I *Told* You Tomorrow: Practical Time-Locked Secrets using Smart Contracts

Enrico Bacis[*]
enrico.bacis@unibg.it
Università degli Studi di Bergamo
Italy

Dario Facchinetti
dario.facchinetti@unibg.it
Università degli Studi di Bergamo
Italy

Marco Guarnieri
marco.guarnieri@imdea.org
IMDEA Software Institute
Spain

Marco Rosa
marco.rosa@sap.com
SAP Security Research
France

Matthew Rossi
matthew.rossi@unibg.it
Università degli Studi di Bergamo
Italy

Stefano Paraboschi
parabosc@unibg.it
Università degli Studi di Bergamo
Italy

ABSTRACT

A Time-Lock enables the release of a secret at a future point in time. Many approaches implement Time-Locks as cryptographic puzzles, binding the recovery of the secret to the solution of the puzzle. Since the time required to find the puzzle's solution may vary due to a multitude of factors, including the computational effort spent, these solutions may not suit all scenarios.

To overcome this limitation, we propose I Told You Tomorrow (ITYT), a novel way of implementing time-locked secrets based on smart contracts. ITYT relies on the blockchain to measure the elapse of time, and it combines threshold cryptography with economic incentives and penalties to replace cryptographic puzzles.

We implement a prototype of ITYT on top of the Ethereum blockchain. The prototype leverages secure Multi-Party Computation to avoid any single point of trust. We also analyze resiliency to attacks with the help of economic game theory, in the context of rational adversaries. The experiments demonstrate the low cost and limited resource consumption associated with our approach.

CCS CONCEPTS

• **Security and privacy** → Security services.

KEYWORDS

time-locks, smart contracts, secure multi-party computation, threshold cryptography, rational adversaries

ACM Reference Format:
Enrico Bacis, Dario Facchinetti, Marco Guarnieri, Marco Rosa, Matthew Rossi, and Stefano Paraboschi. 2021. I *Told* You Tomorrow: Practical Time-Locked Secrets using Smart Contracts. In *The 16th International Conference on Availability, Reliability and Security (ARES 2021), August 17–20, 2021, Vienna, Austria.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3465481.3465765

[*] now at Google

ARES 2021, August 17–20, 2021, Vienna, Austria
© 2021 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9051-4/21/08.
https://doi.org/10.1145/3465481.3465765

1 INTRODUCTION

Many real-world scenarios require disclosing a secret at a specific future point in time. For instance, this happens when we vote for elections or when we dispose our inheritance by will. In these circumstances we typically entrust a notary to keep a secret private until a future time, and then to publish it so that we are no longer needed for disclosure to happen. This however requires that the owner of the secret completely entrusts a single third-party (i.e., the notary).

Early proposals bound the recovery of the secret to the recovery of a key [43], which was split among a number of peers, distributing trust among many parties instead of entrusting a single one.

To completely remove the dependence on one or more trusted third-parties, cryptographers have been working on *timed-release cryptography* [38] proposing schemes that effectively replace notaries with Time-Locks (TL). Time-Lock *puzzles* [7, 34, 45] bind the recovery of the secret to the solution of a cryptographic puzzle. In this setting, a sending party (i.e., the secret's owner) can encrypt the secret so that the receiving party is required to perform multiple decryptions to recover it. Given the assumption that each decryption round takes the same amount of time, and that the number of rounds can be tuned by the sender during the encryption process, it is possible to protect a secret for arbitrarily long periods of time.

The first puzzle proposed by Rivest et al. [45] uses a trapdoor function so that anyone willing to recover the secret had to undergo a computing effort that was orders of magnitude larger compared to the one of the sending party. Also, since the authors imposed the decryption process to be executed in an inherently sequential order, their approach is the first example of a *Proof of Sequential Work* (PoSW) [14, 35].

An alternative to trapdoor functions is using *weak hash-chains* [8], by requiring anyone willing to obtain the secret to brute-force a chain of weak hashes. The use of a chain, rather than a single stronger hash, permits to reduce the variance of the decryption runtime. However, the decryption process remains parallelizable and, thus, the estimated disclosure time far less reliable. These approaches are often classified as *Proof of Work* (PoW) [20] algorithms.

Cryptographic puzzles avoid the need for a trusted party, yet two aspects make them impractical. First, the sending party has to make assumptions on future computing power. This is far from trivial. As an example, Rivest's LCS35 time capsule [44], released in 1999 with an estimated decryption time of 35 years, was opened in 2019 after a decryption process of only 2 months on dedicated hardware [15]. Second, TL puzzles require the receiving party to run the decryption procedure for a long time, which poses a question about economic incentives.

The development of blockchains gives us new opportunities to implement time-locks. Indeed, a blockchain intrinsically defines the concept of time, which was one of the main reasons that led to the creation of cryptographic puzzles. It can also be used to persistently disclose secrets, as it is designed to resist modification of its data. However, the blockchain alone is not enough to deploy a TL because it does not offer any mechanism to keep information confidential. Recent proposals address the problem of dealing with secret data on public blockchains [1, 27] by splitting information among multiple users who have to cooperate to recover the secret information using a pre-defined protocol. The protocol is often programmed as a smart contract [47], which permits to verify the correctness of its outcome and to detect any undefined behavior. Although this approach could resemble the early implementations of timed-release cryptography, the security no longer follows from a single strong trust assumption (i.e., the trust to the notary or to a certification authority) but from the behavior of the users that take part to the protocol. If the users cooperate as intended, the outcome of the protocol will be successful, the secret will be recovered, and the TL function achieved as a consequence. To ensure that users cooperate as intended, several proposals (e.g., [30, 33]) reward them with economic incentives. This permits to analyze the protocols as extended form games whose outcome can be determined based on participants' expected utility.

In this paper we propose I Told You Tomorrow (ITYT), a practical and generic framework to implement time-locks using smart contracts that is based neither on trust assumptions nor on cryptographic puzzles. The basic idea of our approach is to first split a secret into shares using *threshold cryptography* (using Shamir's Secret Sharing [46]), and assign them to users so that no one can recover the secret unless *k-of-n* shares are available. To ensure the TL behavior, we rely on economic incentives and penalties enforced by a smart contract. The contract rewards users for revealing their share only after the disclosure time and penalizes any other misbehavior. As rewards and penalties are associated with the correct management of the shares, ITYT leverages *secure Multi-Party Computation* (sMPC) [50], which ensures confidentiality and avoids the need for trusted users (including the owner of the secret).

Here we summarize our main contributions. We define a protocol that deploys Time-Locked secrets on the blockchain by leveraging an economic model in which every user (or coalition of them) has an expected negative payoff associated with possible misbehavior. We address the problems that arise when combining secure Multi-Party Computation and protocols based on economic incentives and penalties (e.g., use secure Multi-Party Computation protocols to break the protocol bypassing the smart contract's hashlocks). We implemented the protocol based on the Ethereum blockchain [49] and the FRESCO secure Multi-Party Computation framework [16], characterized by low overhead and limited cost of execution. Finally, we compare ITYT with other existing solutions, discussing the key advantages associated with our approach.

2 BACKGROUND

We describe here a few concepts that are used in the design of the protocol.

Threshold cryptography. Threshold cryptography [46] enables the owner of a secret to share it among a group of users. In a *k-of-n* threshold scheme, *n* shares of the secret are created and distributed among the parties. To reconstruct the secret, at least *k* different shares have to be combined. Hence, the advantages of threshold cryptography are (i) distribution of trust, and (ii) fault tolerance.

