Journal of Computer Security 31 (2023) 649–677
DOI 10.3233/JCS-230001
IOS Press

Securing blockchain-based timed data release against adversarial attacks[1]

Jingzhe Wang * and Balaji Palanisamy
School of Computing and Information, University of Pittsburgh, Pittsburgh, PA, USA
E-mails: jiw148@pitt.edu, 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 an 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 a systematic solution for attack-resilient and practical blockchain-based timed data release in a mixed adversarial environment, where both malicious adversaries and rational adversaries exist. We first propose an effective uncertainty-aware reputation measure to capture the behaviors of the peer involved in timed data release activities in the network. In light of such a measure, we present the design of a basic protocol that consists of two critical ingredients, namely *reputation-aware peer recruitment* and *verifiable enforcement protocols*. The former, prior to the start of the enforcement protocols, performs peer recruitment based on the reputation measure to make the design probabilistically attack-resilient to the *post-facto attacks*. The latter is responsible for contractually guarding the recruited peers at runtime by transparently reporting observed adversarial behaviors. However, the basic recruitment design is only aware of the reputation of the peers and it does not consider the working time schedule of the participating peers and as a result, it results in lower attack-resilience. To enhance the attack resilience further without impacting the verifiable enforcement protocols, we propose a temporal graph-based reputation-aware peer recruitment algorithm that carefully determines the peer recruitment plan to make the service more attack-resilient. In our proposed approach, we formally capture the timed data release service as a temporal graph and we develop a novel maximal attack-resilient path-finding algorithm on the temporal graph for the participating peers.

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, we perform simulation experiments to validate the effectiveness of the reputation-aware timed data release protocols as well as our proposed temporal-graph-based improvements. The results demonstrate the effectiveness and strong attack resilience of the proposed mechanisms and our approach incurs only a modest gas cost.

Keywords: Timed data release, blockchain, smart contract, temporal graph

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 applications using timed data release include secure auction systems where important bidding information needs protection until all bids arrive and secure voting

[1]This paper is an extended and revised version of a paper presented at DBSEC 2022.
*Corresponding author. E-mail: jiw148@pitt.edu.

0926-227X © 2023 – The authors. Published by IOS Press. This is an Open Access article distributed under the terms of the Creative Commons Attribution License (CC BY 4.0).

mechanisms where votes are not permitted to be accessed until the close of the polling process. Concretely, in such applications, timed data release can serve as a service that takes charge of protecting sensitive information (bidding and voting) before the release time and releasing such information to the public after hitting the release time. Since the early research on timed information release [29], 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 [4,5,17,18,34] 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 in decentralized applications that require timed data release. Recently, a category of decentralized data systems, namely self-emerging data infrastructures [3], 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 [23] 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 a large number of participants 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, blockchain-based solutions for timed data release include several inherent risks. In this paper, we particularly focus on two major risks of blockchain-based infrastructures for timed data release: *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 misbehaving participants 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 the 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 [22–24] protect the timed data release from peers with a fully rational context in the Ethereum network. *Second*, cryptography-based solutions [31] are 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 involves 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 basic suite of reputation-aware timed-release protocols consisting of two key ingredients: *First*, a reputation-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 life cycle of the timed-release service. Briefly, such recruitment first retrieves peers who are available at a given time point, followed by picking a qualified peer based on his/her reputation. This procedure is operated recursively until the entire life cycle of the timed data release service is covered. *Second*, a suite of decentralized on-chain protocols, namely *verifiable enforcement protocols*, are proposed to guarantee the normal operations of the timed-release protocol. Concretely, during service runtime, such protocols take responsibility for making any adversarial behaviors publicly verifiable and updating reputation transparently. However, the initial reputation-aware peer recruitment explores available peers in a local search fashion while disregarding the big picture that captures the entire time availability of the peers over the life cycle.

To address this limitation, we adopt the notion of *Temporal Graph* [13] to formally capture the peer service availability in the blockchain-based timed data release design. Specifically, we first construct a temporal graph from public information shown in the smart contract, consisting of each peer's available working windows as well as the peer's reputation. We then show the equivalence of our maximal attack-resilient peer recruitment problem with the *minimum temporal path finding* [42,43] problem. However, directly adopting existing techniques for the optimal solution does not serve the purpose as existing solutions only focus on temporal-related metrics (e.g., latency or duration) rather than the attack-resilience properties that is of great importance in our scenario. To this end, we propose a novel efficient path finding algorithm in temporal graph, namely *maximal attack-resilient path finding*, to tailor to the needs of our timed release service requests that maximizes the attack resilience. Concretely, grounded on our resilience metrics, we demonstrate that our problem shows an optimal substructure property [6]. We then propose an efficient algorithm to offer an optimal recruitment that maximizes drop attack resilience or release-ahead attack resilience.

For extensively evaluating our proposed timed-release protocol, we first perform simulation studies using a synthetic dataset to evaluate the effectiveness of the reputation-aware peer recruitment as well

as the temporal graph-based recruitment technique. To demonstrate the gas-efficiency of our proposed verifiable enforcement protocols, we implement a proof-of-concept prototype using real-world smart contracts developed using the *Solidity* programming language and we 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:

- We carefully design an effective uncertainty-aware reputation measure for supporting the needs of blockchain-based timed-release services.
- We propose a suite of novel reputation-aware timed-release protocols to construct an attack-resilient timed-release scheme.
- In order to maximize attack resilience, we propose an enhanced recruitment scheme using the notion of *Temporal Graph*.
- We perform extensive evaluation of our proposed protocol through simulation studies 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 key preliminaries adopted in this work. In Section 3, 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 4, we formally introduce our proposed reputation model. Then, using the proposed reputation model, the full view of the construction of our reputation-aware timed-release protocol is unfolded in Section 5. In Section 6, we formally introduce our temporal graph-based design. In Section 7, we discuss the results of our simulation studies and present the on-chain gas evaluations using our prototype implementation. In Section 8, we discuss the related work and in Section 9, we conclude the paper.

2. Preliminaries

2.1. The Ethereum blockchain

As a pioneering blockchain that embraces the design of smart contracts [38], *Ethereum* has been gaining popularity. In *Ethereum*, a smart contract consists of a piece of computer code executed and stored on *Ethereum*. Such a salient feature provides an effective mechanism for decentralized application (DApp) developers to design decentralized applications.

There are two types of accounts active in *Ethereum*. One is Externally Owned Account (*EOA*). An *EOA* is controlled by a real-world user who is in possession with a unique public-private key pair as well as a balance of *Ether*, the cryptocurrency associated with *Ethereum*. The other is Contract Account (*CA*). A *CA*, without a private key, takes charge of storing the smart contract code and hold a balance of *Ether*. Information flow between EOAs and CAs on the *Ethereum* blockchain is realized in terms of issuing transactions. Specifically, an EOA can transfer some amount of *Ether* to another by privately signing a transaction with his/her owned private key. Also, a transaction can also allow an EOA to invoke a function coded in a smart contract owned by a *CA*. Each transaction is public to the entire network once it is confirmed.

To support the submission and execution of transactions in a decentralized fashion, at a lower level, *Ethereum* constructs and maintains a *peer-to-peer* network that is composed of a set of *Ethereum* workers (miner nodes). Such nodes jointly confirm transactions by following the *Proof-of-Work (PoW)* consensus protocol supported by *Ethereum*. *Ethereum* incentivizes the flow of transactions by introducing

the *Gas* mechanism [10]. The *Gas* is measured by *Ether*. For instance, to submit a new transaction, an EOA needs to pay some gas for Ethereum workers as reward to execute the transaction.

In this paper, we use the account network consisting of both EOAs and CAs to provide a decentralized environment to implement the timed data release.

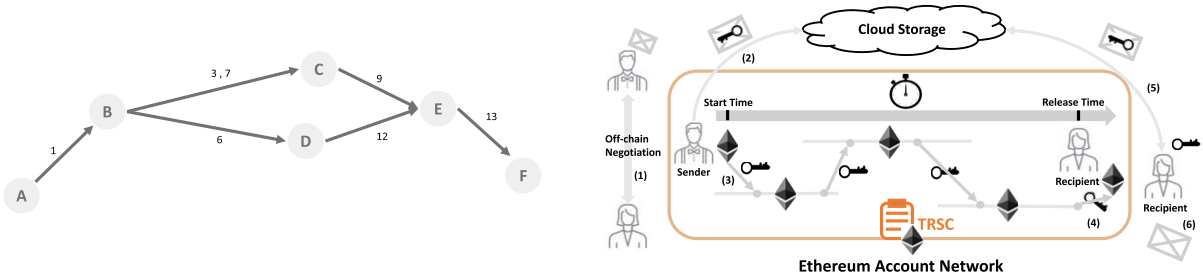
2.2. Cryptographic primitives

In this work, we adopt the following cryptographic primitives:

- *Cryptographic Hash Function*: To support data integrity check, we adopt the cryptographic hash function, *Keccak256* [10], supported by *Ethereum*. For simplicity, in the remaining of this paper, we assume that $hash(\cdot) := Keccak256(\cdot)$.
- *Cryptographic Digital Signature*: We adopt the cryptographic digital signature, *ECDSA*, supported by *Ethereum* to enable public verification. In this paper, we denote *ECDSA* as $Sig(\cdot)$.
- *Whisper Key Protocol*: In contrast to the interactions through issuing transactions that are public, we also allow two EOAs to interact privately. To this end, we follow the Whisper Key Protocol mentioned in [23] to build private channels. Such a scheme supports a symmetrical whisper key share, in which the first EOA encrypts his/her whisper key with the public key of the second one. This design ensures that only the second one can get the whisper key.
- *Onion Routing*: To provide protection to the data, we follow the design in [12] to encrypt the data in multiple layers.

