Keeping Time-Release Secrets through Smart Contracts

Jianting Ning [*] Hung Dang [†] Ruomu Hou [‡] Ee-Chien Chang [§]

November 29, 2018

Abstract

A *time-release* protocol enables one to send secrets into a future release time. The main technical challenge lies in incorporating timing control into the protocol, especially in the absence of a central trusted party. To leverage on the regular heartbeats emitted from decentralized blockchains, in this paper, we advocate an incentive-based approach that combines *threshold secret sharing* and blockchain based *smart contract*. In particular, the secret is split into shares and distributed to a set of incentivized participants, with the payment settlement contractualized and enforced by the autonomous smart contract. We highlight that such approach needs to achieve two goals: to reward honest participants who release their shares honestly after the release date (the "carrots"), and to punish premature leakage of the shares (the "sticks"). While it is not difficult to contractualize a carrot mechanism for punctual releases, it is not clear how to realise the stick. In the first place, it is not clear how to identify premature leakage. Our main idea is to encourage public vigilantism by incorporating an informer-bounty mechanism that pays bounty to any informer who can provide evidence of the leakage. The possibility of being punished constitute a deterrent to the misbehaviour of premature releases. Since various entities, including the owner, participants and the informers, might act maliciously for their own interests, there are many security requirements. In particular, to prevent a malicious owner from acting as the informer, the protocol must ensure that the owner does not know the distributed shares, which is counter-intuitive and not addressed by known techniques. We investigate various attack scenarios, and propose a secure and efficient protocol based on a combination of cryptographic primitives. Our technique could be of independent interest to other applications of threshold secret sharing in deterring sharing.

1 Introduction

A time-release protocol allows a secret to be sent into the future. The secret is encrypted in such a way that it remains so until a particular *release time* to come, which is predefined by the secret owner. Time-release protocols have found plenty of real-world applications such as sealed-bid auctions, electronic auctions, scheduled payment methods, and lotteries.

There are extensive studies on time-release protocols. One approach relies on a trusted authority to control the timing of the decryption [39, 16, 6, 28, 37]. Another approach binds

[*]National University of Singapore, *ningjt@comp.nus.edu.sg.*

[†]National University of Singapore, *hungdang@comp.nus.edu.sg.*

[‡]National University of Singapore, *houruomu@comp.nus.edu.sg.*

[§]National University of Singapore, *changec@comp.nus.edu.sg.*

the decryption of the secret to a computational puzzle that is assumed to require a specific amount of time to solve. Examples of this approach include constructions by Rivest *et al.* [39], Bellare *et al* [4], Boneh *et al* [7] and other subsequent works [35, 19, 20]. Nevertheless, it is not easy to predict the amount of time needed to solve a given computational puzzle, due to variances in hardware design and their performances. Consequently, it is difficult to tune the system to fit into the intended release time.

A recent line of works explore an integration of blockchain-based computational puzzles in time-release protocols [29, 31, 32]. This approach is motivated by the emergence of public blockchains such as Bitcoin and Ethereum, and their block chaining mechanism that emits regular heartbeats. More specifically, in these prominent blockchains, mechanism are in-place to maintain constant expected block time (i.e., the time it takes to mine a block) over time. For instance, the "difficulties" of Bitcoin are adjusted according to the present computing power, so as to have expected block time of ten minutes. One can treat the issuance of a new block in the blockchain as its heartbeat. Potentially, these heartbeats can provide a secure and reliable timing control for time-release protocol, obliviating the need of a trusted third party. Liu *et al.* [31] suggested combining Bitcoin blockchain and witness encryption to implement time-release protocol. In particular, one could encrypt a secret using a "public key" extracted from the current state of the Bitcoin blockchain, while the corresponding "decryption key" is extracted from the chain of blocks that are to be created in the future. However, existing constructions of witness encryption (e.g., [22, 24]) are based on cryptographic multilinear maps [21] and currently, which have high performance overhead [14, 23, 15].

In this work, we consider a practical, incentive-based approach that implements a time-release protocol using threshold secret sharing and smart contract. In particular, the *secret owner* divides the secret into n *shares* using a (t, n) threshold secret sharing scheme [40], and distributes the shares to n *participants* where each participant is entrusted with one share. The participants are expected to reveal the shares on the release time, allowing the reconstruction of the secret. To encourage participation, monetary incentive (or *carrot*) is provided by the secret owner, and the rewarding of such incentive to the participants is enforced by the smart contract. More specifically, based on the secure and reliable heartbeats emitted by the blockchain, the smart contract is able to verify the required conditions regarding the release time, and its autonomous execution enforces the fulfilment of contractual obligations for all parties. Further, the security of (t, n) threshold secret sharing scheme guarantees that if no more than $t-1$ participants collude (i.e., pool their shares together), no single entity (except the secret owner) is able to obtain the secret before the release time.

Nonetheless, the above "carrot-only" incentive overlooks a potential threat arising from the prevalent sharing behaviours in social media. While it is very unlikely to have a large number of mutually trusting participants to collude (e.g., a set of t or more independent participants pool their shares to reconstruct the secret at will, or there is a single entity capable of gaining control over all of them), other weaker forms of sharing misbehaviours exist. For instance, some participants could post their shares on social media platforms (e.g., Twitter or Facebook), either broadcast to the public or to a smaller social media group, to show off their involvement in the time-release protocol. While many participants might value their social media reputation and refrain from the behaviour, the deterrence based on social responsibility is arguably weak and thus additional mechanisms are required.

We propose incorporating monetary penalty as "stick" so as to further deter sharing. In particular, the participants are required to pay a certain security deposit prior to taking part in the protocol. Should the participants faithfully follow the protocol, they are refunded with their deposit and awarded with a reward provided by the secret owner (i.e., the carrot). Conversely, if there is evidence of a participant's wrongdoing (e.g., leaking his share prematurely on social platform), his deposit will be slashed. The slashed deposits constitute a pool of bounties that are paid to individuals or entities, called *informers* that present the wrongdoing evidences. The penalties, together with the rewards, constitute a *carrots-and-sticks* approach. Since any entity can be the informer, this bounty-informer mechanism encourages online vigilantism and can be viewed as a form of crowdsourcing, which enlists the online community to assist in identifying leakage. To an indiscreet participant, the possibility of having an informer in the participant's social group constitutes a deterrent to dishonest publishing. Note that the bounty-informer mechanism provides deterrence, which is a weaker security requirement compares to the assurance that any premature publishing is guaranteed to be punished.

Realising the carrot-and-stick mechanism is not straightforward, due to the suspicions among the owner, participants and the informers, who might selfishly act for their own interests. For instance, a greedy owner may play as the informer in order to claim the bounty. To prevent that, the mechanism must assure that the owner does not know the distributed shares, even though the owner knows the secret, and is involved in the shares distribution. Additionally, the owner might maliciously release wrong shares, so as to deny the participants from receiving their deserved award. On the other hand, a dishonest participant, instead of prematurely publish the exact share it received, might jointly publish some transformed data together with other participants. Thus, the informer-bounty mechanism would not be effective if it gives out bounty only when the exact share is presented as evidence. Furthermore, there are stringent computation requirements, imposed by the expensive cost of smart contract execution.

To address the challenges, we first give a threat model that investigates various attack scenarios. We consider a form of collusion whereby a large number of cliques, each consists of $(t-1)$ or fewer mutually trusting participants, broadcast their transformed shares. This form of collusion captures scenarios where a large number of participants colluded and further posted information in social media. Note the crucial difference from the classical threshold secret sharing which considers a small group (less than t) of mutually trusting participants, and our more general model which considers a large number of small tightly-knit groups who are allowed to publish information on public bulletin board once. In other words, instead of stopping at allowing groups of $(t-1)$ colluders, our model further allows the groups to broadcast some information. The security goal here is to track and penalise the malicious participant(s) via their broadcast behaviours. Our model also consider other scenarios, for instance, a greedy owner who plays the role of the informer, another informer who attempt to "steal" evidence from others, etc.

We give a protocol that achieves our proposed security requirements. One of the key constructions is a mechanism that ensures the owner is not aware of the distributed shares. Our main idea is to let the participant i chooses a secret sample point s_i, and then through some specialised secure 2-party computation, distribute $f(s_i)$ to the participant while hiding s_i from the owner. Our protocol also incorporates a form of verifiable secret sharing and commitment scheme to prevent various forms cheating. The underlying cryptographic primitives in our protocol could be relevant to other applications of threshold secret sharing in deterring large scale causal sharing.

Our protocol requires only one exponentiation to be carried out per share (for claiming the deposit and reward). In our implementation in Solidity and Ethereum, the computation of each share costs 40,000 gas[1] per share for effective key size of 128 bits. Our construction focuses on the essential interactions between involved parties. We abstract away from subtle configuration details such as deposit and reward composition, bounties structures, or potential role of brokerage. We remark that their dispositions are critical to ensure incentive-compatibility and sustainability of the system and would be interesting extensions.

In summary, this work makes the following contributions:

[1]As of July 2018, Ethereum averaged USD450 per ETH. With gas price set to 5Gwei, 40,000 gas amounts to roughly 9 cents.

- We advocate combining threshold secret sharing and smart contract to keep time-release secrets. We highlight the threat of causal social-media sharing and propose an incentive-based carrot-and-stick approach.
- We investigate various attack scenarios for the proposed approach, and further present the corresponding threat model.
- We propose an efficient protocol that realises the proposed approach. The protocol provides deterrence to causal sharing and requires only one on-chain exponentiation per share.

