{FL, DV\}$. If P_j honestly follows the i-th timed-release protocol, $e(j,i) = FL$, otherwise, $e(j,i) = DV$. For counting the evaluations and recording in $\mathcal{SC}$, we denote $I(j,i,FL) = 1$ if $e(j,i) = FL$. Otherwise, $I(j,i,DV) = 1$, where $I(j,i,FL) + I(j,i,DV) = 1$ for each $req_i \in Req$. Then for each peer, we define $\alpha(j,i) = I(j,i,FL) + 1$, and $\beta(j,i) = I(j,i,DV) + 1$. Since P_j may engage in multiple timed-release services, we have $\alpha(j) = \sum_{i=1}^{m} I(j,i,FL) + 1$, and $\beta(j) = \sum_{i=1}^{m} I(j,i,DV) + 1$, where $\alpha(j)$ captures the total observations of honest evaluations of P_j among m requests, and $\beta(j)$ captures the total observations of deviation evaluations for P_j among m requests. Then, based on the above definitions, we have $E[Pr(j)]$ for P_j,where $E[Pr(j)] = \frac{\alpha(j)}{\alpha(j)+\beta(j)}$ denotes the likelihood that P_j honestly follows the protocol when participating in the future service, given the observations of honest evaluations $\alpha(j)$ and the observations of deviating evaluations $\beta(j)$. Based on $E[Pr(j)]$, we define the reputation measure of P_j as follows:

$$Rep(j) = \frac{\alpha(j)}{\alpha(j) + 1 + \xi \cdot [\beta(j) - 1]} \tag{1}$$

where $\xi \geq 1$ is a predetermined penalty factor which is designed to penalize any dishonest behaviors of P_j. The design objectives of this measure are two-fold: first, given that the historical evaluations, $Rep(j)$ reflect the likelihood that P_j honestly follows future engaged services, it measures the reputation with uncertainty. Second, adjusted by ξ, for P_j, $Rep(j)$ is gradually built up when P_j honestly contributes in the protocol and it is rapidly reduced when P_j dishonestly deviates the protocol. For instance, if P_j has been engaged in 12 timed-release services, wherein he/she honestly follows the protocol 8 times and dishonestly deviates the protocol 4 times, the reputation is measured by $Rep(j) = \frac{(8+1)}{(8+1)+1+3*4} = 0.41$, which indicates that for the incoming service engagement, P_j holds 0.41 likelihood to follow such service. In particular, for a peer who does not hold any past observations, the reputation score is initialized with $\frac{1}{2}$.

3.2 Reputation-aware Timed-release Protocol

We now describe the detailed design of our reputation-aware timed-release protocol.

3.2.1 Peer Registration

At any time, each voluntary peer $P_j \in \mathcal{P}$ can register himself with $\mathcal{SC}$. The detailed design is described as follows: 1. Each voluntary peer $P_j \in \mathcal{P}$ can submit his/her personal information $I_j := \{pk_j, w_j, d_j\}$ to the smart contract $\mathcal{SC}$. 2. After receiving the registration request from P_j, $\mathcal{SC}$ performs the following operations: $\mathcal{SC}$ first updates w_j with $\mathcal{SC}.WITree$ (an *Interval Tree* [2] structure to store working windows). Then, $\mathcal{SC}$ initializes an account, namely, $Peer(j)$ for P_j to store the on-chain profile, which consists of the content as follows: the account balance of P_j, the public key pk_j of P_j, the history of service evaluations of P_j, and the initial reputation score $Rep(j) = 0.5$. In parallel, $\mathcal{SC}$ constructs IP, a map that indexes $Peer(j)$ with the key w_j. After the transactions corresponding to the registration procedure are confirmed, the information regarding P_j is stored on the public on-chain, which can be observed by any peer within the network.