2.3. Temporal graphs

In this section, we introduce the fundamentals of temporal graph [13]. We formally capture our timed data release work by adopting such a graph. In particular, we consider directed temporal graphs. Let $G = (V, E)$ be a temporal graph where V is the set of vertices and E is the set of edges containing time-respecting information. Specifically, an edge $e \in E$ is captured by a tuple (u, v, t, λ, w), where $u, v \in V$. t is the contact starting time between u and v, λ is the contact duration between u and v, and $t + \lambda$ is the contact ending time between u and v. Each edge $e \in E$ is also assigned a weight, namely w. The weight varies upon different application requirements. As an example, in Fig. 1a, we assume that the temporal graph captures contact scenarios in social networks. Each vertex in this example represents



(a) Temporal Graph (b) Framework

Fig. 1. The framework of timed data release.

an user in the social network, and the number attached to each edge represents the contact start time between two users. For instance, User-A contacts with User-B in day 1, and User-B contacts with User-C in day 3 and day 7. If the interaction between User-A and User-B lasts 1 day without being aware of w, we can denote such an edge as $(A, B, 1, 1)$.

In our work, we carefully adopt the path related properties in temporal graphs to investigate our framework. Several key notions are described as follows: In a temporal graph G, we define a temporal path TP as a sequence of vertices $\langle v_1, v_2, \ldots, v_k, v_{k+1}\rangle$. Specifically, we describe $(v_i, v_{i+1}, t_i, \lambda_i, w_i) \in E$ where $1 \leqslant i \leqslant k$ as the i-th temporal edge on TP and $(t_i + \lambda_i) \leqslant t_{i+1}$, where $1 \leqslant i \leqslant k$. As an example, in Fig. 1a, there exists a temporal path, namely P, where $P = \{(A, B, 1, 1), (B, C, 3, 1), (C, E, 9, 1), (E, F, 13, 1)\}$.

3. Background & motivation

In this section, we first introduce blockchain-based self-emerging data infrastructures. We then present the adversary models and discuss the limitations of existing solutions.

3.1. 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 Storage*, and *Blockchain Infrastructure* respectively. Without loss of generality, we denote multiple timed-release service requests as $\mathcal{RQ} = \{req_1, \ldots, req_m\}$, where $m \in \mathbb{Z}^+$. A pair of data sender and data recipient as well as a group of peers over the Ethereum network participate in each request. We show a sketch view of the framework in Fig. 1b. We describe the four components as follows:

Data Sender: A new timed-release service is initialized by a data sender S_i. S_i first privately negotiates with the recipient R_i through a private channel to confirm the start time T_s^i and the release time T_r^i of the data that needs a timed release service (step-1).Then, S_i encrypts the data with a secret key, and uploads the encrypted data to a trusted cloud storage platform (step 2). At the start time T_s^i of the service req_i, S_i sends the encrypted secret key to the Ethereum account network (step 3) and the key will be released at a prescribed release time T_r^i. By taking into consideration multiple service requests, we denote $\mathcal{S} = \{S_1, \ldots, S_m\}$ 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 data. We assume that S_i and R_i can perform negotiation before a new req_i through off-chain interactions. We denote $\mathcal{R} = \{R_1, \ldots, R_m\}$ as the data recipient set for different timed-release service requests.

Cloud Storage: 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 account network as the core infrastructure to decentralize the protection of the secret key. Specifically, a carefully designed smart contract for the service requirement, denoted as $\mathcal{SC}$, is deployed to the account network. The data sender S_i will select a set of peers who have registered with the smart contract to enforce them to jointly protect the secret key. Without loss of generality, we denote $\mathcal{P} = \{P_1, \ldots, P_j\}$ as the set of registered peers in $\mathcal{SC}$ who expect to participate to multiple timed-release services.

3.2. Adversary model

3.2.1. Mixed adversarial environment

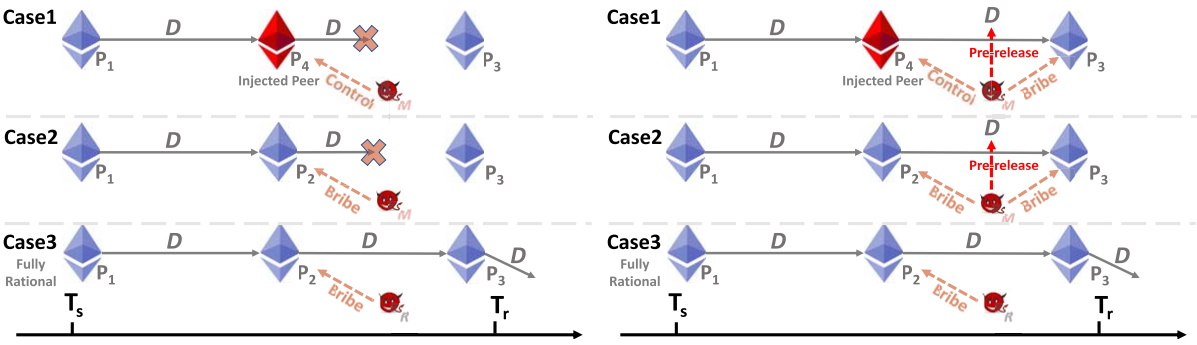
We consider three different types of peer accounts,[2] namely *honest peers*, *rational peers*, and *malicious peers*, existing in the *Ethereum* account network. Specifically, every *honest peer* always participates in timed-release service protocols with absolutely honest actions. This type of peer 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 allows to 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 [23], 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. In the above cases, successful bribery or injection happens with monetary cost that is charged from M. Henceforth, we assume that M is upper-bounded by finite budget in terms of cryptocurrency. We formally define the attacks next.

3.2.2. Post-facto attacks

We consider two concrete *post-facto attacks* in our framework. One is *drop attack*, 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. 2a, M controls the injected malicious peer P_4 to drop the data D,[3] and after T_r, D will be missing. In addition, M also can adopt bribery



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

Fig. 2. *Post-facto* attack examples.

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

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

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. 2b 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*.

3.3. Limitations of existing solutions

We present an example to illustrate the fully rational environment [23], described in Case 3 in Fig. 2a and Fig. 2b, in which the global-view adversary M also acts rationally. The incentive-only solution [23] can regulate each rational peer's behavior based on the existence of *Nash Equilibrium* [30], 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. 2a and Fig. 2b, 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 limitation motivates us to design an attack-resilient protocol to protect the data during a timed-release service in our practical mixed adversarial environment.

4. Uncertainty-aware reputation measure in timed data release

In this section, we introduce our proposed uncertainty-aware reputation measure. In our framework, $\mathcal{SC}$ judges peers' behaviors that are formally bounded by a set $S = \{FL, DV, NS\}$. Specifically, if a peer is involved in a timed release service, he/she will get the following two distinct post-service evaluations: One is honestly following the protocol, denoted by FL, and the other is dishonestly deviating the protocol, denoted by DV. If one is not engaged in a service, we treat his/her behavior as NS. Such evaluations are governed by our *Service Enforcement* and *Service Summary* protocols, which will be described in detail in Section 5.1.3 and Section 5.1.4 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 [16] to measure the reputation of each engaged peer from an uncertainty perspective. Here, leveraging *Beta* distribution is natural. In the literature [16], when measuring reputation in cases involving binary assessment, *Beta* distribution has typically been adopted to represent probability distributions of binary events. Based on the *Beta* distribution, we propose a novel reputation measure for our framework next.

We start with the sketch of the *Beta* distribution, which is a two-parameter family of functions represented by α and β, defined as

$$f(Pr|\alpha, \beta) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} Pr^{\alpha-1}(1 - Pr)^{\beta-1}, \tag{1}$$

where $0 \leqslant Pr \leqslant 1$, $\alpha > 0$, $\beta > 0$ with the constraint that the probability variable $Pr \neq 0$ if $\alpha < 1$, and $Pr \neq 1$ if $\beta < 1$. The notion of Γ represents the *Gamma* function [8]. Then, given α and β, the expectation of Pr is described as follows:

$$E[Pr] = \frac{\alpha}{\alpha + \beta}. \tag{2}$$

Next, we detail the establishment of our reputation measure. We first define a behavior evaluator to reflect the resultant evaluations.

Definition 4.1 (Behavior Evaluator)**.** Let $e : \mathcal{P} \times \mathcal{RQ} \rightarrow \mathcal{S} = \{FL, DV, NS\}$ be a surjective function, for every $P_j \in \mathcal{P}$ and $req_i \in \mathcal{RQ}$, defined by

$$e(P_j, req_i) = \begin{cases} FL, & \text{if } P_j \text{ acts honestly in } req_i, \\ DV, & \text{if } P_j \text{ acts maliciously in } req_i, \\ NS, & \text{if } P_j \text{ is not selected in } req_i. \end{cases} \tag{3}$$

The detailed criterion, such as what behaviors are treated as FL, will be discussed in Section 5.1.3. Then, for counting the evaluations in $\mathcal{SC}$, based on e, we define the following counter function

Definition 4.2 (Counter Function)**.** For every $Pr_j \in \mathcal{P}$, $req_i \in \mathcal{RQ}$, $S \in \mathcal{S}$, the counter function $I : \mathcal{P} \times \mathcal{RQ} \times \mathcal{S} \rightarrow \{0, 1\}$ is defined as follows:

- If $e(P_j, req_i) = FL$, then $I(P_j, req_i, FL) = 1$, $I(P_j, req_i, DV) = 0$, and $I(P_j, req_i, NS) = 0$.
- If $e(P_j, req_i) = DV$, then $I(P_j, req_i, FL) = 0$, $I(P_j, req_i, DV) = 1$, and $I(P_j, req_i, NS) = 0$.
- If $e(P_j, req_i) = NS$, then $I(P_j, req_i, FL) = 0$, $I(P_j, req_i, DV) = 0$, and $I(P_j, req_i, NS) = 1$.