The remaining of the paper is structured as follows. In Section 2, we briefly summarize preliminaries on smart contract and secret sharing scheme. We then state our problem definition and its security requirements in Section 3, before presenting our proposed protocol in Section 4. The security analysis of the proposed protocol is presented in Section 5. We discuss implementation details and performance optimization of our solution in Section 6, and survey the related works in Section 7. Finally, we conclude our work in Section 8.

2 Preliminaries

In this section, we give a brief overview smart contracts. Although our discussion focuses on the Ethereum platform, the solution and security arguments presented in this work are also applicable to other smart contract platforms.

2.1 Smart Contract

A *smart contract* (or contract) in the Ethereum blockchain is an "autonomous agent" whose private storage and executable code are stored on the blockchain. The contract's storage is private in a sense that data or variables stored within the storage can only be manipulated by the contract's executable code. The read access, however, is publicly available, due to the public nature of the blockchain. A contract is created by a "creation transaction" posted to the blockchain, and uniquely identified by a *contract address*. In addition to the private storage, the contract *state* also consists of a *balance*, which is essentially an amount of native virtual coins (i.e., ETH) that it holds.

The code of an Ethereum contract is typically written in high-level programming languages, such as Solidity [2] or Serpent [1], and then compiled into a low-level, stack-based bytecode language known as Ethereum Virtual Machine (EVM) code. To invoke a contract at address *addr*, one sends a transaction *tx* to *addr* with payload typically containing payment (in ETH) for the contract execution, and input data for the invocation [33]. The security of the Ethereum blockchain guarantees that as long as a majority of its miners behave honestly following the Ethereum protocol, the contract code is executed properly, and its state is recorded correctly on the blockchain. In other words, contract execution integrity is guaranteed.

Gas System. Since the Ethereum smart contact execution consumes certain computational expense, to ensure fair compensation, Ethereum pays miners a certain fee which is proportional to the computation expended. This fee is paid for by the user who wishes to invoke the contract. In particular, each instruction in the EVM code has a pre-set amount of *gas*. When a user sends a transaction *tx* to invoke a contract, she has to specify the `gasPrice` indicating how much she is willing to pay for each *gas* (in the native virtual coin ETH), and the `gasLimit` indicating how much she is willing to spend for the entire smart contract execution. A miner who "mines" *tx* (i.e., executes the computation that *tx* entails, and includes *tx* and the updated state resulted from such computation in his proposed block) is paid the transaction fee proportional to the amount of gas required by the execution multiplied by `gasPrice`, subject to a condition that such a fee is within the limit set by `gasLimit`. If the contract invocation consumes a fee larger than `gasLimit`, the execution is halted, and the contract state is reverted to the initial state prior to the invocation. It is worth noting that the transaction fee is still collected by the miner to compensate for the expended but then rolled back computation. This mandatory payment is also a means to mitigate resource-exhaustion attacks [34].

2.2 Shamir's (t, n) Secret-Sharing Scheme

Adi Shamir [40] introduced a threshold secret sharing scheme. The scheme is parameterized by two value, namely n and t. Under this scheme, a secret s is split into n "shares" such that any set of t shares is sufficient to reconstruct s. The security of the scheme dictates that no adversary that possesses fewer than t shares could reveal any information about s.

3 Time-release Mechanism via Smart Contract and Secret Sharing

In this section, we first state the system model and assumptions that we make in designing a time-release protocol, before giving an overview of our proposed carrots-and-sticks approach. We then discuss the adversary model, as well as the security and performance requirements that justify and motive our protocol design.

3.1 Setting and Assumptions

3.1.1 Entities

Our time-release protocol assumes a model that comprises three main entities:

- *Secret owners* would like to seal their secrets such that they can only be revealed at some predefined times in the future. A owner of the secret s delegates its safe keeping to a set of n *participants* using a threshold secret sharing scheme. The secret owner provides a reward to incentivise the participants in upholding the safe keeping. It is important to note that the secret owner may not involve in the reveal or reconstruction of the secret s. That is, after distributing the shares, he could go offline and might not rejoin the system.
- *Participants* make financial gain by keeping the shares issued by the secret owner private until the prescribed *release time*. At the release time, the participants reveal their share, claiming the reward provided by the secret owner. The obligation of a participant is to safe keep and not revealing its share, and back online on the release date to reveal the secret.
- *Informers* police the system, earning bounties by exposing malicious participants (i.e., they capture and report evidences of participants' wrongdoings). The secret owner and the participants can also play the role of informer.

We assume that participants may form cliques, and there could be multiple cliques. All members of a clique mutually trust each other[2]. Further, a member is aware of internal states and shares held by other members of the clique. Consequently, if an adversary is able to control one member of the clique, it will know the internal states of all clique's members. In other words, a clique can be deemed as a single entity that possesses multiple shares. We

[2]We assume that trust is transitive, and that cliques are non-overlapping.

assume that the size of each clique is not larger than a threshold t. One can imagine a worst case scenario where every clique is of size $(t-1)$.

The main motivation of grouping participants into cliques in our formulation is to model a scenario where a malicious entity is able to gather a small group of mutually trusting colluders or a small group of colluders who are willing to carry out expensive secure multiparty computation protocol. The reason of considering secure multiparty computation protocol is that, it is possible for the malicious entity to orchestrate a secure multi-party computation [26] among mutually distrusting participants to jointly compute any function on their individual input data without revealing it. However, such computation requires a large number of interaction rounds, incurs a significant overhead, and cannot scale to large number of participants (at a relatively low cost). Thus, it is reasonable to assume that the malicious entity can only gather a small group (less than t) of such participants.

3.1.2 Smart Contract

We assume that all entities have access to a public blockchain-based smart contract system such as Ethereum [8]. They can submit transactions to the smart contract, and see all transactions in the system. Similar to earlier works [31, 32], our formultation assumes that the underlying consensus protocol of the blockchain on which the smart contract runs attains both safety and liveness. At the time of writing, the underlying consensus protocol in Ethereum essentially implements a replicated state machine, wherein the smart contracts and their transactions are to be executed by every miner in the network. Further, each transaction execution entails a transaction fee that correlates with the execution's complexity. Hence, it is desirable to minimize on-chain cost by moving as many expensive computations off the blockchain as possible.

3.1.3 Communication models

We consider three communication models. The first model features highly interactive communications among a small group of colluders. The second model features posting the blockchain transactions, whereas the third model captures causal sharing or broadcast (e.g., posting to a social media platform such as Twitter or Facebook) of information. In our formulation, participants within a clique can access each other's internal state, and their communications assume the highly interactive model. The communication models are illustrated in Figure 1.

To model the more restrictive causal sharing, we assume that participants can post anonymous messages to a public bulletin after they have received their shares. In this restrictive model, we assume that there is no out-of-band communication channel. Consequently, cross-clique communications can be performed by having the senders posting the messages to the bulletin, follows by reading from the builletin. One may argue that in practice, interactions can be carried out without the public bulletin. We treat these forms of communication as interaction within the clique, and thus there is no loss of generality.

For the purpose of capture the scenario that participants are performing casual sharing, we restrict user to posting once and then read transactions.

3.2 Carrots-and-Sticks Approach

We advocate an incentive-based approach, which we call *carrots-and-sticks*, to reward honest participants and penalise dishonest participants. This approach requires all participants to pay deposits prior to participating in the protocol. If a participant behaves honestly, the deposit will be refunded, and a reward (provided by the secret owner) will be allotted to him. On the other hand, if there exists evidence of his wrongdoings, his deposit is slashed.

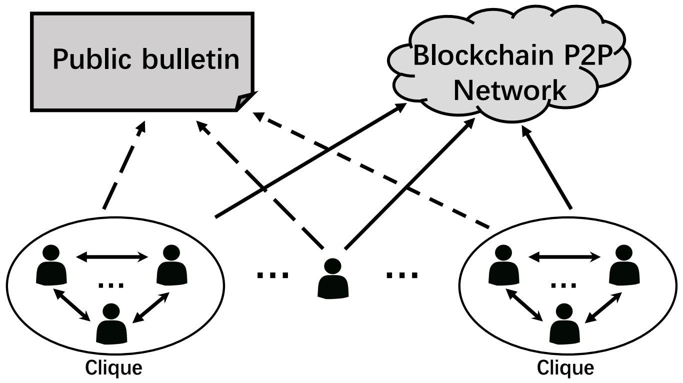


Figure 1: Overview of communication model. The solid arrows represent sending transactions to the smart contract, and the dashed arrows represent posting messages to the public bulletin.

The slashed deposits of dishonest participants constitute a pool of bounties that informers who police the system[3].

The approach consists of three phases:

- *Handshake phase.* The purpose of this phase is for the secret owner to initiate the protocol, recruiting participants and distributing the shares computed using the (t, n) secret sharing scheme. This phase could be carried out in multiple rounds. Upon the completion of this phase, the secret owner and the recruited participants would have made their deposits into the smart contract.
- *Waiting phase.* During this phase, any informer can report malicious behaviour by submitting transactions containing wrongdoing evidences to the smart contract. The smart contract verifies the transactions. If a transaction is valid, the smart contract extracts the identities of the dishonest participant(s) from the transaction's payload, slashes their deposits, and pays the informer a bounty.
- *Reveal phase.* This phase starts at the release date. In this phase, each participant is to submit its share, and other meta information if any, to the smart contract system. An honest participant will get back the deposit and the deserved reward. With sufficient number of shares submitted, anyone can retrieve the shares from the smart contract system and recover the secret. Slashed deposits and unclaimed rewards can also be redistributed to other honest entities during this phase.

