Timed Secret Sharing

Alireza Kavousi[1], Aydin Abadi[2], and Philipp Jovanovic[3]

[1] University College London
a.kavousi@cs.ucl.ac.uk
[2] Newcastle University
aydin.abadi@newcastle.ac.uk
[3] University College London
p.jovanovic@ucl.ac.uk

Abstract. This paper introduces the notion of *timed secret sharing* (TSS), which establishes *lower* and *upper* time bounds for secret reconstruction in a threshold secret sharing scheme. Such time bounds are particularly useful in scenarios where an early or late reconstruction of a secret matters. We propose several new constructions that offer different security properties and show how they can be instantiated efficiently using novel techniques. We highlight how our ideas can be used to break the *public goods game*, which is an issue inherent to threshold secret sharing-based systems, without relying on incentive mechanism. We achieve this through an upper time bound that can be implemented either via *short-lived proofs*, or the *gradual release of additional shares*, establishing a trade-off between time and fault tolerance. The latter independently provides *robustness* in the event of dropout by some portion of shareholders.

1 Introduction

Threshold secret sharing [54] is a widely used primitive in cryptography and distributed computing. A (t, n)-threshold secret sharing scheme lets a dealer distribute a secret s among n shareholders such that any subset of at least $t + 1$ shares can recover s, whereas no subset of at most t shares reveal any information about s. This primitive is useful in a wide range of applications from password-protection [8,37] and federated learning [42], to verifiable management of on-chain secrets [39] and many more. Protocols using secret sharing usually specify conditions under which shareholders release their shares to reconstruct the secret [20,30]. In many cases, these conditions depend on the notion of time in one way or another. In practice, however, shareholders may violate these time-dependent conditions intentionally or unintentionally by releasing their shares too early or late. These issues may arise due to the use of unsynchronized clocks by the shareholders [7,12,34] or due to a (temporary) dishonest majority [22,23]. The latter could occur particularly when incentives are misaligned so that shareholders collude and reconstruct secrets earlier than what specified [36,44].

The practical applications of threshold secret sharing motivate this work, where elaborate on two concrete scenarios as follows.

Maximal Extractable Value. In cryptocurrency platforms, consensus nodes such as proof-of-stake validators may engage in *maximal extractable value* (MEV) processes [27] to gain some benefit from users by learning their transactions and affect their ordering in the block. A principal MEV countermeasure deploys threshold secret sharing to protect the privacy of transactions up to a time where their inclusion/ordering in a block is ensured. This is done by encrypting the transaction using a random key and then sharing the key towards validators with a threshold secret sharing scheme [44, 61].

However, it largely overlooks the fact that consensus nodes have significant incentives to prematurely reconstruct the secrets to capitalize on MEV rewards. Observe that this type of collusion (*i.e.,* dishonest majority) does not violate the protocol's liveness (*i.e.,* reconstruction) as the success of MEV depends on the completion of the secret reconstruction, and thus colluding parties are incentivized to make progress. In many cases, such behavior is particularly problematic since corrupt shareholders can carry out the process without leaving any public traces and thus collusion is *unobservable* [51].[4]

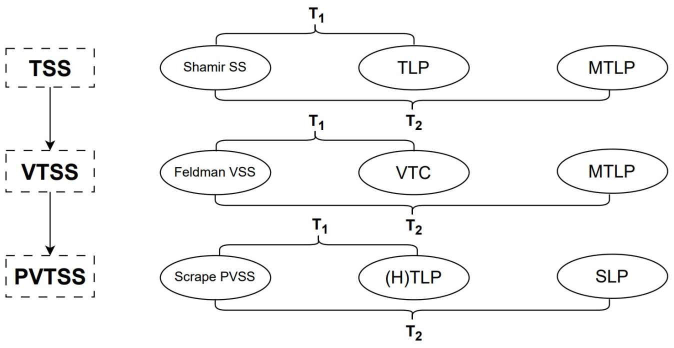
Public goods game. An independent issue with threshold secret sharing-based schemes is that they could constitute a *public goods game* [4, 11]. This is essentially because only a subset of the shareholders needs to release their shares to reconstruct the secret. Consequently, the shareholders may choose to remain inactive, hoping that others will step forward and contribute. As a mitigation mechanism an incentive system is usually assumed [6, 39] which may, however, not be available or feasible to implement under all circumstances.

Our schemes with lower and upper time bounds T_1 and T_2, respectively, address the aforementioned issues: T_1 prevents shareholders from reconstructing the secret early, and T_2 prevents public goods game dilemma without having to rely on financial incentives, providing an alternative solution. We stress that the motivations for lower and upper time bounds are different and independent. In the case of the former, we must *ensure* that the reconstruction does not occur before T_1. In the case of the latter, the goal is to *encourage* (rational) shareholders to appear early and initiate the reconstruction. For the sake of better consistency, we present the schemes with both time bounds rather than treating them separately.

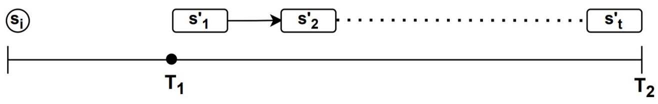
1.1 Technical Overview

Our constructions enjoy novel techniques and build upon time-based primitives with efficient instantiation in a modular way. In particular, we use *time-lock puzzles* (TLPs) [1, 43, 50], *verifiable timed commitments* (VTCs) [57], and *verifiable delay functions* (VDFs) [48, 60]. In the remainder of this section, we give an overview of our proposed constructions.

[4] Using time-lock puzzles (TLPs) [50] are not sufficient to address the issue as protected transactions may actually not make it into the block and then lose confidentiality after the TLP has been opened, demanding *pending transaction privacy* [24].



(a) A visual representation of our constructions. It depicts the underlying tools and techniques to establish the lower and upper time bounds for different variants of secret sharing protocols.



(b) A visual representation of secret sharing with gradual release of additional shares that could break public goods game and provide robustness.

Timed Secret Sharing (TSS). This is our basic construction, where the dealer encapsulates the shares into TLPs [43,50] to realize a lower time bound T_1. Consequently, no computationally bounded adversary can learn the secret before T_1, even if it corrupts *all the shareholders* [23]. Moreover, TLPs provide a consistent relative measure of time (*i.e.,* computational timing), eliminating the need for a shared global clock. For the upper time bound, we rely on the underlying timing assumption of the secret sharing scheme and later show how to relax it.

Verifiable Timed Secret Sharing (VTSS). We enhance TSS with verification mechanisms, to deal with *malicious* dealers and shareholders. First, we ensure that a malicious dealer cannot distribute malformed puzzles, *i.e.,* puzzles that either are not extractable or contain invalid shares. Second, we ensure that malicious shareholders cannot send invalid shares during the reconstruction phase. Here we need to tackle technical challenges in realizing lower and upper time bounds. We overcome the former with a novel trick in using verifiable secret sharing (VSS) [31] and verifiable timed commitment (VTC) [57] that allows checking the validity of embedded share before the shareholder invests computational effort to retrieve it. For the latter, we introduce the novel idea of secret sharing with *gradual release of additional shares* that relaxes the assumption made in the previous scheme and also that could be of independent interest.

Publicly Verifiable Timed Secret Sharing (PVTSS). We further extend our schemes to support *public verifiability.* To do so, we take a different route in realizing the lower and upper time bounds. First, we deploy an efficient non-interactive zero-knowledge (NIZK) protocol, and the cut-and-choose technique [40] to let *anyone* (not just shareholders) ensure the validity of the embedded encrypted shares and the extractability of puzzles. Second, we bind the attestation of the distributed shares to time and impose an upper bound T_2 by utilizing short-lived proofs (SLPs) [5] that come with time-sensitive soundness and public verifiability. We crucially rely on the observation that the secret (and shares) are uniformly distributed, allowing us to securely use SLPs that require indistinguishability property. This essentially puts an upper time bound by making the system usable up to some time T_2, *i.e.,* the correct reconstruction is only guaranteed before T_2.

It is worth mentioning that our idea of secret sharing with additional shares could be useful in scenarios where a sufficient number of (honest) shareholders is not available for reconstruction and thus the additional shares allow the remaining parties to nevertheless reconstruct the secret, providing *robustness* to the system. As an application, this could help with *dropout resilience* in secure aggregation protocols for federated learning [42].

1.2 Our Contributions

- We formally define and construct (t, n)-*timed secret sharing* (TSS) which enables a timely reconstruction of a secret shared by a dealer to a set of n shareholders within the time interval $[T_1, T_2]$.
- We enhance TSS with verifiability by formally defining and constructing *verifiable timed secret sharing* (VTSS), which protects against a malicious dealer during share distribution and against malicious shareholders during secret reconstruction.
- We further extend VTSS with public verifiability by formally defining and constructing *publicly verifiable timed secret sharing* (PVTSS).
- We introduce two novel ideas to break the public goods game in threshold secret sharing systems. One is based on using *short-lived proofs* and the other is based on *gradual release of additional shares.* As a side contribution, we formally define and propose a construction for the latter which is also useful to provide *robustness* against shareholder's dropouts.

2 Related Work

There is a large body of literature on the combination of computational timing and cryptographic primitives such as commitment [3, 14, 29, 45, 58], encryption [17, 25, 41], signature [9, 28, 33, 57], and more. The essence of almost all of these works is to enable the receiver(s) to forcefully open the locked object after a predefined period by working through some computational operation.

The work of [57] proposed efficient constructions for encapsulating a signature into a TLP, ensuring the receiver can extract the valid signature after carrying out sequential computation. Roughly speaking, the sender secret shares the signature and embeds each share in a linearly homomorphic TLP [43]. Then, the sender and receiver run a cut-and-choose protocol for verifying the correctness of the puzzles. Moreover, to enable the receiver to compact all the pieces of time-locked signatures and solve one single puzzle, a range proof is used to guarantee that no overflow occurs. Manevich and Akavia in [45] augment the timed commitment of Boneh and Naor [14] with zero-knowledge proofs, enabling the sender to prove *any* arbitrary attribute regarding the committed value.

With a focus on reducing the interaction in MPC protocols with limited-time secrecy, the authors in [3] developed a gage time capsule (GaTC), allowing a sender to commit to a value that others can obtain after putting a total computational cost which is parallelizable to let solvers claim a monetary reward in exchange for their work. The security guarantee of GaTC resemble ours when using secret sharing with additional shares in the sense that over time it gradually decays, as the adversary can invest more and more computational resources. Doweck and Eyal [29] constructed a multi-party timed commitment that enables a group of parties to jointly commit to a secret to be opened by an aggregator later on via brute-force computation.

The authors in [10] explore multi-party computation with output-independent abort, having each participant in an MPC protocol lock their output until some time in the future. This is to force the adversary to decide whether to cause an abort before learning the output. As performing sequential computations might be beyond the capacity of some users, Thyagarajan et al. [59] developed a system to allow users to outsource their tasks to some servers in a privacy-preserving manner. Srinivasan et al. [56] constructed a TLP that supports unbounded batch-solving while enjoying a transparent setup and a puzzle size independent of the batch size. Although their construction is of theoretical interest and does not have practical efficiency due to the reliance on indistinguishability obfuscation, it enables a party to solve many puzzle instances simultaneously at the cost of solving one puzzle. It is worth noting that such a setting is not applicable to our PVTSS as each shareholder just needs to know their own share and solving other parties' puzzles gives her no information as they are already encrypted under the parties' public keys. One of the motivating reasons for batch-solving is to enable a party to solve the puzzles of others in case a large number of parties abort. We refer the reader to [47] for a more detailed overview of relevant works.

3 Preliminaries

3.1 Threat Model and Assumptions

We consider a standard synchronous network where each pair of parties in a set $\mathcal{P} = \{P_1, \ldots, P_n\}$ is connected via an authenticated channel, and each message is delivered at most by a known delay. There is also a dealer D that takes the role of distributing the secret among participating parties.

As common in the literature for verifiable secret sharing, we assume the existence of broadcast channels. For a publicly verifiable scheme, we assume the existence of an authenticated public bulletin board. In this work, we consider a static adversary that may corrupt up to t out of n parties before the start of protocol execution. D may also be corrupted. We consider both semi-honest and malicious types of adversaries. In the former, the corrupted parties are assumed to follow the protocol but may try to learn some information by observing the protocol execution. In the latter, however, the corrupted parties are allowed to do any adversarial action of their choice. The adversary's computational power is bounded with respect to a security parameter λ that gives it a negligible advantage in breaking the security of underlying primitives. Such algorithms are often known as probabilistic polynomial time (PPT). Finally, we denote by $[n]$ the set $\{1, \ldots, n\}$ an by $\mathbf{v}$ a vector of elements $\{v_i\}_{i\in[n]}$.

3.2 Secret Sharing

A (threshold) secret sharing scheme is a cryptographic protocol that enables a dealer D to distribute a secret s among n parties. The scheme typically consists of two main phases; *distribution* and *reconstruction*. In the former, D sends each party their corresponding share, and in the latter, any proper subset of parties reconstruct the secret by pooling their shares.

A (t, n)-threshold secret sharing offers two main properties: (1) correctness: the secret is reconstructed by any subset of at least $t+1$ shares, and (2) t-security: no information is revealed about the secret by gathering t or fewer shares. In this work, we develop our protocols based on the popular Shamir secret sharing [54]. We note that our proposed definitions can capture any (linear) secret sharing.

Verifiable Secret Sharing (VSS). The basic (t, n)-threshold secret sharing scheme (e.g., [54]) only provides security against a *semi-honest* adversary. When dealing with malicious adversaries, it is essential for (1) the dealer to prove the validity of the shares it produces in the distribution phase, and (2) the shareholders to prove the validity of the shares they provide in the reconstruction phase. To satisfy these properties, various VSS schemes have been proposed, following the celebrated work by Feldman [31].