Such definitions aim at quantifying the evaluation results for further usage. As an example, if P_j participates in the i-th service req_i and gets an FL evaluation, then we have $I(P_j, req_i, FL) = 1$. After updating $I(P_j, req_i, DV) = 0$ and $I(P_j, req_i, NS) = 0$, we complete the evaluations for P_j in req_i. Based on this definition, we then define two behavior observers, namely *honest observer* and *malicious observer*. Such observers serve as a bridge to integrate our quantified evaluation results with the two parameters, α and β, in *Beta* distribution. Without loss of generality, we have

Definition 4.3.

- Honest Observer: We define an honest observer as the following function $\alpha' : \mathcal{P} \times \mathcal{RQ} \rightarrow \{0, 1\}$ defined by

$$\alpha'(P_j, req_i) = I(P_j, req_i, FL). \tag{4}$$

- Malicious Observer: We define a malicious observer as the following function $\beta' : \mathcal{P} \times \mathcal{RQ} \rightarrow \{0, 1\}$ defined by

$$\beta'(P_j, req_i) = I(P_j, req_i, DV). \tag{5}$$

Remark. In our design, as a special case where P_j does not receive evaluations in req_i, we have $\alpha'(P_j, req_i) = 0$ and $\beta'(P_j, req_i) = 0$.

Then, based on the definition of the observers, we focus on a specific peer P_j to count his/her historical observations and we have the following formal definition:

Definition 4.4 (Historical Observations)**.** For a given peer $P_j \in \mathcal{P}$, we define his/her historical honest observations among m requests as follows

$$\alpha(P_j) = \sum_{i=1}^{m} \alpha'(P_j, req_i) + 1. \tag{6}$$

Similarly, for the historical malicious observations, we have

$$\beta(P_j) = \sum_{i=1}^{m} \beta'(P_j, req_i) + 1. \tag{7}$$

As an example, for the peer P_j, $\alpha(P_j)$ records how many times he/she acted honestly among m engaged requests. Then, based on the above definitions, we have $E[Pr(j)]$ for every P_j,where $E[Pr(j)] = \frac{\alpha(P_j)}{\alpha(P_j)+\beta(P_j)}$ denotes the likelihood that P_j honestly follows the protocol when participating in an incoming future service, given the historical honest observations $\alpha(P_j)$ and the historical malicious observations $\beta(P_j)$. Based on $E[Pr(j)]$, we finalize the reputation measure of P_j as follows:

Definition 4.5 (Uncertainty-aware Reputation Measure)**.** For all $P_j \in \mathcal{P}$, given his/her historical honest observations $\alpha(P_j)$, historical malicious observations $\beta(P_j)$, and a pre-defined penalty factor ξ, where $\xi \geqslant 1$, we define uncertainty-aware reputation measure for P_j as follows,

$$Rep(P_j) = \frac{\alpha(P_j)}{\alpha(P_j) + 1 + \xi \cdot [\beta(P_j) - 1]}. \tag{8}$$

$\xi \geqslant 1$ is a predetermined penalty factor which is designed to penalize any dishonest behaviors of P_j. The design objectives of this measure are three-fold: first, given that the historical evaluations, $Rep(P_j)$ reflects the likelihood that P_j honestly follows future services and it measures the reputation with uncertainty. Second, adjusted by ξ, for P_j, $Rep(P_j)$ is gradually built up when P_j honestly contributes in the protocol and it is rapidly reduced when P_j dishonestly deviates the protocol. Third, to make the initial measure of each peer as the value of $\frac{1}{2}$, in the denominator, we assign two constants $+1$ and -1.

***An Example*:** For instance, if P_j has engaged in 12 timed-release services, wherein he/she honestly followed the protocol 8 times and dishonestly deviated the protocol 4 times, the reputation is measured by $Rep(P_j) = \frac{(8+1)}{(8+1)+1+3*4} = 0.41$, which indicates that for the incoming service engagement, P_j holds 0.41 likelihood to follow the service. In particular, for a peer who does not hold any past observations, the reputation score is initialized with $\frac{1}{2}$.

With our proposed reputation measure (Equation (8)) in hand, we next uncover our basic reputation-aware timed data release design.

5. Attack-resilient timed data release design: A basic protocol

In this section, we demystify the detail of our reputation-aware timed-release service protocol. Specifically, on top of our uncertainty-aware reputation mechanism built in Section 4, we carefully design

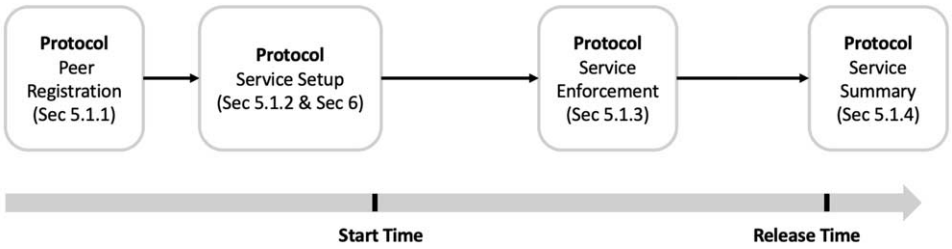


Fig. 3. Life cycle of reputation-aware timed data release protocol.

four tightly coupled subprotocols, namely ***Peer Registration***, ***Service Setup***, ***Service Enforcement*** , and ***Service Summary***, that jointly take charge of safeguarding a timed data release service. Roughly, as shown in Fig. 3, a service starts from the peer registration, followed by executing the service setup protocol from the sender's end, the service formally starts at the start time. Later, when the service is in operation, the service enforcement protocol comes to govern. Finally, after the release time, the service summary closes the current service. Next, we uncover the four protocols.

5.1. Reputation-aware timed-release protocol

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

5.1.1. Peer registration

Our basic protocols start peer registration protocol that aims at making any peer in the network get an opportunity to engage in our timed data release service. We detail our design as follows:

Step 1. At any time, any voluntary peer P_j can submit his personal service information $IF_j := \{pk_j, d_j, wd_j\}$ to the smart contract $\mathcal{SC}$. Specifically, pk_j denotes P_j's public key, d_j refers to a deposit in *Ether*, and wd_j captures his/her working window availability. In particular, we hereby assume that wd_j is represented by a time interval, reflecting the potential time availability of P_j when providing store-and-forward services. Concrete examples of such an availability will be shown in Fig. 4

Step 2. After confirming the registration request from P_j, $\mathcal{SC}$ initializes an account for P_j, denoted as $Peer(j)$, to store P_j's on-chain profile. Specifically, $Peer(j)$ is updated by $\mathcal{SC}$ as follows: (1) Update $P_j s$'s wd_j; (2) update pk_j for P_j; (3) update the historical observations, $\alpha(P_j)$ and $\beta(P_j)$, of P_j; (4) assign an initial reputation score $Rep(P_j) = 0.5$

Step 3. Followed by step 2, the information regarding P_j is public on-chain, which is able to be observed by every peer within the network.

5.1.2. 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. The 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 [23]. The novel feature that differentiates our protocol from the one in [23] 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 retrieve the available peer windows. S then locally constructs an *Interval Tree*

[15], namely *WITree*, to store such 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 $[T_s^i, T_r^i]$. 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. The 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 selected peers, we formally define such metrics as follows:

Definition 5.1 (Drop Attack Resilience)**.** The *drop attack resilience* of E, denoted as F_d^E, captures the failure likelihood of an adversary who launches drop attacks, quantified as

$$F_d^E = \prod_{k=1}^{l} \big[Rep(P_k)\big]. \tag{9}$$

Such a definition reflects that, to successfully launch the drop attack, the malicious adversary M must control at least one peer involved in the scheme E. Then, for the release-ahead attack resilience, we have

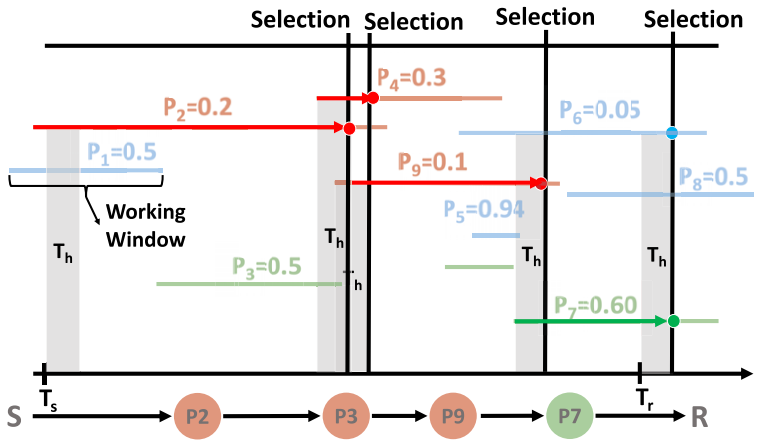
Definition 5.2 (Release-ahead Attack Resilience)**.** The *release-ahead attack resilience* of E, denoted as F_r^E, captures the failure likelihood of an adversary who launches release-ahead attacks at the start time, quantified as

$$F_r^E = 1 - \prod_{k=1}^{l} \big[1 - Rep(P_k)\big]. \tag{10}$$

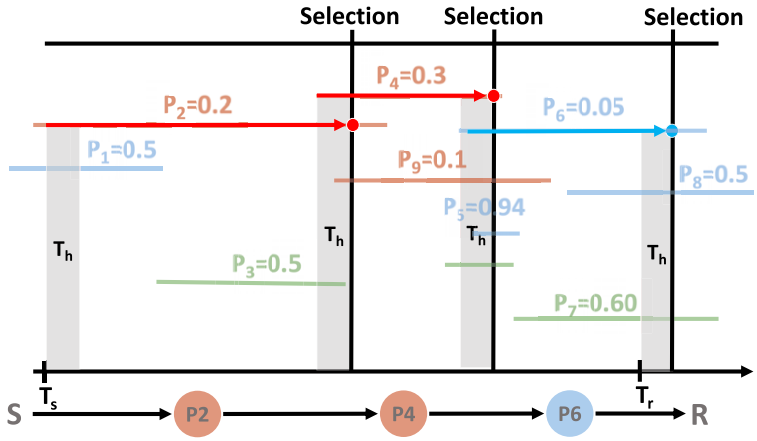
In the above definition, by directly following the adversarial setting of [20,21,23], we only consider that the adversary M intends to pre-release the data at the start time. Since the scheme E is protected with the *Onion Routing* [32] scheme, an adversary must control all peers to prematurely release 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. 4a. In the example, there are totally 8 peers P_1–P_9 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



(a) Reputation-aware Recruitment



(b) Time-aware Recruitment

Fig. 4. Peer recruitment example.