Remarks.
(1) Implementing the carrots and sticks approach that rewards the honest and punishes the dishonest is challenging. A simple implementation may reward any entity who submits a share with a valid authentication tag signed by the secret owner on the release date. Unfortunately, this simple implementation cannot prevent a malicious secret owner from framing the participant by presenting a participant's share and the corresponding authentication tag as evidence to steal the informer bounty. Coding conditions of the penalty of pre-mature release is arguably even more challenging. Section 3.3 investigates the attack scenarios and Section 4 proposes a protocol that realise the carrots and sticks approach.
(2) It is essential to codify the conditions of rewards and bounties settlements in the smart contract. This is to enforce autonomous fulfillment of contractual obligations, especially

[3]The slashed deposits can also be redistributed to other honest participants. We leave the concrete compositions of the deposits and structures of the bounties to future investigations.

in a scenario where the secret owner goes offline after the handshaking phase. In such situation, the smart contract autonomously and automatically enforces and carries out the owner's obligations, such as payment to honest participants on release data, even in the secret owner's absence.
(3) The obligations of the participants include keeping their shares private until the release date, and publishing it on such a date. Security deposit made by a participant remains "locked" in the smart contract until the participant claims it after the release date, or being slashed in case of misbehaviour. A participant might decide to act as the informer giving evidence against itself to receive the informer bounty, at a cost of losing its own deposit. This would lead to a negative or zero net return and damaged reputation, if any, depending on the protocol design. Although such action seems counter-intuitive, the participant may opt to so behave in case of urgent liquidity need. Hence, the act of committing a security deposit and joining the protocol is somewhat analogous to making a fixed-rate deposit with guaranteed interest earned on maturity date, with a prerequisite that early withdrawal is subject to a penalty.

3.3 Adversary model and security

An entity might deviate from the protocol so as to maximise its monetary return. Generally, we need to consider the following three major security invariants in the presence of malicious secret owner and/or participants.

- Confidentiality of the secret before the release date.
- Guaranteed reward for honest participants.
- Rightful claim on informer's bounty.

Note that the above three goals correspond to the interests of the secret owner, participants and informers, respectively. The remaining of this section articulates the attack scenarios and their counter measures as well as the security requirements of our protocol.

3.3.1 Confidentiality of shared secret

The shares held in any clique (of at most $(t-1)$ participants) must not reveal any information of the secret. More specifically, consider the scenario where all participants first carry out the handshaking phase honestly, and then the participants in a clique attempt to jointly extract some information of the secret. Clearly, the protocol needs to protect the confidentiality of the secret from such attempts.

3.3.2 Deterrence against premature release

Here, we consider the scenario where participants in a clique might jointly post a message to the public bulletin board. As highlighted earlier, deterrence against such dishonest sharing is implemented using a combination of bounties given to informers who assist in policing the system, and penalties imposed on the dishonest participants. From the perspective of a participant tempted to share its data, the possibility that the receiving end leverages the shared data against him (i.e., using the data to claim the informer bounty at the cost of the sharer's deposit being slashed) constitutes the deterrence. Note that the receiving end (who is the informer) could be any other participants, or even the secret owner. In the worst case scenario to the sharer, the receiving end is the rest of the system.

To this end, we need to formulate the requirements of the deterrence mechanism. Such formulation needs to handle cases wherein the posted messages are transformations of the shares and they could contain only partial information of the original. To achieve deterrence, we require that if a clique C_0 posts a message x that contains useful information, then there exists a mechanism to trace x to some participants in the clique C_0. Intuitively, a piece of data x is useful if it can be combined with some other data y to reconstruct the secret, while the secret cannot be derived from either x or y alone.

In our security analysis, we capture the above intuition under a game between a challenger and an adversary. The game comprises the following steps:

1. Handshake phase is honestly carried out by the challenger who acts as honest secret owner and participants.
2. The adversary chooses $(t-1)$ participants to form a clique C_0 and is given access to the randomness and the internal state of each participant in C_0. Next, the adversary plays the role of the participants in C_0 and posts a message x.

The adversary wins if there exists a polynomial time algorithm that can recover the secret on input x, internal states of other $(t-1)$ participants that are not in C_0 and the public parameters, and yet there does not exists an informer algorithm that can provide evidence against at least one of the participant in C_0. A precise description of this game is presented in Section 5.

3.3.3 Framing and Reward stealing

To encourage participation, it is crucial to have strong protection of the participants' interests. To protect a honest participant, we have to ensure that no other entities, including other participants and the secret owner, can claim or or prevent the honest participants from claiming the reward on the release date. It turns out that this requirements can be easily achieved in smart contract by ensuring that only the identity who made the deposit can claim the reward during reveal phase. Nevertheless, such requirement should be explicitly stated, since there could be some constructions which attempt to dis-associate identities involved in both phases.

Likewise, it is important to prevent framing of honest participants, that is, forging evidence against the honest participant during the waiting phase. In a certain sense, this requirement is on authenticity of the share, in contrast to the previous two requirements.

Our security analysis (Section 5) models this security invariant using a game between a challenger and a strong adversary who controls the owner and all the participants except for one honest participant (i.e., victim). In such a game, all entities behave honestly during handshaking. Next, the adversary practicing control over the secret owner and all other participants' internal states attempts to forge the share to claim the victim's deposit and reward.

3.3.4 Unclaimable shares

Recall that participants can only reclaim their deposits by submitting valid shares on the release data. A malicious secret owner may sabotage honest participants' deposits by creating and distributing inconsistent or corrupted shares that cannot be verified on the release date. In such an attack scenario, the honest participants lose their deposits, which could be redistributed to the owners and its colluders depending on the reward and deposit compositions of the system.

3.3.5 Evidences stealing

A common attack surface inherent to blockchains, in particular Ethereum, involves transaction-ordering dependence (TOD)[4] and front-running[5]. In particular, if there are two competing

[4]`https://consensys.github.io/smart-contract-best-practices/known_attacks/`
[5]`http://swende.se/blog/Frontrunning.html`

transactions submitted to the blockchain (e.g., two informers submitting wrongdoing evidences of the same dishonest participant, but only one of them can claim the bounty), which of them is executed/settled is at miners' discretion. Worse yet, if an adversary is aware of an honest informer's evidence submission transaction before such transaction is mined, it can front-run the latter by submitting a transaction containing the exact same evidence "in front" (i.e., associating the transaction with higher gas price, colluding with the miners to prioritize its transaction or censoring that of the informer). In light of these potential attack scenarios, the protocol must be designed to be robust against TOD and front-running so as to guarantee fairness in the system.

3.4 Performance requirement

In addition to the security requirements discussed above, the protocol should attain efficiency at scale. We detail in the following desired properties that benefit performance of the protocol:

3.4.1 Low on-chain costs

Given the scalability barrier of the blockchain system [18], and the cost of on-chain smart contract execution, the proposed mechanism needs to move as many expensive computations off the blockchain as possible.

3.4.2 Non-reliance of direct communication channels

Since the participants and the owner may not be online throughout the handshaking phase, solutions that require real-time communication channels between the involved parties should be avoided. Such direct communication might compromise the participants' privacy (for instance, revealing ip-address) in the absence of additional layer of protection.

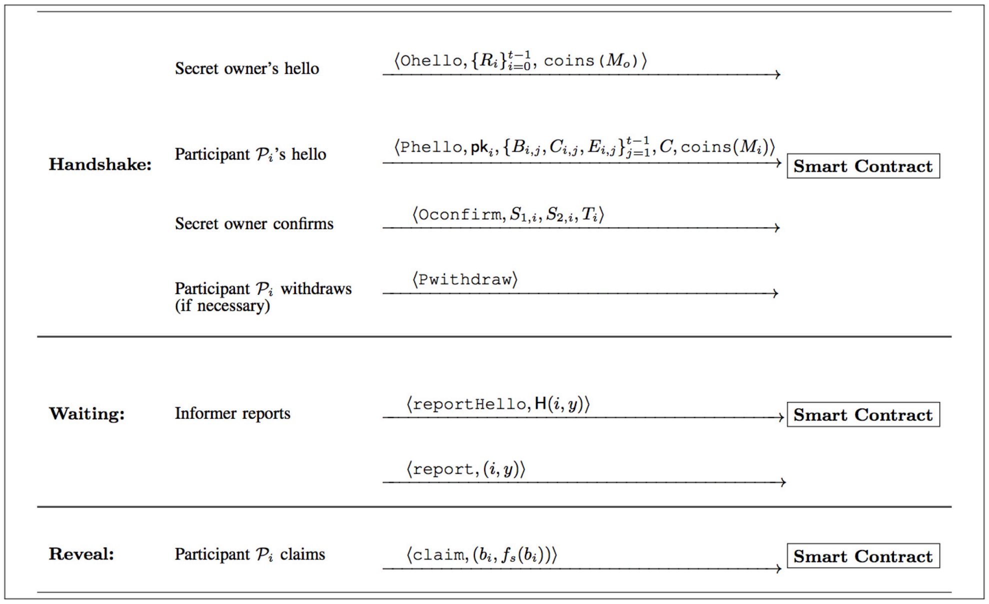
4 The proposed protocol

This section presents a protocol realising the carrots-and-stick approach.