Publicly Verifiable Secret Sharing (PVSS). To extend the scope of verifiability to the public and not only participating parties, PVSS schemes [18, 19, 53] deploy cryptographic primitives such as encryption and NIZK proofs. PVSS enables anyone to verify the distribution and reconstruction phases. Cascudo and David [18] proposed an efficient scheme called Scrape PVSS, which is an improvement over [53] and has been deployed extensively in many recent cryptographic protocols. The Scrape protocol works as follows. The dealer D chooses a random value $s \overset{\$}{\leftarrow} \mathbb{Z}_q$, sets the secret as a group element of form $S = h^s$, splits s into shares $\{s_i\}_{i\in[n]}$, and computes the encrypted shares $\{\hat{s}_i\}_{i\in[n]}$ using corresponding parties' public keys $\{pk_i\}_{i\in[n]}$.

Then, D publishes a set of commitments to shares $\{v_i\}_{i\in[n]}$ together with a proof π_D, enabling anyone to check the consistency of the shares (*i.e.,* shares are evaluations of the same polynomial of proper degree) and validity of the ciphertexts (*i.e.,* encrypted shares correspond to the committed shares). Upon receiving a threshold number of valid shares (*i.e.,* shares with correct decryptions), anyone can use Lagrange interpolation [2] in the exponent to reconstruct the secret S. The authors proposed two versions, one in the random oracle model under the Decisional Diffie-Hellman (DDH) assumption and the other in the plain model under the Decisional Bilinear Squaring (DBS) assumption. We use the non-pairing variant which offers *knowledge soundness*. This is vital to ensure the secret chosen by the adversary is independent of those of honest parties. Also, we require the knowledge soundness property for deploying short-live proofs [5].

3.3 Time-Lock Puzzles (TLPs)

The idea of TLPs was introduced by Rivest et al. [50]. TLP locks a secret such that it can only be retrieved after a predefined amount of sequential computation. It consists of two algorithms: $\mathsf{TLP.Gen}$, which takes as input a time parameter T and a secret s, and returns a puzzle Z, and $\mathsf{TLP.Solve}$, that takes as input a puzzle Z and returns a secret s. A TLP must satisfy *correctness* and *security*. The correctness ensures that the solution is indeed obtained if the protocol gets executed as specified. The security ensures that no PPT adversary running in parallel obtains the solution within the time bound T, except with negligible probability. We provide the formal definitions in Appendix A.

Homomorphic Time-lock Puzzles (HTLP). Malavolta and Thyagarajan [43] proposed homomorphic TLP, enabling one to homomorphically combine many instances of TLPs into a single TLP. An HTLP consists of a tuple of algorithms ($\mathsf{HTLP.Setup}$, $\mathsf{HTLP.Gen}$, $\mathsf{HTLP.Solve}$, $\mathsf{HTLP.Eval}$). In particular, $\mathsf{HTLP.Setup}$ generates public parameters pp on input a security parameter, and $\mathsf{HTLP.Eval}$ performs a homomorphic operation on input a set of puzzles to output a single puzzle.

Multi-instance Time-lock Puzzle (MTLP). Abadi and Kiayias [1] proposed a primitive called multi-instance TLP. This variant of TLP is suitable for the case where the solver is given multiple puzzles at the same time but must discover each solution at different points in time. It allows solving the instances sequentially one after the other without needing to run parallel computations on them. An MTLP consists of a tuple of algorithms ($\mathsf{MTLP.Setup}$, $\mathsf{MTLP.Gen}$, $\mathsf{MTLP.Solve}$, Prove, Verify), where the last two algorithms are used to check the correctness of a solver's claimed solution.

3.4 Timed Commitment

An inherent limitation of the well-known time-lock puzzles such as [43,50] is the lack of verifiability, meaning that the receiver cannot check the validity of the received puzzle unless after putting time and effort into solving it. To fill this gap, a timed commitment scheme [14] enables the receiver to make sure about the well-formedness (*i.e.,* extractability) of the puzzle before performing a sequential computation. In an attempt to make the timed commitment of [14] efficiently verifiable, the recent work of Thyagarajan et al. [57] proposed verifiable timed commitment (VTC), enabling the sender to verifiably[5] commit to signing keys of form $pk = g^{sk}$, $sk \in \{0,1\}^\lambda$. The VTC primitive consists of a tuple of algorithms ($\mathsf{VTC.Setup}$, $\mathsf{VTC.Commit}$, $\mathsf{VTC.Verify}$, $\mathsf{VTC.Solve}$). Note that we deploy VTC to design construction for our verifiable time secret sharing (VTSS) scheme.

3.5 Sigma Protocols

A zero-knowledge protocol enables proving the validity of a claimed statement by the prover P to the verifier V without revealing any information further. While zero-knowledge protocols involve various settings and notions, we particularly consider the well-known Sigma protocols which are useful building blocks in many cryptographic constructions. Let v denote an instance that is known to both parties and w denote a witness that is only known to the P. Let $R = \{(v;w)\} \in \mathcal{V} \times \mathcal{W}$ denote a relation containing the pairs of instances and corresponding witnesses. A Sigma protocol Σ on $(v;w) \in R$ is an interactive protocol with three movements between P and V. Using Fiat-Shamir heuristic [32] in the random oracle model, one can make the protocol non-interactive with public verifiability. A Sigma protocol satisfies two security properties: (1) *soundness*, ensuring the verifier about the validity of the statement v, and (2) *zero-knowledge*, ensuring the prover about the secrecy of the witness w.

Zero Knowledge proof of equality of discrete logarithm. One of the well-used Sigma protocols is discrete logarithm equality (DLEQ) proof. It considers a tuple of publicly known values (g_1, x, g_2, y), where g_1, g_2 are random generators and x, y are two elements of the cyclic group $\mathbb{G}$ of order q. DLEQ proof enables a prover P to prove to the verifier V that it knows a witness α such that $x = g_1^\alpha$ and $y = g_2^\alpha$. A DLEQ proof is an AND-composition of two Sigma protocols for relation $R = \{(v_i; w) : v_i = g_i^w\}$ with the *same* witness and challenge. The following protocol is a Sigma protocol for generating a DLEQ proof due to Chaum-Pedersen [21].

1. P chooses a random element $u \xleftarrow{\$} \mathbb{Z}_q$, computes $a_1 = g_1^u$ and $a_2 = g_2^u$, and sends them to the V.
2. V sends back a randomly chosen challenge $c \xleftarrow{\$} \mathbb{Z}_q$.
3. P computes $r = u + c\alpha$ and sends it to V.
4. V checks if both $g_1^r = a_1 x^c$ and $g_2^r = a_2 y^c$ hold.

Throughout the paper we use the non-interactive version of this protocol which produces a single message $\mathsf{DLEQ.P}(\alpha, g_1, x, g_2, y)$ as proof π verified via $\mathsf{DLEQ.V}(\pi, g_1, x, g_2, y)$. The challenge is computed by the prover as $c = H(x, y, a_1, a_2)$, where H is a cryptographic hash function modeled as a random oracle.

3.6 Short-lived Proofs

Arun et al. [5] recently introduced the notion of *short-lived proofs* (SLPs) which can be roughly defined as types of proofs with expiration, such that their sound-

[5] Ensuring the extractability together with validity of the committed message that is the discrete logarithm of a public key.

ness will disappear after certain time. They are only sound if being observed before a determined time, afterwards, they may be forgery indistinguishable from the valid proofs. At a high level, an SLP is proof of an OR-composition $R \vee R_{VDF}$, where R is an arbitrary relation and R_{VDF} is a VDF evaluation relation. Interestingly, this proof is only convincing to the verifier for a determined time T as forging the proof is possible for anybody after evaluating the VDF. Due to the nature of VDF, short-lived proofs offer efficient public variability. One notable point is that the primitive makes use of a *randomness beacon* [26] which outputs unpredictable values b periodically.

An SLP scheme consists of four algorithms ($\mathsf{SLP.Setup}$, $\mathsf{SLP.Gen}$, $\mathsf{SLP.Forge}$, $\mathsf{SLP.Verify}$) with the following descriptions. $\mathsf{SLP.Setup}$ generates public parameters pp on input the security parameter and time parameter T. $\mathsf{SLP.Eval}$ takes pp, an input x, a random beacon value b, and generates a proof π. $\mathsf{SLP.Forge}$ takes pp, x, b, and produces a proof π. Lastly, $\mathsf{SLP.Verify}$ validates the proof π on input pp, x, π, and b. A short-lived proof must satisfy four security properties including *forgeability*, enabling anyone running in time $(1+\epsilon)T$ to generate a valid proof, *soundness*, preventing a malicious prover P^* running with parallel processors to generate a convincing proof in time less than T, *zero knowledge*, preserving the privacy of the witness w, and *indistinguishability*, making the real and forged proofs indistinguishable from the actual proof.

4 Timed Secret Sharing (TSS)

With timed secret sharing (TSS), we make a secret sharing scheme dependent on time, having the reconstruction phase occur within a determined time interval, $[T_1, T_2]$, where T_1 is the lower time bound and T_2 is the upper time bound. These time bounds might be required by the dealer or as part of the system requirements, or even a combination of these two. An important consideration, however, is that the dealer's *availability* should not be affected by making the scheme time-based, meaning that the dealer's role should finish after the distribution phase similar to the original setting.

4.1 TSS Definition

In this section, we present a formal definition of TSS. This definition builds upon the original definition of threshold secret sharing.

Definition 1 (Timed Secret Sharing). *A timed secret sharing (TSS) scheme involves the following algorithms.*

1. **Initialization:**
   – Setup: $\mathsf{TSS.Setup}(1^\lambda, T_1, T_2) \rightarrow pp$, *on input security parameter* λ, *lower time bound* T_1, *and upper time bound* T_2, *outputs public parameters* pp.
2. **Distribution:**

- Sharing: $\mathsf{TSS.Sharing}(pp, s) \to \{C_i\}_{i\in[n]}$, *on input pp and secret $s \in S_\lambda$, outputs a locked share C_i with time parameter T_1 for each party P_i in the set $\mathcal{P}$.*

3. **Reconstruction:**
   - Recovering: $\mathsf{TSS.Recover}(pp, C_i) \to s_i$, *on input pp and C_i, recovers the share s_i. The algorithm is run by each party P_i in $\mathcal{P}$.*
   - Pooling: $\mathsf{TSS.Pool}(pp, \mathcal{S}, T_2) \to s$, *on input pp and a set $\mathcal{S}$ of shares (where $|\mathcal{S}| > t$ and $t \in pp$), outputs the secret s if T_2 has not elapsed. Otherwise, it outputs $\perp$.*

A correct TSS scheme must satisfy *privacy*, ensuring no share is obtained before T_1 and *security*, ensuring any set of shares less than a threshold $t + 1$ reveals no information about the secret before T_2.

Definition 1.1 (Correctness) *A TSS satisfies correctness if for all secret $s \in S_\lambda$ and a set of shares $|\mathcal{S}| > t$ it holds*

$$\Pr\left[\mathsf{TSS.Pool}(pp, \mathcal{S}, T_2) \to s : \begin{array}{c}\mathsf{TSS.Setup}(1^\lambda, T_1, T_2) \to pp,\\ \mathsf{TSS.Sharing}(pp, s) \to \{C_i\}_{i\in[n]},\\ \mathsf{TSS.Recover}(pp, C_i) \to s_i\end{array}\right] = 1$$

Definition 1.2 (Privacy) *TSS satisfies privacy if for all parallel algorithms $\mathcal{A}$ whose running time is at most less than T_1 there exists a simulator Sim and a negligible function μ such that for all secret $s \in S_\lambda$, all $\lambda \in \mathbb{N}$, and all $i \in [n]$ it holds*

$$\left|\Pr\left[\mathcal{A}(pp, s, C_i) = 1 \;:\; \begin{array}{l}\mathsf{TSS.Setup}(1^\lambda, T_1, T_2) \to pp,\\ \mathcal{A}(pp, 1^\lambda) \to s,\\ \mathsf{TSS.Sharing}(pp, s) \to \{C_i\}_{i\in[n]}\end{array}\right] - \right.$$
$$\left.\Pr\left[\mathcal{A}(pp, s', C_j) = 1 \;:\; \begin{array}{l}\mathsf{TSS.Setup}(1^\lambda, T_1, T_2) \to pp,\\ \mathcal{A}(pp, 1^\lambda) \to s',\\ \mathsf{Sim}(pp) \to \{C_j\}_{j\in[n]}\end{array}\right]\right| \leq \mu(\lambda)$$

Definition 1.3 (Security) *TSS satisfies security if an adversary $\mathcal{A}$ controlling a set $\mathcal{S}'$ of parties, where $|\mathcal{S}'| \leq t$ and $s \in S_\lambda$, learns no information about s. Thus, it must hold*

$$\Pr\left[\mathcal{A}(pp, \mathcal{S}', T_2) \to s : \begin{array}{c}\mathsf{TSS.Setup}(1^\lambda, T_1, T_2) \to pp,\\ \mathsf{TSS.Sharing}(pp, s) \to \{C_i\}_{i\in[n]},\\ \mathsf{TSS.Recover}(pp, C_i) \to s_i\end{array}\right] \leq \mu(\lambda) + \frac{1}{|S_\lambda|}$$

4.2 TSS Construction

We present an instantiation of TSS in Figure 2. To enforce a lower time bound T_1, the dealer uses TLPs [43, 50] to lock the shares into puzzles, enforcing a computational delay for each party to recover their corresponding share. Note that we treat T_2 mostly as a matter of formalization and rely on the underlying assumption of having common knowledge of time for participating parties. We later in Section 5 show how to relax this assumption using computational timing.