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 $Rep(P_7) = 0.6$, 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_6 and P_9, cover such a time point. The one holding a higher reputation P_9, $Rep(P_5) = 0.1$, is selected. Followed by the third-round selection, P_4 is selected. Finally, the last-round selection picks P_2 as well as checks that the start time of P_2 is prior to the start time of the timed-release service, T_s, which ends the reputation-aware recruitment procedure. The order of peers, $P_2 \rightarrow P_4 \rightarrow P_9 \rightarrow P_7$, 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 [23] as the baseline for comparison. To analyze its resilience, we take the release-ahead attack resilience as an example. The release-ahead attack resilience of E^1 from our proposed reputation-aware peer recruitment is derived from $F_r^{E^1} = 1 - (1 - 0.2) \cdot (1 - 0.3) \cdot (1 - 0.1) \cdot (1 - 0.6) = 0.7984$.

For the reputation-unaware recruitment in Fig. 4b, 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 $F_r^{E^2} = 1-(1-0.2)\cdot(1-0.3)\cdot(1-0.05) = 0.468$. Compared with our reputation-aware recruitment, $F_r^{E^2}$ shows significant degradation. By following a similar procedure, one can easily verify that the same results hold for F_d. After providing all information mentioned above, the service is ready to go.

5.1.3. 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 [23]. 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 [23] 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 [23]. 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.

We stress that the above procedures work under the assumption that malicious peers only launch the drop attack or the release-ahead attack. In fact, there may be other scenarios that impact the evaluation results, such as bad reputation evaluations on P_k, launched by P_{k+1}. Such scenarios, however, are out of the scope of this paper. It will be an interesting extension to our protocol, and we will leave this part as our potential future works.

5.1.4. 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 [23] 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 (8).

Taking into consideration the entire protocol, we would like to underscore the underlying philosophy of our design. In contrast to aiming to achieve immunity to any misbehavior, our focus lies in safeguarding the service by offering satisfactory service quality in terms of attack resilience. While the reputation measure, a key pillar in our design, cannot completely prevent misbehavior, it works in conjunction with peer recruitment to provide senders with valuable guidance regarding the service's attack resilience. For example, if a sender anticipates a 95% drop attack resilience for their service, but the recruitment policy can only guarantee 75%, the sender may opt to decline the service and thereby reduce the risk of data loss during its execution. Conversely, without the support of the reputation measure, in an environment plagued by malicious adversaries, senders are compelled to adhere to the recruitment scheme, which may involve these adversaries. Unfortunately, this significantly increases the likelihood of failures in timed data release services.

5.2. Limitations of the basic reputation-aware peer recruitment

In comparison with the baseline peer recruitment policy, our proposed straightforward peer recruitment approach does offer a scheme providing a better attack-resilient metrics. Such a design, however, inevitably suffers from a limitation that it cannot fully make use of the peers' working window distribution. Specifically, given a predetermined hand-off duration T_h, the recruited peers from the straightforward approach may not be the best scheme showing the highest release-ahead attack resilience or drop attack resilience. By taking the release-ahead resilience as an example, in Fig. 5, when facing such a working window distribution, the straightforward solution gives us the scheme $P_2 \rightarrow P_4 \rightarrow P_9 \rightarrow P_7$, which holds $F_r = 0.7984$. However, in Fig. 5, there exists another scheme, $P1 \rightarrow P_3 \rightarrow P_4 \rightarrow P_5 \rightarrow P_7$, which holds a higher release-ahead attack resilience, $F_r = 0.9958$. Based on this observation, we propose an improved version next to help senders with higher attack resilience.

6. Improved attack-resilient timed data release design: A temporal graph-based protocol

In this section, we discuss the design of our blockchain-based timed data release by adopting the notion of temporal graphs. We emphasize that approaches proposed in this section serve as an alternative, showing higher attack resilience, to the reputation-aware peer recruitment in the service setup protocol.

6.1. Framework descriptions

We begin by first formally capturing our blockchain-based timed data release based on the discussion of temporal graphs in Section 2.3. Before an incoming timed data release service, S first constructs a

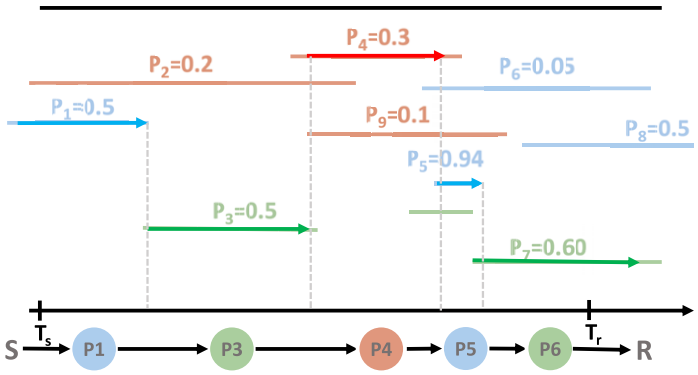


Fig. 5. A higher release-ahead attack-resilient path.

temporal graph based on the information published on the smart contract $\mathcal{SC}$. The resultant temporal graph, namely G_{TDR}, captures available peers as well as their future interactions. In this section, we assume that, if selected, each peer will contact his/her postdecessor on the service path to transfer the onion at the start time of his/her postdeceesor. Such an application scenario naturally follows the contact sequence representation in which temporal-related contact information in G_{TDR} is captured by a type of edge, namely, *temporal edge*, denoted as $e_i = \{se_i, de_i, t_i, \lambda_i, w_i, wd_{se_i}, wd_{de_i}\}$. Here, in comparison with the definition of the temporal edge in [42,43], our definition attaches two new items, wd_{se_i} and wd_{de_i}, to e_i. Such items represent working windows of se_i and de_i corresponding to e_i, which helps us make further decisions in Algorithm 1. We next unfold e_i as follows: se_i and de_i represent the source peer of e_i and destination peer of e_i, respectively. se_i can interact with de_i starting from t_i and ending at $(t_i + \lambda_i)$ to transfer the corresponding onion. Being aware of our two attack resilience metrics, here, we assign w_i two different types of non-negative weight, where $w_i = (w_{ir}, w_{id})$, in which w_{ir} reflects edge release-ahead attack success rate and w_{id} captures edge drop attack resilience. Formally, we have

$$w_{ir} = \begin{cases} -\ln[((1 - Rep(se_i)) \cdot (1 - Rep(de_i))], & \text{if } se_i \neq S \ \& \ de_i \neq R, \\ -\ln[1 - Rep(de_i)], & \text{if } se_i = S, \\ -\ln[1 - Rep(se_i)], & \text{if } de_i = R \end{cases} \tag{11}$$

and for w_{id}, we have

$$w_{id} = \begin{cases} -\ln[Rep(se_i) \cdot Rep(de_i)], & \text{if } se_i \neq S \ \& \ de_i \neq R, \\ -\ln[Rep(de_i)], & \text{if } se_i = S, \\ -\ln[Rep(se_i)], & \text{if } de_i = R. \end{cases} \tag{12}$$

Given the notion of the *temporal edge*, together with the temporal path definition in Section 2, we next define the notion of *Timed Data Release Service Path* that seamlessly fits in our service requirement.

Definition 6.1 (Timed Data Release Service Path)**.** Given a constructed temporal graph G_{TDR} containing registered peers' working time information, the sender vertex S and the recipient vertex R in G_{TDR}, the prescribed timed data release service interval $[T_s, T_r]$, let $\mathbb{E}(S, R, [T_s, T_r]) = \{TP$: TP is a temporal path from S to R containing a set of selected peers such that start(TP) $= T_s$, end(TP) $= T_r\}$. We define the

following two types of service paths that have the maximal attack resilience corresponding to our two attacks:

- Maximal Drop Attack-resilient Path: $TP \in \mathbb{E}$ is a maximal drop attack resilience path if $w_d(TP) = \min\{\sum_{i=1}^{|TP'|} w_{id}, TP' \in \mathbb{E}(S, R, [T_s, T_r])\}$.
- Maximal Release-ahead Attack-resilient Path: $TP \in \mathbb{E}$ is a maximal release-ahead attack resilience path if $w_r(TP) = \max\{\sum_{i=1}^{|TP'|} w_{ir}, TP' \in \mathbb{E}(S, R, [T_s, T_r])\}$.

Keeping such a definition in mind, to build an attack-resilient timed data release service, our peer recruitment procedure aims at exploring a service path consisting of a set of peers in G_{TDR} holding the highest release-ahead resilience or drop attack resilience within our prescribed service interval $[T_s, T_r]$. To systematically investigate such a requirement, the following two lemmas direct us to focus on the path related problems in temporal graph [42,43].