Secure Multi-Party Computation. A *secure Multi-Party Computation (*sMPC) protocol [48, 50] is a cryptographic protocol that allows multiple parties to jointly compute a function over their inputs while keeping them private. Current sMPC frameworks can execute binary or modular arithmetic algorithms computed among several parties (even with dishonest majority) by leveraging *semi-homomorphic encryption* [17, 26] and *oblivious transfer* [25, 42].

Smart contracts. A blockchain is an append-only list of blocks linked together via cryptographic properties. The blocks are non-mutable and keep a permanent history of transactions. Smart contracts [47] are programming frameworks built on top of blockchains. A smart contract permits to program tamper-proof protocols whose outcome is verifiable by the whole network. We rely on smart contracts to pay incentives, trigger penalties, and enforce the correct execution of our scheme without relying on a trusted party. Specifically, our approach makes use of the *time* and *hash* primitives, to conditionally execute actions based on time and submitted data (e.g., reward users participating in a successful execution of the protocol at protocol termination time).

Rational adversaries. A malicious adversary [24] is someone who is willing to perform any action to attack a protocol. A rational adversary [22], instead, subverts the protocol only if it is economically convenient. Modeling the participants as rational enables the use of game theory concepts to analyze cryptographic protocols [5, 6, 9]. ITYT models each participant as rational.

3 THE ITYT PROTOCOL

In this section, we overview the ITYT protocol, introducing the preliminary definitions, the roles of the participants, and the main functions.

ITYT is an instance of the *TL abstraction*: a mechanism that keeps a secret $\mathcal{S}$ private until its *disclosure time* t_d and publishes it afterward. ITYT implements TL by splitting the secret, provided by the *owner*, among several users named *shareholders* (each obtaining a share h), so that none of them can recover $\mathcal{S}$ before the disclosure time. To ensure that (i) each user keeps its share secret until t_d, and (ii) each user publicly discloses its share immediately after t_d, ITYT introduces a set of economic incentives and penalties. Thus, the TL function is achieved as a consequence of the rational economic behavior of the involved parties (see Section 4).

3.1 Definitions

Principals. We denote by $\mathcal{U}$ the set of users that take part in an instance of ITYT. Additionally, we denote by $\mathcal{SC}$ the set of the smart contract identifiers. We then denote by $\mathcal{P}$ the set of principals consisting of $\mathcal{U} \cup \mathcal{SC}$.

Wallets. Each principal $p \in \mathcal{P}$ is associated with a wallet $wlt(p)$, accessible only by p, that can be used to receive or issue payments.

Protocol parameters. In the following table we report the definition of all the parameters that characterize an instance of ITYT.

$\mathcal{S}$	secret
V	economic value assigned to the secret
n	number of shareholders
k	number of shares needed to reconstruct $\mathcal{S}$
h_i	share of the secret issued to the i-th shareholder
t_d	disclosure time
t_{term}	termination time
F_O	fee deposited by the owner to use the service
$B_{\mathcal{H}}$	bid deposited by the shareholder to get a share
$R_{\mathcal{H}}$	reward paid to the shareholder in case of success
W_h	reward paid when whistleblowing a share
$W_{\mathcal{S}}$	reward paid when whistleblowing the secret

Smart contract state. The ITYT smart contract keeps track of the protocol status through the following data structures.

$[shares]$	array of shares submitted to the contract
$C_{\mathcal{S}}$	commitment of the secret (i.e., its hash)
$[C_h]$	array of share commitments
$state$	global state of the protocol
$[states]$	array of states for each share
$num\_pending$	count of the pending shares
$num\_disclosed$	count of the disclosed shares

Primitives. We now define the primitives used by the smart contract.

- `time()`: returns the current time as witnessed by the blockchain (generally defined in terms of block height).
- `hash`(d): returns the result of the application of a chosen cryptographic hash function over the data d.
- `pay`(p_1, p_2, v): transfers the amount v from $wlt(p_1)$ to $wlt(p_2)$.
- `initialize_sc`$([params])$: instantiates an ITYT smart contract, and deploys it to the blockchain. The primitive is executed by the owner and returns the smart contract identifier $sc \in \mathcal{SC}$.
- `generate_shares`$(\mathcal{S}, [users])$: generates shares $h_1, \ldots, h_n$ and securely distributes them to the parties. The primitive guarantees that (i) the i-th shareholder is the only principal who learns the share h_i, and (ii) the owner learns only the commitment $hash(h_i)$, for each share. We discuss in Section 5 how our prototype implements this primitive by leveraging sMPC and secret sharing.

3.2 Roles

In ITYT, each user $u \in \mathcal{U}$ plays one of the following roles: *owner*, *shareholder*, and *whistleblower*.

Owner. An owner O delegates the disclosure of a secret $\mathcal{S}$ to a timelock at disclosure time t_d. The owner O configures and deploys the TL providing all the required parameters. In particular, O sets (i) the total number n of shares h of $\mathcal{S}$, (ii) the number k of shares needed to recover $\mathcal{S}$, (iii) the disclosure time t_d, and (iv) all the bids and the rewards that define the instance.

Algorithm 1 Protocol initialization (executed by the owner).

```
    $[params]$                        protocol parameters of the ITYT instance
    $\mathcal{S}$                     secret
    $O$                               user identifier of the owner
    $[\mathcal{H}_1, \ldots, \mathcal{H}_n]$   list of shareholder identifiers
procedure INIT($[params], \mathcal{S}, O, [\mathcal{H}_1, \ldots, \mathcal{H}_n]$)
    $sc \leftarrow$ initialize_sc($[params]$)                    ▷ Create $sc$
    pay($O, sc, sc.F_O$)                           ▷ Transfer owner's fee to $sc$
    $C_h \leftarrow$ generate_shares($\mathcal{S}, [O, \mathcal{H}_1, \ldots, \mathcal{H}_n]$)
    $sc.C_{\mathcal{S}} \leftarrow$ hash($\mathcal{S}$)                       ▷ Set secret commitment
    $sc.state \leftarrow$ PENDING
    for $i \leftarrow 1, n$ do                              ▷ Set share commitments
        $sc.C_h[i] \leftarrow C_h[i]$
        $sc.states[i] \leftarrow$ PENDING
    end for
    $sc.num\_pending \leftarrow n$
    $sc.num\_disclosed \leftarrow 0$
    return $sc$                               ▷ The smart contract identifier
end procedure
```

Algorithm 2 Shareholder commitment to participate in ITYT

```
    $sc$      smart contract identifier
    $\mathcal{H}_i$      user identifier of the $i$-th shareholder
    precondition: $\mathcal{H}_i$ checks that hash($h_i$) $= sc.C_h[i]$
procedure PARTICIPATE($sc, \mathcal{H}_i$)
    if $sc.state =$ PENDING and $sc.states[i] =$ PENDING then
        pay($\mathcal{H}_i, sc, sc.B_{\mathcal{H}}$)
        $sc.states[i] \leftarrow$ BID
        $sc.num\_pending -= 1$
        if $sc.num\_pending = 0$ then
            $sc.state \leftarrow$ LOCKED
        end if
    end if
end procedure
```

Shareholder. A shareholder $\mathcal{H}$ is entrusted by the owner O to keep a share h of the secret $\mathcal{S}$ confidential until t_d, and to publicly disclose it afterward. In exchange for her service, $\mathcal{H}$ receives a reward $R_{\mathcal{H}}$ paid by the smart contract whenever the following two conditions hold: (i) her share h is disclosed only after t_d, and (ii) the secret $\mathcal{S}$ is not revealed before t_d.