Π_{TSS}

1. **Initialization:**
   - Setup: $\mathsf{TSS.Setup}(1^\lambda, T_1, T_2) \to pp$, the protocol works over $\mathbb{Z}_q$, where $q > n$. The dealer D runs $\mathsf{TLP.Setup}(1^\lambda, T_1)$ and publishes public parameters pp.
2. **Distribution:**
   - Sharing: $\mathsf{TSS.Sharing}(pp, s) \to \{Z_i\}_{i \in [n]}$, the dealer D picks a secret $s \in Z_p$ to be shared among n parties. It samples a degree-t Shamir polynomial $f(\cdot)$ such that $f(0) = s$ and $f(i) = s_i$ for $i \in [n]$. It runs $\mathsf{TLP.Gen}(1^\lambda, T_1, s_i)$ to create puzzle Z_i with time parameter T_1, locking the share s_i for all $i \in [n]$. Finally, D privately sends each party P_i their corresponding puzzle Z_i.
3. **Reconstruction:**
   - Recovering: $\mathsf{TSS.Recover}(pp, Z_i) \to s_i$, upon receiving the puzzle Z_i, party P_i starts solving it by running $\mathsf{TLP.Solve}(T_1, Z_i)$ to recover the share s_i.
   - Pooling: $\mathsf{TSS.Pool}(pp, \mathcal{S}, T_2) \to s$, upon having sufficient number of shares $(\geq t+1)$ received before T_2, the reconstructor (a party in $\mathcal{P}$) reconstructs the secret s using Lagrange interpolation at $f(0)$; otherwise, it returns $\perp$.

Fig. 2: Timed Secret Sharing (TSS) protocol

Theorem 1. If the time-lock puzzle TLP and Shamir secret sharing are secure, then timed secret sharing protocol Π_{TSS} presented in Figure 2 satisfies privacy and security, w.r.t. definitions 1.2 and 1.3 respectively.

Proof. Correctness is straightforward. The privacy property follows directly from that of the underlying TLP which implies the indistinguishability of a puzzle produced by algorithm $\mathsf{TSS.Sharing}$ and the one produced by Sim. Since all the puzzles are communicated through private channels, no party can learn the other party's share after T_1. Finally, the security stems from the underlying threshold secret sharing, where a subset of shares $\mathcal{S}'$ whose size is less than t reveals no information about the secret s.

5 Secret Sharing with Additional Shares

A threshold secret sharing scheme guarantees *t-security.* There is also $t+1$-*robustness* assumption, ensuring the availability of a sufficient number of valid shares during the reconstruction phase. However, it is natural to challenge such a liveness assumption and consider a scenario in which a *large* fraction of honest parties goes offline, particularly when having a determined period for reconstruction, putting the system under threat of failure (*i.e.,* lack of liveness). To be concrete, a possible scenario that may lead to having less than a threshold of (honest) parties available is explored in [57] known as *denial of spending* (DoSp) attack where the set of available parties cannot reach the threshold and their investment will remain locked. In a federated learning setting [42], real-world factors such as hardware failure or poor network coverage can also cause this issue, leading to shareholders' dropouts.

Our goal is to provide *robustness* using the capabilities of time-based cryptography. We observe this is feasible by having the dealer provide parties with *additional time-locked shares*. By additional, we mean some shares other than the individual one each party already receives during the distribution phase of the protocol. Thus, even if there is less than a threshold of parties (even a single one) available at the reconstruction period (*i.e.,* $[T_1, T_2]$), they will be able to open the additional time-locked shares after carrying out some computation and retrieve the secret. We remark that a large body of literature on threshold secret sharing assumes all the parties, not only those interacting in the reconstruction phase, learn the secret [18, 38]. Given this, we argue that the availability of a (threshold) number of additional time-locked shares at the proper time (*i.e.,* T_2) does not violate the security of the system since it enables all the parties to eventually learn the secret at the same time if they have not already learned it.

5.1 Decrementing-threshold Timed Secret Sharing (DTSS)

It is possible to derive an interesting *trade-off* between time and fault tolerance by having some additional time-locked shares to be realized periodically at *different* points in time. The consequence of this *gradual* release is twofold. Firstly, it enables (honest) parties requiring some more shares (not necessarily t) to reconstruct the secret without going through the sequential computation for the whole period, *i.e.,* $[T_1, T_2]$. They can stop working up to a point where a sufficient number of additional shares is gained, as T_2 might be considerably later than T_1. Secondly, as time goes by and the reconstruction is not initiated, the adversary may get more additional shares by investing computational effort, causing security decay over time [3]. Looking ahead, this feature happens to be useful to impose an upper time bound and thus *break the public goods game* as it ties the security of the system to time; the later parties initiate the reconstruction, the more chances the adversary learns the secret.[6]

5.2 DTSS Definition

Now, we present a formal definition for our scheme called decrementing-threshold timed secret sharing (DTSS).

Definition 2 (Decrementing-threshold Timed Secret Sharing). *A (t, n) DTSS scheme consists of a tuple of algorithms (DTSS.Setup, DTSS.Sharing, DTSS.ShaRecover, DTSS.Verify, DTSS.AddRecover, DTSS.Pool) as follows.*

1. **Initialization:**

[6] It is clear that since all parties can recover the secret by T_2, this essentially puts an upper time bound for the system. We use this technique to relax the assumption made to realize an upper time bound for TSS.

– Setup: $\mathsf{DTSS.Setup}(1^\lambda, T_1, T_2, t) \to \{pp, pk, sk\}$, *on input security parameter* λ*, lower time bound* T_1*, and a value* t*, outputs public parameters* pp *and key pair* (pk, sk) *to be used for generating additional locked shares by the dealer* D*.*

2. **Distribution:**
   – Sharing: $\mathsf{DTSS.Sharing}(pp, s, pk, sk) \to \{\{C_i\}_{i\in[n]}, \{O_j\}_{j\in[t]}\}$, *on input* pp*, a secret* s*, and a key pair* (pk, sk)*, outputs locked share* C_i *with time parameter* T_1*. Moreover, it outputs* t *additional locked shares* $\{O_j\}_{j\in[t]}$*, with* O_j *being locked with time parameter* $(j+1)T_1$*.*
3. **Reconstruction:**
   – Share recovery: $\mathsf{DTSS.ShaRecover}(pp, C_i) \to s_i$, *on input* pp *and* C_i*, outputs a share* s_i*. The algorithm is run by each party* P_i*.*
   – Additional share recovery: $\mathsf{DTSS.AddRecover}(pp, pk, \{O_j\}_{j\in[t]}) \to \{s'_j\}$, *on input* pp*,* pk*, and* $\{O_j\}_{j\in[t]}$*, forcibly outputs the additional share* s'_j *at time* $(j+1)T_1$*. The algorithm is run by anyone in* $\mathcal{P}$ *wishing to obtain additional shares.*
   – Pooling: $\mathsf{DTSS.Pool}(pp, \mathcal{S}, T_2) \to s$, *on input* pp *and a set* $\mathcal{S}$ *of shares (where* $|\mathcal{S}| > t$ *and* $t \in pp$*), outputs the secret* s *if* T_2 *has not elapsed.*

A correct DTSS scheme must satisfy *privacy*, *security*, and *robustness* with the following definitions.

Definition 2.1 (Privacy) *A DTSS satisfies privacy if for all algorithms* $\mathcal{A}$ *running in time* $T < jT_1$*, where* $1 \le j \le t$*, with at most* T_1 *parallel processors, there exists a simulator* Sim *and a negligible function* μ *such that for all secret* $s \in S_\lambda$ *and* $\lambda \in \mathbb{N}$ *it holds that*

$$\left| \Pr\left[\begin{array}{l} \mathcal{A}(pp, pk, s, \\ C_i, \{O_j\}_{j\in[t]}) = 1 \end{array} : \begin{array}{l} \mathsf{DTSS.Setup}(1^\lambda, T_1) \to \{pp, pk, sk\}, \\ \mathcal{A}(1^\lambda, pp) \to s \\ \mathsf{DTSS.Sharing}(pp, s) \\ \to \{\{C_i\}_{i\in[n]}, \{O_j\}_{j\in[t]}\} \end{array} \right] - \Pr\left[\begin{array}{l} \mathcal{A}(pp, pk, s', \\ C_i, \{O_j\}_{j\in[t]}) = 1 \end{array} : \begin{array}{l} \mathsf{DTSS.Setup}(1^\lambda, T_1) \to \{pp, pk, sk\}, \\ \mathcal{A}(1^\lambda, pp) \to s' \\ \mathsf{Sim}(pp) \to \{\{C_i\}_{i\in[n]}, \{O_j\}_{j\in[t]}\} \end{array} \right] \right| \le \mu(\lambda)$$

Definition 2.2 (t-Security) *Let* $2T_1, \dots, (t+1)T_1$ *be times at which each additional time-locked share is forcibly obtained. A DTSS is* t*-secure if prior to* $(j+1)T_1$*, where* $1 \le j \le t$*, the adversary controlling a set* $|\mathcal{S}'| \le t - (j-1)$ *of parties learns no information about* $s \in S_\lambda$ *in a computational sense. Thus, it holds*

$$\Pr\left[\mathcal{A}(pp, pk, \mathcal{S}', T_2) \to s \ : \begin{array}{l} \mathsf{DTSS.Setup}(1^\lambda, T_1, t) \to \{pp, pk, sk\} \\ \mathsf{DTSS.Sharing}(pp, s) \\ \to \{\{C_i\}_{i\in[n]}, \{O_j\}_{j\in[t]}\}, \\ \mathsf{DTSS.ShaRecover}(pp, C_i) \to s_i, \\ \mathsf{DTSS.AddRecover}(pp, pk, \{O_j\}_{j\in[t]}) \\ \to \{s'_j\}, 1 \le j \le t. \end{array} \right] \le \mu(\lambda) + \frac{1}{|S_\lambda|}$$

Definition 2.3 (Robustness) *A DTSS is robust if each party in $\mathcal{P}$ can* eventually *reconstruct the secret s, after receiving a sufficient number of other parties' shares and/or obtaining the additional time-locked shares.*

$$\Pr\left[\mathsf{DTSS.Pool}(pp, \mathcal{S}, T_2) \to s : \begin{array}{l} \mathsf{DTSS.Setup}(1^\lambda, T_1, t) \to \{pp, pk, sk\} \\ \mathsf{DTSS.Sharing}(pp, s) \\ \to \{\{C_i\}_{i\in[n]}, \mathbf{v}, \{O_j\}_{j\in[t]}\}, \\ \mathsf{DTSS.ShaRecover}(pp, C_i) \to s_i, \\ \mathsf{DTSS.AddRecover}(pp, pk, \{O_j\}_{j\in[t]}) \\ \to \{s'_j\}_{j\in[t]} \end{array}\right] = 1$$

5.3 DTSS Construction

We present a construction for DTSS in Figure 3. We would like a protocol in which anyone can obtain each additional share s'_j at time $(j+1)T_1$ given that the dealer's role must end with the distribution phase.[7] In a naive way, the dealer should create t puzzles each embedding one additional share to be opened at t different points in time. However, this inefficient solution comes with a high computation cost as anyone wishing to access the shares needs to solve each puzzle separately in parallel, demanding up to $T_1 \sum_{j=1}^{t} j$ operations. To get away with this issue, we use multi-instance time-lock puzzle (MTLP) [1], a primitive allowing sequential (chained) release of solutions where the overall computation cost of solving t puzzles is equal to that of solving only the last one.

Theorem 2. If the multi-instance time-lock puzzle MTLP and timed secret sharing TSS are secure, then our DTSS protocol Π_{DTSS} presented in Figure 3 satisfies the properties described in Section 5.2.

Proof. Privacy follows from that of Π_{TSS} together with the underlying Π_{MTLP} protocol for additional time-locked shares. The t-security is satisfied concerning the gradual release of additional time-locked shares s'_j over time. That is, the adversary can forcibly learn s'_j by $(j+1)T_1$, reducing fault tolerance to $t-j$. The protocol is robust as each party P_i can eventually learn the secret by the time T_2 due to the t additional time-locked shares.

6 Verifiable Timed Secret Sharing (VTSS)

In this section, we present verifiable timed secret sharing (VTSS), an enhanced TSS which considers malicious adversaries. It protects against a malicious dealer who may send incorrect shares (or even no shares) during the distribution phase and against a malicious shareholder who may send an incorrect share during the reconstruction phase.

[7] Without loss of generality we assume $T_2 = (t+1)T_1$, accommodating the periodic release of additional shares.

Π_{DTSS}

1. **Initialization:**
   - Setup: $\mathsf{DTSS.Setup}(1^\lambda, T_1, t) \rightarrow \{pp, pk, sk\}$, the dealer D invokes two algorithms of $\mathsf{TSS.Setup}(1^\lambda, T_1, T_2)$ and $\mathsf{MTLP.Setup}(1^\lambda, T_1, t+1)$, and publishes the set of public parameters pp, pk.
2. **Distribution:**
   - Sharing: $\mathsf{DTSS.Sharing}(pp, s, pk, sk) \rightarrow \{\{C_i\}_{i\in[n]}, \{O_j\}_{j\in[t]}\}$, the dealer D first picks a secret $s \leftarrow \mathbb{Z}_q$ and invokes $\mathsf{TSS.Sharing}(pp, s)$ to generate n locked shares $\{C_i\}_{i\in[n]}$. Moreover, it computes t additional shares $f(a_j) = s'_j$ for $j \in [t]$, where $f(0) = s$ and $\{a_1, \dots, a_t\}$ are some known distinct points. Finally, it invokes $\mathsf{MTLP.Gen}(\mathbf{m}, pk, sk)$, where $\mathbf{m} = \{\perp, s'_1, \dots, s'_t\}$ to generate an MTLP containing $\{s'_j\}_{j\in[t]}$.
3. **Reconstruction:**
   - Share recovery: $\mathsf{DTSS.ShaRecover}(pp, C_i) \rightarrow s_i$, each party P_i runs $\mathsf{TSS.Recover}(pp, C_i)$ to recover their share s_i.
   - Additional share recovery: $\mathsf{DTSS.AddRecover}(pp, pk, \{O_j\}_{j\in[t]}) \rightarrow \{s'_j\}_{j\in[t]}$, anyone wishing to obtain additional time-locked shares $\{s'_j\}_{j\in[t]}$ runs $\mathsf{MTLP.Solve}(pp, \{O_j\}_{j\in[t]})$.
   - Pooling: $\mathsf{DTSS.Pool}(pp, \mathcal{S}, T_2) \rightarrow s$, upon having sufficient number of valid shares (*i.e.*, $\geq t+1$), the reconstrctor $V \in \mathcal{P}$ reconstructs the secret s using Lagrange interpolation at $f(0)$.