Lemma 6.1. *The minimal weight temporal path with the consideration of w_{id} per edge in G_{TDR} provides a service path with the maximal drop attack resilience.*

Proof. Assume that there exists a temporal path TP' holding the minimal weight value $w_d = \sum_{i=1}^{|TP'|} w_{id} = -2\ln[\prod_{i=1}^{|TP'|} Rep(se_i)Rep(de_i)]$. With the strictly decreasing property of the logarithm function $-\ln(\cdot)$, when $\prod_{i=1}^{|TP'|} Rep(se_i)Rep(de_i)$ arrives at the maximal value, we get the minimal value of $\sum_{i=1}^{|TP'|} w_{id}$. Therefore, based on the definition of F_d such a path gives us a maximal drop attack resilience scheme. □

Lemma 6.2. *The maximal weight temporal path with the consideration of w_{ir} per link in G_{TDR} provides a service path with the maximal release-ahead attack resilience.*

Proof. Assume that there exists a path TP' holding the maximal weight value $w_r = \sum_{i=1}^{|TP'|} w_{ir} = -2\ln[\prod_{i=1}^{|TP'|}(1 - Rep(se_i))(1 - Rep(de_i))]$. With the strictly increasing property of the logarithm function, when $\prod_{i=1}^{|TP'|}(1 - Rep(se_i))(1 - Rep(de_i))$ arrives at the minimal value, we get the maximal value of $\sum_{i=1}^{|TP'|} w_{id}$. Based on the definition of F_r, such a path provides a maximal release-ahead attack resilience scheme. □

We next formally investigate our timed data release service path finding problem.

6.2. Timed data release service path finding

6.2.1. Problem statement & analysis

Our timed data release service path finding problem is described as follows: Given a timed data release temporal graph G_{TDR}, a pair of vertex S and R in G_{TDR}, and a prescribed service interval $[T_s, T_r]$, can we find a temporal path from S to R with the maximal drop attack resilience or maximal release-ahead attack resilience?

We address this problem by showing the existence of an optimal substructure described as follows:

Lemma 6.3. *Assume that $TP = \{S, P_1, P_2, \ldots, R\}$ is a timed data release service path from S to R given a service interval $T = [T_s, T_r]$. Then, every prefix-subpath $TP_i = \{S, P_1, P_2, \ldots, P_i\}$, is a timed data release service path from S to P_i given a new service interval $[T_s, end(TP_i)]$.*

Proof. Since a timed data release service path may be maximal in drop attack resilience or release-ahead attack resilience, we only show the proof of the drop attack case here and the other is trivial.

We prove by contradicting. Now, TP is a service path with the maximal drop attack resilience. Assume that the prefix sub-path TP_i is not the maximal drop attack-resilient path. Then, there exists another path TP'_i holding a higher drop attack resilience such that $end(TP'_i) \leqslant end(TP_i)$. Such a path TP'_i will render a new path TP' holding higher drop attack-resilient compared with the original one TP, which contradicts our assumptions and this completes the proof. □

With such a property in hand, we propose a greedy algorithm for determining the service path.

6.2.2. An efficient algorithm

We design an efficient one-pass algorithm to derive the timed data release service path. We follow the convention in [42] to represent G_{TDR} in an edge stream manner, in which we sort $e_i \in G_{TDR}$ in a non-decreasing order of t_i. In Algorithm 1, per execution, we aim at deriving a timed data release service path that maximizes either w_r or w_d.

We take w_r as an example to illustrate our algorithm. Prior to scanning the edge stream, we first initialize several key attributes for every P_i. Specifically, we adopt $t[P_i]$ to denote the time that a path arrives at P_i, $w[P_i]$ to denote the weight w_r of that path, and $prev[P_i]$ to denote a predecessor of P_i that is used for retrieving peers on that path (line 4). Totally, we denote $R[P_i]$ to record every entry of $(w[P_i], prev[P_i], t[P_i])$. Our algorithm only accepts the edge satisfying our service time interval $[T_s, T_r]$ (line 8). At the heart of our algorithm, our greedy strategy (line 16) retrieves all the temporal paths that arrive at P_k before the start time, t_i, of the current edge and finds the one with the maximal w_{ir}, followed by updating P_m's weight and corresponding attributes (lines 19–20). In the comment line above line 26, we update $W[P_m]$ to let $W[P_m]$ record the maximal value. When the algorithm scans the final edge, where $P_m = R$, the algorithm will terminate and give us the maximal release-ahead resilience, stored in $W[R]$ (line 33). To retrieve the peers on that path, starting from $prev[R]$, we can recursively find predecessors.

Note that this proposed approach will be adopted by the sender before the start time of a service, which means that the entire procedure does not incur any on-chain operations that are expensive computationally.

***Comparisons*:** We provide an example in Fig. 6c to show our design rationale as well as the benefits of our proposed temporal graph-based design. In Fig. 6a, given the public peer working window distribution, we adopt the basic reputation-aware recruitment in Section 5.1.2 to get a routing path $S \to P_1 \to P_2 \to P_3 \to R$ that shows $F_r = 0.95$.

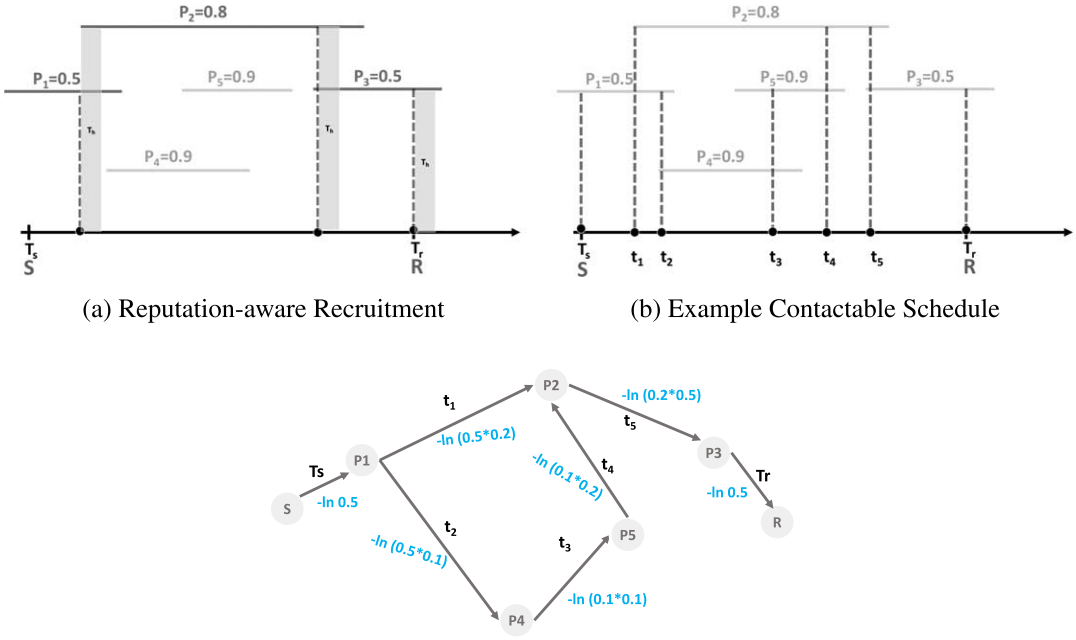
Then, for our temporal graph-based recruitment, in Fig. 6b, based on Algorithm 1, we can retrieve a timed data release service path from G_{TDR} with the maximal release-ahead attack resilience, namely $TP = \{S, P_1, P_4, P_5, P_2, P_3, R\}$. For simplicity, we skip to label duration λ_i here. Given the attached w_{ir} (highlighted in blue) for each edge, we can get the maximal $w_r = -2\ln(0.5*0.1*0.1*0.2*0.5)$ in G_{TDR}. Then, for this scheme, we have $F'_r = 1 - 0.5*0.1*0.1*0.2*0.5 = 0.9995$, which is higher than $F_r = 0.95$. For the drop attack resilience, following the similar procedure, one can easily verify such a benefit.

7. Evaluations

In this section, we evaluate our proposed framework using our simulation environment as well on the real-world *Ethereum* test network. Specifically, in Section 7.1, we perform simulation studies to demon-

Algorithm 1: Timed data release service path finding

Input: edge stream representation of G_{TDR}, sender vertex S, recipient vertex R, timed data release service interval $T = [T_s, T_r]$;
Output: The Service Path E with maximal w_d or w_r ;
foreach $P_i \in V_{G_{TDR}}$ **do**
    Initialize $w[P_i]$, $t[P_i]$, and $prev[P_i]$;
    Initialize $R[P_i] \leftarrow \{(w[P_i], prev[P_i], t[P_i])\}$;
end
/* $W[P_i] = -\infty$ is for w_r */
$W[P_i] = \infty$, $\boxed{W[P_i] = -\infty}$, $W[S] = 0$, $P_i \in V_{G_{TDR}} \setminus S$;
foreach *incoming* $e_i = \{P_k, P_m, t_i, \lambda_i, w_i, wd_{P_k}, wd_{P_m}\}$ **do**
    if $T_s \leqslant t_i \&\& (t_i + \lambda_i) \leqslant T_r$ **then**
        if $P_k == S$ **then**
            $w[P_k] = 0$;
            $t[P_k] = T_s$, $prev[P_k] = null$;
            $R[P_k].insert((w[P_k], prev[P_k], t[P_k]))$;
        end
        Find $w'[P_k] = \min\{w[P_k] : (w[P_k], prev[P_k], t[P_k]) \in R[P_k], wd_{P_k}.start \leqslant t[P_k] \leqslant t_i\}$;
        /* Replace line 15 with line 16 for w_r */
        $\boxed{\text{Find } w'[P_k] = \max\{w[P_k] : (w[P_k], prev[P_k], t[P_k]) \in R[P_k], wd_{P_k}.start \leqslant t[P_k] \leqslant t_i\}}$;
        $w[P_m] = w'[P_k] + w_{id}$;
        /* Replace line 17 with line 18 for w_r */
        $\boxed{w[P_m] = w'[P_k] + w_{ir}}$;
        $t[P_m] = t_i + \lambda_i$;
        $prev[P_m] = P_k$;
        if $t[P_m]$ *is in* $R[P_m]$ **then**
            update corresponding $w[P_m]$;
        else
            $R[P_m]$.insert($(w[P_m], prev[P_m], t[P_m])$);
        end
        /* For w_r, replace the condition in line 26 with $w[P_m] > W[P_m]$ */
        if $w[P_m] < W[P_m]$ **then**
            $W[P_m] = w[P_m]$;
        end
    else
        break and goto final;
    end
end
return $W[R]$



(a) Reputation-aware Recruitment (b) Example Contactable Schedule

(c) Corresponding Temporal Graph G_{TDR}

Fig. 6. An working example of temporal graph-based design.

strate the effectiveness and efficacy of our proposed basic protocol and the temporal graph-based peer recruitment algorithm. In Section 7.2, we provide a proof-of-concept implementation to demonstrate the practical applicability of our framework.