4.1 Main Idea

Similar to Shamir's secret sharing scheme, in our protocol, each participant $\mathcal{P}_i$ holds a share $\langle b_i, f_s(b_i)\rangle$ where $f_s()$ is a $(t-1)$-degree polynomial with $f_s(0) = s$, and b_i is the sample point associated to the participant $\mathcal{P}_i$. However, to protect from a malicious owner who attempts to use the shares as evidences against the participants, the tuple $\langle b_i, f_s(b_i)\rangle$ has to be hidden from the secret owner, which is challenging since the shares are to be generated by the secret owner.

Our solution is to let a participant's share to be jointly determined by the secret owner and the participant himself. Specifically, we let the participant choose a secret sample point as b_i that is unknown to the secret owner. By doing so, besides preventing the malicious owner from withholding the reward, we can also show that if the participant has dishonestly posted useful information, then the value b_i can be extracted from the posted message. Any informer who has successfully extracted b_i from the posted message can submit b_i to open a "commitment" previously made by the participant and obtain the informer's bounty. Since the shares $\langle b_i, f_s(b_i)\rangle$ are to be computed between the participant and secret owner who are suspicious of each other, cryptographic primitives such as additive homomorphic encryption are employed to preserve confidentiality and consistency of the data.



Secret owner's hello
$\langle \texttt{Ohello}, \{R_i\}_{i=0}^{t-1}, \texttt{coins}(M_o) \rangle$
Handshake:
Participant $\mathcal{P}_i$'s hello
$\langle \texttt{Phello}, \mathsf{pk}_i, \{B_{i,j}, C_{i,j}, E_{i,j}\}_{j=1}^{t-1}, C, \texttt{coins}(M_i) \rangle$
Smart Contract
Secret owner confirms
$\langle \texttt{Oconfirm}, S_{1,i}, S_{2,i}, T_i \rangle$
Participant $\mathcal{P}_i$ withdraws (if necessary)
$\langle \texttt{Pwithdraw} \rangle$
Waiting:
Informer reports
$\langle \texttt{reportHello}, \mathsf{H}(i, y) \rangle$
Smart Contract
$\langle \texttt{report}, (i, y) \rangle$
Reveal:
Participant $\mathcal{P}_i$ claims
$\langle \texttt{claim}, (b_i, f_s(b_i)) \rangle$
Smart Contract

Figure 2: Summary of the proposed protocol.

Note that there are a number of requirements and cryptographic primitives involved in the above description, which have to be neatly incorporated without compromising each other.

4.2 Notation

Every entity in the smart contract system has an identity. Without loss of generality, let us assign the participants the identity $1, 2, \ldots, n$ and the secret owner the identity 0. Let us denote the participant with identity i as $\mathcal{P}_i$ for each i. An informer can create new unique identity.

Each transaction submitted to the smart contract is represented in this format:

$$\langle \texttt{TransType}, pars, \texttt{coins}(M) \rangle,$$

where TransType specifies the transaction type, *pars* are the list of paramenters, and $\texttt{coins}(M)$ specifies M coins to be transferred to the smart contract system.

Each transaction is bound to the sender's identity. We assume that authenticity of the sender is protected, that is, it is difficult to forge the sender of the transaction.

We define $[t_1, t_2] = \{t_1, t_1 + 1, ..., t_2\}$ for $t_1, t_2 \in \mathbb{N}$.

4.3 Proposed Protocol

We denote by HE a public key additive homomorphic encryption which provides semantic security against chosen-plaintext attacks (abbreviated IND-CPA), and by H a collision-resistant hash function. Let G be a group of prime order p, such that the discrete logarithm problem is hard in this group. Let g be the group generator of G, which is the public parameter of the protocol. The secret sharing scheme in used is parameterized by (t, n), in which n indicates the number of shares to be created, and by its extension, number of participants necessitated in the secret safe keeping, whereas t indicates the number of shares needed to reconstruct the secret. Let M_i denote a security deposit made by participant $\mathcal{P}_i$ prior to participating in the protocol, and M_o denote the reward provided by the secret owner to compensate the participants for the safe keeping of his secret. For simplicity, we assume that all participants' deposits are equal. We abuse the notation and refer to participant's deposit by M.

Our proposed protocol comprises three phases, namely *Handshake*, *Waiting*, and *Reveal*. It relies on a smart contract to enforce contractual obligations fulfillment of all entities. The entities interact with the smart contract using seven types of transactions: `Ohello`, `Phello`, `Oconfirm`, `Pwithdraw`, `reportHello`, `report` and `claim`. We depict in Figure 2 a summary of our protocol.

4.3.1 Handshake phase

The handshake phase consists of the following rounds:

- (*Owner's hello*) The secret owner first chooses random $s, \alpha \in \mathbb{Z}_p$ and computes $h = g^{\alpha}$. Let h^s be the secret to be shared and h be the public parameter of the protocol. The secret owner then submits a transaction `Ohello` to recruit participants. The secret owner chooses random $r_1, ..., r_{t-1} \in \mathbb{Z}_p$ and submits a transaction:

  $$\left\langle \texttt{Ohello}, \{R_i = g^{r_i}\}_{i\in[0,t-1]}, \texttt{coins}(M_o)\right\rangle$$

  Let $f_s()$ be the polynomial

  $$f_s(x) = \sum_{i=0}^{t-1} r_i x^i$$

  where $r_0 = s$.

- (*Participant's hello*) A participant indicates interest in joining by submitting a transaction `Phello`. Suppose participant $\mathcal{P}_i$ wants to join, $\mathcal{P}_i$ first chooses random $b_i, c_i \in \mathbb{Z}_p$ as its secret and computes

  $$\{B_{i,j} = g^{b_i^j}, C_{i,j} = h^{c_i b_i^j}\}_{j\in[1,t-1]}, C = g^{c_i}.$$

  $\mathcal{P}_i$ then initializes an instance of the additive homomorphic encryption scheme HE. Let pk_i be the public parameters of HE instaniated by $\mathcal{P}_i$. Next, $\mathcal{P}_i$ computes

  $$\{E_{i,j} = \mathsf{HE.Enc}(b_i^j)\}_{j\in[1,t-1]}$$

  where $\mathsf{HE.Enc}()$ is the encryption algorithm of HE. Finally, $\mathcal{P}_i$ submits the transaction:

  $$\langle \texttt{Phello}, \mathsf{pk}_i, \{B_{i,j}, C_{i,j}, E_{i,j}\}_{j\in[1,t-1]}, C, \texttt{coins}(M)\rangle$$

- (*Owner's confirmation*) The secret owner selects a set of n participants, and submits a transaction `Oconfirm` for each of them. Without loss of generality, let us assume the selected participants are $\{\mathcal{P}_i\}_{[1,n]}$. For each $\mathcal{P}_i$, the secret owner computes

  $$\begin{aligned} S_{1,i} &= g^s \prod_{j=1}^{t-1} (B_{i,j})^{r_j}, \\ S_{2,i} &= C^{\alpha s} \prod_{j=1}^{t-1} (C_{i,j})^{r_j}, \\ T_i &= \mathsf{HE.Enc}(s) + \prod_{j=1}^{t-1} r_j E_{i,j} \end{aligned}$$

and submits the transaction:

$$\langle \texttt{Oconfirm}, S_{1,i}, S_{2,i}, T_i \rangle.$$

- (*Participant's confirmation*) The participant $\mathcal{P}_i$ verifies whether the values $(S_{1,i}, S_{2,i}, T_i)$ submitted by the secret owner is consistent. If not, $\mathcal{P}_i$ submits a transaction `Pwithdraw`. Specifically, using the corresponding secret key of pk_i, $\mathcal{P}_i$ checks whether the following two equalities both hold:

$$\begin{aligned} g^{\mathsf{HE.Dec}(T_i)} &\overset{?}{=} S_{1,i}, \\ h^{\mathsf{HE.Dec}(T_i)} &\overset{?}{=} S_{2,i}, \quad \text{and} \\ \prod_{j=0}^{t-1} (R_j)^{b_i^j} &\overset{?}{=} S_{1,i} \end{aligned}$$

where $\mathsf{HE.Dec}()$ is the decryption algorithm of HE. If not, $\mathcal{P}_i$ submits the transaction

$$\langle \texttt{Pwithdraw} \rangle$$

to withdraw. In response to this transaction, the smart contract returns the deposit back to $\mathcal{P}_i$.

If both equalities hold, $\mathcal{P}_i$ does not submit any transaction. Note that in this case, the participant can obtain the value $f_s(b_i)$ by decrypting T_i:

$$\mathsf{HE.Dec}(T_i) = f_s(b_i).$$

Hence, at this point, $\mathcal{P}_i$ has the share

$$\langle b_i, f_s(b_i) \rangle.$$

- (*Restart if necessary*) If less than n participants remain at this round, the smart contract returns all deposits and restarts handshaking.

4.3.2 Waiting phase

During this phase, any entity may submit evidence to the smart contract to report dishonest sharing. Suppose an informer knows the value of $y = f_s(b_i)$ hold by participant $\mathcal{P}_i$. The informer claims the informer's bounty in two rounds. First, the following transaction is submitted:

$$\langle \texttt{reportHello}, \mathsf{H}(i, y) \rangle$$

After the transaction is processed by the smart contract, the informer submits:

$$\langle \texttt{report}, (i, y) \rangle$$

In response to the transaction, the smart contract verifies that the digest in `reportHello` matches the (i, y) in `report`. If so, verify that the evidence is valid by checking the equality

$$g^y \overset{?}{=} S_{1,i}.$$

If the equality holds, informer bounty is transferred to the informer.