Fig. 3: Decrementing-threshold Timed Secret Sharing (DTSS) protocol

6.1 VTSS Definition

We present a formal definition of VTSS. Our definition extends the original verifiable secret sharing (VSS) of Feldman [31], incorporating the notion of time.

Definition 3 (Verifiable Timed Secret Sharing). *A verifiable timed secret sharing (VTSS) scheme involves the following algorithms.*

1. **Initialization:**
   - Setup: $\mathsf{VTSS.Setup}(1^\lambda, T_1, T_2) \rightarrow pp$, *on input security parameter λ, lower time bound T_1 and upper time bound T_2, outputs public parameters pp.*
2. **Distribution:**
   - Sharing: $\mathsf{VTSS.Sharing}(pp, s) \rightarrow \{C_i, \pi_i\}_{i\in[n]}$, *on input pp and a secret s, outputs locked share C_i with time parameter T_1 and a proof of validity π_i for each party $P_i \in \mathcal{P}$.*
   - Share verification: $\mathsf{VTSS.Verify}_1(pp, C_i, \pi_i) \rightarrow 1/0$, *on input pp, C_i, and π_i, checks the validity of share to ensure the locked share C_i is well-formed and contains a valid share of secret s. The algorithm returns 1 if both checks pass. Otherwise, it returns 0.*
3. **Reconstruction:**
   - Recovering: $\mathsf{VTSS.Recover}(pp, C_i) \rightarrow s_i$, *on input pp and C_i, forcibly outputs a share s_i. The algorithm is run by each party P_i.*

– Recovery verification: $\mathsf{VTSS.Verify}_2(pp, s_i, \pi_i) \to 1/0$, *on input pp, s_i, and π_i, checks the validity of submitted share. The algorithm is run by a verifier $V \in \mathcal{P}$.*
– Pooling: $\mathsf{VTSS.Pool}(pp, \mathcal{S}, T_2) \to s$, *on input pp and a set $\mathcal{S}$ of shares (where $|\mathcal{S}| > t$ and $t \in pp$), outputs the secret s if T_2 has not elapsed and $\perp$ otherwise.*

A correct VTSS scheme must satisfy *soundness*, ensuring extractability and verifiability of the shares, *privacy*, and *security.*

Definition 3.1 (Correctness) *A VTSS satisfies correctness if for all secret $s \in S_\lambda$ and a set of shares $|\mathcal{S}| > t$ it holds*

$$\Pr\left[\begin{matrix}\mathsf{VTSS.Verify}_1(pp, C_i, \pi_i) = 1 \\ \mathsf{VTSS.Verify}_2(pp, s_i, \pi_i) = 1 \\ \mathsf{VTSS.Pool}(pp, \mathcal{S}, T_2) \to s\end{matrix} : \begin{matrix}\mathsf{VTSS.Setup}(1^\lambda, T_1, T_2) \to pp, \\ \mathsf{VTSS.Sharing}(pp, s) \to \{C_i, \pi_i\}_{i\in[n]}, \\ \mathsf{VTSS.Recover}(pp, C_i) \to s_i\end{matrix}\right] = 1$$

Definition 3.2 (Soundness) *A VTSS scheme is sound if there exists a negligible function μ such that for all PPT adversaries $\mathcal{A}$ and all $\lambda \in \mathbb{N}$ it holds*

$$\Pr\left[b_1 = 1 \vee b_2 = 1 : \begin{matrix}\mathsf{VTSS.Setup}(1^\lambda, T_1, T_2) \to pp, \\ \mathcal{A}(pp) \to (\{C_i, \pi_i\}_{i\in[n]}, \{s_i, \pi_i'\}), \\ b_1 := \mathsf{VTSS.Verify}_1(pp, C_i, \pi_i) \wedge \nexists s \text{ s.t.} \\ \mathsf{VTSS.Sharing}(pp, s) \to (\{C_i\}_{i\in[n]}, \cdot), \\ b_2 := \mathsf{VTSS.Verify}_2(pp, s_i, \pi_i') \wedge \nexists C_i \text{ s.t.} \\ \mathsf{VTSS.Recover}(pp, C_i) \to s_i\end{matrix}\right] \le \mu(\lambda)$$

Definition 3.3 (Privacy) *A VTSS satisfies privacy if for all parallel algorithms $\mathcal{A}$ whose running time is at most T_1 there exists a simulator Sim and a negligible function μ such that for all secret $s \in S_\lambda$ and all $\lambda \in \mathbb{N}$, it holds*

$$\left|\Pr\left[\mathcal{A}(pp, s, \{C_i, \pi_i\}) = 1 : \begin{matrix}\mathsf{VTSS.Setup}(1^\lambda, T_1, T_2) \to pp, \\ \mathcal{A}(1^\lambda, pp) \to s \\ \mathsf{VTSS.Sharing}(pp, s) \to \{C_i, \pi_i\}_{i\in[n]}\end{matrix}\right] - \Pr\left[\mathcal{A}(pp, s', \{C_j, \pi_j\}) = 1 : \begin{matrix}\mathsf{VTSS.Setup}(1^\lambda, T_1, T_2) \to pp, \\ \mathcal{A}(1^\lambda, pp) \to s' \\ \mathsf{Sim}(pp) \to \{C_j, \pi_j\}_{j\in[n]}\end{matrix}\right]\right| \le \mu(\lambda)$$

Definition 3.4 (Security) *A VTSS satisfies security if there exists a negligible function μ such that for an adversary controlling a subset $\mathcal{S}'$ of parties, where $|\mathcal{S}'| \le t$ and $s \in S_\lambda$ it holds*

$$\Pr\left[\mathcal{A}(pp, \mathcal{S}', T_2) \to s : \begin{matrix}\mathsf{VTSS.Setup}(1^\lambda, T_1, T_2) \to pp, \\ \mathsf{VTSS.Sharing}(pp, s) \to \{C_i, \pi_i\}_{i\in[n]}, \\ \mathsf{VTSS.Recover}(pp, C_i) \to s_i\end{matrix}\right] \le \mu(\lambda) + \frac{1}{|S_\lambda|}$$

$\mathit{\Pi}_{\mathsf{VTSS}}$

1. **Initialization:**
   - Setup: $\mathsf{VTSS.Setup}(1^\lambda, T_1, T_2) \to pp$, let g be a generator of a group $\mathbb{G}$ of order q. The dealer D runs $\mathsf{VTC.Setup}(1^\lambda, T_1)$ and publishes a set of public parameters pp.
2. **Distribution:**
   - Sharing: $\mathsf{VTSS.Sharing}(pp, s) \to \{C_i, \pi_i\}_{i\in[n]}$, D picks a secret $s \xleftarrow{\$} \mathbb{Z}_q$ to be shared among n parties. It samples a degree-t random polynomial $f(\cdot)$ such that $f(0) = s$ and $f(i) = s_i$ for $i \in [n]$. It then commits to f by computing $v_i = g^{s_i}$ and broadcasting $\mathbf{v} = \{v_i\}_{i\in[n]}$. Then, D runs $\mathsf{VTC.Commit}(pp, s_i)$ to create a locked share C_i and a corresponding proof of validity π'_i with respect to v_i, locking the share s_i to be opened forcibly at T_1, $\forall i \in [n]$. Let $\pi_i = \{\pi'_i, \mathbf{v}\}$. D privately sends each party P_i their sharing $\{C_i, \pi'_i\}$.
   - Share verification: $\mathsf{VTSS.Verify}_1(pp, C_i, \pi_i) \to 1/0$, party P_i runs $\mathsf{VTC.Verify}(pp, v_i, C_i, \pi'_i)$ to check the locked share C_i is well-formed and embeds the share s_i corresponding to v_i. They then validate the consistency of the shares by sampling a code word $\mathbf{y}^\perp \in \mathcal{C}^\perp$, where $\mathbf{y}^\perp = \{y_1^\perp, \dots, y_n^\perp\}$, and checking if $\prod_{j=1}^{n} {v_j}^{y_j^\perp} = 1$.
   - Complaint round: If a set of parties of size $\geq t+1$ complain about sharing, then D is disqualified. Otherwise, D reveals the corresponding locked shares with proofs by broadcasting $\{C_i, \pi'_i\}$. If the verification fails (or D does not broadcast), the dealer is disqualified.
3. **Reconstruction:**
   - Recovering: $\mathsf{VTSS.Recover}(pp, C_i) \to s_i$, each P_i wishing to participate in reconstruction runs $\mathsf{VTC.Solve}(pp, C_i)$ to obtain a share s_i.
   - Recovery verification: $\mathsf{VTSS.Verify}_2(pp, s_i, \pi_i) \to 1/0$, for each received share s_i from P_i, the reconstructor checks its validity by computing g^{s_i} and comparing it with v_i.
   - Pooling: $\mathsf{VTSS.Pool}(pp, \mathcal{S}, T_2) \to s$, upon having sufficient number of valid shares (*i.e.*, $\geq t+1$) received before T_2, the reconstrctor (a party in $\mathcal{P}$) reconstructs the secret s using Lagrange interpolation at $f(0)$ or aborts otherwise.

Fig. 4: Verifiable Timed Secret Sharing (VTSS) protocol

6.2 VTSS Construction

We present a protocol for VTSS in Figure 4. Following Feldman VSS [31], we make a crucial change in the protocol to adapt it for VTSS so that the dealer coudl convince each individual shareholder about the validity of their shares. Notably, in VTSS we have the dealer commit to the *shares* rather than the *coefficients* of the Shamir polynomial. This modification has two consequences.

First, it allows shareholders to check the consistency of the shares (*i.e.,* all lie on a polynomial of degree t) using properties of error-correcting code, particularly the Reed-Solomon code [49]. This is due to the equivalency of the Shamir secret sharing with Reed-Solomon encoding observed by [46].[8] We restate the basic fact of linear error correcting code in Lemma 1. We remark that in Feldman VSS the checking of each share is done against the commitment to the whole polynomial, but here it is done with respect to an individual commitment to each share, requiring the this step to ensure the sharing phase has been performed correctly.

Lemma 1. *Let $\mathcal{C}^{\perp}$ be the dual code of $\mathcal{C}$ that is a linear error correcting code over $\mathbb{Z}_q$ of length n. If $\boldsymbol{x} \in \mathbb{Z}_q^n \backslash \mathcal{C}$, and $\boldsymbol{y}^{\perp}$ is chosen uniformly at random from $\mathcal{C}^{\perp}$, the probability that the inner product of the vectors $\langle \boldsymbol{x}, \boldsymbol{y}^{\perp} \rangle = 0$ is exactly $1/q$.*

Second, it enables us to make use of VTC primitive [57] to *non-interactively* ensure each party P_i that they indeed obtains its correct share s_i at T_1. As mentioned, VTC allows committing to a signing key sk where its corresponding public key $pk = g^{sk}$ is publicly known. Our main insight is that we can think of $v_i = g^{s_i}$ published by the dealer as a public key for each share s_i committed by VTC. So, each party P_i can check the verifiability of its locked share C_i while ensuring the consistency of the shares $\{s_i\}_{i\in[n]}$.

Remark 1. We can realize the upper time bound in VTSS similarly to TSS by using the idea of secret sharing with additional shares (Section 5.1). We implicitly assume the additional time-locked shares are honestly generated due to our motivation which is realizing an upper time bound (and thus breaking public goods game). [9]

Theorem 3. If the verifiable timed commitments VTC and Feldman verifiable secret sharing [31] are secure, then verifiable timed secret sharing protocol Π_{VTSS} presented in Figure 4 satisfies soundness, privacy, and security, w.r.t. definitions 3.2, 3.3, and 3.4 respectively.

Proof. Correctness is straightforward. The soundness property of the protocol follows directly from that of the underlying Π_{VTC} primitive for every single share s_i committed with respect to the v_i in $\mathbf{v}$. A maliciously generated $\mathbf{v}$ can

[8] We refer the reader to [18] for a detailed description of the verification procedure.

[9] Should a malicious dealer attempt to misbehave, this assumption could be lifted by using less efficient cryptograhpic protocols.

pass the verification check $\mathsf{VTSS.Verify}_1$ only with probability $1/q$. A maliciously submitted s_i by P_i cannot pass the verification check $\mathsf{VTSS.Verify}_2$, except with negligible probability. The privacy property also follows directly from that of the underlying Π_{VTC} which implies the indistinguishability of a puzzle produced by $\mathsf{VTC.Sharing}$ and the one produced by Sim. Note that the commitment to shares $\mathbf{v}$ does not reveal any information about the secret s under the discrete logarithm assumption. It is important to note that for the assumption to hold the secret s should have a random distribution. Observe that before T_1 the privacy property essentially implies the security; afterward, the security follows directly from that of Feldman VSS due to the security of the commitment $\mathbf{v}$.