3.2.2 Service Setup

Before initializing a new timed-release service, the sender S_i and recipient R_i first negotiate through off-chain connections to achieve an agreement for a new timed-release service. Such an agreement consists of the following configurations of the incoming timed release service: a prescribed release time T_r^i and minimum deposit needed. After negotiating, the responsible sender S_i makes provision for the incoming service with $\mathcal{SC}$ by means of the *Service Setup* protocol. The logic behind the *Service Setup* in our protocol is similar to one presented in [13]. The novel feature that differentiate our protocol from the one in [13] is the newly designed reputation-aware peer recruitment policy which we describe next.

(i)Reputation-aware Peer Recruitment: For a specific timed-release service request req_i, the sender S_i first interacts with $\mathcal{SC}$ to observe the available peer windows. Then, S_i will make a decision to select a number of peers such that

the set union of the working windows of such peers must cover $\left[T_s^i, T_r^i\right]$. The set of selected peers forms a routing-like path to store the data and perform the data hand-off between each peer. In parallel, the sender S_i is also concerned with the performance of data protection provided by such a set of peers. Such a protection can be specifically represented as follows: given a set of selected peers, what is the likelihood that the data is prevented from *drop attack* as well as *release-ahead attack*?

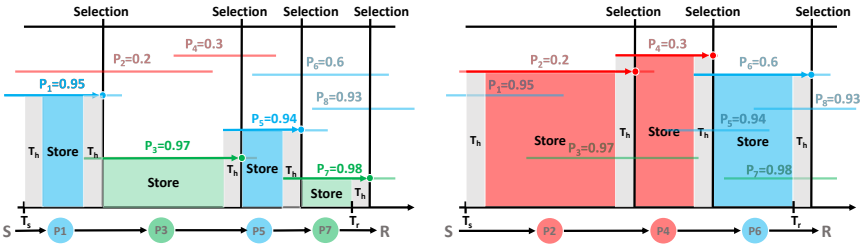
We formally quantify such likelihood with two distinct attack resilience metrics, namely *drop attack resilience* and *release-ahead attack resilience* respectively. With the help of our uncertainty-aware reputation measure $Rep(\cdot)$, given a peer recruitment scheme E that consists of l peers, we define such metrics as follows: The *drop attack resilience* of E, denoted as F_d^E, captures the failure likelihood of the adversary M when he/she intends to launch a drop attack, quantified as $F_d^E = \prod_{k=1}^{l} [Rep(k)]$, which reflects that M must control or bribe at least one peer in E to successfully launch a drop attack. The *release-ahead attack resilience* of E, denoted as F_r^E, captures the failure likelihood of M when he/she intends to launch release-ahead attacks. It is quantified as $F_r^E = 1 - \prod_{k=1}^{l} [1 - Rep(k)]$. Since the scheme E is protected with the *Onion Routing* [20] scheme, an adversary must control all peers to prematurely release the data at the start time. We discuss this later in the *Service Enforcement* protocol.

Our peer recruitment for the timed-release service consists of multiple rounds. In each round of selection, the sender will take a time point as the input. By retrieving all the available registered peers whose working window covers the input time point in terms of $WITree$, the sender will pick the one having the highest reputation score as the responsible peer for the current round. Then, the start time of the selected peer and the hand-off time zone will be taken as the input of the next round selection. In particular, two conditions will end the recruitment. When the $WITree$ finds that no available peers are in a selection round or when the start time of the selected peer is earlier than T_s.

We illustrate our design approach using an example in Fig.2a. In the example, there are totally 7 peers P_1-P_7 who have already registered with $\mathcal{SC}$ as well as available within $[T_s, T_r]$. The numerical value associated with each peer represents the current reputation measure, which is publicly known. The different numerical measures associated with the peers indicate their historical behaviors when they engaged in timed-release requests in the past. The blue color represents rational peers, the red color represents the malicious peers, and the green color represents the honest peers. T_h denotes a predefined hand-off timezone. The sender S first takes T_r+T_h as the input of the first-round selection. There are totally 3 peers, P_6, P_7, and P_8, who are available at this time point. Then, S picks the highest reputation one, P_7, whose reputation is $Rep(P_7)$=0.98, as one of the peer candidates. Then, S performs the second-round selection, which takes $T_s(P_7)$+T_h, where $T_s(P_7)$ represents the start time of the working window of P_7, as the input time point. The working windows of two peers, P_5 and P_6, cover such a time point. The one holding a higher reputation P_5, $Rep(P_5)$=0.94, is selected. Followed by the third-round selection, P_3 is selected. Finally, the last-