Whistleblower. A whistleblower $\mathcal{W}$ reports user misbehavior in return for payment. Whenever $\mathcal{W}$ captures a share h or the secret $\mathcal{S}$ before t_d, $\mathcal{W}$ can submit it to the contract and receive a reward.

3.3 Setup

In the early stage of ITYT, the owner initializes and deploys a smart contract sc on the blockchain using the `initialize_sc()` primitive. Along with the protocol parameters, she writes to the sc the identifiers of all the shareholders, which the owner has selected beforehand[1], that will take part in the TL instance. She also deposits

[1] How to randomly choose competing players in adversarial settings such as blockchains has been addressed in many literature works (e.g., [19, 39])

Algorithm 3 SC function to whistleblow a share before t_d

```
$sc$     smart contract identifier
$h_i$    the $i$-th share to be whistleblown
procedure WhistleblowShare($sc, h_i$)
    if $sc.state$ = LOCKED and time() < $sc.t_d$ then
        if $sc.states[i]$ = BID and hash($h_i$) = $sc.C_h[i]$ then
            $sc.shares\ [i] \leftarrow h_i$
            $sc.states[i] \leftarrow$ WHISTLEBLOWN
            $sc.num\_disclosed$ + = 1
            if $sc.num\_disclosed = sc.k$ then
                $sc.state \leftarrow$ FAILED
            end if
            pay($sc, caller, sc.W_h$)
        end if
    end if
end procedure
```

to the contract the amount F_O, a fee that will be used to pay the rewards at protocol termination time t_{term}.

After the contract is deployed, the owner and the shareholders jointly execute the `generate_shares()` primitive. As a result, the owner gets the hash of the secret and the hash of each share (i.e., the commitments), while each shareholder $\mathcal{H}$ gets her share along with the hash of the secret. The owner submits to the *sc* contract the commitments (Algorithm 1). Then, each shareholder reads the contract and checks whether her commitment matches what received from `generate_shares()`. If so, she agrees and deposits her bid $B_{\mathcal{H}}$ (Algorithm 2).

As soon as each shareholder has committed, the TL is activated (i.e., the global state is set to `LOCKED`). If any party refuses to commit, the funds already deposited are returned to their proprietaries and the instance setup aborts (as discussed in Section 5).

The reader may have noticed that in this setup, the secret shares are exposed to the shareholders prior to the activation of the TL (i.e., the shares are distributed before the state is set to `LOCKED`). In Section 5 we show how to overcome this issue using a key instead of the actual secret. We also point out that the verification of protocol parameters (especially the economic ones) can be done by the shareholders at commit time, as their values are publicly available in the contract.

3.4 Actions

When the TL is active, the actions performed by users determine the status of the protocol. Each action is perfomed executing a smart contract function whose effects are public. Four actions are available to users: `WhistleblowShare`, `WhistleblowSecret`, `Disclose`, and `Withdraw`, with the last two reserved to shareholders.

`WhistleblowShare`: This action (Algorithm 3) enables the whistleblower $\mathcal{W}$ to report the misbehavior of a single shareholder. Whenever $\mathcal{W}$ obtains a share h_i, she submits it to collect a reward. If the commitment $sc.C_h[i]$ matches, the share whistleblow reward W_h is paid to the whistleblower and the shareholder $\mathcal{H}_i$ loses her reward $R_{\mathcal{H}}$. Additionally, if the number of whistleblown shares equals k, then the TL is marked as `FAILED` (i.e., no further actions allowed).

Algorithm 4 SC function to whistleblow the secret before t_d

```
$sc$     smart contract identifier
$\mathcal{S}$      the secret to be whistleblown
procedure WhistleblowSecret($sc, \mathcal{S}$)
    if $sc.state$ = LOCKED and time() < $sc.t_d$ then
        if hash($\mathcal{S}$) = $sc.C_{\mathcal{S}}$ then
            $sc.state \leftarrow$ FAILED
            pay($sc, caller, sc.W_{\mathcal{S}}$)
        end if
    end if
end procedure
```

Algorithm 5 SC function to disclose the share after t_d

```
$sc$     smart contract identifier
$h_i$    the $i$-th share
procedure Disclose($sc, h_i$)
    if time() ≥ $sc.t_d$ and time() < $sc.t_{term}$ then
        if $sc.state$ = LOCKED and $sc.states[i]$ = BID then
            if hash($h_i$) = $sc.C_h[i]$ then
                $sc.shares\ [i] \leftarrow h_i$
                $sc.num\_disclosed$ + = 1
                $sc.states[i] \leftarrow$ DISCLOSED
            end if
        end if
    end if
end procedure
```

Algorithm 6 SC function to withdraw the reward

```
$sc$     smart contract identifier
$h_i$    the $i$-th share
procedure Withdraw($sc, h_i$)
    if time() ≥ $sc.t_{term}$ and $sc.num\_disclosed \geq sc.k$ then
        if $sc.states[i]$ = DISCLOSED then
            $sc.states[i] \leftarrow$ WITHDRAWN
            pay($sc, caller, R_{\mathcal{H}}$)
        end if
    end if
end procedure
```

`WhistleblowSecret`: This action (Algorithm 4) enables the whistleblower $\mathcal{W}$ to prove the possession of the secret ahead of the disclosure time t_d, thereby reporting the misbehavior of a group of at least k shareholders. In detail, $\mathcal{W}$ submits the secret $\mathcal{S}'$ to the contract. If the commitment $sc.C_{\mathcal{S}}$ matches, then the TL is marked as `FAILED` and the secret whistleblow reward $W_{\mathcal{S}}$ is paid to the whistleblower. Moreover, all the shareholders lose their bid, and the remaining smart contract funds are destroyed.

`Disclose.` After the disclosure time t_d, each shareholder $\mathcal{H}_i$ is required to publicly reveal its share h_i to enable the retrieval of the secret. The submission is successful if (i) the TL was not previously marked as `FAILED`, and (ii) `hash`(h_i) matches the commitment $sc.C_h[i]$, otherwise submission fails (as shown in Algorithm 5).

`Withdraw.` Conditionally to the outcome of the TL instance, and immediately after the termination time t_{term}, the shareholders are authorized to claim their rewards. Rewards are paid to all shareholders that correctly completed the disclosure procedure. Algorithm 6 illustrates how shareholders can request to withdraw their reward.

4 ECONOMIC MODEL

ITYT models each participant as rational. The interest of the involved parties in the secret is represented by an economic value V associated with it. The use of an economic value permits to analyze the behavior of the parties involved in the protocol, under the assumption that each wants to maximize her reward. In this section we illustrate how to constrain the economic parameters to push rational actors to strictly adhere to the protocol, hence achieving the desired TL function. Since the participants could form alliances, we focus on the behavior of groups of users, considering a generic coalition $\mathcal{M}$ of users that team up to recover the secret ahead of disclosure time.

We provide a description of the structure, the strategy and the collective payoff users can gain as members of $\mathcal{M}$. To ensure that breaking the TL is not profitable, we develop a set of constraints guaranteeing that the attack cost is greater than the maximimun achievable payoff. To this end, we focus on the best attack scenario, that is, the one in which a coalition ideally completes its entire strategy, maximizing the payoff, with no interference by other groups of users.