7 Publicly Verifiable Timed Secret Sharing (PVTSS)

In this section, we make our timed secret sharing scheme publicly verifiable, meaning that anyone, not only a participating party, is able to verify different phases of the scheme. To achieve this, we use a publicly verifiable secret sharing (PVSS) scheme as the main building block that compels parties to behave correctly by non-interactively proving the validity of the messages sent during the distribution and reconstruction phases.

7.1 PVTSS Definition

In this section, we present a formal definition of PVTSS according to the existing ones in the literature such as [18, 19, 53].

Definition 4 (Publicly Verifiable Timed Secret Sharing). *A PVTSS scheme involves the following algorithms.*

1. **Initialization:**
   – Setup: $\mathsf{PVTSS.Setup}(1^\lambda, T_1, T_2) \to pp$, *on input security parameter λ, lower time bound T_1, and upper time bound T_2, outputs public parameters pp. Each party P_i announces a registered public key pk_i which the corresponding secret key sk_i is only known to them.*
2. **Distribution:**
   – Sharing: $\mathsf{PVTSS.Sharing}(pp, S, \{pk_i\}_{i\in[n]}) \to \{\{C_i\}_{i\in[n]}, \pi_D\}$, *on input pp, $\{pk_i\}_{i\in[n]}$, and a secret S, generates locked encrypted share C_i with time parameter T_1 for each party $P_i \in \mathcal{P}$. It also generates a proof π_D for the validity of shares.*
   – Share verification: $\mathsf{PVTSS.Verify}_1(pp, \{pk_i, C_i\}_{i\in[n]}, \pi_D) \to 1/0$, *on input pp, $\{pk_i, C_i\}_{i\in[n]}$, and π_D, checks the validity of the shares. This includes verifying the published locked encrypted shares are well-formed and contain correct shares of secret S. The algorithm is run by any verifier V.*
3. **Reconstruction:**
   – Recovering: $\mathsf{PVTSS.Recover}(pp, C_i, pk_i, sk_i) \to \{\tilde{s}_i, \pi_i\}$, *on input pp, C_i, pk_i, and sk_i, outputs a decrypted share $\tilde{s}_i$ together with proof π_i of valid decryption. The algorithm is run by each party $P_i \in \mathcal{P}$.*

– Recovery verification: $\mathsf{PVTSS.Verify}_2(pp, C_i, \tilde{s}_i, \pi_i) \to \{0,1\}$, *on input pp, C_i, $\tilde{s}_i$, and π_i, checks the validity of the decryption. The algorithm is run by any verifier V.*
– Pooling: $\mathsf{PVTSS.Pool}(pp, \mathcal{S}, T_2) \to S$, *on input pp and a set $\mathcal{S}$ of decrypted shares $\tilde{s}_i$ (where $|\mathcal{S}| > t$ and $t \in pp$), outputs the secret S if T_2 has not elapsed.*

A PVTSS scheme must satisfy the following properties.

Definition 4.1 (Correctness) *A PVTSS satisfies correctness if for all secret $s \in S_\lambda$ and a set of shares $|\mathcal{S}| > t$ it holds that*

$$\Pr\left[\begin{array}{l|l} \mathsf{PVTSS.Verify}_1(pp, \{C_i\}_{i\in[n]}, & \mathsf{PVTSS.Setup}(1^\lambda, T_1, T_2) \to pp, \\ \pi_D, \{pk_i\}_{i\in[n]}) = 1 & \mathsf{PVTSS.Sharing}(pp, S, \{pk_i\}_{i\in[n]}) \\ \mathsf{PVTSS.Verify}_2(pp, C_i, \tilde{s}_i, \pi_i) = 1 & \to \{\{C_i\}_{i\in[n]}, \pi_D\}, \\ \mathsf{PVTSS.Pool}(pp, \mathcal{S}, T_2) \to S & \mathsf{PVTSS.Recover}(pp, C_i, pk_i, sk_i) \\ & \to \{\tilde{s}_i, \pi_i\} \end{array}\right] = 1$$

Definition 4.2 (Soundness) *A PVTSS scheme is sound if there exists a negligible function μ such that for all PPT adversaries $\mathcal{A}$ and all $\lambda \in \mathbb{N}$ it holds that*

$$\Pr\left[b_1 = 1 \lor b_2 = 1 \; : \begin{array}{l} \mathsf{PVTSS.Setup}(1^\lambda, T_1, T_2) \to pp, \\ \mathcal{A}(pp) \to (\{pk_i, C_i\}_{i\in[n]}, \pi_D, \tilde{s}, \pi), \\ b_1 := \mathsf{PVTSS.Verify}_1(pp, \{pk_i, C_i\}_{i\in[n]}, \pi_D) \\ \land \nexists s \text{ s.t.} \\ \mathsf{PVTSS.Sharing}(pp, S, \{pk_i\}_{i\in[n]}) \\ \to \{\{C_i\}_{i\in[n]}, \cdot\}, \\ b_2 := \mathsf{PVTSS.Verify}_2(pp, C, \tilde{s}, \pi) \land \nexists sk \text{ s.t.} \\ \mathsf{PVTSS.Recover}(pp, C, pk, sk) \to \{\tilde{s}, \cdot\}, \end{array} \right] \le \mu(\lambda)$$

Definition 4.3 (t-Privacy) *A PVTSS satisfies t-privacy if for all parallel algorithms $\mathcal{A}$ whose running time is at most T_1, and set $I \subset [n]$ with $|I| = t+1$, there exists a simulator Sim and a negligible function μ such that for all secret $s \in S_\lambda$ and $\lambda \in \mathbb{N}$ it holds that*

$$\left| \Pr\left[\mathcal{A}(pp, S, \{C_i\}_{i\in[I]}, \pi_D) = 1 \; : \begin{array}{l} \mathsf{PVTSS.Setup}(1^\lambda, T_1, T_2) \to pp, \\ \mathcal{A}(1^\lambda, pp) \to S, \\ \mathsf{PVTSS.Sharing}(pp, S, \{pk_i\}_{i\in[I]}) \\ \to \{\{C_i\}_{i\in[I]}, \pi_D\} \end{array} \right] - \right.$$

$$\left. \Pr\left[\mathcal{A}(pp, S', \{C_j\}_{j\in[I]}, \pi_D) = 1 : \begin{array}{l} \mathsf{PVTSS.Setup}(1^\lambda, T_1, T_2) \to pp, \\ \mathcal{A}(1^\lambda, pp) \to S', \\ \mathsf{Sim}(pp) \to (\{C_j\}_{j\in[I]}, \pi_D) \end{array} \right] \right| \le \mu(\lambda)$$

Definition 4.4 (Security) *A PVTSS satisfies security if there exists a negligible function μ such that for an adversary controlling a set $\mathcal{S}'$ of parties, where $|\mathcal{S}'| \le t$ and $s \in S_\lambda$, together with the public information denoted by PI, it holds that* [10]

$$\Pr\left[\mathcal{A}(pp, \mathcal{S}', \mathsf{PI}, T_2) \to S : \begin{array}{l} \mathsf{PVTSS.Setup}(1^\lambda, T_1, T_2) \to pp, \\ \mathsf{PVTSS.Sharing}(pp, s, \{pk_i\}_{i\in[n]}) \\ \to \{\{C_i\}_{i\in[n]}, \pi_D\}, \\ \mathsf{PVTSS.Recover}(pp, C_i, pk_i, sk_i) \\ \to \{\tilde{s}_i, \pi_i\} \end{array}\right] \le \mu(\lambda) + \frac{1}{|S_\lambda|}$$

An indistinguishability game given in [35,52] and adopted by [18] formalizes the security definition.

7.2 PVTSS Construction

We present a detailed description of the PVTSS protocol in Figure 5. In what follows, we elaborate on several techniques used in our construction. In particular, it turns out that the public verifiability requirement of the scheme demands taking different approaches toward realizing the lower and upper time bounds.

Dealing with a Malicious Dealer. What makes the protection mechanism challenging for PVTSS is that *anyone*, before performing sequential computation, should be able to check the correctness of shares including consistency, validity, and extractability of the shares having a set of *encrypted* shares locked by the dealer. That is to say, a solution should *simultaneously* ensure (1) all shares lie on the same polynomial of degree t, (2) locked encrypted shares contain the committed shares, and (3) shares are obtainable in time T_1, all concerning some public information. We first discuss how to guarantee consistency and verifiability followed by our approach regarding extractability.

Blinded DLEQ. Our solution to meet the first two aforementioned requirements is based on having the dealer blind each encrypted shares $\tilde{s}_i$ using some randomness β_i, put the randomness into a puzzle Z_i, and publish all the puzzles together with locked encrypted shares and commitments for $i \in [n]$. The dealer needs to show that the locked encrypted shares contain the same shares as the commitments, while the consistency of the shares can be checked using the commitments (as discussed in Section 6.2). To do so, we slightly modify the DLEQ proof (Section 3.5) and make it blinded. It allows proving simultaneous knowledge of two witnesses, one of which is common in two statements. The following is a protocol Π_{BDLEQ} for the language

$$L_{\mathsf{BDLEQ}} = \{(g_1, x, g_2, g_3, y) \mid \exists(\alpha, \beta) : x = g_1^\alpha \wedge y = g_2^\alpha g_3^\beta\}$$

1. P chooses two random elements $u_1, u_2 \xleftarrow{\$} \mathbb{Z}_q$, computes $a_1 = g_1^{u_1}$ and $a_2 = g_2^{u_1} g_3^{u_2}$, and sends them to V.

[10] This property is presented as IND1-Secrecy in [35,52].

2. V sends back a randomly chosen challenge $c \xleftarrow{\$} \mathbb{Z}_q$.
3. P computes $r_1 = u_1 + c\alpha$ and $r_2 = u_2 + c\beta$ and sends them to V.
4. V checks if both $g_1^{r_1} = a_1 x^c$ and $g_2^{r_1} g_3^{r_2} = a_2 y^c$ hold.

Theorem 4. Protocol Π_{BDLEQ} is a public-coin honest-verifier zero-knowledge argument of knowledge corresponding to the language L_{BDLEQ}.

Proof. We show that the Π_{BDLEQ} satisfies the properties of a Sigma protocol. Completeness holds, as

$$g_1^{r_1} = g_1^{u_1+c\alpha} = g_1^{u_1} g_1^{c\alpha} = a_1 x^c$$
$$g_2^{r_1} g_3^{r_2} = g_2^{u_1+c\alpha} g_3^{u_2+c\beta} = g_2^{u_1} g_3^{u_2} (g_2^{u_1} g_3^{u_2})^c = a_2 y^c$$

For knowledge soundness, given two accepting transcripts $(a_1, a_2; c; r_1, r_2)$ and $(a_1, a_2; c'; r_1', r_2')$ the witness (α, β) can be found as follows

$$g_1^{r_1} = a_1 x^c,\ g_2^{r_1} g_3^{r_2} = a_2 y^c \quad ; \quad g_1^{r_1'} = a_1 x^{c'},\ g_2^{r_1'} g_3^{r_2'} = a_2 y^{c'}$$
$$g_1^{r_1 - r_1'} = x^{c-c'} \Leftrightarrow x = g_1^{\frac{r_1 - r_1'}{c-c'}}$$
$$g_2^{r_1 - r'_1} g_3^{r_2 - r_2'} = y^{c-c'} \Leftrightarrow y = g_2^{\alpha} g_3^{\frac{r_2 - r_2'}{c-c'}}$$

Hence, the witness β can be found as $\beta = (r_2 - r_2')/(c - c')$ given the witness $\alpha = (r_1 - r_1')/(c - c')$.
Let c be a given challenge. Zero-knowledge property is implied by the fact that the following two distributions, namely real protocol distribution and simulated distribution, are identically distributed.

$$\texttt{Real} : \{(a_1, a_2; c; r_1, r_2) : u_1, u_2 \xleftarrow{\$} \mathbb{Z}_q, a_1 = g_1^{u_1}, a_2 = g_2^{u_1} g_3^{u_2}; r_1 = u_1 + c\alpha, r_2 = u_2 + c\beta\}$$
$$\texttt{Sim} : \{(a_1, a_2; c; r_1, r_2) : r_1, r_2 \xleftarrow{\$} \mathbb{Z}_q; a_1 = g_1^{r_1} x^{-c}, a_2 = g_2^{r_1} g_3^{r_2} y^{-c}\}$$

Note that the probability of occurring for each distribution is the same and equals $1/q^2$.

Cut-and-choose. The dealer needs to convince the parties they can obtain their shares at time T_1. This is equivalent to saying that Z_i has indeed the value β_i embedded. A natural way to show the correctness of puzzle generation is by utilizing the cut-and-choose technique as in previous works [9,56]. This technique forces a sender to behave correctly by randomly opening a (fixed) set of puzzles it has already sent to the receiver based on the receiver's choice.

We remark that it is possible to deploy the cut-and-choose technique in our construction without sacrificing security. Given that opening just reveals a (random) set of size t of encrypted shares, we are still guaranteed that the secret remains hidden up to time T_1 as $t+1$ shares are needed for reconstruction. Each party is supposed to open their corresponding locked encrypted share, which is not among the opened ones by the dealer. Given public verification, we can stick to an honest majority assumption (*i.e.,* $\mathrm{t} < \mathrm{n}/2$) while ensuring soundness. We can borrow concrete numbers from related work in the same setting: For example, setting $n = 40$ would give a soundness error of 10^{-12} (Table 3, [57]).

Realizing an Upper Time Bound. Due to the public verifiability, PVTSS protocol is executed over a public bulletin board. As a result, the secret may be reconstructed/used by any external party after T_2. This demands taking a different approach towards realizing the upper time bound to make it more strict. Our solution is based on deploying *short-lived proofs* (SLPs) [5]. We Observe that the use of SLPs allows tying the *correctness* of the system to time, meaning that the secret is only guaranteed to be correct if it is reconstructed before the upper time bound. Correctness intuitively states if the distribution phase succeeds, then the reconstruction phase will output the *same* secret initially shared by the dealer. Let us now briefly explain how we make use of SLPs in our construction.