round selection picks P_1 as well as checks that the start time of P_1 is prior to the start time of the timed-release service, T_s, which ends the reputation-aware recruitment procedure. The order of peers, $P_1 \rightarrow P_3 \rightarrow P_5 \rightarrow P_7$, jointly take charge of the incoming timed-release service. We denote this scheme as E^1.

(ii) Resilience Analysis: We provide a concrete example to show the resilience analysis of our proposed reputation-aware peer recruitment policy as well as the time-aware recruitment proposed in [13] as the baseline for comparison. To analyze its resilience, we independently consider the drop attack and the release-ahead attack. The release-ahead attack resilience of E^1 from our proposed reputation-aware peer recruitment is derived from $F_r^{E^1} = 1 - (1-0.98)\cdot(1-0.94)\cdot(1-0.97)\cdot(1-0.95) = 0.999$, and the drop attack resilience is calculated as $F_d^{E^1} = 0.98 \cdot 0.94 \cdot 0.97 \cdot 0.95 = 0.85$. For the reputation-



(a) Reputation-aware Recruitment (b) Time-aware Recruitment

Fig. 2: Peer Recruitment Example

unaware recruitment in Fig. 2b, in each round of selection, only the peer who has the longest working time is selected. The peers $P_2 \rightarrow P_4 \rightarrow P_6$ take charge of the service. We denote this scheme as E^2. The release-ahead attack resilience is $F_r^{E^2} = 1-(1-0.2)\cdot(1-0.3)\cdot(1-0.6) = 0.77$, and the drop attack resilience is $F_d^{E^2} = 0.2 \cdot 0.3 \cdot 0.6 = 0.036$. Compared with our reputation-aware recruitment, both $F_r^{E^2}$ and $F_d^{E^2}$ show significant degradation. After providing all information mentioned above, S_i and R_i deposit d and register the information with $\mathcal{SC}$.

3.2.3 Service Enforcement

After completing *Service Setup*, a timed-release service moves into execution. *Service Enforcement* takes charge of monitoring the correctness and timeliness of the executions after the data is released into the blockchain network. Moreover, with the help of the *Service Enforcement* protocol, $\mathcal{SC}$ can abort the service in time if any misbehavior is detected. We adopt the basic procedures of the *Service Enforcement* protocol described in [13]. We describe it next.

We denote $E^i = \{P_k\}_{k\in[1,l]}$ as the selected peers from the reputation-aware recruitment policy from S_i, and S_i generates an *onion* by encrypting the original data with the public keys of the peers in E^i and transfer it to P_1 to initialize the *Service Enforcement* protocol.Without loss of generality, under the