Single-user constraints. Before going into details, we set some constraints to discourage misbehavior of single users. As the shareholder takes part to the protocol to get a payoff, her expected reward $R_{\mathcal{H}}$ has to be greater than the bid $B_{\mathcal{H}}$ paid to get the share. Moreover, the reward W_h paid to a share whistleblower has to be lower than $B_{\mathcal{H}}$, as each shareholder is expected to keep her share confidential before t_d. Then, to incentivize each shareholder to report misbehavior, $W_{\mathcal{S}}$ has to be greater than $R_{\mathcal{H}}$. Furthermore, since the owner is a priori able to perform the `WhistleblowSecret` action, the owner's fee to get the service F_O must be greater than the secret whistleblow bonus $W_{\mathcal{S}}$ (since the owner is considered rational). Inequality 1 captures, in a single expression, all the considerations made so far.

$$W_h < B_{\mathcal{H}} < R_{\mathcal{H}} < W_{\mathcal{S}} < F_O \tag{1}$$

In the following, we address how a coalition of shareholders could try to break the TL behavior based on when the recostruction of the secret is performed. For the sake of clarity, Section 4.1 does not take into account the `WhistleblowShare` action, which is discussed separately in Section 4.2. Finally, Section 4.3 addresses how to constrain the fee paid by the owner.

4.1 Prevent the reconstruction of the secret

Rational shareholders will consider if it is worth breaking the TL ahead of t_d or not. To perform a successful attack, a shareholder has to team up with other $k-1$ shareholders to recover $\mathcal{S}$. In such an event, the coalition $\mathcal{M}$ would earn the most by monetizing $\mathcal{S}$ and then by performing the `WhistleblowSecret` action, getting an expected payoff of $V + W_{\mathcal{S}}$.[2] The alternative, i.e., $\mathcal{M}$ does not break the TL and submits the k shares after t_d, would lead to an expected payoff equal to the sum of the rewards. To make the second alternative more advantageous, we could require: $k \cdot R_{\mathcal{H}} > V + W_{\mathcal{S}}$. Yet, in order to earn the rewards, the coalition $\mathcal{M}$ should wait until termination time, whereas $V + W_{\mathcal{S}}$ could be collected earlier. For this reason, we use a stricter formulation of the constraint based on the cost already paid by $\mathcal{M}$ to participate in the protocol:

$$k \cdot B_{\mathcal{H}} > V + W_{\mathcal{S}} \tag{2}$$

Constraint (2) addresses secrecy (time $t < t_d$), however, when the disclosure time expires, the constraints are used to promote the release of the secret. However, a coalition of $n-k+1$ shareholders could wait for $k-1$ others to submit the shares and then lead the TL instance to a stall by refusing to submit their own shares. To prevent them from waiting for a buyer of $\mathcal{S}$ and distribute the collective payoff, we introduce the constraint:

$$(n-k+1) \cdot R_{\mathcal{H}} > V \tag{3}$$

Here the contribution of the term $W_{\mathcal{S}}$ disappears, as the role of whistleblower is no longer admissible after t_d. Inequality 3 promotes the disclosure of the shares, as shareholders are rewarded only in case the TL terminates successfully (i.e., at least k shares have been submitted succesfully before termination time t_{term}).

4.2 Impact of share whistleblowing action

The `WhistleblowShare` action increases the number of available strategies. Indeed, a coalition could maximize the payoff by submitting to the contract some of the share it holds before performing the `WhistleblowSecret` action. However, as share whistleblowing implies writing to the blockchain, the public event may trigger strategies of other participants. Thus, the extra revenue $W_{er} = j^o \cdot W_h$ can be gained, where j^o stands for the optimal number of shares to be submitted that do not enable other coalitions' strategies. To address this case, we formulate a stricter version of Inequality (2):

$$k \cdot B_{\mathcal{H}} > V + W_{\mathcal{S}} + W_{er} \tag{4}$$

The optimal number of shares a coalition $\mathcal{M}$ could submit before incurring into penalties or favor other participants, is function of the economic amounts $B_{\mathcal{H}}$, V and $W_{\mathcal{S}}$, and parameters n and k. There are two cases: (a) multiple coalitions are able to recover the secret, and (b) independently from the ratio between the economic amounts there is only one coalition holding at least k shares.

In the first case (a), when i shares have been whistleblown by coalition $\mathcal{M}$, a quiescent coalition $\mathcal{M}'$ formed by $k-i$ shareholders would gain the ability to recover the secret having paid only $(k-i) \cdot B_{\mathcal{H}}$ to get its shares. Therefore, the coalition $\mathcal{M}$ performing the submissions needs to determine the optimal number of shares j^o to be whistleblown so that $\mathcal{M}'$ does not end up having a positive payoff. To compute it $\mathcal{M}$ can solve:

$$j_a^o = \max_i \{i | (k-i) \cdot B_{\mathcal{H}} > V + W_{\mathcal{S}},\ i \in 1, \ldots, k-1\}$$

In the second case (b), the k-shareholders coalition $\mathcal{M}$ is the only one able to recover the secret, since $k > \lfloor n/2 \rfloor$. This condition holds until $2k-n-1$ shares are submitted to the contract. Note this number of submissions could be smaller compared to the one identified in case (a). So, $\mathcal{M}$ can compute j^o by:

$$j_b^o = \max \{2k-n-1;\ j_a^o\}$$

[2] The coalition $\mathcal{M}$ could setup an additional external contract with the buyer to be sure to gain both V and $W_{\mathcal{S}}$.

In both cases, under the assumption of rational agents, the coalition $\mathcal{M}$ can submit j^o shares while still being sure that no other smaller coalition will break the TL. Inequality (4) ensures this strategy is associated with a negative payoff.

4.3 Rewards and bonuses

Now that we have introduced how to constrain the amounts to prevent misbehavior, we discuss some additional requirements to consider an instance of the ITYT protocol well-formed.

In the typical scenario, rational users will strictly adhere to the protocol. To accommodate for this case, the fee paid by the owner has to be enough to remunerate the shareholders:

$$F_O \geq n \cdot (R_{\mathcal{H}} - B_{\mathcal{H}}) \tag{5}$$

In case of failure (i.e., the secret has been recovered before t_d), at least $k - 1$ shares and the secret have been submitted to the contract. To ensure the smart contract has enough currency to pay the whistleblower bonuses, we impose the following constraint:

$$F_O + n \cdot B_{\mathcal{H}} \geq (k - 1) \cdot W_h + W_{\mathcal{S}} \tag{6}$$

In case less than $k - 1$ shares are submitted to the contract before disclosure time, inequalities (5-6) still hold, since $B_{\mathcal{H}} > W_h$.

The constraints (1-6) must all hold for any well-formed ITYT instance. They determine the acceptance area for the economic amounts. The owner may desire to minimize the fee F_O, while the shareholders may desire to maximize the profit $R_{\mathcal{H}} - B_{\mathcal{H}}$. In Table 1 we show three sample configurations obtained by constraint programming. We highlight that the fee paid by the owner is less than the value of the secret, which is a desirable property.

V	k	n	W_h	$B_{\mathcal{H}}$	$R_{\mathcal{H}}$	$W_{\mathcal{S}}$	F_O
1	10	20	0.0031	0.1122	0.1153	0.1184	0.1216
1	15	30	0.0013	0.0717	0.0730	0.0743	0.0756
1	20	50	0.0006	0.0527	0.0533	0.0538	0.0544

Table 1: Sample configurations (economic amounts are expressed as ratio of V)

5 IMPLEMENTATION

In this section, we illustrate how to implement ITYT leveraging existing frameworks. Specifically, Section 5.1 details the ITYT smart contract, while Section 5.2 explains how to use sMPC to implement the share generation primitive.