4.3.3 Reveal Phase

During this phase, the participant $\mathcal{P}_i$ can submit a `claim` transaction to reveal the share and claim the reward. Let $f_i = f_s(b_i)$ which $\mathcal{P}_i$ obtained during the handshaking phase. $\mathcal{P}_i$ submits the following transaction:

$$\langle \texttt{claim}, (b_i, f_i) \rangle$$

In response to the transaction, the smart contract verifies the following conditions:

- $\mathcal{P}_i$ has not been reported during the waiting phase.
- The equalities
$$g^{f_i} \stackrel{?}{=} S_{1,i},$$

If both conditions hold, then the deposit and reward is released to $\mathcal{P}_i$.

Note that the secret h^s can be obtained if more than $t-1$ participants have successfully claimed their deposits.
Remark.
(1) Our exposition thus far has abstracted away from subtle configuration details of the system, for examples concrete deposit and reward composition, or bounties structures. We also have not discussed in details how the slashed deposits are to be distributed. Such dispositions are critical to ensure incentive-compatibility and sustainability of the system. We believe that their thorough analysis require economics and game theory expertise, which is beyond the scope of this work. Nonetheless, for completeness, we include in our discussion one intuitive setting.

Recall that M_o denotes the reward provided by the secret owner to compensate the safe keeping the secret s, and M denotes a deposit made by a participant prior to participating in the protocol. If the participant behaves honestly, at the end of the protocol, he should receive his deposit together with a portion of the reward M_o (i.e., $M + (M_o/n)$). On the contrary, if he commits a wrongdoing whose evidence is presented to the smart contract, his deposit is slashed, and the informer is paid a bounty of value $M/2$. The remaining portion of the slashed deposits can be equally distributed to all honest participants who have faithfully participated in the protocol.
(2) During the waiting phase, only the first informer to submit the valid evidence of some wrongdoing obtains the informer bounty. Nonetheless, should there be two informers submitting evidence of the same wrongdoing to the smart contract, the order by which those submissions are processed depends on various factor (e.g., gasPrice, network connectivity and propagation mechanisms), and thus is potentially subject to adversarial manipulation. Even worse, the adversary can even hijack the informer bounty by front-running the evidence submission transaction as discussed in Section 3.3.

Our construction resolves this attack scenario by including the `reportHello` step. Nonetheless, it brings forth attack scenario wherein a participant submits `reportHello` against itself, without further submitting the `report` transaction. In so doing, it denies other informers of the smart contract's inform-function. This can be deterred by imposing a deposit on the first `reportHello` transaction, and a time limit to complete the second transaction.
(3) To illustrate how an informer can obtain the evidence, let us consider the following example. Assume a participant $\mathcal{P}_i$ dishonestly posted a message $\mathcal{M}$ on the public bulletin. If $\mathcal{M}$ is useful (i.e., there exists an algorithm $\mathcal{B}$ that, on input $\mathcal{M}$ and $(t-1)$ other shares, outputs s), the informer can carry out the following steps to determine $f_s(b_i)$:

(a) Randomly chooses $t-1$ sample points. Feed $\mathcal{B}$ with $\mathcal{M}$ and the chosen $t-1$ points, and obtain the secret s_1 whose value is $f_1(0)$ where f_1 is the polynomial fitting the randomly chosen points and s_1.

(b) Repeats Step (a) obtain $f_2, ..., f_d$ ($d \geq 2$), until there is only one common intersection point(s) among the curves $f, f_1, ..., f_d$.

The common intersection point must be at the P_i's secret sample point, which the informer can present to the smart contract as evidence.

5 Security Analysis

5.1 Confidentiality of shared secret

First of all, for the secret owner's interest, the proposed protocol in Section 4 must address the confidentiality of shared secret. The requirement for preventing the leakage of secret from shares is essentially the security requirement of the underlying Shamir's (t, n) Secret Sharing scheme. That is, the secret cannot be recovered as long as at most $t-1$ participants are compromised. Nevertheless, some messages submitted by the secret owner might leak information on the secret s, and thus we need to investigate the protocol carefully.

To model the prevention of the leakage of secret from shares, we define a security model based on a security game Game_{SC} between a challenger and an adversary. The phases of the game are the following:

- *Handshake:* The challenger acts as honest secret owner and participants in this phase.
- *Waiting phase:* During the Waiting phase, the challenger and the adversary interact as follows:
  - *Collude query*: The adversary sends a $\mathsf{Collude}$ query to the challenger. The challenger sends $t-1$ participants' shares and internal random coins to the adversary.
  - *Challenge*: The adversary outputs a challenge secret S.

We define that the adversary wins the Game_{SC} game if S equals the secret chosen by the secret owner. We say that the proposed protocol in Section 4 achieves the confidentiality of shared secret if no PPT adversary can win the Game_{SC} game.

Lemma 1 *If the computational Diffie-Hellman assumption holds and* HE *is IND-CPA secure, the shared secret of the proposed protocol in Section 4 is computationally hidden.*

The proof sketch of this lemma is given Appendix A.1.

5.2 Deterrence against premature release

In case of participants in a clique post a message to the public bulletin board, the proposed protocol provides a deterrence mechanism. We define the security model for deterrence against premature release via a security game Game_{DPR} between a challenger and an adversary. The phases of the game are the following:

- *Handshake:* The challenger acts as honest secret owner and participants in this phase.
- *Waiting phase:* During the Waiting phase, the challenger and the adversary interact as follows:
  - *Collude query*: The adversary sends a $\mathsf{Collude}$ query to the challenger. The challenger sends $t-1$ participants' shares and internal random coins to the adversary.
  - *Challenge*: The adversary posts a message on the public bulletin board. An informer reports this malicious behaviour via submitting `reportHello` and `report` transactions to the smart contract. The smart contract identifies the colluded participants through the `reportHello` and `report` transactions.

The adversary wins the above Game_{DPR} game if no participant(s) it colluded can be identified by the smart contract. The proposed protocol in Section 4 achieves deterrence against premature release if no PPT adversary can win the Game_{DPR} game.

Lemma 2 *If the computational Diffie-Hellman assumption holds and* HE *is IND-CPA secure, the proposed protocol in Section 4 achieves deterrence against premature release.*

The proof of this lemma is given in Appendix A.2.

5.3 Framing and Reward stealing

The proposed protocol ensures that a honest participant's deposit can only be claimed by the participant itself. That is, no entities (including other participants and the secret owner) can claim its deposit via submitting a `claim` transaction with an incorrect share. We called the entity who submits a `claim` transaction with an incorrect share the *incorrect-share* entity. The security model for security against *incorrect-share* entity for the above protocol is described by a security game Game_{ISE} between a challenger and an adversary. The phases of the game are the following:

- *Handshake:* The challenger acts as honest secret owner and participants in this phase.
- *Waiting phase:* The challenger acts as honest participants in this step.
- *Reveal phase:* During the Reveal phase, the challenger and the adversary interact as follows:
  - *Collude query*: The adversary sends a Collude query along with a deposit $\mathtt{coins}(M)$ to the challenger, where M equals to the number of coins that a participant needs to join a secret share game. The challenger sends the secret owner and all participants' share and internal random coins to the adversary, excepting for one participant P.
  - *Challenge*: The adversary submits a `claim` transaction with an incorrect share, which aims to claim P's deposit and the corresponding reward.

We define that the adversary wins the Game_{ISE} game if the "net profit(s)" of the adversary is greater than 0. That is, the adversary can claim his (or other participant's) deposit if it wins the Game_{ISE} game. The proposed protocol in Section 4 is secure against *incorrect-share* entity if no PPT adversary can win the Game_{ISE} game.

Lemma 3 *If l-Strong Diffie-Hellman assumption holds and* HE *is IND-CPA secure, the proposed protocol in Section 4 is secure against* incorrect-share *entity.*

The proof of this lemma is given in Appendix A.3.

5.4 Unclaimable Shares

For economic benefit, a malicious secret owner may issue fake shares to participants. The proposed protocol provides a mechanism that ensures the security against such malicious secret owner. We formalize its security below. Specifically, the security against *malicious* secret owner for the above protocol is described by a security game Game_{MSO} between a challenger and an adversary. The game proceeds as follows.

- *Handshake:*
  1. (*Owner's hello*) The adversary acts as a malicious secret owner in this step.
  2. (*Participant's hello*) The challenger acts as honest participants in this step.

Table 1: Comparison of price for different method [1]

Method	Effective Key Size	Gas Consumption	Price in Ether	USD Price
MM	56	600,000	0.003	1.35
ACE	128	480,000	0.0024	1.08
ECCC	128	40,000	0.0002	0.09

[1] MM stands for Montgomery Multiplication, ACE stands for Assembly Code Exponentiation, and ECCC stands for ECC Cryptography on alt_bn128.

3. (*Owner's confirmation*) The adversary acts as a malicious secret owner in this step.
4. (*Participant's confirmation*) The challenger acts as honest participants in this step.

We define that the adversary wins Game_{MSO} if the participants accept the fake shares issued by the adversary during the Handshake phase.

Lemma 4 *If computing discrete log is hard in G and* HE *is IND-CPA secure, the proposed protocol in Section 4 is secure against malicious secret owner.*

The proof of this lemma is given in Appendix A.4.