scope of the timed-release service, we capture the action space of each peer with $\mathcal{A} = \{cs, ws, tr, vr, rp\}$, where cs represents that P_k verifies the received certification from the P_{k-1}. The certification is used to detect drop attacks. ws indicates that P_k builds private channel using the *Whisper Key* protocol [13] with the subsequent peer P_{k+1} to transfer the *onion*, and tr represents that P_k transfers the onion to P_{k+1}, vr aims at verifying the on-time results of cs and ws, and rp captures the possible misbehavior actions. In order to detect misbehavior in time, we bound the major actions within $\mathcal{A}$ with corresponding deadlines, which enables our protocol to be aborted timely if any misbehavior is detected. Concretely, we associate each peer with a series of pre-determined hard deadlines. For example, for the peer $P_k \in E^i$, before T_1, P_k must perform cs. Then, P_k must perform vr for the verification of cs, and perform ws before T_2. The verification of ws, and the delivery of the *onion* to the subsequent peer P_{k+1} must be executed before T_3. Such deadlines follow the ordering: $T_1 < T_2 < T_3$. In particular, if P_{k+1} does not receive the corresponding *onion* from P_k, he/she must report the issue (rp) before performing cs, otherwise, if a failed submission of the certification is detected from P_{k+1}, P_{k+1} will be judged as launching the *drop attack*. Our *Misbehavior Detection* protocol is similar to the one described in [13]. Corresponding to the results from the verification mentioned above, we denote the finite state space of each $P_k \in E^i$ as $\mathcal{B}_k^i = \{INIT, VF, WS, TR, FAIL\}$. It is described as follows: all peers within E^i start with the $INIT$ state when engaged in req_i. For P_k, if he/she passes the verification for cs, then the state of P_k will be updated to VF from $INIT$. Then, if P_k passes the verification for ws, the state will be updated to WS from VF. If P_{k+1} does not report any misbehavior of P_k, the state of P_k will be updated to TR from VF. If any previous verification fails or a drop report is emitted, the state of P_k will be moved to $FAIL$ and the current service will be aborted. If all peers in E^i honestly follow the protocol, R_i will get the original data and close the current service by following *Service Summary* which is described next.

3.2.4 Service Summary

There are two scenarios that trigger *Service Summary*:(1) A timed-release service is successfully finished. Under this scenario, R_i is responsible for the final evaluations of each peer as well as updating the on-chain reputation. (2) A timed-release service is aborted at a time point before the prescribed release time T_r^i. We adopt a remuneration policy similar to one described in [13] for the incentive refund. For the behavior evaluation as well as for updating the reputation, $\mathcal{SC}$ follows the rules to perform evaluations: $\mathcal{SC}$ checks the state of each peer within E^i. If the final state of a peer is TR, the binary assessment for the peer is FL, and if the final state of a peer is $FAIL$, the assessment is DV. Otherwise, if a peer has $INIT$ state, $\mathcal{SC}$ does not perform any evaluations. Finally, the reputation measure of each peer in E^i will be updated using by following equation 1.

4 Evaluations

4.1 Simulation Evaluations

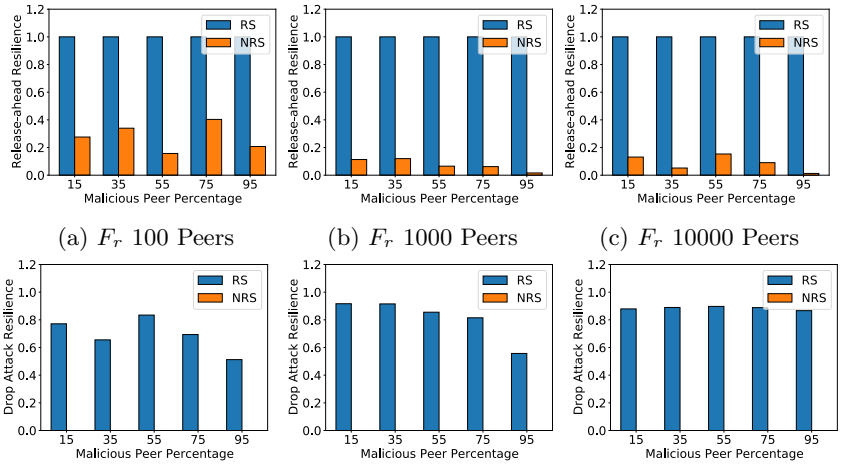
We first perform extensive simulation experiments to show the benefits and effectiveness of our reputation measure as well as the reputation-aware peer recruitment policy. We evaluate our protocol, namely $\underline{RS}$, on two metrics, F_r and F_d. Specifically, we expect the protocol to demonstrate high F_r as well as F_d. For comparisons, we also implement the timed-aware peer recruitment proposed in [13], namely $\underline{NRS}$, as a baseline, which does not consider peer reputation when performing peer recruitment. We implement a simulator in JAVA, and the generated synthetic data as well as default experimental settings in our evaluations are described below:

Synthetic Peer Profile: The default number of registered peers in the simulation is set to 1000. In the simulator, we first randomly select 5 registered peers as the honest peers. Then, given a malicious peer percentage, we randomly select malicious peers based on that percentage. The remaining peers are all rational peers. For each peer, the duration of his/her working window is drawn from a *normal* distribution with the mean of 50 hours and 5-hour standard deviation.

Protocol Configurations: Following the assumptions in Section 2.2, in our simulated protocol, the malicious peers in the registered pool 100% deviate from the protocol, and the honest peers 100% follow the protocol. The rational peers only launch attack with respective of the bribery value from the malicious adversaries. The penalty factor $\xi = 10$.

Experimental Evaluation Settings: We evaluate the resilience by varying the percentage of malicious peers from 15% to 95%. In each percentage setting, the simulator first generates a new peer account network (registered peers) based on the descriptions mentioned above. Then 500 requests are executed for the evaluation, where 70% of them are training requests, and 30% of them are validation requests. We calculate the average resilience as the result of each peer malicious percentage setting. Each experiment is executed 10 times and the average resilience is obtained.

Effectiveness under Different Scales of Registered Peer: The first set of experiments demonstrates effectiveness of our proposed reputation-aware peer selection protocol under three different scales of peer accounts network, namely 100 peers, 1000 peers and 10000 peers. We set the number of honest peers as 5 in each setting. The time duration of each service is set to 300 hours, and the handoff time zone is set to 4 hours. When there are 100 peers, compared with NRS, under each setting of malicious peer percentage, RS shows significantly higher release-ahead attack resilience (Fig. 3a) and drop attack resilience (Fig. 3d). We increase the size of the registered list towards 1000 peers in Fig. 3b. Under each setting of malicious peer percentage, our proposed RS achieves a higher release-ahead attack resilience than NRS. Also in Fig. 3e, compared with NRS, our proposed RS achieves a significantly higher drop attack resilience. Finally, under a relatively large scale peer network with 10000 peers in Fig. 3c and Fig. 3f, our proposed RS scheme achieves a higher release-ahead attack resilience and drop



(a) F_r 100 Peers (b) F_r 1000 Peers (c) F_r 10000 Peers

(d) F_d 100 Peers (e) F_d 1000 Peers (f) F_d 10000 Peers

Fig. 3: Different Scales of Registered Peers

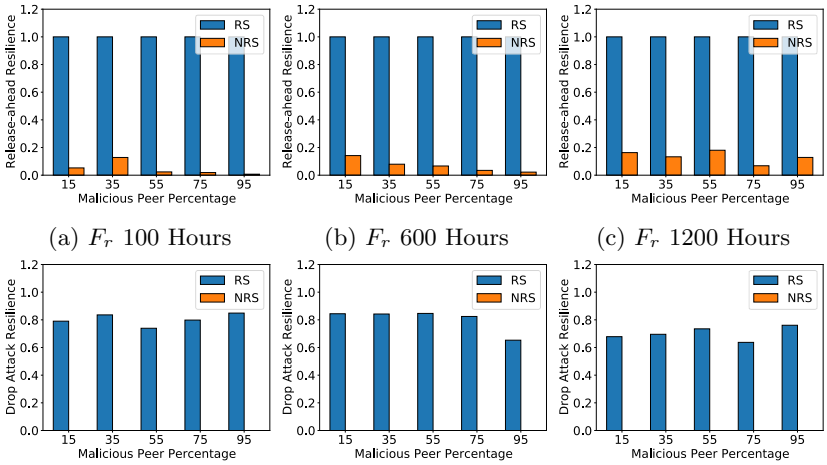
attack resilience. In summary,under different scale of peer network, the proposed approach *RS* can effectively protect the data with higher attack resilience.