5.1 Smart contract implementation

We designed ITYT as a finite state machine (FSM) within an Ethereum smart contract whose functions match the actions available to users. If successful, each action entails a write to the blockchain and possibly a change of the global state. The FSM state regulates the actions available to users. Additionally, some actions are reserved to the owner. Figure 1 depicts a simplified version of the state machine that shows only valid transitions. It can be divided into five macro phases: (i) `setup`, in which the owner has to deploy the contract and the shareholders subscribe to it; (ii) `share generation`, that involves *off-chain* operations (i.e., not directly performed by smart contract functions) to confidentially split the secret; (iii) `activation`, in which the shareholders attest they have received their shares and give their go-ahead; (iv) `lock`, in which the shareholders keep the shares confidential; and (v) `termination`, where the secret is finally disclosed.

Setup. Initially, the owner deploys a smart contract instance of ITYT to the Ethereum blockchain [49]. Then, she calls the contract initialization primitive, transfers F_O to the contract, and specifies all the parameters as detailed in Section 3. This advances the global state to PRE_INITIALIZED. Afterwards, each shareholder subscribes to the contract by depositing the proper amount of Ether that corresponds to the bid $B_{\mathcal{H}}$ and invoking the contract function `participate`, as part of Algorithm 2. After all the shareholders have deposited the bid, the owner executes the `startMPC` function, which advances the state to `INITIALIZED`.

Share generation. At this point, owner and shareholders cooperate to generate the shares. Since the economic penalties have not been activated yet, a random key $\mathcal{K}$ is used in place of the secret. From the shareholder's perspective there is no difference, as rewards and penalties are now associated with the management of $\mathcal{K}$, but from the owner perspective, the use of $\mathcal{K}$ avoids the exposure of $\mathcal{S}$ until TL activation (see Figure 2). Only after that, the owner will write to the contract an encrypted version (i.e., the ciphertext) of the secret, $CT = \mathcal{E}nc_{\mathcal{K}}(\mathcal{S})$. The share generation primitive returns to the owner the commitment of the key $C_{\mathcal{K}}$, along with the commitments of all the shares $\{C_1, ..., C_n\}$; while each shareholder gets her share h_i and the commitment of the key. Further details on the sMPC primitive are reported in Section 5.2.

Activation. The owner calls the function `finalizeMPC` and updates the contract with the output received from the shares generation primitive, turning the global state to `PRE_COMMITTED`. Each shareholder verifies the share commitment value written to the contract, if it matches the one obtained from the sMPC (i.e., the owner did not tamper it), then she invokes the `commit` function. After all the participants have given their go-ahead, the owner executes `enforce`, activating the TL (i.e., `LOCKED` state). The economic incentives and penalties are activated as a consequence.

Lock. Before disclosure time t_d, it is only possible to: (i) whistleblow a share, and (ii) whistleblow the key $\mathcal{K}$. To whistleblow a single share, a user can call `WhistleblowShare` submitting h' (Algorithm 3). If the commitment matches, then W_h is immediately paid to the whistleblower. The whistleblow of a share is permitted only k times, as the *k-th* submission leads to the global state `FAILED`. To whistleblow the key, a user can call the `WhistleblowSecret` function submitting $\mathcal{K}'$ (Algorithm 4). If its commitment matches $C_{\mathcal{K}}$, then $W_{\mathcal{S}}$ is paid to the whistleblower, and the protocol is marked as `FAILED`. When the protocol fails, the remaining amount is no longer withdrawable.

Termination. If protocol is not marked as `FAILED` at time t_d, the shareholders can invoke the `disclosure` function to submit their share (Algorithm 5). The disclosure is successful if the share matches the corresponding commitment and it was not published before. The reward can be withdrawn by each shareholder that correctly disclosed her share after the protocol has terminated successfully.

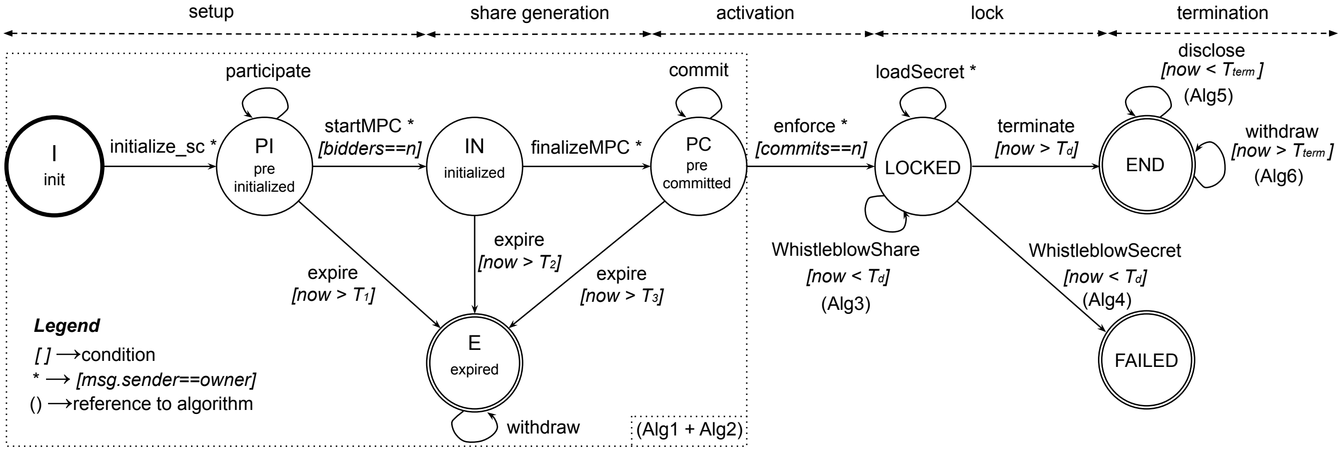


Figure 1: State machine representing the valid state transitions of the ITYT protocol. Each transition name maps to an action (an Ethereum smart contract function) that can be invoked by participants to update the state. Square brackets state additional conditions that must be met to consider the transition valid

To claim the reward, the user can call the function `withdraw` (Algorithm 6). We remark that there is no need to materialize $\mathcal{K}$ in the contract, as all the valid shares will be permanently accessible. Anyone can recover $\mathcal{K}$ just by assembling the shares (e.g., using Lagrange's interpolation in the case of secret sharing).

5.2 Share generation and distribution

In this section, we describe how to implement the `generate_shares` function by using secure Multi-Party Computation frameworks.

In the sMPC setting, each party joins the protocol as a network host. Each ITYT user is then provided with a virtual machine containing an application able to communicate via network following a pre-defined protocol. The application is implemented using FRESCO [4][16], a FRamework for Efficient and Secure COmputation that aims to ease the development of prototypes based on secure computation. FRESCO offers several secure computation techniques, referred to as suites. Depending on the model of computation, each of them is classified into binary or arithmetic. The arithmetic suites permit to efficiently perform additions and multiplications on values that are defined over a finite field, a desirable feature for protocols like ITYT that rely on Secret Sharing. SPDZ [18] is the arithmetic suite we selected. In addition to high performance, SPDZ also ensures protection against *active adversaries* that can deviate in arbitrary ways. This grants ITYT the ability to securely open (i.e., release) partial results of the computation only to some of the parties executing the sMPC protocol, enabling us to send each share only to the legitimate shareholder. To further increase performance, MASCOT [25] was used as SPDZ pre-processing strategy.