7.1. Simulation evaluations

The objective of our simulation evaluations are two-fold: *first*, with the help of our proposed reputation measure, we would like to show the benefits regarding the attack-resilient metrics in our proposed framework; *second*, in comparison with our basic protocols, we aim at showing and validating the optimal attack resilience from our temporal graph-based design.

7.1.1. Synthetic dataset

To the best of our knowledge, there is no existing public data set that reflects our application settings completely. We thus decide to synthesize a data set by closely following the settings of [23]. Detailed description on our synthetic data is elaborated as follows:

Synthetic Dataset: 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. Also, we consider that the generated peers follow a duty cycle-aware working fashion [9], in which the total working period of each peer is separate for two different cycles. One is online cycle that is represented by the generated working window associated with such a peer, the other is offline cycle that represents the offline time of such a peer. In general, we use a scalar value τ to represent the fraction of the online cycle. For example,

for a peer with a generated 10-hour online cycle (working window), given that $\tau = 50\%$, he/she will next have a 10-hour offline cycle. Totally, his/her working period is 20 hours. Based on τ, we synthesize two different peer profile with different duty-cycle characterizations. One peer data set has the setting of $\tau = 80\%$ for all generated peers, namely type 1 peer profile, $\mathcal{D}1$ for short. The other has the setting of $\tau = 40\%$ for all generated peers, namely type 2 peer profile, $\mathcal{D}2$ for short. For our temporal graph data, based on $\mathcal{D}1$ and $\mathcal{D}2$, we follow a similar procedure shown in Figs 6b, 6c. Specifically, if two peers' working window overlap, we treat them contactable and assign a corresponding start time. The temporal graph data is transformed to the edge representation.

7.1.2. Experimental settings

Metrics: To quantify the attack-resilient performance of our design, we study two attack-resilient metrics, *release-ahead attack resilience* (F_r) and *drop attack resilience* (F_d).

Comparisons: In our evaluations, we consider three different groups of protocols:

- *Non Reputation-aware Peer Recruitment (NRS)*: The most relevant work in literature is that of [23], where the authors propose a time-aware peer recruitment policy without the help of any reputation measure. We refer to it as *NRS*. In our evaluations, *NRS* serves as a baseline protocol. We consider two variants with 2 pre-defined parameters $|T_h| = 1$ hour and $|T_h| = 5$ hours. For simplicity, we denote such two variants as *NRS*1 and *NRS*5, respectively.
- *Basic Reputation-aware Peer Recruitment (RS)*: *RS* refers to our proposed basic reputation-aware peer recruitment policy proposed in Section 5.1.2. Based on two predefined values of $|T_h| = 1$ hour and $|T_h| = 5$ hours, we have two variants, namely *RS*1 and *RS*5, respectively.
- *Temporal graph-based Peer Recruitment (TGRS)*: *TGRS* refers to our proposed temporal graph-based policy in Section 6.2. Being aware of two different metrics F_r and F_d, we evaluate the two variants. One is to derive a maximal drop attack resilience path, namely *TGRSD*, the other is to derive a maximal release-ahead attack resilience path, namely *TGRSR*.

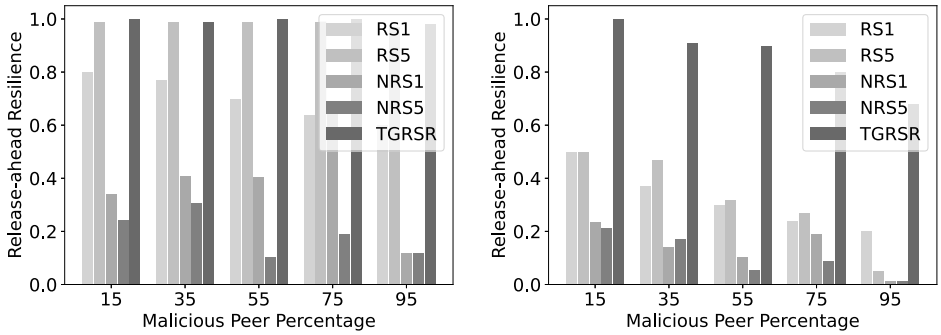
Protocol Configurations: Following the assumptions in Section 3.2, in our simulated protocol, the malicious peers in the registered pool deviate from the protocol completely, and the honest peers follow the protocol entirely. The rational peers only launch attack based on the bribery value from the malicious adversaries. The penalty factor $\xi = 10$.

7.1.3. Evaluation results

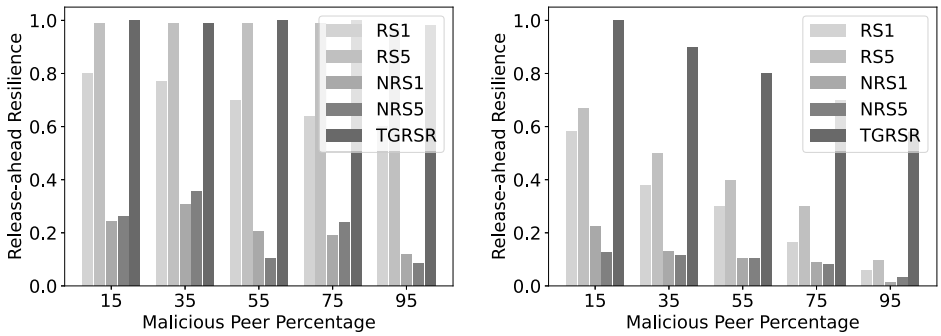
Global Experimental Evaluation Settings: For each evaluation, our simulator generates 500 requests for the evaluation, where 70% of them are training requests, and 30% of them are validation requests. We calculate the average resilience as the result of each peer malicious percentage setting. Each experiment is executed 10 times and the average resilience is obtained.

(1) Effectiveness under Different Malicious Peer Percentage

This suite of experiments demonstrates effectiveness of our proposed reputation-aware peer recruitment policy (RS1 and RS5) as well as the optimum of our proposed temporal graph-based ones (TGRSR/TGRSD) under various percentage of malicious peers. To offer a fine-grained analysis, we split our experiments into two sub-groups to observe resultant F_r and F_d. We first show the results yield by the first sub-group observing F_r. In Fig. 7a, we can observe that with 100 peers with $\tau = 0.8$, TGRSR always offers the highest release-ahead attack resilience than all other ones under different malicious peer percentage. The results for RS1 and RS5, though are less effective compared to TGRSR, they can provide in some sense good resilience against release-ahead attack. The remaining ones, NRS1 and



(a) F_r 100 Peers ($\mathcal{D}1$) (b) F_r 100 Peers ($\mathcal{D}2$)

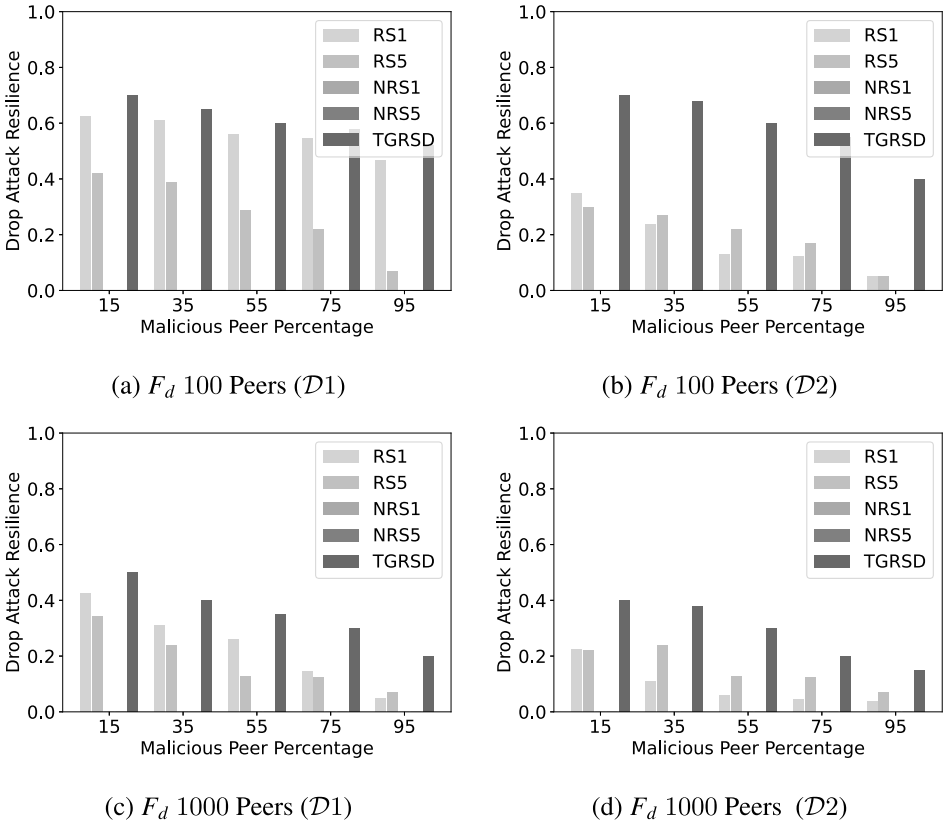


(c) F_r 1000 Peers ($\mathcal{D}1$) (d) F_r 1000 Peers ($\mathcal{D}2$)

Fig. 7. F_r evaluations under different malicious peer percentage.