Effectiveness under Different Duration of Service: In this set of experiments, we learn the effectiveness of our proposed protocol under different duration of service time. Apart from the default 300 hours service time, in this set of experiments, we set a short duration, 100 hours, a moderate length duration 600 hours, and a longer duration 1200 hours. We fix the number of registered peers as 1000, and the hand-off time zone is set to 4 hours. In Fig. 4a and Fig. 4d, we observe that compared with NRS, RS significantly improves the release-ahead attack resilience and drop attack resilience under different percentage of malicious peers. Then, from the results of 600-hour service time, compared with NRS, RS achieves a significantly higher release-ahead attack resilience and drop attack resilience under different malicious peer percentage. Finally, the results corresponding to the 1200-hour setting also shows similar benefits for RS. In summary, RS can achieve significantly better resistance for both the release-ahead attacks as well as the drop attacks under different duration of service.

4.2 Prototype Implementation

4.2.1 Implementation Description

In this section, we present our prototype implementation. We implement our smart contract in the official programming language *Solidity* [4]. We deploy our smart contract on Ethereum official test network *Rinkeby* [3] through the interface, *Infura* [1]. We use the data from our simulation studies as the input for on-chain validations. We have 7 selected peers in total taking charge of 1200 hour-service. The results from Table I show the detailed information of our im-



(a) F_r 100 Hours (b) F_r 600 Hours (c) F_r 1200 Hours

(d) F_d 100 Hours (e) F_d 600 Hours (f) F_d 1200 Hours

Fig. 4: Different Duration of Timed-release Service

plementations. The implementation of our reputation-aware timed-release protocol consist of five different modules. ①**Registration:** this module provides interfaces for every peer to register with $\mathcal{SC}$ as well as to manage their own information. *peerRegister* is invoked by any peers to register with $\mathcal{SC}$. *deposit* and *withdraw*, are invoked by the registered peers to manage the account balance. *updatePubKey* and *updateWindow* are invoked by the registered peers to manage the information regarding the timed-release service. ②**Setup:** this module provides interfaces for senders and recipients to perform service provisioning for timed-release requests. *senderSign* is invoked by senders to register with $\mathcal{SC}$. The second function *recipientSign* is invoked by the corresponding recipients. The third function *setup* is invoked by senders to finish the provisioning of a service. ③**Enforce:** this module guarantees the normal executions of a timed-release service. *setCertificate* is invoked by senders to submit the certification. *submitCertificate* is invoked by any engaged peers or the recipients to verify the correctness of the certification. *setWhisperKey* is invoked by the engaged peers or the senders to build private channels. The fourth function *verification* is invoked by any peers to actively verify the behaviors of each peer. ④**Detection** This module consists of four functions to detect and reward the reporting of behaviors. *releaseReport* is invoked by any peer to report the evidence of the release-ahead attack, and after verification by $\mathcal{SC}$, *releaseReward* is invoked by the reporter to receive rewards. *dropReport* is invoked by any peer to report the drop attack evidences. After the verification, the reporter invokes the function *dropReward* to get rewards. ⑤**Summary** The last module consists of two functions. The first one, *peerEvaluation*, is invoked by the last responsible peer of each service to assess peers' behaviors, and the second function *reputationUp-*

date is invoked by the same peer to finally update the peers' reputation measure, which is publicly recorded on-chain.

4.2.2 Cost Analysis

From Table I, we have two significant observations regarding the cost: first, there are four functions incurring relatively higher cost namely *peerRegister(229669)*, *senderSign(355534)*, *setup(122682)*, and *peerEvaluation(139363)*. Specifically, *peerRegister* is invoked by any peers, *senderSign* and *setup* are both invoked by a sender. *peerRegister* is invoked by the last peer in the routing scheme. We note that those four functions are only invoked once per request and therefore it is an one-time cost. Second, our proposed on-chain reputation mechanisms only incurs a modest gas cost from the summary module where *peerEvaluation* incurs 139363 and *reputationUpdate* incurs 48620.