To execute the share generation primitive, owner and shareholders start the sMPC application. The owner inputs the random 128 bits key $\mathcal{K}$, together with the total number of shareholders n, and the reconstruction threshold k. Each shareholder, instead, submits only a random 128 bits seed. The sMPC selects k of the n seeds received, using it as the $a_1, \ldots, a_k$ coefficients of the Shamir Secret Sharing polynomial [46], while $\mathcal{K}$ is interpreted as a_0. Then, the sMPC generates n random x values of 128 bits and computes the associated y coordinates using the Horner's method, which permits to evaluate a polynomial of degree k with only k multiplications and k additions. Each i-th share is built as the concatenation of the xy coordinates $h_i = x_i||y_i$. To compute the commitments of the key and the shares, we used MiMC [3], a cryptographic primitive characterized by low multiplicative complexity implemented by FRESCO and compatible with SPDZ. Finally, dedicated output is opened to the parties: the owner gets the commitment C_i of any share generated, while each shareholder gets her share h_i, the commitment of the key $C_{\mathcal{K}}$, n and k.

6 DISCUSSION

In this section, we discuss how ITYT ensures the methods to report misbehavior are not bypassable, and how ITYT mitigates denial of service (DOS) and prevents deadlocks.

6.1 Misbehavior detection

The economic penalties ITYT relies on are triggered when there is a user that is able to prove someone else's misbehavior, and this happens for example when a share is released improperly. Up to now, we have considered secure Multi-Party Computation as a mean to securely deliver dedicated output to users that take part in the ITYT protocol (i.e., to generate and distribute the shares confidentially). However, sMPC can also be used to subvert the protocol, as it enables a group of parties to jointly compute a function while keeping their input confidential. Indeed, a coalition of shareholders could use it to recover the secret ahead of disclosure time without leaking any share nor the key, thus bypassing smart contract commitments. This is an interesting scenario as it applies to most protocols played by rational users that involve rewards and penalties (e.g., [41]).

In our setting this happens when there is a coalition that is able to recover $\mathcal{S}$ without releasing $\mathcal{K}$, thus preventing anyone to perform the `WhistleblowSecret` action. To do that, the coalition inputs to

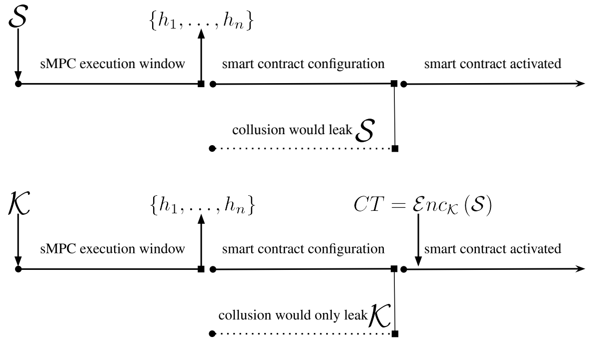

$\mathcal{S}$
$\{h_1, \ldots, h_n\}$
sMPC execution window
smart contract configuration
smart contract activated
collusion would leak $\mathcal{S}$
$\mathcal{K}$
$\{h_1, \ldots, h_n\}$
$CT = \mathcal{E}nc_{\mathcal{K}}(\mathcal{S})$
sMPC execution window
smart contract configuration
smart contract activated
collusion would only leak $\mathcal{K}$

Figure 2: Avoid the exposition of $\mathcal{S}$ before sc activation

Function	Gas		
	$n = 2$	$n = 5$	$n = 10$
`participate`	89545	90308	100207
`startMPC`	50399	50399	50399
`finalizeMPC`	122096	191666	307617
`commit`	56834	59597	69496
`enforce`	50425	50425	50425
`loadSecret`	51367	51367	51367
`WhistleblowShare`	125492	125676	125715
`WhistleblowSecret`	121587	121587	121587
`terminate`	29657	29657	29657
`disclose`	125492	125676	125715
`withdraw`	42634	48709	62451

Table 2: Gas cost for each smart contract function with $k = 2$

the sMPC protocol k shares along with the ciphertext CT (which is publicly available as it is written to the contract by the owner). Then, the protocol performs the reconstruction of the Shamir polynomial, recovers $\mathcal{K}$, and extracts the secret by $\mathcal{S} = \mathcal{D}ec_{\mathcal{K}}(CT)$, opening it to the parties as the only result.

There are two alternatives to prevent this attack: (i) use an encryption scheme vulnerable to the *Known Plaintext Attack*, and (ii) use an encryption scheme that is practically incompatible with the sMPC setting. As an example of (i), with the *One-Time Pad*, given two among $\{CT, \mathcal{S}, \mathcal{K}\}$ the third is implied; then, a coalition of shareholders cannot avoid to release $\mathcal{K}$ by recovering $\mathcal{S}$. The drawback of using *OneTimePad* is that $|\mathcal{S}| = |\mathcal{K}|$ by construction. This limitation can be overcome by selecting an encryption scheme that satisfies (ii).

6.2 DOS and deadlock prevention

A denial of service attack is performed by users that participate in multiple ITYT instances and refuse to deposit their bids, to commit, or to correctly execute the sMPC protocol. To mitigate these kinds of disruptions it is possible to introduce a reputation system. However, this requires to discriminate with high accuracy between misbehaving users and users that follow the protocol as intended. Therefore, we decided to include an additional step in the FSM *setup* phase, in which all participants (including the owner) are required to pay an additional small service pawn that will be returned only at activation time. It has been proven that the introduction of a small fee to access a service can mitigate many DOS attacks [32, 36].

Any other misbehavior, malfunction, or network error could result in a failure to meet the *setup* time threshold set by the owner. The deadlock, to which the protocol leads to, can be managed introducing the final state `EXPIRED` (see Figure 1). In this state, all the participants are allowed to withdraw their funds locked by the contract, except for the small service pawn (as the TL was never activated).

7 EXPERIMENTAL RESULTS

Our experimental analysis is organized into: (i) smart contract deployment and testing, and (ii) simulation of sMPC network protocols. The tests have been executed on a dual Intel Xeon E5 server with 256 GB memory and 512 GB SSD drive running Ubuntu 20.04 LTS. The size of the shares was set to 256 bits.

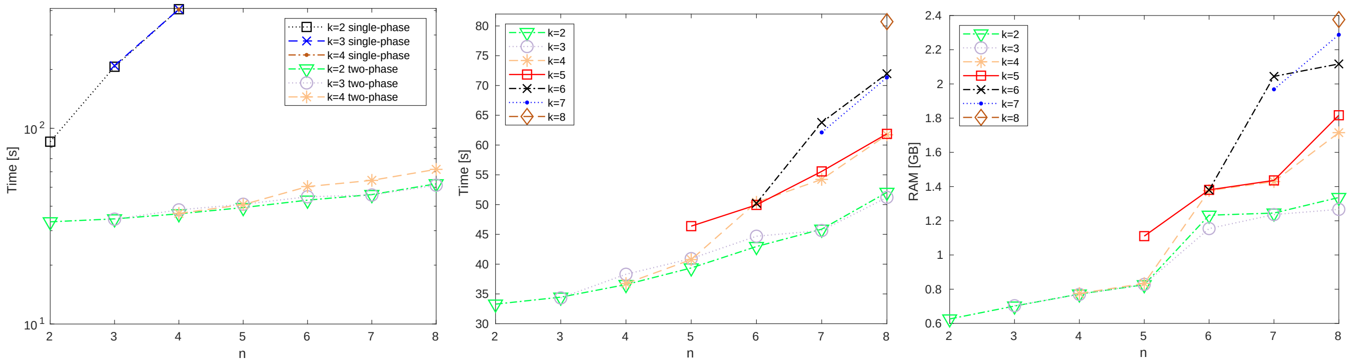
Smart contract. A preliminary version of the smart contract was built with FSolidM [37], a tool that automatically generates Ethereum smart contracts code from high-level Finite State Machine representations. To deploy, test, and debug the contract generated, we relied on Brownie [23], a Python framework that allows us to create wallets, inspect transactions and automatize tests. To provide such functionalities, Brownie interacts with Ganache [2]: a personal Ethereum blockchain used to facilitate development.