Upper time bound with SLPs. Our approach is to take advantage of the *forgeability* property of SLPs in the PVTSS construction. We piggyback on the *proof of decryptions* π_i generated by each party P_i as part of the reconstruction phase, turning them into SLPs where their expiration time matches the upper time bound T_2. Therefore, given the properties of short-lived proofs and also relying on that the secret has *uniformly random* distribution in Scrape PVSS,[11] the correctness of a share submitted by a party P_i is only guaranteed if being observed before T_2, otherwise it could be an invalid share accompanied with a valid proof. A short-lived proof for any arbitrary relation R for which there exists a Sigma protocol can be efficiently constructed [5]. For completeness, we present the short-lived proof for a relation R using pre-computed VDFs in Figure 6.

In our protocol, we make a black box use of short-lived DLEQ proof generation denoted by $\mathsf{DLEQ.SLP}$ and verification denoted by $\mathsf{DLEQ.SLV}$. It is required that the beacon value b used to compute π_i is not known until the time T_1, with $T = T_2 - T_1$ being the time parameter for the underlying VDF. Therefore, anyone verifying the proof before T_2 knows that it could have not been computed through forgery. We highlight that, to deploy short-lived proofs we need to use the DDH-based version of Scrape PVSS which its DLEQ proof comes with *knowledge soundness* property.

Remark 2. Several recent works focus on the notion of forgeability over time, particularly for developing short-lived signature or forward-forgeable signature [5,55]. To the best of our knowledge, Arun et al. [5] is the only one exploring the time-based forgeability in proof systems. This in turn enables us to deploy their primitive to provide the upper time bound for PVTSS, binding the correctness of the secret reconstruction to time.

Remark 3. We do not assume the availability of an *online* verifier who observes the protocol over time. In fact, due to the characteristic of SLPs, their use is

[11] This essentially implies any set of shares is indistinguishable from a set of random strings. Note that in normal Shamir secret sharing this is limited to a set of size at most t shares as the secret is not uniformly distributed [13].

meaningful when the verifier does not necessarily remain online during the reconstruction period $[T_1, T_2]$; otherwise, it can always reject the proofs sent afterward, negating the forgeability property. Moreover, as pointed out in [5], convincingly timestamping the messages published on the bulletin board is opposed to the usability of SLPs.

In our PVTSS construction, we explicitly feed the upper time bound T_2 and a beacon value b in two algorithms, $\mathsf{PVTSS.Recover}$ and $\mathsf{PVTSS.Verify}_2$. This is essentially due to the necessity of the knowledge of time parameters $T = T_2 - T_1$ and b for short-lived proof generation and verification. Moreover, as discussed in [5], T does not need to be hardcoded when $\mathsf{PVTSS.Setup}$ is run. This allows the use of VDFs with any time parameter $T' > T$, while still generating short-lived proofs with respect to time T. That is, even if different parties use different time parameters with $T' > T$ for their VDF evaluations, only those proofs observed before time T are convincing.

Theorem 5. If the time-lock puzzle TLP, short-lived proofs SLP, and Scrape PVSS are secure, then publicly verifiable timed secret sharing protocol $\mathit{\Pi}_{\mathit{PVTSS}}$ (presented in Figure 5) satisfies soundness, t-privacy, and security, w.r.t. definitions 4.2, 4.3, and 4.4 respectively.

Proof. Before T_2, the correctness is straightforward. Afterward, the correctness may fail with overwhelming probability due to the forgeability and indistinguishability properties of the underlying SLPs together with the uniform distribution of the secret s (and thus shares s_i). Anyone observing the public bulletin board after T_2 cannot distinguish an erroneous decryption share $\tilde{s_i}$ from a valid one as both pass the verification check $\mathsf{PVTSS.Verify}_2$. The soundness of the protocol follows from the underlying cut-and-choose argument and BDLEQ's soundness property. Note that by choosing parameters properly the soundness error for the cut-and-choose technique can be negligible in n. The property of t-privacy stems from the fact that given a random set of t opened locked encrypted shares produced by $\mathsf{VTC.Sharing}$, the simulator Sim can produce a locked encrypted share indistinguishable from any locked encrypted share that remained unopened due to the privacy properties of the underlying TLP. Security of the protocol follows directly from the underlying PVSS protocol. Note that blinded encrypted shares c_i distributed by the dealer provide semantic security due to the independent randomness β_i, while the original encryption method used in [18] to generate $\hat{s_i}$ is not IND-CPA-secure.

8 Discussion

In the following, we explore and discuss several aspects of our constructions.

On the setup phase. In all of our schemes, Setup algorithm is responsible for generating a set of public parameters pp, encapsulating the parameters for the underlying secret sharing and time-based cryptographic primitive. In particular, our VTSS construction in Figure 4 requires a trusted setup to generate

$\mathit{\Pi}_{\mathsf{PVTSS}}$

1. **Initialization:**
   - Setup: $\mathsf{PVTSS.Setup}(1^\lambda, T_1) \to pp$, the public parameters pp include independently chosen generators g_1, g_2, g_3 in a DDH-hard group $\mathbb{G}$, a field $\mathbb{Z}_q$, a hash function $H : \{0,1\}^* \to I \subset [n]$ with $|I| = t$, and a public bulletin board. Each party P_i announces a registered public key $pk_i = g_1^{sk_i}$ which its secret key sk_i is only known to them.
2. **Distribution:**
   - Sharing: $\mathsf{PVTSS.Sharing}(pp, S, \{pk_i\}_{i\in[n]}) \to \{\{C_i\}_{i\in[n]}, \pi_D\}$, the dealer D randomly chooses $s \xleftarrow{\$} \mathbb{Z}_q$ and defines the secret $S = g_1^s$ to be shared among n parties with public keys $\{pk_i\}_{i\in[n]}$. D computes Shamir shares $f(i) = s_i$, commitments $v_i = g_2^{s_i}$, and encrypted shares $\hat{s}_i = pk_i^{s_i}$ for all $i \in [n]$ using a degree-t Shamir polynomial $f(\cdot)$, where $f(0) = s$. It blinds the encrypted shares $\{\hat{s}_i\}_{i\in[n]}$ using some independent randomness β_i, resulting in $\{c_i\}_{i\in[n]}$, where $c_i = \hat{s}_i g_3^{\beta_i}$. The dealer then locks every randomness β_i in a TLP by running $\mathsf{TLP.Gen}(1^\lambda, T_1, \beta_i)$. Let denote $C_i = \{c_i, Z_i\}$. To show the consistency and validity of the locked encrypted shares, D runs $\mathit{\Pi}_{\mathsf{BDLEQ}}$, resulting in proof $\pi =: (v_i, e, r_{1,i}, r_{2,i})$ for $i \in [n]$. Finally, D publishes the locked encrypted shares $\{C_i\}_{i\in[n]}$ and proof π_D on a public bulletin board. Moreover, D computes $H(\{C_i\}_{i\in[n]}, \pi) \to I$ as a random challenge (for cut and choose) and outputs $\pi_D = \{I, \pi, \beta_i, \hat{s}_i\}_{i\in[I]}$.
   - Share verification: $\mathsf{PVTSS.Verify}_1(pp, \{C_i\}_{i\in[n]}, \pi_D, \{pk_i\}_{i\in[n]}) \to 1/0$, the verifier V first validates the consistency of the shares by sampling a code word $\mathbf{y}^\perp \in C^\perp$, where $\mathbf{y}^\perp = \{y_1^\perp, \ldots, y_n^\perp\}$, and checking if $\prod_{j=1}^n v_j{}^{y_j^\perp} = 1$. V then checks the proof π_D is valid. After re-computing I, the verifier checks the puzzles are correctly constructed by invoking $\mathsf{TLP.Gen}$ algorithm and comparing the encrypted share sent by the dealer with the one being unlocked using β_i.
3. **Reconstruction:**
   - Recovering: $\mathsf{PVTSS.Recover}(pp, C_i, pk_i, sk_i, b, T_2) \to \{\tilde{s}_i, \pi_i\}$, after checking the validity of sharing phase, any party P_i wishing to obtain their share at T_1, unlocks the blinding factor β_i by running $\mathsf{TLP.Solve}(pp, Z_i)$, and obtains their share $\tilde{s}_i$ after decrypting $\hat{s}_i$ as $\tilde{s}_i = \hat{s}_j{}^{1/sk_i}$. Then, the party P_i reveals the share $\tilde{s}_i$ together with a short-lived proof $\pi_i =: \{\mathsf{DLEQ.SLP}(sk_i, g_1, pk_i, \tilde{s}_i, \hat{s}_i), \beta_i\}$ of valid decryption. Note that $\mathsf{DLEQ.SLP}$ involves calling $\mathsf{SLP.Gen}$ for the relation $R_{DLEQ} = \{(g_1, pk_i, \tilde{s}_i, \hat{s}_i; sk_i)\}$ given a beacon value b publicly known no sooner than T_1.
   - Recovery verification: $\mathsf{PVTSS.Verify}_2(pp, C_i, \tilde{s}_i, \pi_i, b, T_2) \to 1/0$, any (external) verifier V can check the validity of published share $\tilde{s}_i$ via $\mathsf{DLEQ.SLV}(\pi_i, g_1, pk_i, \tilde{s}_i, \hat{s}_i)$. Note that having C_i, the verifier first obtains $\hat{s}_i$ with β_i.
   - Pooling: $\mathsf{PVTSS.Pool}(pp, \mathcal{S}, T_2) \to S$, upon having sufficient number of shares ($\geq t+1$) received before time T_2, denoted by $\mathcal{S}$, anyone can reconstruct the secret $S = g_1^s$ using Lagrange interpolation in the exponent.

Fig. 5: Publicly Verifiable Timed Secret Sharing (PVTSS) protocol

the parameters for the underlying VTC primitive. This is due to the linearly homomorphic TLP of [43] deployed in VTC construction. The functionality of the primitive depends on such an assumption; otherwise, either the puzzle is not solvable or one can efficiently solve it upon receipt. Using class groups of imaginary quadratic fields [16] as a family of groups of unknown order instead of the well-known RSA group is an option to reduce the trust, but comes with higher (offline) computational investment for the puzzle generator to compute the parameters through sequential computation [43]. Deploying the class groups solely does not eliminate the need for a trusted setup as it is still feasible that a malicious sender fools a receiver into accepting locked shares that will never be opened. Moreover, the VDF used in SLPs can be instantiated efficiently via class groups [60] without making any trusted setup assumption.

On the use of SLPs. As previously mentioned, the use of SLPs necessitates the availability of a reconstructor prior to the upper time bound for a correct reconstruction. Moreover, we deploy short-lived proofs using precomputed VDFs [5] which do not offer reusable forgeability, *i.e.,* forging a proof for any statement v without computing a new VDF. However, this essentially fits a secret sharing setting (in particular, PVSS) which is inherently one-time use, *i.e.,* after reconstruction the secret is known and the system is not reusable.

Failure probability. Although just some chances of reconstruction failure after T_2 should be enough to break the public goods game, here We briefly analyze the probability of a reconstruction failure after T_2 when deploying SLPs with an honest majority assumption. Let t be the number of adversarial shares and n be the total number of shares publicly available. Given that the incorporation of even *one* invalid share results in an invalid reconstruction and the fact that shares are uniformly distributed, the success probability can be computed as $p = \frac{p_1}{p_2}$, where $p_1 = \binom{n-t}{t+1}$ and $p_2 = \binom{n}{t+1}$. We can easily show that by a proper choice of the parameters n, t the reconstruction fails with overwhelming probability. Setting $t = \lceil \frac{n}{2} \rceil - 1$, we have $p \leq n2^{-(\lceil \frac{n}{2} \rceil + 1)}$ which is a negligible value in λ for a choice of $n = \lambda$.

Breaking public goods game. A common method to break the public goods game is to reward those parties who publish their shares sooner via harnessing the financial capabilities of the blockchain systems [6, 11, 39]. That is, the shareholder receives some *reward* if their submitted share is among the first $t+1$ shares published on the chain. This in turn creates a race and motivates the shareholder to show up sooner. Our two solutions, namely gradual release of additional shares and using short-lived proofs, can be considered as *orthogonal* methods that are *off-chain.* More precisely, the former approach essentially binds the security of the protocol to time by causing security reduction over time. The latter approach binds the correctness of the protocol to time, meaning that if the reconstruction does not occur sometime before T_2, then the correctness is not guarantee.[12]. As a

[12] This is a generic argument, independent of the adversarial behavior.

result, in both approaches the shareholders are pushed to act as soon as possible to avoid any pitfalls.

Acknowledgements. The authors would like to thank Dan Ristea for the helpful discussions and Asiacrypt 2024 anonymous reviewers for their useful comments. Aydin Abadi was supported in part by REPHRAIN: The National Research Centre on Privacy, Harm Reduction and Adversarial Influence Online, under UKRI grant: EP/V011189/1.