NRS5 show the worst resilience among all. Next, in Fig. 7b, we make $\tau = 0.4$. TGRSR still offers the best resilience among all under different malicious peer percentage. Compared with TGRSR, RS1 and RS5 show a significant resilience degradation. NRS1 and NRS5 always offer the worst resilience among all. When comparing the results in Fig. 7a and Fig. 7b, we find that RS1/RS5 and NRS1/NRS5 show a significant variations in F_r when we vary τ to redistribute time windows. This validates our observations in Section 5.1.2 that these two sets of techniques are highly respective of the pre-determined $|T_h|$, which is sensitive to the variations of τ. Finally, in Figs 7c, 7d, we show evaluations under a larger scale with 1000 registered peers. The results show a similar observations as the ones reported from Figs 7a, 7b. In summary, under different malicious percentage settings, in face of the release-ahead attack, TGRSR effectively protects against such an attack. RS1 and RS5, with their non-deterministic performance, lie at the second position. Such observations remain invariant when meeting different τ and the scale of the peers.

In the second sub-group, we investigate F_d. First, in Fig. 8a, TGRSD always realizes the highest drop attack resilience under various malicious peer percentage. RS1 and RS5 lie at the second position which, in regards to F_d, is less effective compared to TFRSD while significantly surpassing NRS1 and NRS5. When we make $\tau = 0.4$ with 100 peers, the observations remain almost invariant. By horizontally comparing Fig. 7b with Fig. 7a, the changes in τ renders a significant variation in F_d, which, again, is attributed to the pre-determined $|T_h|$ sensitive to the time window distribution. In Figs 8c, 8d, we increase



(a) F_d 100 Peers ($\mathcal{D}1$) (b) F_d 100 Peers ($\mathcal{D}2$)

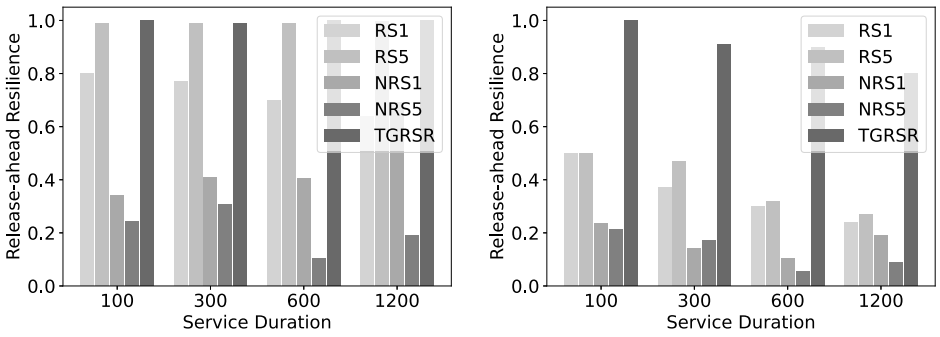
(c) F_d 1000 Peers ($\mathcal{D}1$) (d) F_d 1000 Peers ($\mathcal{D}2$)

Fig. 8. F_d evaluations under different malicious peer percentage.

the number of peers. The corresponding observations are similar as the ones mentioned in Figs 8a, 8b. Therefore, under different malicious percentage settings, when meeting the drop attack, TGRSD is the best performer. RS1 and RS5 lie at the next position. These observations are similar with different τ and different number of peers.

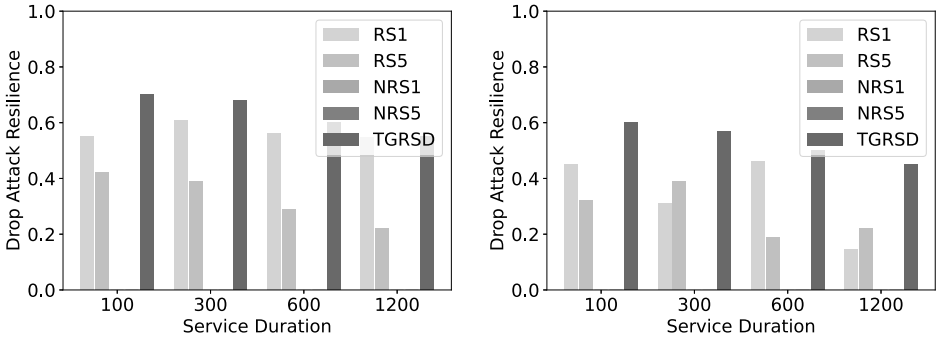
(2)Effectiveness under Different Duration of Service: In this set of experiments, we aim at learning the effectiveness of our proposed protocol under different duration of service time. Apart from the default 300 hours service time, in this set of experiments, we set a short duration, 100 hours, a moderately long duration 600 hours, and a long duration 1200 hours. We fix the number of registered peers as 1000 and the malicious peer percentage as 35%.

In Figs 9a, 9b, in the evaluation of F_r, under different time duration settings, TGRSR always achieves the highest F_r. Though RS1 and RS5 show a lower F_r in comparison with TGRSR, they significantly outperform NRS1 and NRS5 which struggles to handle the release-ahead attack. Figures 10a, 10b show the evaluations of F_d under different service duration settings. TGRSD always offers the highest F_d. RS1 and RS5 give a lower F_d compared with TGRSD. Both TGRSD as well as RS1 and RS5, compared with NS1 and NS5, show a significant improvement in F_d. In summary, under different service duration, TGRSD always offer the best drop attack resilience and TGRSR achieves the best release-ahead attack resilience.



(a) F_r ($\mathcal{D}1$) (b) F_r ($\mathcal{D}2$)

Fig. 9. F_r evaluations under different timed-release service durations.



(a) F_d ($\mathcal{D}1$) (b) F_d ($\mathcal{D}2$)

Fig. 10. F_d evaluations under timed-release service durations.

7.2. Prototype implementation

7.2.1. Smart contract implementation

In this section, we present our prototype implementation. We implement our smart contract in the official programming language *Solidity* [36]. We deploy our smart contract on Ethereum official test network *Rinkeby* [33] through the interface, *Infura* [14]. We use the data from our simulation studies as the input for on-chain validations. We have 7 selected peers in total that take charge of 1200 hour service. The results from Table 1 show the details of our implementations. 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

Table 1
Gas cost

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

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

7.2.2. Monetary cost analysis

From Table 1, 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.

8. Related work

8.1. Blockchain-based timed data release

With recent advancements in decentralized cryptocurrency solutions, adopting blockchains to develop decentralized timed data release has been gaining attention. There has been two major lines of work on this topic. The first line of work strives to provide effective cryptographic constructions. Specifically, Liu et al. [26–28] proposed to construct a variant of timed-lock encryption [34] by incorporating the notion of *computational reference clocks*, derived from *Bitcoin* with *Witness Encryption* [11]. Later, Ning et al. [31] constructed a provably-secure scheme using secret sharing [35] and *Ethereum* smart contracts [10]. Recent solution, named i-TiRE [2], offers a gas-efficient timed-encryption construction on *Ethereum*. Though the above-mentioned theoretical constructions provide provably-secure guarantee, their scalability is often limited and therefore, it is hard to directly adopt such schemes for practical blockchain-based timed data release. The second line of work on this topic has focused on developing practical alternative solutions for timed data release. Conretely, CTDRB [41] provides a controllable blockchain-based timed data release solution by carefully considering the tradeoff between data privacy and data control. By characterizing adversaries as economically rational, Li et al. [23] designed protocols in smart contracts as an extensive-form game with imperfect information [19]. Bacis et al. [1] developed an alternate construction with the help of *Secure Multiparty Computation* [7]. However, in a real-world decentralized network, there may be also malicious adversaries in addition to rational adversaries. To mitigate such an adversarial environment, our earlier work [39,40] has introduced the preliminary concepts and techniques related to uncertainty-aware reputation mechanisms for peer recruitment for tackling the *mixed adversarial environment* comprising of both rational as well as malicious adversaries.

8.2. Reputation measure in blockchains

The transparent nature of blockchains provides a suitable context to build reputation systems. There are several representative blockchain-based reputation mechanisms in the literature. Based on social norm, Li,et al. [25] build a reputation model on *Ethereum*-based crowdsourcing framework to regulate the worker's behaviors. Zhou, et al [45] leverage a simple reputation measure to block the witnesses with misbehaviors in a blockchain-based witness model. RTChain [37] integrates a reputation system into the blockchains that focus on e-commerce to achieve transaction incentives and distributed consensus. RepuCoin [44] proposed *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.