The experiments mainly focused on estimating the execution cost of each ITYT instance. The cost is measured in gas, the unit that measures the amount of computational effort required to execute specific operations on the Ethereum network. Table 2 shows, depending on the number of participants, the gas required to run each ITYT function.

sMPC. First, we implemented a sMPC protocol compliant with the description in Section 5.2. We refer to this version as single-phase. Each party was provided with a different application acting either as client or server, and the network communication round trip time (RTT) was set to 10 ms. As it is illustrated in Figure 3a, strictly adhering to this protocol leads to a sudden performance degradation when the number of shareholders increases. This is because computing the MiMC primitive among several participants is highly affected by network latency (the parties have to exchange several messages to carry out even simple operations in the sMPC setting). To improve performance, we implemented a two-phase algorithm composed by two sMPC protocols: *Step 1* and *Step 2.*

Step 1. An n-to-n sMPC is jointly executed by all participants. The owner inputs the random 128 bits key $\mathcal{K}$, together with n and k. Each shareholder submits only a 128 bits seed. As detailed in Section 5.2, the sMPC selects the coefficients to generate the Shamir polynomial and computes the shares. Finally, the output is opened to the parties: the owner gets the polynomial coefficients of $f(x)$, while each shareholder gets her share h_i, n and k.

Step 2. A 1-to-1 sMPC is computed between the owner and each i-th shareholder. The owner inputs $f(x)$, while the shareholder



(a) Single-phase vs two-phase sMPC execution time

(b) Two-phase time consumption

(c) Two-phase memory consumption

Figure 3: sMPC protocol time and memory consumption

inputs her $(x_i; y_i)$ coordinate. If $(x_i; y_i)$ belongs to $f(x)$, commitments are produced. The owner gets the commitment of the share C_i, while the shareholder gets the commitment of the key $C_{\mathcal{K}}$.

The difference between the single-phase and the two-phase sMPC lies in the evaluation of the MiMC primitive. Unlike the single-phase version, the two-phase solution separates the generation of the shares from the production of commitments. It follows that the first step can be carried out even in scenarios with several participants, as it is not computationally intensive, whereas the second step, which instead is computationally intensive, is always performed among two users. The comparison between the two sMPC protocol variants is shown in Figure 3a. More details about the two-phase execution time and memory consumption for each participant, in case of polynomial of higher degree, are illustrated in Figures 3b and 3c, respectively.

8 RELATED WORK

The use of cryptography to solve the problem of unveiling private data at a specific time in the future was first envisioned in 1993 by Timothy May [38]. Since then, many researchers have proposed solutions to this problem. Based on the assumptions and technologies used, we can classify the proposals into four main categories.

Trust and Honesty — Chan et al. [11] and Cheon et al. [13] proposed single point of trust schemes, in which the owner encrypts the secret using public-key cryptograpyhy, and relies on a trusted time-server to release the private decryption key in the future. Rabin et al. described Time-Lapse Cryptography [40, 43] that overcomes the single point of trust assumption by splitting the single authority into a group of users that have to cooperate to release the keys. Li et al. [29] proposed a solution that relies on Distributed Hash Tables to route the secret among peers. However, these proposals entail the peers to be honest as they do not consider the possible economic benefits that the parties would obtain by colluding.

Time-Lock Puzzles — They require the recipient to solve an inherently sequential mathematical puzzle to prove the elapse of time. Starting from Rivest et al. seminal work [45], many other puzzles have been proposed [7, 14, 34]. All these techniques require to run the decryption procedure for a long time, and to make assumptions on future computing power.

Smart Contracts — Similarly to our proposal, the third category leverages smart contracts [47] to replace the trusted party. Kimono [10] and Keep Network [33] rely on threshold cryptography to split the secret among participants that can earn a remuneration by keeping their shares private until disclosure time. However, they do not introduce security deposits, thus failing at preventing misbehavior. Li et al. proposal [30] overcomes some of these limitations by modeling the protocol as an extensive-form game with imperfect information [28]. Yet, as each peer is a single point of failure, and as the owner has perfect information about the shares, they require every participant to pay a security deposit that exceeds the value of the secret, limiting the applicability of the protocol. Compared to our solution, all the proposals in this category do not consider that coalitions of users can reconstruct the secret ahead of disclosure time inside an sMPC protocol without exposing the shares, thus effectively avoiding penalties and safeguarding remunerations.

Witness Encryption — This category of solutions leverages witness encryption [21], in which the sender can encrypt a message so that it can only be opened by a recipient who knows a witness to an NP relation. Liu et al. [31] showed how to construct a computational reference clock from large public computations, such as those made by the Bitcoin network, and couple it with witness encryption to achieve a TL encryption mechanism. Yet, this proposal relies on the availability of a practical witness encryption scheme.

Other Contributions — Several recent proposals address the problem of dealing with secret data on public blockchains. Enigma [51] and Hawk [27] leverage sMPC to allow multiple actors to execute an algorithm on private inputs and store the proof of correct execution on the blockchain. However, these proposals require the data holder to actively participate in the computation, thus they cannot be used to solve the problem of data disclosure at a future point in time. *Proof of Elapsed Time* (*PoET*) is a network consensus algorithm often used in permissioned blockchains, like Hyperledger Sawtooth [1, 12], that avoids wasting computational resources by using a fair lottery system run inside a Trusted Execution Environment (TEE), such as Intel SGX. Each participant runs an algorithm in the TEE that waits for a random amount of time, thus proving the elapse of time without the need of PoW. Even if this approach resembles ITYT, as it prevents cheating on the chosen time, it is not able to store secret data. Another recent contribution, *Bitcoin Lightning Network* [41], shows how economic constraints enforced by TL primitives can be successfully integrated with blockchains. Lightning Network can be used to instantly exchange bitcoins among peers by using off-chain transactions while effectively preventing misbehavior.

9 CONCLUSIONS

In this paper we presented I Told You Tomorrow (ITYT), a practical framework that leverages the rationality assumption to deploy Timed-Locked secrets on the blockchain. In contrast to other Time-Lock mechanisms, ITYT does not rely on a trusted third-party, neither it requires a receiving party to run a decryption algorithm until disclosure time, nor it demands guessing about future computing power. The implementation and experimental evaluation show the low cost and limited resource consumption associaced with our approach.

As future work, we are considering the use of homomorphic encryption for the share generation primitive. By doing so, we expect that the time and resources needed to setup the protocol will be further reduced.

Acknowledgments. This work was supported by the EC within the H2020 Program under project MOSAICrOWN. Marco Guarnieri was also supported by an Atracción de Talento Investigador grant 2018-T2/TIC-11732A, a Juan de la Cierva-Formación grant FJC2018-036513-I, Spanish project RTI2018-102043-B-I00 SCUM, and Madrid regional project S2018/TCS-4339 BLOQUES.

Availability. The open source implementation of ITYT is freely available at: https://github.com/unibg-seclab/ityt

REFERENCES

[1] 2018. Hyperledger Sawtooth. https://sawtooth.hyperledger.org.

[2] 2019. Ganache – Personal blockchain for Ethereum development. https://github.com/trufflesuite/ganache.

[3] M. Albrecht, L. Grassi, C. Rechberger, A. Roy, and T. Tiessen. 2016. MiMC: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In *ASIACRYPT*.