References

1. A. Abadi and A. Kiayias. Multi-instance publicly verifiable time-lock puzzle and its applications. In *International Conference on Financial Cryptography and Data Security*, pages 541–559. Springer, 2021.
2. A. V. Aho and J. E. Hopcroft. *The design and analysis of computer algorithms.* Pearson Education India, 1974.
3. G. Almashaqbeh, F. Benhamouda, S. Han, D. Jaroslawicz, T. Malkin, A. Nicita, T. Rabin, A. Shah, and E. Tromer. Gage mpc: Bypassing residual function leakage for non-interactive mpc. *Proceedings on Privacy Enhancing Technologies*, 2021.
4. M. Archetti and I. Scheuring. Game theory of public goods in one-shot social dilemmas without assortment. *Journal of theoretical biology*, 299:9–20, 2012.
5. A. Arun, J. Bonneau, and J. Clark. Short-lived zero-knowledge proofs and signatures. In *Advances in Cryptology–ASIACRYPT 2022: 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5–9, 2022, Proceedings, Part III*, pages 487–516. Springer, 2023.
6. Z. Avarikioti, E. Kokoris-Kogias, R. Wattenhofer, and D. Zindros. B rick: Asynchronous incentive-compatible payment channels. In *Financial Cryptography and Data Security: 25th International Conference, FC 2021, Virtual Event, March 1–5, 2021, Revised Selected Papers, Part II 25*, pages 209–230. Springer, 2021.
7. C. Badertscher, P. Gaži, A. Kiayias, A. Russell, and V. Zikas. Dynamic ad hoc clock synchronization. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 399–428. Springer, 2021.
8. A. Bagherzandi, S. Jarecki, N. Saxena, and Y. Lu. Password-protected secret sharing. In *Proceedings of the 18th ACM conference on Computer and Communications Security*, pages 433–444, 2011.
9. W. Banasik, S. Dziembowski, and D. Malinowski. Efficient zero-knowledge contingent payments in cryptocurrencies without scripts. In *Computer Security–ESORICS 2016: 21st European Symposium on Research in Computer Security, Heraklion, Greece, September 26-30, 2016, Proceedings, Part II 21*, pages 261–280. Springer, 2016.
10. C. Baum, B. David, R. Dowsley, R. Kishore, J. B. Nielsen, and S. Oechsner. Craft: C omposable r andomness beacons and output-independent a bort mpc f rom t ime. In *IACR International Conference on Public-Key Cryptography*, pages 439–470. Springer, 2023.
11. D. Beaver, K. Chalkias, M. Kelkar, L. K. Kogias, K. Lewi, L. de Naurois, V. Nicolaenko, A. Roy, and A. Sonnino. Strobe: Stake-based threshold random beacons. *Cryptology ePrint Archive*, 2021.

12. A. Beimel, Y. Ishai, and E. Kushilevitz. Ad hoc psm protocols: Secure computation without coordination. In *Advances in Cryptology–EUROCRYPT 2017: 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30–May 4, 2017, Proceedings, Part III 36*, pages 580–608. Springer, 2017.
13. M. Bellare, W. Dai, and P. Rogaway. Reimagining secret sharing: Creating a safer and more versatile primitive by adding authenticity, correcting errors, and reducing randomness requirements. *Proceedings on Privacy Enhancing Technologies*, 2020(4), 2020.
14. D. Boneh and M. Naor. Timed commitments. In *Annual international cryptology conference*, pages 236–254. Springer, 2000.
15. J. Bonneau, J. Clark, and S. Goldfeder. On bitcoin as a public randomness source. *Cryptology ePrint Archive*, 2015.
16. J. Buchmann and H. C. Williams. A key-exchange system based on imaginary quadratic fields. *Journal of Cryptology*, 1(2):107–118, 1988.
17. J. Burdges and L. D. Feo. Delay encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 302–326. Springer, 2021.
18. I. Cascudo and B. David. Scrape: Scalable randomness attested by public entities. In *International Conference on Applied Cryptography and Network Security*, pages 537–556. Springer, 2017.
19. I. Cascudo, B. David, L. Garms, and A. Konring. Yolo yoso: fast and simple encryption and secret sharing in the yoso model. In *Advances in Cryptology–ASIACRYPT 2022: 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5–9, 2022, Proceedings, Part I*, pages 651–680. Springer, 2023.
20. M. Chase, H. Davis, E. Ghosh, and K. Laine. Acsesor: A new framework for auditable custodial secret storage and recovery. *Cryptology ePrint Archive*, 2022.
21. D. Chaum and T. P. Pedersen. Wallet databases with observers. In *Annual international cryptology conference*, pages 89–105. Springer, 1992.
22. M. Chen, C. Hazay, Y. Ishai, Y. Kashnikov, D. Micciancio, T. Riviere, A. Shelat, M. Venkitasubramaniam, and R. Wang. Diogenes: lightweight scalable rsa modulus generation with a dishonest majority. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 590–607. IEEE, 2021.
23. Y.-H. Chen and Y. Lindell. Feldman's verifiable secret sharing for a dishonest majority. *IACR Communications in Cryptology*, 1(1), 2024.
24. A. R. Choudhuri, S. Garg, J. Piet, and G.-V. Policharla. Mempool privacy via batched threshold encryption: Attacks and defenses. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 3513–3529. USENIX Association, 2024.
25. P. Chvojka, T. Jager, D. Slamanig, and C. Striecks. Versatile and sustainable timed-release encryption and sequential time-lock puzzles. In *European Symposium on Research in Computer Security*, pages 64–85. Springer, 2021.
26. J. Clark and U. Hengartner. On the use of financial data as a random beacon. *Evt/wote*, 89, 2010.
27. P. Daian, S. Goldfeder, T. Kell, Y. Li, X. Zhao, I. Bentov, L. Breidenbach, and A. Juels. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 910–927. IEEE, 2020.
28. Y. Dodis and D. H. Yum. Time capsule signature. In *International Conference on Financial Cryptography and Data Security*, pages 57–71. Springer, 2005.

29. Y. Doweck and I. Eyal. Multi-party timed commitments. *arXiv preprint arXiv:2005.04883*, 2020.
30. S. D. Dwilson. What happened to julian assange's dead man's switch for the wikileaks insurance files? `https://heavy.com/news/2019/04/julian-assange-dead-mans-switch-wikileaks-insurance-files/`, Apr. 2019. Section: News.
31. P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pages 427–438. IEEE, 1987.
32. A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the theory and application of cryptographic techniques*, pages 186–194. Springer, 1986.
33. J. A. Garay and M. Jakobsson. Timed release of standard digital signatures. In *International Conference on Financial Cryptography*, pages 168–182. Springer, 2002.
34. J. Y. Halpern, B. Simons, R. Strong, and D. Dolev. Fault-tolerant clock synchronization. In *Proceedings of the third annual ACM symposium on Principles of distributed computing*, pages 89–102, 1984.
35. S. Heidarvand and J. L. Villar. Public verifiability from pairings in secret sharing schemes. In *International Workshop on Selected Areas in Cryptography*, pages 294–308. Springer, 2008.
36. L. Heimbach and R. Wattenhofer. Sok: Preventing transaction reordering manipulations in decentralized finance. *arXiv preprint arXiv:2203.11520*, 2022.
37. S. Jarecki, A. Kiayias, and H. Krawczyk. Round-optimal password-protected secret sharing and t-pake in the password-only model. In *Advances in Cryptology–ASIACRYPT 2014: 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, ROC, December 7-11, 2014, Proceedings, Part II 20*, pages 233–253. Springer, 2014.
38. A. Kate, G. M. Zaverucha, and I. Goldberg. Constant-size commitments to polynomials and their applications. In *Advances in Cryptology-ASIACRYPT 2010: 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings 16*, pages 177–194. Springer, 2010.
39. E. Kokoris-Kogias, E. C. Alp, L. Gasser, P. Jovanovic, E. Syta, and B. Ford. Calypso: private data management for decentralized ledgers. *Proceedings of the VLDB Endowment*, 14(4):586–599, 2020.
40. Y. Lindell. Fast cut-and-choose-based protocols for malicious and covert adversaries. *Journal of Cryptology*, 29(2):456–490, 2016.
41. A. F. Loe, L. Medley, C. O'Connell, and E. A. Quaglia. Tide: A novel approach to constructing timed-release encryption. *Cryptology ePrint Archive*, 2021.
42. Y. Ma, J. Woods, S. Angel, A. Polychroniadou, and T. Rabin. Flamingo: Multi-round single-server secure aggregation with applications to private federated learning. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 477–496. IEEE, 2023.
43. G. Malavolta and S. A. K. Thyagarajan. Homomorphic time-lock puzzles and applications. In *Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part I*, pages 620–649. Springer, 2019.
44. D. Malkhi and P. Szalachowski. Maximal extractable value (mev) protection on a dag. In *4th International Conference on Blockchain Economics, Security and Protocols*, page 1, 2023.

45. Y. Manevich and A. Akavia. Cross chain atomic swaps in the absence of time via attribute verifiable timed commitments. In *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*, pages 606–625. IEEE, 2022.
46. R. J. McEliece and D. V. Sarwate. On sharing secrets and reed-solomon codes. *Communications of the ACM*, 24(9):583–584, 1981.
47. L. Medley, A. F. Loe, and E. A. Quaglia. Sok: Delay-based cryptography. In *2023 IEEE 36th Computer Security Foundations Symposium (CSF)*, pages 169–183. IEEE, 2023.
48. K. Pietrzak. Simple verifiable delay functions. In *10th innovations in theoretical computer science conference (itcs 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.
49. I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics*, 8(2):300–304, 1960.
50. R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock puzzles and timed-release crypto. 1996.
51. A. Rondelet and Q. Kilbourn. Threshold encrypted mempools: Limitations and considerations. *arXiv preprint arXiv:2307.10878*, 2023.
52. A. Ruiz and J. L. Villar. Publicly verifiable secret sharing from paillier's cryptosystem. In *WEWoRC 2005–Western European Workshop on Research in Cryptology*. Gesellschaft für Informatik eV, 2005.
53. B. Schoenmakers. A simple publicly verifiable secret sharing scheme and its application to electronic voting. In *Annual International Cryptology Conference*, pages 148–164. Springer, 1999.
54. A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.
55. M. A. Specter, S. Park, and M. Green. {KeyForge}:{Non-Attributable} email from {Forward-Forgeable} signatures. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1755–1773, 2021.
56. S. Srinivasan, J. Loss, G. Malavolta, K. Nayak, C. Papamanthou, and S. A. Thyagarajan. Transparent batchable time-lock puzzles and applications to byzantine consensus. In *IACR International Conference on Public-Key Cryptography*, pages 554–584. Springer, 2023.
57. S. A. K. Thyagarajan, A. Bhat, G. Malavolta, N. Döttling, A. Kate, and D. Schröder. Verifiable timed signatures made practical. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1733–1750, 2020.
58. S. A. K. Thyagarajan, G. Castagnos, F. Laguillaumie, and G. Malavolta. Efficient cca timed commitments in class groups. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 2663–2684, 2021.
59. S. A. K. Thyagarajan, T. Gong, A. Bhat, A. Kate, and D. Schröder. Opensquare: Decentralized repeated modular squaring service. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 3447–3464, 2021.
60. B. Wesolowski. Efficient verifiable delay functions. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 379–407. Springer, 2019.
61. H. Zhang, L.-H. Merino, Z. Qu, M. Bastankhah, V. Estrada-Galiñanes, and B. Ford. F3b: A low-overhead blockchain architecture with per-transaction front-running protection. In *5th Conference on Advances in Financial Technologies*, 2023.

A Cryptographic Primitives and Definitions

A.1 Time-lock Puzzles (TLP)

Definition 5 (Time-lock Puzzle). *A time-lock puzzle (TLP) consists of the following two algorithms:*

1. $\mathsf{TLP.Gen}(1^\lambda, T, s) \rightarrow Z$*, a probabilistic algorithm that takes time parameter* T *and a secret* s*, and generates a puzzle* Z*.*
2. $\mathsf{TLP.Solve}(T, Z) \rightarrow s$*, a deterministic algorithm that solves the puzzle* Z *and retrieves the secret* s*.*

We recall the correctness and security definition of standard time-lock puzzles:

Correctness [43]**.** A TLP scheme is correct if for all $\lambda \in \mathbb{N}$, all polynomials $T(\cdot)$ in λ, and all $s \in S_\lambda$, it holds that

$$\Pr\left[\mathsf{TLP.Solve}(T(\lambda), Z) \rightarrow s : \mathsf{TLP.Gen}(1^\lambda, T(\lambda), s) \rightarrow Z\right] = 1$$

Security [43]**.** A TLP scheme is secure with gap $\epsilon < 1$ if there exists a polynomial $\tilde{T}(\cdot)$ such that for all polynomials $T(\cdot) \geq \tilde{T}(\cdot)$ and every polynomial-size adversary $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda \in N}$ of depth $\leq T^\epsilon(\lambda)$, there exists a negligible function $\mu(\cdot)$, such that for all $\lambda \in \mathbb{N}$ and $s_0, s_1 \in \{0,1\}^\lambda$ it holds that $\Pr\left[\mathcal{A}(Z) \rightarrow b : \mathsf{TLP.Gen}(1^\lambda, T(\lambda), s_b) \rightarrow Z, b \stackrel{\$}{\leftarrow} \{0,1\}\right] \leq \frac{1}{2} + \mu(\lambda)$.

In particular, the seminal work of [50] introduced the notion of *encrypting to the future* using an RSA-based TLP. Loosely speaking, the sender encrypts a message m under a key k derived from the solution s to a puzzle Z. So, anyone can obtain m after running $\mathsf{TLP.Solve}(T, Z)$, and learning the key.