8.3. Temporal graph and applications

The notion of Temporal Graph [13] provides an effective approach for modeling real-world network activities such as social contact networks. *Temporal Rechability*, as a critical temporal-related property, measures whether two given vertices are reachable within a given time interval. There has been a group of work aiming at exploring minimum temporal path in a given temporal graph. One can treat such a path as a counterpart of shortest path in static graph but with more fine-grained awareness of temporal measure. Our proposed temporal graph-based timed data release construction is closely related to these problems. Xuan et al. [43] suggest three important types of temporal path, namely *earliest-arrival time*,

foremost path, as well as *shortest traversal path*. They also propose efficient algorithms to derive such paths. Such algorithms, however, inevitably incur computational complexity because of their complicated graph representation. By observing such a limitation, the work, proposed by [42], addresses this by representing a temporal graph in an edge-stream manner. With their proposed streaming algorithms, the computational efficiency of the three types of paths is improved. However, directly adopting their designs for our optimal attack-resilient path is not feasible as the greedy techniques in [42] are based on temporal metrics and are not aware of our edge resilience metrics. In summary, prior techniques strive to boost computational efficiency within a constraint scope focusing on the three types of paths and as a result, they are not suitable as effective solutions in our timed data release design.

9. 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 are proposed to take charge of monitoring the states of the 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.

Acknowledgment

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] E. Bacis, D. Facchinetti, M. Guarnieri, M. Rosa, M. Rossi and S. Paraboschi, I told you tomorrow: Practical time-locked secrets using smart contracts, in: *The 16th International Conference on Availability, Reliability and Security, ARES 2021*, Association for Computing Machinery, New York, NY, USA, 2021.

[2] L. Baird, P. Mukherjee and R. Sinha, i-TiRE: Incremental timed-release encryption or how to use timed-release encryption on blockchains? in: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 235–248. doi:10.1145/3548606.3560704.

[3] P. Balaji and C. Li, Self-emerging data infrastructures, in: *2019 IEEE 5th International Conference on Collaboration and Internet Computing (CIC)*, 2019, pp. 256–265.

[4] N. Bitansky, S. Goldwasser, A. Jain, O. Paneth, V. Vaikuntanathan and B. Waters, Time-lock puzzles from randomized encodings, in: *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, 2016, pp. 345–356. doi:10.1145/2840728.2840745.

[5] D. Boneh and M. Naor, Timed commitments, in: *Annual International Cryptology Conference*, Springer, 2000, pp. 236–254.

[6] T.H. Cormen, C.E. Leiserson, R.L. Rivest and C. Stein, *Introduction to Algorithms*, MIT Press, 2022.
[7] R. Cramer, I.B. Damgård et al., *Secure Multiparty Computation*, Cambridge University Press, 2015.
[8] P.J. Davis, Leonhard Euler's integral: A historical profile of the gamma function: In memoriam: Milton Abramowitz, *The American Mathematical Monthly* **66**(10) (1959), 849–869.
[9] Duty cycle. https://en.wikipedia.org/wiki/Duty_cycle.
[10] Ethereum. https://ethereum.org/en/.
[11] S. Garg, C. Gentry, A. Sahai and B. Waters, Witness encryption and its applications, in: *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing*, 2013, pp. 467–476. doi:10.1145/2488608.2488667.
[12] D. Goldschlag, M. Reed and P. Syverson, Onion routing, *Communications of the ACM* **42**(2) (1999), 39–41. doi:10.1145/293411.293443.
[13] P. Holme and J. Saramäki, Temporal networks, *Physics Reports* **519**(3) (2012), 97–125. doi:10.1016/j.physrep.2012.03.001.
[14] Infura. https://infura.io/.
[15] Interval tree. https://github.com/gnidan/interval-trees-solidity.
[16] A. Josang and R. Ismail, The beta reputation system, in: *Proceedings of the 15th Bled Electronic Commerce Conference*, Vol. 5, 2002, pp. 2502–2511.
[17] K. Kasamatsu, T. Matsuda, K. Emura, N. Attrapadung, G. Hanaoka and H. Imai, Time-specific encryption from forward-secure encryption, in: *International Conference on Security and Cryptography for Networks*, Springer, 2012, pp. 184–204. doi:10.1007/978-3-642-32928-9_11.
[18] R. Kikuchi, A. Fujioka, Y. Okamoto and T. Saito, Strong security notions for timed-release public-key encryption revisited, in: *International Conference on Information Security and Cryptology*, Springer, 2011, pp. 88–108.
[19] K. Leyton-Brown and Y. Shoham, Essentials of game theory: A concise multidisciplinary introduction, *Synthesis Lectures on Artificial Intelligence and Machine Learning* **2**(1) (2008), 1–88. doi:10.1007/978-3-031-01545-8.
[20] C. Li and B. Palanisamy, Emerge: Self-emerging data release using cloud data storage, in: *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, IEEE, 2017, pp. 26–33. doi:10.1109/CLOUD.2017.13.
[21] C. Li and B. Palanisamy, Timed-release of self-emerging data using distributed hash tables, in: *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, IEEE, 2017, pp. 2344–2351. doi:10.1109/ICDCS.2017.109.
[22] C. Li and B. Palanisamy, Decentralized privacy-preserving timed execution in blockchain-based smart contract platforms, in: *2018 IEEE 25th International Conference on High Performance Computing (HiPC)*, IEEE, 2018, pp. 265–274. doi:10.1109/HiPC.2018.00037.
[23] C. Li and B. Palanisamy, Decentralized release of self-emerging data using smart contracts, in: *2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS)*, IEEE, 2018, pp. 213–220. doi:10.1109/SRDS.2018.00033.
[24] C. Li and B. Palanisamy, SilentDelivery: Practical timed-delivery of private information using smart contracts, *IEEE Transactions on Services Computing* **15**(6) (2022), 3528–3540. doi:10.1109/TSC.2021.3098858.
[25] M. Li, J. Weng, A. Yang, W. Lu, Y. Zhang, L. Hou, J.-N. Liu, Y. Xiang and R.H. Deng, Crowdbc: A blockchain-based decentralized framework for crowdsourcing, *IEEE Transactions on Parallel and Distributed Systems* **30**(6) (2018), 1251–1266. doi:10.1109/TPDS.2018.2881735.
[26] J. Liu, F. Garcia and M. Ryan, Time-release protocol from bitcoin and witness encryption for sat, *Korean Circulation Journal* **40**(10) (2015), 530–535.
[27] J. Liu, T. Jager, S.A. Kakvi and B. Warinschi, How to build time-lock encryption, *Designs, Codes and Cryptography* **86**(11) (2018), 2549–2586. doi:10.1007/s10623-018-0461-x.
[28] J. Liu, S.A. Kakvi and B. Warinschi, Extractable witness encryption and timed-release encryption from bitcoin, *IACR Cryptol. ePrint Arch.* **2015** (2015), 482.
[29] T.C. May, Timed-release crypto, 1993. http://www.hks.net/cpunks/cpunks-0/1460.html.
[30] J.F. Nash Jr., Equilibrium points in n-person games, *Proceedings of the National Academy of Sciences* **36**(1) (1950), 48–49. doi:10.1073/pnas.36.1.48.
[31] J. Ning, H. Dang, R. Hou and E.-C. Chang, Keeping time-release secrets through smart contracts, *IACR Cryptol. ePrint Arch.* (2018), 1166.
[32] M.G. Reed, P.F. Syverson and D.M. Goldschlag, Anonymous connections and onion routing, *IEEE Journal on Selected Areas in Communications* **16**(4) (1998), 482–494. doi:10.1109/49.668972.
[33] Rinkeby. https://www.rinkeby.io/#stats.
[34] R.L. Rivest, A. Shamir and D.A. Wagner, Time-lock puzzles and timed-release crypto.
[35] A. Shamir, How to share a secret, *Communications of the ACM* **22**(11) (1979), 612–613. doi:10.1145/359168.359176.
[36] Solidity. https://docs.soliditylang.org/en/v0.8.10/.
[37] Y. Sun, R. Xue, R. Zhang, Q. Su and S. Gao, Rtchain: A reputation system with transaction and consensus incentives for e-commerce blockchain, *ACM Transactions on Internet Technology (TOIT)* **21**(1) (2020), 1–24. doi:10.1145/3432249.
[38] N. Szabo, Formalizing and securing relationships on public networks, *First Monday* (1997).

[39] J. Wang and B. Palanisamy, 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, pp. 123–140.
[40] J. Wang and B. Palanisamy, Protecting blockchain-based decentralized timed release of data from malicious adversaries, in: *2022 IEEE International Conference on Blockchain and Cryptocurrency*, 2022.
[41] J. Wang and B. Palanisamy, CTDRB: Controllable timed data release using blockchains, in: *SecureComm 2022*, 2023, pp. 231–249.
[42] H. Wu, J. Cheng, S. Huang, Y. Ke, Y. Lu and Y. Xu, Path problems in temporal graphs, *Proceedings of the VLDB Endowment* **7**(9) (2014), 721–732. doi:10.14778/2732939.2732945.
[43] B.B. Xuan, A. Ferreira and A. Jarry, Computing shortest, fastest, and foremost journeys in dynamic networks, *International Journal of Foundations of Computer Science* **14**(02) (2003), 267–285. doi:10.1142/S0129054103001728.
[44] J. Yu, D. Kozhaya, J. Decouchant and P. Esteves-Verissimo, Repucoin: Your reputation is your power, *IEEE Transactions on Computers* **68**(8) (2019), 1225–1237. doi:10.1109/TC.2019.2900648.
[45] H. Zhou, X. Ouyang, Z. Ren, J. Su, C. de Laat and Z. Zhao, A blockchain based witness model for trustworthy cloud service level agreement enforcement, in: *IEEE INFOCOM 2019 – IEEE Conference on Computer Communications*, IEEE, 2019, pp. 1567–1575. doi:10.1109/INFOCOM.2019.8737580.