[4] Alexandra Institute. 2019. FRESCO - A Framework for Efficient Secure Computation. https://github.com/aicis/fresco.

[5] J. Alwen, C. Cachin, O. Pereira, A.R. Sadeghi, B. Schoenmakers, A. Shelat, and I. Visconti. 2007. Summary Report on Rational Cryptographic Protocols.

[6] G. Asharov, R. Canetti, and C. Hazay. 2011. Toward a Game Theoretic View of Secure Computation. *IACR* (2011).

[7] N. Bitansky, S. Goldwasser, A. Jain, O. Paneth, V. Vaikuntanathan, and B. Waters. 2016. Time-Lock Puzzles from Randomized Encodings. In *ITCS*.

[8] G. Branwen. 2018. Time-Lock Encryption. https://www.gwern.net/Self-decrypting-files.

[9] P. Caballero-Gil, C. Hernández-Goya, and C. Bruno-Castañeda. 2010. A Rational Approach to Cryptographic Protocols. *CoRR* (2010).

[10] F. Mert Celebi, P. Fletcher-Hill, G. Kaemmer, and D. Que. 2018. Kimono Time Capsule. https://kimono.network.

[11] A.C.F. Chan and I.F. Blake. 2005. Scalable, Server-Passive, User-Anonymous Timed Release Cryptography. In *ICDCS*.

[12] L. Chen, L. Xu, N. Shah, Z. Gao, Y. Lu, and W. Shi. 2017. On Security Analysis of Proof-of-Elapsed-Time (PoET). In *SSS*.

[13] J.H. Cheon, N. Hopper, Y. Kim, and I. Osipkov. 2006. Timed-Release and Key-Insulated Public Key Encryption. In *FC*.

[14] B. Cohen and K. Pietrzak. 2018. Simple Proofs of Sequential Work. In *EUROCRYPT*.

[15] A. Conner-Simons. 2019. Programmers solve MIT's 20-year-old cryptographic puzzle. https://www.csail.mit.edu/news/programmers-solve-mits-20-year-old-cryptographic-puzzle.

[16] I. Damgård, K. Damgård, K. Nielsen, P.S. Nordholt, and T. Toft. 2017. Confidential Benchmarking Based on Multiparty Computation. In *FC*.

[17] I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Scholl, and N.P. Smart. 2013. Practical Covertly Secure MPC for Dishonest Majority – Or: Breaking the SPDZ Limits. In *ESORICS*.

[18] I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Scholl, and N.P. Smart. 2013. Practical covertly secure MPC for dishonest majority–or: breaking the SPDZ limits. In *ESORICS*.

[19] C. Dong, Y. Wang, A. Aldweesh, P. McCorry, and A. van Moorsel. 2017. Betrayal, distrust, and rationality: Smart counter-collusion contracts for verifiable cloud computing. In *SIGSAC*.

[20] C. Dwork and M. Naor. 1993. Pricing via Processing or Combatting Junk Mail. In *CRYPTO*.

[21] S. Garg, C. Gentry, A. Sahai, and B. Waters. 2013. Witness Encryption and Its Applications. In *STOC*.

[22] A. Groce, J. Katz, A. Thiruvengadam, and V. Zikas. 2012. Byzantine Agreement with a Rational Adversary. In *ICALP*.

[23] B. Hauser. 2019. Introducing Brownie: A Python framework for testing, deploying and interacting with Ethereum smart contracts. https://medium.com/hyperlink-technology/introducing-brownie-a763859409ca.

[24] C. Hazay and Y. Lindell. 2010. A Note on the Relation between the Definitions of Security for Semi-Honest and Malicious Adversaries.

[25] M. Keller, E. Orsini, and P. Scholl. 2016. MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer. In *SIGSAC*.

[26] M. Keller, V. Pastro, and D. Rotaru. 2018. Overdrive: making SPDZ great again. In *EUROCRYPT*.

[27] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou. 2016. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *IEEE S&P*.

[28] K. Leyton-Brown and Y. Shoham. 2008. Essentials of game theory: A concise multidisciplinary introduction. *Synthesis lectures on artificial intelligence and machine learning* (2008).

[29] C. Li and B. Palanisamy. 2017. Timed-Release of Self-Emerging Data Using Distributed Hash Tables. In *ICDCS*.

[30] C. Li and B. Palanisamy. 2018. Decentralized Release of Self-Emerging Data using Smart Contracts. In *SRDS*.

[31] J. Liu, T. Jager, S.A. Kakvi, and B. Warinschi. 2018. How to build time-lock encryption. *Designs, Codes and Cryptography* (2018).

[32] G. Loukas and G. Öke. 2010. Protection against denial of service attacks: A survey. *Comput. J.* (2010).

[33] M. Luongo and C. Pon. 2019. The Keep Network: A Privacy Layer for Public Blockchains. https://keep.network/whitepaper.

[34] M. Mahmoody, T. Moran, and S. Vadhan. 2011. Time-Lock Puzzles in the Random Oracle Model. In *CRYPTO*.

[35] M. Mahmoody, T. Moran, and S. Vadhan. 2013. Publicly verifiable proofs of sequential work. In *ITCS*.

[36] D. Mankins, R. Krishnan, C. Boyd, J. Zao, and M. Frentz. 2001. Mitigating distributed denial of service attacks with dynamic resource pricing. In *ACSAC*.

[37] A. Mavridou and A. Laszka. 2017. Designing Secure Ethereum Smart Contracts: A Finite State Machine Based Approach. *ArXiv* (2017).

[38] Timothy May. 1993. Timed-release crypto. http://cypherpunks.venona.com/date/1993/02/msg00129.html.

[39] M. Nojoumian, A. Golchubian, L. Njilla, K. Kwiat, and C. Kamhoua. 2019. Incentivizing Blockchain Miners to Avoid Dishonest Mining Strategies by a Reputation-Based Paradigm. In *ICIC*.

[40] D.C. Parkes, M.O. Rabin, S.M. Shieber, and C. Thorpe. 2008. Practical secrecy-preserving, verifiably correct and trustworthy auctions. *ECRA* (2008).

[41] J. Poon and T. Dryja. 2016. The Bitcoin lightning network: Scalable off-chain instant payments. https://www.bitcoinlightning.com/wp-content/uploads/2018/03/lightning-network-paper.pdf.

[42] M.O. Rabin. 2005. How To Exchange Secrets with Oblivious Transfer. *IACR* (2005).

[43] M.O. Rabin and C. Thorpe. 2006. *Time-lapse cryptography.* Technical Report.

[44] R.L. Rivest. 1999. Description of the LCS35 Time Capsule Crypto-Puzzle. https://people.csail.mit.edu/rivest/lcs35-puzzle-description.

[45] R.L. Rivest, A. Shamir, and D.A. Wagner. 1996. *Time-lock Puzzles and Timed-release Crypto.* Technical Report.

[46] A. Shamir. 1979. How to share a secret. *Commun. ACM* (1979).

[47] N. Szabo. 1997. Formalizing and securing relationships on public networks. *First Monday* (1997).

[48] M. von Maltitz and G. Carle. 2018. A Performance and Resource Consumption Assessment of Secret Sharing Based Secure Multiparty Computation. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology.*

[49] G. Wood. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper* (2014).

[50] A.C. Yao. 1982. Protocols for Secure Computations. In *SFCS*.

[51] G. Zyskind, O. Nathan, and A. Pentland. 2015. Enigma: Decentralized computation platform with guaranteed privacy. *arXiv* (2015).