6 Implementation

6.1 Prototype Implementation

We implemented a prototype of the proposed smart contract in Solidity [2], and then compiled it into EVM. The proposed smart contract is straight forward to implement, except for the exponentiation function. Even though Ethereum includes a built-in contract that supports large number exponentiation[6], it lacks high-level language support. We sidestep this issue by implementing a wrapper for the exponentiation function using assembly code shown in Listing 1.

```
function largeExpMod(bytes base, bytes expn, bytes modn) public returns (bytes
    result){
      // ... details hidden for simplicity
      // setup the input array to the correct format
      bytes memory inputArray = new bytes(totalInputLength);
      assembly{
          inputPtr := add(inputArray, 0x20) // start of the array value
          mstore(inputPtr, lenBase)
          mstore(add(inputPtr, 32), lenExp)
          mstore(add(inputPtr, 64), lenMod)
      }
      uint tempLoc = 96;
      uint value;
      // copy the base to the memory
      for(i = tempLoc; i < lenBase + tempLoc; i++){
        value = uint(base[i - tempLoc]);
        assembly{
          mstore8(add(inputPtr, i), value)
        }
      }
      // copy other values also ...
      // a function call to the EVM code for exponentiation
      assembly{
```

[6]This is available in the Byzantium upgrade.

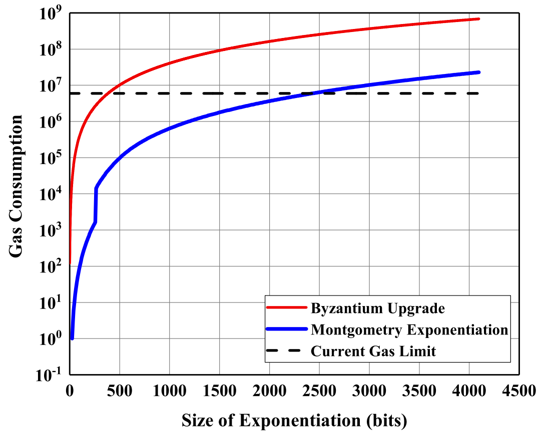


Figure 3: Comparison of the gas consumption by different implementation of large number exponentiation.

```
        let res := call(costExp, 0x05, 0, inputPtr, mul(totalInputLength, 0x20
            ), add(result,0x20), lenMod)
    }}
```

Listing 1: The code snippet for exponentiation

Optimizing share verification. In order to verify the validity of each participant's share, the smart contract has to perform $\mathcal{O}(t)$ exponentiations (i.e., the threshold of the secret sharing scheme). Recall that each operation that the smart contract executes consumes a certain amount of gas. Given the current Ethereum's block gas limit of 8 million[7], each function call can perform no more than 20 exponentiations. (Figure 1 plots gas consumption of different implementation of the large number exponentiation). In order to overcome this problem, we have derived a zero-knowledge proof system that reduces the number of necessary checks per each share to *one*, regardless of the threshold of the underlying secret sharing scheme. This not only overcome the block gas limit issues, but also reduces the transaction fee by a factor of t.

The zero-knowledge proof effectively proves that the B_i values published by the participants follows the specified relation. As a result, in the claim phase, the contract only needs to check whether $g^b = B_1$.

- Participant chooses a random number n and send to owner $T_1 = g^n, T_2 = B_1^n, ..., T_t = B_{t-1}^n$;
- The secret owner chooses a random c and send to participant c;
- Participant sends over $bc + n$;
- Owner verifies: $g^s = B_1^c T_1, B_1^s = B_2^c T_2, ..., B_{t-1}^s = B_t^c T_t$;
- If owner verifies, he proceeds with sending a share to the participant, otherwise he aborts and the participants withdraws the deposit back.

Notice that the verification can be done off-chain to reduce transaction fee incurred by on-chain execution, but at a cost of anonymity. A detailed proof of the correctness and zero-knowledge properties of the proof system are included in Appendix A.5.

We also note that one could implement the zero-knowledge proof and the commitment scheme on **alt_bn128** elliptical curve. Implemented in elliptic curve groups, the required

[7]`https://etherscan.io/blocks`.

exponentiations essentially reduced to multiplications, whose gas consumption is as low as 40,000 per operation. At the time of writing, Ethereum averaged USD450 per ETH, and the recommended gas price is 5Gwei. We summarize the price (in both ETH and USD) of different method in Table 1.

6.2 Performance Extensions

In this section, we discuss two performance extensions that potentially improve efficiency of the proposed protocol.

Coordinating the choice of participants: Our protocol abstracts away from mechanism by which secret owner discovers and recruits participants to take part in the safe keeping of his secret. Our security guarantees hold regardless of such mechanism. To enable a scalable and performant system, we propose to introduce an extra entity called *broker*. The broker facilitates participant discovery, load balancing, and random assignment of participants to secret safe keeping tasks.

The implementation of the broker is critical to the security of the underlying secret sharing scheme. In particular, if an adversary can bias the assignment of participants to secret safe keeping tasks, he could subvert the secret confidentiality by gaining control over more than t of its shares. Fortunately, various techniques exist to ensure randomized and fair assignments, including the use of cryptographic protocols [30, 25] or trusted execution environment [17].

Leveraging Micro Payment Channel: Another avenue for improvement is a mechanism by which deposits, rewards and bounties are transacted. Recall that each transaction on the blockchain incurs a fee. If the protocol requires a large number of micro-transactions (i.e., transactions that carry micro payments) to transacts the deposits, rewards, and bounties, it would incur unnecessary high transaction fee and overload the blockchain. We propose to employ payment channels [38, 3, 27] to address this issue. In particular, payment channel enables two parties to securely transact a large number of micro payments with minimal on-chain transaction. The two parties first establish a channel by depositing a maximum amount of value that they wish to be transacted into an escrow on the blockchain. Subsequently, they can transact micro payments by issuing digitally signed and hash-locked transfers, called payment promises, that are fully collateralized by the on-chain deposit. The parties can (unilaterally) decide to close the channel at anytime, settling the payment promises transacted thus far using a single on-chain transaction. The payments can be routed through the broker (discussed previously) so as to reduce the number of channels need to be established in the entire system.

7 Related Works

May [36] introduced the notion of *timed-release cryptography*. Later, the notion of *time-lock puzzles* was introduced by Rivest, Shamir, and Wagner [39] to further explore this problem. Generally, there are two main techniques that served as the solution to the problem of sending information into the future. One is mainly based on the time-lock puzzles [39, 4, 35, 7, 19, 20], where the receiver must perform expensive, non-stop computation to solve the relative time problem and obtain a message. There is no trusted server involved in this approach, but the receiver has to invest in a significant computational effort before recovering the message. Recently, Bitansky *et al.* [5] showed that various flavors of randomized encodings give rise to time-lock puzzles of varying strengths, and instantiated the construction with different randomized encodings from the literature. The other one employs a trusted server [16, 6, 11, 28, 9, 37], which is used to provide time reference to synchronize senders and receivers. Chalkias *et at.* [10] improved upon approaches introduced by [11, 9] by reducing the number of bilinear pairings as well as by enabling additional pre-computations. Chow *et al.* [12] proposed an encryption scheme in the standard model which can be used as timed-release encryption, where only confidentiality is guaranteed. Chow *et al.* [13] later demonstrated how to support pre-open capability, which is often desirable in applications of timed-release encryption. In the context of secret sharing, Watanabe *et al.* [41] proposed a construction of timed-release computational secret sharing, which is based on identity-based key encapsulation mechanism (IB-KEM). However, in most of current works, the trusted server needs to have lots of interaction with senders and receivers. It is desirable to reduce the interactions to a ideal level. Recently, several works [29, 31, 32] showed how to build time-release encryption from bitcoin and witness encryption. However, the efficiency is the main concern of of these works. Indeed, they are impractical since they cannot be deployed in the real bitcoin network efficiently.

8 Conclusion

In this paper, we have proposed a practical carrots-and-sticks approach that implements a time-release protocol. Our protocol leverages threshold secret sharing scheme to delegate safe keeping of time-release secrets to a large number of participants, and incorporates incentive-based mechanism to encourage honest participation while deterring misbehaviour by dishonest parties. The protocol attains timing control based on regular heartbeats emitted by a blockchain, and enforces fulfillment of involved parties' contractual obligations using a smart contract. Although our study abstracts away from subtle protocol configuration details such as deposit and reward compositions, bounties structures, and potential role of brokerage, we remark that their dispositions are critical to ensure incentive-compatibility and sustainability of the system. It would be an interesting future work to address these elements from economics and game theory viewpoints.

References

[1] Ethereum Foundation. The serpent contract-oriented programming language. `https://github.com/ethereum/serpent`.

[2] Ethereum Foundation. The solidity contract-oriented programming language. `https://github.com/ethereum/solidity`.

[3] Raiden network. `http://raiden.network`.

[4] Mihir Bellare and Shafi Goldwasser. Encapsulated key escrow, 1996.

[5] Nir Bitansky, Shafi Goldwasser, Abhishek Jain, Omer Paneth, Vinod Vaikuntanathan, and Brent Waters. Time-lock puzzles from randomized encodings. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, 2016*, pages 345–356, 2016.

[6] Dan Boneh and Matt Franklin. Identity-based encryption from the weil pairing. In *Advances in Cryptology–CRYPTO 2001*, pages 213–229. Springer, 2001.

[7] Dan Boneh and Moni Naor. Timed commitments. In *Advances in Cryptology–Crypto 2000*, pages 236–254. Springer, 2000.

[8] Vitalik Buterin. Ethereum: A next-generation smart contract and decentralized application platform. *https://github.com/ethereum/wiki/wiki/White-Paper*, 2014.

[9] Julien Cathalo, Benoît Libert, and Jean-Jacques Quisquater. Efficient and non-interactive timed-release encryption. In *Information and Communications Security, 7th International Conference, ICICS 2005, Proceedings*, pages 291–303, 2005.