Module	**Function**	**Gas Cost**
Registration	*peerRegister*	229669
	deposit	42349
	withdraw	29283
	updatePubKey	43556
	updateWindow	83884
Setup	*senderSign*	355534
	recipientSign	62292
	setup	122682
Enforce	*setCertificate*	70539
	submitCertificate	44460
	setWhisperKey	64585
	verification	26036
Detection	*releaseReport*	69541
	releaseReward	29596
	dropReport	65362
	dropReward	29662
Summary	*peerEvaluation*	139363
	reputationUpdate	48620

Table 1: Gas Cost

5 Related Work

Practically constructing decentralized timed-release services has gained attention recently. *Self-emerging infrastructure* [19] provides a promising solution. Two representative constructions were developed using this design philosophy. Based on *Distributed Hash Table (DHT)*, the first set of solutions, [10, 11], design a set of mechanisms within DHT to statistically resist potential attacks launched by adversaries. The second type of solution [13, 14] suggests the use of blockchain technologies to support timed release of data by using the Ethereum network. Besides timed release of data, there are several other practical contributions. The decentralized protocol presented in [12] schedules a transaction that is

executed at a future point of time while protecting the information regarding the transaction before the release time. Ning et.al [18] proposed a carrot-and-stick mechanism using smart contracts to tackle premature release issues when developing timed-release applications. While such practical constructions advance the development of timed-release applications in real world, aggressive adversarial impacts on such paradigm inevitably exist and raise major concerns.

A preliminary discussion about the importance of protecting timed-release services in mixed adversarial environments is recently introduced by the authors in [23]. In this research paper, we focus on developing a comprehensive solution to the problem by designing a reputation-aware scheme for protecting against attackers in mixed adversarial settings consisting of both malicious and rational peers. The reputation-based scheme is designed to detect dishonest behaviors and reward and incentivize honest behavior in the context of supporting timed release of data. The transparent nature of blockchains provides a convenient context to build reputation measure. There are several representative blockchain-based reputation mechanism in the literature. Based on social norm, Li,et.al [15] builds a reputation model on *Ethereum*-based crowdsourcing framework to regulate the worker's behaviors. Zhou, et al [25] leverages a simple reputation measure to block the witnesses with misbehaviors in a blockchain-based witness model. RTChain [22] integrates a reputation system into the blockchains that focus on e-commerce to achieve a transaction incentives and distributed consensus. RepuCoin [24] designs *proof-of-reputation* in a permissionless blockchain to achieve a strong deterministic consensus, which is robust to attacks. Unlike the reputation scheme developed in this work, these existing schemes primarily focus on blocking the participants with misbehaviors and are not designed to directly provide quantitative analysis to aid peer selection in mixed adversarial settings.

6 Conclusion

In this paper, we develop techniques to protect blockchain-based timed data release in mixed adversarial environments where both malicious adversaries and rational adversaries exist. An uncertainty-aware reputation measure is developed to quantitatively capture the behavior of peers engaging in timed release service. Based on the reputation measure, a novel reputation-aware timed-release protocol is designed to handle such mixed adversarial settings. First, an off-chain reputation-aware peer recruitment is performed by carefully considering the impact on attack resilience. Then, a suite of on-chain mechanisms jointly take charge of monitoring the states of the proposed protocol and evaluate the behavior of each engaged peer. Our extensive simulations demonstrate the effectiveness of our proposed protocol. Compared with the existing solutions, the approach achieves significantly higher attack resilience. We develop a prototype of the proposed protocol using smart contracts and we deploy it on the *Ethereum* official test net, *Rinkeby*. The on-chain evaluations show that our protocol incurs only a moderate amount of gas cost and demonstrates its cost-effectiveness.