[10] Konstantinos Chalkias, Dimitrios Hristu-Varsakelis, and George Stephanides. Improved anonymous timed-release encryption. In *Computer Security - ESORICS 2007, 12th European Symposium On Research In Computer Security, 2007, Proceedings*, pages 311–326, 2007.

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

[12] Sherman S. M. Chow. Token-controlled public key encryption in the standard model. In *Information Security, 10th International Conference, ISC 2007, 2007, Proceedings*, pages 315–332, 2007.

[13] Sherman S. M. Chow and Siu-Ming Yiu. Timed-release encryption revisited. In *Provable Security, Second International Conference, ProvSec 2008. Proceedings*, pages 38–51, 2008.

[14] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Practical multilinear maps over the integers. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, 2013. Proceedings, Part I*, pages 476–493, 2013.

[15] Jean-Sébastien Coron, Tancrede Lepoint, and Mehdi Tibouchi. New multilinear maps over the integers. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, 2015, Proceedings, Part I*, pages 267–286, 2015.

[16] Giovanni Di Crescenzo, Rafail Ostrovsky, and Sivaramakrishnan Rajagopalan. Conditional oblivious transfer and timed-release encryption. In *Advances in Cryptology - EUROCRYPT '99*, pages 74–89, 1999.

[17] Hung Dang, Anh Dinh, Ee-Chien Chang, and Beng Chin Ooi. Chain of trust: Can trusted hardware help scaling blockchains? *arXiv preprint arXiv:1804.00399*, 2018.

[18] Tien Tuan Anh Dinh, Ji Wang, Gang Chen, Rui Liu, Beng Chin Ooi, and Kian-Lee Tan. Blockbench: A framework for analyzing private blockchains. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1085–1100. ACM, 2017.

[19] Juan A. Garay and Markus Jakobsson. Timed release of standard digital signatures. In *Financial Cryptography, 6th International Conference, FC 2002*, pages 168–182, 2002.

[20] Juan A. Garay and Carl Pomerance. Timed fair exchange of standard signatures: [extended abstract]. In *Financial Cryptography, 7th International Conference, FC 2003*, pages 190–207, 2003.

[21] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, 2013. Proceedings*, pages 1–17, 2013.

[22] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In *Symposium on Theory of Computing Conference, STOC'13, June 1-4, 2013*, pages 467–476, 2013.

[23] Craig Gentry, Sergey Gorbunov, and Shai Halevi. Graph-induced multilinear maps from lattices. In *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, 2015, Proceedings, Part II*, pages 498–527, 2015.

[24] Craig Gentry, Allison B. Lewko, and Brent Waters. Witness encryption from instance independent assumptions. In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, August 17-21, 2014, Proceedings, Part I*, pages 426–443, 2014.

[25] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 51–68. ACM, 2017.

[26] Oded Goldreich. Secure multi-party computation. *Manuscript. Preliminary version*, 78, 1998.

[27] Matthew Green and Ian Miers. Bolt: Anonymous payment channels for decentralized currencies. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 473–489. ACM, 2017.

[28] Yong Ho Hwang, Dae Hyun Yum, and Pil Joong Lee. Timed-release encryption with pre-open capability and its application to certified e-mail system. In *Information Security, 8th International Conference, ISC 2005, Proceedings*, pages 344–358, 2005.

[29] Tibor Jager. How to build time-lock encryption. *IACR Cryptology ePrint Archive*, 2015:478, 2015.

[30] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, and Bryan Ford. Omniledger: A secure, scale-out, decentralized ledger. *IACR Cryptology ePrint Archive*, 2017:406, 2017.

[31] Jia Liu, Flavio Garcia, and Mark Ryan. Time-release protocol from bitcoin and witness encryption for sat. *IACR Cryptology ePrint Archive*, 2015:482, 2015.

[32] Jia Liu, Saqib A Kakvi, and Bogdan Warinschi. Extractable witness encryption and timed-release encryption from bitcoin. Technical report, Cryptology ePrint Archive, Report 2015/482, 2015. http://eprint. iacr. org/2015/482, 2015.

[33] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. Making smart contracts smarter. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 254–269. ACM, 2016.

[34] Loi Luu, Jason Teutsch, Raghav Kulkarni, and Prateek Saxena. Demystifying incentives in the consensus computer. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 706–719. ACM, 2015.

[35] Wenbo Mao. Timed-release cryptography. In *Selected Areas in Cryptography*, pages 342–357. Springer, 2001.

[36] Timothy C. May. Timed-release crypto. *http://www.hks.net/cpunks/cpunks-0/1460.html*, 1993.

[37] Kenneth G. Paterson and Elizabeth A. Quaglia. Time-specific encryption. In *Security and Cryptography for Networks, 7th International Conference, SCN 2010. Proceedings*, pages 1–16, 2010.

[38] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments. 2016.

[39] Ronald L Rivest, Adi Shamir, and David A Wagner. Time-lock puzzles and timed-release crypto. *Massachusetts Institute of Technology*, 1996.

[40] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[41] Yohei Watanabe and Junji Shikata. Timed-release computational secret sharing scheme and its applications. In *Provable Security - 8th International Conference, ProvSec 2014. Proceedings*, pages 326–333, 2014.

A Proof sketch

A.1 Proof of Lemma 1

To prove this lemma, we will assume that there exists a PPT adversary $\mathcal{A}$ that colludes with $t-1$ participants after the Handshake phase, which can win the Game_{SC} game.

Given g^α and g^β, we will make use of $\mathcal{A}$ to obtain $g^{\alpha\beta}$, where α, β are random. Let $h = g^\alpha$ and $R_0 = g^\beta$. Let $(b_1, c_1), ..., (b_{t-1}, c_{t-1})$ be the secrets of the colluded participants. This allow us to directly compute $S_{1,i}, S_{2,i}$ for $i = 1, ..., t-1$. We then randomly choose (b'_i, c'_i) for $i = t, ..., n$, and sets $\{B_{i,j} = g^{(b'_i)^j}, C_{i,j} = h^{c'_i (b'_i)^j}\}_{j\in[1,t-1]}, C = g^{c'_i}$. Since HE is IND-CPA secure, $E_{i,j}$ can be simulated using dummy messages. The complete view of the protocol is now defined (with the right distribution), which is consistent with the private view of participants P_i for $i = 1, ..., t-1$. Suppose that they are able to calculate $h^{f_s(0)}$. Since $h = g^\alpha$ and $R_0 = g^\beta$, we are able to obtain $g^{\alpha\beta}$. This contradicts the computational Diffie-Hellman assumption.

A.2 Proof of Lemma 2

Claim 1: Suppose $\mathbb{I}_{(t,n)}$ is the Shamir's secret sharing scheme used in the proposed protocol in Section 4, Z is a premature release outputted by $\mathcal{A}$, if there exists a PPT algorithm that takes Z, $t-1$ valid shares and public information can recover the secret of $\mathbb{I}_{(t,n)}$, then there exists a PPT algorithm that takes Z (with additional inputs) can output the underlying share(s) that $\mathcal{A}$ takes.

Proof: Suppose there exists a PPT algorithm $\mathcal{Z}$ that takes Z, $t-1$ valid shares and public information can recover the secret of $\mathbb{I}_{(t,n)}$, then we have that there exists a non-uniform PPT algorithm $\mathcal{B}$ that takes Z, $t-1$ valid shares and public information can output a curve f, where f is a curve defined by the $t-1$ valid shares and the underlying point(s) of Z. Define Procedure (1) as:

(1) Randomly chooses $t-1$ sample points. Feed $\mathcal{B}$ with Z and the chosen $t-1$ points, and obtain the secret s_1 whose value is $f_1(0)$ where f_1 is the polynomial fitting the randomly chosen points and s_1.

We then construct an extractor which works as follows:

- Run Procedure (1) repeatedly and obtain $f_2, ..., f_d$ ($d \geq 2$), until there is only one common intersection point(s) among the curves $f, f_1, ..., f_d$.

We conclude that common intersection point(s) must be the underlying point(s) of Z. That is, there exists an extractor that can extract the underlying point(s) of Z.

From Claim 1 and Lemma 1, we have that any $\mathcal{A}$ cannot win the the Game_{DPR} game.

A.3 Proof of Lemma 3

To prove this lemma, we will assume that there exists a PPT adversary $\mathcal{A}$ that colludes with the secret owner and $n-1$ participants after the Waiting phase (excepting for one participant P), which can win the Game_{ISP} game. Using this adversary we will build a PPT simulator $\mathcal{B}$ that takes in the l-SDH challenge input $g, D_1 = g^b, D_2 = g^{b^2}, ..., D_{t-1} = g^{b^{t-1}}$ can attack the l-SDH assumption.

- *Handshake:*
    1. (*Owner's hello*) $\mathcal{B}$ chooses random $\alpha, r_1, ..., r_{t-1} \in \mathbb{Z}_p$ and sets $h = g^\alpha$. Next, $\mathcal{B}$ submits a transaction:

$$\left\langle \texttt{Ohello}, \{R_i = g^{r_i}\}_{i\in[0,t-1]}, \texttt{coins}(M_o) \right\rangle,$$

where $f_s()$ is the polynomial

$$f_s(x) = \sum_{i=0}^{t-1} r_i x^i$$

with $r_0 = s$.

2. (*Participant's hello*)
For each $i \in [n-1]$, $\mathcal{B}$ does as follows:
It first chooses a random $b_i, c_i \in \mathbb{Z}_p$ and computes
$$\{B_{i,j} = g^{b_i^j}, C_{i,j} = h^{c_i b_i^j}\}_{j\in[1,t-1]}, C = g^{c_i}.$$
It then initializes an instance of the additive homomorphic encryption scheme HE, where pk_i is the public parameters of HE. Next, it computes
$$\{E_{i,j} = \mathsf{HE.Enc}(b_i^j)\}_{j\in[1,t-1]}$$
where $\mathsf{HE.Enc}()$ is the encryption algorithm of HE. Finally, it submits the transaction:
$$\langle \texttt{Phello}, \mathsf{pk}_i, \{B_{i,j}, C_{i,j}, E_{i,j}\}_{j\in[1,t-1]}, \\ C, \texttt{coins}(M)\rangle$$
For the non-colluded participant P, $\mathcal{B}$ chooses random $c^* \in \mathbb{Z}_p$ and sets
$$\{B_j^* = D_j, C_j^* = D_j^{c^* h'}\}_{j\in[1,t-1]}, C^* = g^{c^*}.$$
$\mathcal{B}$ then chooses random $b^* \in \mathbb{Z}_p$ and initializes an instance of the additive homomorphic encryption scheme HE, where pk^* is the public parameters of HE. Next, it computes
$$\{E_j^* = \mathsf{HE.Enc}((b^*)^j)\}_{j\in[1,t-1]}$$
Finally, it submits the transaction:
$$\langle \texttt{Phello}, \mathsf{pk}^*, \{B_j^*, C_j^*, E_j^*\}_{j\in[1,t-1]}, \\ C^*, \texttt{coins}(M)\rangle$$

3. (*Owner's confirmation*) For each $i \in [n-1]$, $\mathcal{B}$ does as follows:
It computes
$$\begin{aligned} S_{1,i} &= g^s \prod_{i=1}^{t-1} (B_{i,j})^{r_j}, \\ S_{2,i} &= C^{\alpha s} \prod_{i=1}^{t-1} (C_{i,j})^{r_j}, \\ T_i &= \mathsf{HE.Enc}(s) + \prod_{j=1}^{t-1} r_j E_{i,j} \end{aligned}$$
and submits the transaction:
$$\langle \texttt{Oconfirm}, S_{1,i}, S_{2,i}, T_i\rangle.$$
For the non-colluded participant P, $\mathcal{B}$ computes
$$\begin{aligned} S_1^* &= g^s \prod_{i=1}^{t-1} (B_j^*)^{r_j}, \\ S_2^* &= g^s \prod_{i=1}^{t-1} (C_j^*)^{r_j}, \\ T^* &= \mathsf{HE.Enc}(s) + \prod_{j=1}^{t-1} (E_j^*)^{r_j} \end{aligned}$$

and submits the transaction:

$$\langle \texttt{Oconfirm}, S_1^*, S_2^*, T^* \rangle.$$

4. (*Participant's confirmation*) $\mathcal{B}$ does not submit any transaction.

- *Waiting phase:* During the Waiting phase, $\mathcal{B}$ behaviors honestly. No `reportHello` and `report` transactions are submitted.
- *Reveal phase:* $\mathcal{B}$ and $\mathcal{A}$ interact as follows:
  - *Collude query*: $\mathcal{A}$ sends a $\mathsf{Collude}$ query with a deposit `coins(M)` to $\mathcal{B}$. $\mathcal{B}$ sends the secret owner and all participants' share and internal random coins to $\mathcal{A}$, excepting for the participant P.
  - *Challenge*: $\mathcal{A}$ submits the following transaction:

    $$\langle \texttt{claim}, (b_i', f_s'(b_i')) \rangle$$

    which aims to claim the deposit and (the corresponding) reward of P.

If $\mathcal{A}$ can obtain a deposit, then we have that $g^{f_s'(b_i')} = S^* = g^{f_s(b)}$. Hence, $\mathcal{B}$ can obtain the value of b and further compute g^{b^t}, breaking the l-SDH assumption.

A.4 Proof of Lemma 4

To prove this lemma, we will assume that there exists a PPT adversary $\mathcal{A}$ that controls the secret owner, which can win the Game_{MSO} game. Using this adversary we will build a PPT simulator $\mathcal{B}$ that can compute discrete log in G.

- *Handshake:*
  1. (*Owner's hello*) $\mathcal{A}$ chooses random $r_1, ..., r_{t-1} \in \mathbb{Z}_p$. Next, $\mathcal{A}$ submits a transaction:

     $$\left\langle \texttt{Ohello}, \{R_i = g^{r_i}\}_{i\in[0,t-1]}, \texttt{coins}(M_o) \right\rangle,$$

     where $f_s()$ is the polynomial

     $$f_s(x) = \sum_{i=0}^{t-1} r_i x^i$$

     with $r_0 = s$.

  2. (*Participant's hello*) For each $i \in [n]$, $\mathcal{B}$ does as follows:
     It first chooses a random $b_i, c_i \in \mathbb{Z}_p$ and computes

     $$\{B_{i,j} = g^{b_i^j}, C_{i,j} = h^{c_i b_i^j}\}_{j\in[1,t-1]}, C = g^{c_i}.$$

     It then initializes an instance of the additive homomorphic encryption scheme HE, where pk_i is the public parameters of HE. Next, it computes

     $$\{E_{i,j} = \mathsf{HE.Enc}(b_i^j)\}_{j\in[1,t-1]}$$

     where $\mathsf{HE.Enc}()$ is the encryption algorithm of HE. Finally, it submits the transaction:

     $$\langle \texttt{Phello}, \mathsf{pk}_i, \{B_{i,j}, C_{i,j}, E_{i,j}\}_{j\in[1,t-1]},$$
     $$C, \texttt{coins}(M) \rangle$$

3. (*Owner's confirmation*) For each $i \in [n]$, $\mathcal{A}$ randomly chooses $S_{1,i}, S_{2,i}, T_i$ and submits the transaction:

$$\langle \texttt{Oconfirm}, S_{1,i}, S_{2,i}, T_i \rangle.$$

4. (*Participant's confirmation*) For each $i \in [n]$, $\mathcal{B}$ verifies the value S_i, T_i submitted by $\mathcal{A}$, and submit a transaction `Pwithdraw` if otherwise. Using the corresponding secret key of pk_i, $\mathcal{B}$ checks whether the following two equalities both holds:

$$\begin{aligned} g^{\mathsf{HE.Dec}(T_i)} &\overset{?}{=} S_{1,i}, \\ h^{\mathsf{HE.Dec}(T_i)} &\overset{?}{=} S_{2,i}, \quad \text{and} \\ \prod_{j=0}^{t-1} (R_j)^{b_i^j} &\overset{?}{=} S_{1,i} \end{aligned}$$

If not, $\mathcal{B}$ submits the transaction

$$\langle \texttt{Pwithdraw} \rangle$$

to withdraw. In response to this transaction, the smart contract returns the deposit back to $\mathcal{B}$. If both equalities hold, $\mathcal{B}$ does not submit any transaction.

Note that if a fake share is accepted by a participant, we have

$$g^{\mathsf{HE.Dec}(T_i)} = S_{1,i} = \prod_{j=0}^{t-1} (R_j)^{b_i^j}$$

Hence, $\mathcal{B}$ can compute the discrete log of $S_{1,i}$.

A.5 Zero-Knowledge Proof

For the zero-knowledge proof involved in the protocol, we need to prove three properties: Soundness, Completeness and (honest-verifier) Zero-knowledge.

- *Soundness*: We need to prove that if the $B_1, B_2, ..., B_{t-1}$ are indeed related in such a way, and the prover (i.e., the participant) honestly follows the protocol, the verifier (i.e, the secret owner) will verify his result. With the setting of the protocol,

$$B_i^s = g^{(b^i)(bc+n)} = g^{(b^{i+1}c + b^i n)} = B_{i+1}^c T_i.$$

- *Completeness*: We need to prove that if $B_1, B_2, ..., B_{t-1}$ are not related in the way stipulated by the protocol, with a large probability, the prover (i.e., the participant) cannot be verified. We do a proof by induction. WOLG, denote $\log_g B_1 = g$ and $\log_g T_1 = n$, and assume that the prover follows the protocol for $B_1, ..., B_{i-1}$, but violate the protocol at the index i, which means either $B_i \neq g^{b^i}$. Notice that the relation $s = bc + n$ must be true, otherwise the first equation cannot be verified.
  Let us denote $B_i = g^x$ and $T_i = g^y$, then the equation:

$$B_{i-1}^s = B_i^c T_i$$

  must be satisfied. If we expand the equation, we get

$$g^{b^{i-1}(bc+n)} = g^{xc+y},$$

and we have a solution:

$$(b^i - x)c = y - nb^{i-1},$$

which has a single solution for c if $x \neq b^i$. Hence, the probability that the prover successfully tricks the protocol with $B_1, B_2, ..., B_{t-1}$ with different relation is no more than $|G|^{-1}$, which is very low given a large group.

- *Zero-knowledge*: We show that there is a simulator generating a interactive proof procedure without the knowledge of b and with random $B_1, B_2, ..., B_n$. He first choose random c, s, and then generate $T_1 = g^s B_1^{-c}, T_2 = B_1^s B_2^{-c}, ...$, and it is easy to verify that the equations are all satisfied. To see the verification is indistinguishable from a true interactive procedure, observe that by the extended diffie-hellman assumption, $T_1, T_2, ..., T_t$ are indistinguishable from random elements in the group G.