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. Infura. https://infura.io/, [Online].
2. Interval tree. https://github.com/gnidan/interval-trees-solidity, [Online].
3. Rinkeby. https://www.rinkeby.io/stats, [Online].
4. Solidity. https://docs.soliditylang.org/en/v0.8.10/, [Online].
5. Nir Bitansky, Shafi Goldwasser, Abhishek Jain, Omer Paneth, Vinod Vaikuntanathan, and Brent Waters. Time-lock puzzles from randomized encodings. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, pages 345–356, 2016.
6. Dan Boneh and Moni Naor. Timed commitments. In *Annual international cryptology conference*, pages 236–254. Springer, 2000.
7. Audun Josang and Roslan Ismail. The beta reputation system. In *Proceedings of the 15th bled electronic commerce conference*, volume 5, pages 2502–2511, 2002.
8. Kohei Kasamatsu, Takahiro Matsuda, Keita Emura, Nuttapong Attrapadung, Goichiro Hanaoka, and Hideki Imai. Time-specific encryption from forward-secure encryption. In *International Conference on Security and Cryptography for Networks*, pages 184–204. Springer, 2012.
9. Ryo Kikuchi, Atsushi Fujioka, Yoshiaki Okamoto, and Taiichi Saito. Strong security notions for timed-release public-key encryption revisited. In *International Conference on Information Security and Cryptology*, pages 88–108. Springer, 2011.
10. 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.
11. 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.
12. 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.
13. 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.
14. Chao Li and Balaji Palanisamy. Silentdelivery: Practical timed-delivery of private information using smart contracts. *IEEE Transactions on Services Computing, (to appear)*.
15. Ming Li, Jian Weng, Anjia Yang, Wei Lu, Yue Zhang, Lin Hou, Jia-Nan Liu, Yang Xiang, and Robert H Deng. Crowdbc: A blockchain-based decentralized framework for crowdsourcing. *IEEE Transactions on Parallel and Distributed Systems*, 30(6):1251–1266, 2018.
16. Timothy C. May. Timed-release crypto. http://www.hks.net/cpunks/cpunks-0/1460.html, 1993.

17. John F Nash Jr. Equilibrium points in n-person games. *Proceedings of the national academy of sciences*, 36(1):48–49, 1950.
18. Jianting Ning, Hung Dang, Ruomu Hou, and Ee-Chien Chang. Keeping time-release secrets through smart contracts. *IACR Cryptol. ePrint Arch.*, page 1166, 2018.
19. Balaji Palanisamy and Chao Li. Self-emerging data infrastructures. In *2019 IEEE 5th International Conference on Collaboration and Internet Computing (CIC)*, pages 256–265, 2019.
20. M.G. Reed, P.F. Syverson, and D.M. Goldschlag. Anonymous connections and onion routing. *IEEE Journal on Selected Areas in Communications*, 16(4):482–494, 1998.
21. Ronald L Rivest, Adi Shamir, and David A Wagner. Time-lock puzzles and timed-release crypto.
22. You Sun, Rui Xue, Rui Zhang, Qianqian Su, and Sheng Gao. Rtchain: A reputation system with transaction and consensus incentives for e-commerce blockchain. *ACM Transactions on Internet Technology (TOIT)*, 21(1):1–24, 2020.
23. Jingzhe Wang and Balaji Palanisamy. Protecting blockchain-based decentralized timed release of data from malicious adversaries. In *2022 IEEE International Conference on Blockchain and Cryptocurrency*, 2022.
24. Jiangshan Yu, David Kozhaya, Jeremie Decouchant, and Paulo Esteves-Verissimo. Repucoin: Your reputation is your power. *IEEE Transactions on Computers*, 68(8):1225–1237, 2019.
25. Huan Zhou, Xue Ouyang, Zhijie Ren, Jinshu Su, Cees de Laat, and Zhiming Zhao. A blockchain based witness model for trustworthy cloud service level agreement enforcement. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 1567–1575. IEEE, 2019.