A.2 Homomorphic Time-Lock Puzzles (HTLP)

Definition 6 (Homomorphic Time-Lock Puzzles [43]**).** *Let* $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ *be a class of circuits and* S_λ *be a finite domain. A homomorphic time-lock puzzle (HTLP) with respect to* $\mathcal{C}$ *and with solution space* S_λ *is a tuple of algorithms (HTLP.Setup, HTLP.Gen, HTLP.Solve, HTLP.Eval) as follows.*

1. $\mathsf{HTLP.Setup}(1^\lambda, T) \rightarrow pp$*, a probabilistic algorithm that takes a security parameter* 1^λ *and time parameter* T*, and generates public parameters* pp*.*
2. $\mathsf{HTLP.Gen}(pp, s) \rightarrow Z$*, a probabilistic algorithm that takes public parameters* pp *and a solution* $s \in S_\lambda$*, and generates a puzzle* Z*.*
3. $\mathsf{HTLP.Solve}(pp, Z) \rightarrow s$*, a deterministic algorithm that takes public parameters* pp *and puzzle* Z*, and retrieves a secret* s*.*
4. $\mathsf{HTLP.Eval}(C, pp, Z_1, \ldots, Z_n) \rightarrow Z'$*, a probabilistic algorithm that takes a circuit* $C \in \mathcal{C}_\lambda$ *and a set of* n *puzzles* $(Z_1, \ldots, Z_n)$*, and outputs a puzzle* Z'*.*

Security [43]**.** An HTLP scheme ($\mathsf{HTLP.Setup}, \mathsf{HTLP.Gen}, \mathsf{HTLP.Solve}, \mathsf{HTLP.Eval}$) is secure with gap $\epsilon < 1$ if there exists a polynomial $\tilde{T}(\cdot)$ such that for all polynomials $T(\cdot) \geq \tilde{T}(\cdot)$ and every polynomial-size adversary $(\mathcal{A}_1, \mathcal{A}_2) = \{(\mathcal{A}_1, \mathcal{A}_2)_\lambda\}_{\lambda \in N}$ where the depth of $\mathcal{A}_2$ is bounded from above by $T^\epsilon(\lambda)$, there exists a negligible function $\mu(\cdot)$, such that for all $\lambda \in \mathbb{N}$ it holds that

$$\Pr\left[\mathcal{A}_2(pp, Z, \tau) \to b : \begin{array}{l} \mathcal{A}_1(1^\lambda) \to (\tau, s_0, s_1) \\ \mathsf{HTLP.Setup}(1^\lambda, T(\lambda)) \to pp \\ b \xleftarrow{\$} \{0,1\} \\ \mathsf{HTLP.Gen}(pp, s_b) \to Z \end{array}\right] \leq \frac{1}{2} + \mu(\lambda)$$

The puzzle is defined over a group of unknown order and is of the form $Z = (u, v)$, where $u = g^r$ and $v = h^{r \cdot N}(1+N)^s$. One notable point regarding the construction is that a trusted setup assumption is needed to generate the public parameters $pp = (T, N, g, h)$, where N is a safe modulus [13] and $h = g^{2^T}$. Such a setup phase is responsible for generating the parameters as specified and keeping the random coins secret; otherwise, either the puzzle is not solvable or one can efficiently solve it in time $t \ll T$. Having said that, the authors in [43] point out that this assumption can be removed if construction gets instantiated over class groups instead of an RSA group of unknown order. However, this comes at the cost of a higher computational overhead by the puzzle generator.

A.3 Multi-instance Time-lock Puzzle (MTLP)

Definition 7 (Multi-instance Time-lock Puzzle [1]**).** *A Multi-instance Time-lock Puzzle (MTLP) consists of the following five algorithms.*

1. $\mathsf{MTLP.Setup}(1^\lambda, T, z) \to \{pk, sk, \mathbf{d}\}$, *a probabilistic algorithm that takes a security parameter* λ*, a time parameter* T*, and the number of puzzle instances* z*, and outputs a key pair* (pk, sk) *and a secret witness vector* $\boldsymbol{d}$*.*
2. $\mathsf{MTLP.Gen}(\mathbf{m}, pk, sk, \mathbf{d}) \to \{\mathbf{o}, \mathbf{h}\}$, *a probabilistic algorithm that takes a message vector* $\boldsymbol{m}$*, the public-private key* (pk, sk)*, secret witness vector* $\boldsymbol{d}$*, and outputs a puzzle vector* $\boldsymbol{o}$ *and a commitment vector* $\boldsymbol{h}$*.*
3. $\mathsf{MTLP.Solve}(pk, \mathbf{o}) \to \mathbf{s}$, *a deterministic algorithm that takes the public key* pk *and the puzzle vector* $\boldsymbol{o}$*, and outputs a solution vector* $\boldsymbol{s}$*, where* s_j *is of form* $m_j \mid\mid d_j$*.*
4. $\mathsf{Prove}(pk, s_j) \to \pi_j$, *a deterministic algorithm that takes the public key* pk *and a solution* s_j*, and outputs a proof* π_j*.*
5. $\mathsf{Verify}(pk, \pi_j, h_j) \to \{0,1\}$, *a deterministic algorithm that takes the public key* pk*, proof* π_j*, and commitment* h_j*. If verification succeeds, it outputs 1, otherwise 0.*

Security [1]**.** A multi-instance time-lock puzzle is secure if for all λ and T, any number of puzzle: $z \geq 1$, any j (where $1 \leq j \leq z$), any pair of randomised

[13] A safe modulus is a product of two safe primes $P = 2p' + 1, Q = 2q' + 1$, where p' and q' are prime numbers.

algorithm $\mathcal{A} : (\mathcal{A}_1, \mathcal{A}_2)$, where $\mathcal{A}_1$ runs in time $O(poly(jT, \lambda))$ and $\mathcal{A}_2$ runs in time $\delta(jT) < jT$ using at most $\pi(T)$ parallel processors, there exists a negligible function $\mu(.)$ such that

$$\Pr\left[\begin{array}{l} \mathcal{A}_2(pk, \ddot{o}, \tau) \rightarrow \ddot{a} \\ \text{s.t.} \\ \ddot{a} : (b_i, i) \\ m_{b_i,i} = m_{b_j,j} \end{array} : \begin{array}{l} \mathsf{MTLP.Setup}(1^\lambda, \Delta, z) \rightarrow (pk, sk, \mathbf{d}) \\ \mathcal{A}_1(1^\lambda, pk, z) \rightarrow (\tau, \mathbf{m}) \\ \forall j', 1 \leq j' \leq z : b_{j'} \xleftarrow{\$} \{0,1\} \\ \mathsf{MTLP.Gen}(\mathbf{m}', pk, sk, \mathbf{d}) \rightarrow \ddot{o} \end{array}\right] \leq \frac{1}{2} + \mu(\lambda)$$

A.4 Verifiable Delay Function

Definition 8 (Verifiable Delay Function). *A verifiable delay function (VDF) consists of the following three algorithms:*

1. $\mathsf{VDF.Setup}(1^\lambda, T) \rightarrow pp$, *a probabilistic algorithm that takes security parameter λ and time parameter T, and generates system parameters pp.*
2. $\mathsf{VDF.Eval}(pp, x) \rightarrow \{y, \pi\}$, *a deterministic algorithm that given system parameters pp and a randomly chosen input x, computes a unique output y and a proof π.*
3. $\mathsf{VDF.Verify}(pp, x, y, \pi) \rightarrow \{0, 1\}$, *a deterministic algorithm that verifies y indeed is a correct evaluation of the x. If verification succeeds, the algorithm outputs 1, and otherwise 0.*

Intuitively, there are three security properties that a valid VDF should satisfy. There must be a run time constraint of $(1+\epsilon)T$ for a positive constant ϵ to limit the evaluation algorithm, called *ϵ-evaluation*. The VDF should have *sequentially*, meaning no adversary using parallel processors can successfully compute the output without executing proper sequential computation. Lastly, the VDF evaluation should be a function with *uniqueness* property. That is, the verification algorithm must accept only one output per input.

VDF constructions Among a variety of constructions, VDFs based on repeated squaring have gained more attention as they offer a simple evaluation function that is more compatible with the hardware and provides better accuracy in terms of the time needed to perform the computation. The two concurrent works of [48, 60] suggest evaluating the function $y = x^{2^T}$ over a hidden-order group. Despite similarities in construction, they present two independent ways of proof generation. Particularly, the one proposed by Wesolowski [60] enjoys the luxury of having a constant size proof and verification cost. In addition, Wesolowski's construction can be instantiated over class groups of imaginary quadratic fields [16] which do not require a trusted setup assumption.

A.5 Verifiable Timed Commitment

Definition 9 (Verifiable Timed Commitment [57]). *A verifiable timed commitment consists of the following algorithms:*

1. $\mathsf{VTC.Setup}(1^\lambda, T) \to pp$, *a probabilistic algorithm that takes a security parameter* 1^λ *and time parameter* T*, and generates public parameters* pp*.*
2. $\mathsf{VTC.Commit}(pp, s) \to \{C, \pi\}$, *a probabilistic algorithm that takes public parameters* pp *and a secret* s*, and generates a commitment* C *and proof* π*.*
3. $\mathsf{VTC.Verify}(pp, pk, C, \pi) \to \{0, 1\}$, *a deterministic algorithm that takes public parameters* pp*, a public key* pk*, the commitment* C*, and proof* π*, and checks if the commitment contains a valid* s *with respect to* pk*.*
4. $\mathsf{VTC.Solve}(pp, C) \to s$, *a deterministic algorithm that takes commitment* C*, and outputs a secret* s*.*

Intuitively, a correct VTC should satisfy *soundness*, ensuring the commitment C indeed embeds a valid secret s with respect to pk, and *privacy*, ensuring that no parallel adversary with a running time of less than T succeeds in extracting s, except with negligible probability.

A.6 Sigma Protocols

Let $R = \{(v; w)\} \in \mathcal{V} \times \mathcal{W}$ denote a relation containing the pairs of instances and corresponding witnesses. A Sigma protocol Σ on the $(v; w) \in R$ is an interactive protocol with three movements between P and V as follows.

1. $\Sigma.\mathsf{Ann}(v, w) \to a$, runs by P and outputs a message a to V.
2. $\Sigma.\mathsf{Cha}(v) \to c$, runs by V and outputs a message c to P.
3. $\Sigma.\mathsf{Res}(v, w, c) \to r$, runs by P and outputs a message r to V.
4. $\Sigma.\mathsf{Ver}(v, a, c, r) \to \{0, 1\}$, runs by V and outputs 1 if statement holds.

A Sigma protocol has three main properties including *completeness*, *knowledge soundness*, and *zero-knowledge*. Completeness guarantees the verifier gets convinced if parties follow the protocol. Special soundness states that a malicious prover P^* cannot convince the verifier of a statement without knowing its corresponding witness except with a negligible probability. This is formalized by considering an efficient algorithm called *extractor* to extract the witness given a pair of valid protocol transcripts with different challenges showing the computational infeasibility of having such pairs and therefore guaranteeing the knowledge of the witness by P. The notion of zero-knowledge ensures that no information is leaked to the verifier regarding the witness. This is formalized by considering an efficient algorithm called *simulator* which given the instance v, and also the challenge c, outputs a simulated transcript that is indistinguishable from the transcript of the actual protocol execution. Note that this property only needs to hold against an *honest verifier* which seems to be a limitation of the description, but allows for having much more efficient constructions compared to generic models. The interactive protocol described above can be easily turned into a non-interactive variant using the Fiat-Shamir heuristic [32] in the random oracle model, making it publicly verifiable with no honest verifier assumption.

A.7 Short-lived Proofs

Definition 10 (Short-lived Proofs [5]). *A short-lived proof scheme includes a tuple of the following algorithms:*

1. $\mathsf{SLP.Setup}(1^\lambda, T) \to pp$, *a probabilistic algorithm that takes security parameter* λ *and time parameter* T*, and generates public parameters* pp*.*
2. $\mathsf{SLP.Gen}(pp, v, w, b) \to \pi$, *a probabilistic algorithm that takes a* $(v; w) \in R$ *and a random value* b*, and generates a proof* π*.*
3. $\mathsf{SLP.Forge}(pp, v, b) \to \pi$, *a probabilistic algorithm that takes any instance* v *and a random value* b*, and generates a proof* π*.*
4. $\mathsf{SLP.Verify}(pp, v, \pi, b) \to 1/0$, *a probabilistic algorithm verifying that* π *indeed is a valid short-lived proof of the instance* v*. If verification succeeds, the algorithm outputs 1, and otherwise 0.*

Note that the definition assumes there exists a *randomness beacon* which outputs an unpredictable value b periodically at certain times. There are various ways to implement such beacons including using a public blockchain [15], financial market [26], and more. Such an assumption is necessary to eliminate the need for having a shared global clock (*i.e.,* timestamping). As parties agree on the initial point in time (implied by b), the proof π tied to b must have been observed before time T to be convincing, otherwise might be a forgery.

SLP using Sigma protocols. Short-lived proofs can be instantiated both using generic (non-interactive) zero-knowledge proofs and efficient Sigma protocols. However, as shown in [5], making a Sigma protocol short-lived is rather tricky as it needs some modification in the protocol for OR-composition to be secure according to SLP properties. The modification is done in such a way to let the honest prover create an SLP in a short time without needing to wait for time T to compute the VDF but forces the malicious prover to do the sequential computation, preventing her from computing a forgery before time T. More accurately, in an Or-composition the prover can convince the verifier even if it only knows the witness to one of the relations. To do so, the verifier lets the prover somehow cheat by using the simulator for the relation that it does not know the witness for. Thus, having one degree of freedom the prover chooses two sub-challenges c_1 and c_2 under the constraint that $c_1 + c_2 = c$. Note that the prover is free to fix one of them and compute the other one under the constraints. The observation made in [5] to let the honest prover quickly generate the short-lived proof is to involve the beacon b in the generation of the challenge. Therefore, an honest prover just needs to pre-compute the VDF on a random value b^* allowing her to use it when computing the forgery by freely setting one of the sub-challenges, say c_2, to $b^* \oplus b$ and letting $c_1 = c \oplus c_2$. A malicious prover, however, should compute the VDF on demand as it does not know a witness w for the relation R and c_1 gets fixed by the simulator, taking away the possibility of setting c_2 as specified.

Π_{SLP}

1. Initialization: On input a random value b^*, compute $\mathsf{VDF.Eval}(pp, b^*) \rightarrow \{y^*, \pi^*_{VDF}\}$
2. Proof generation: $\mathsf{SLP.Gen}(pp, v, w, b) \rightarrow \pi$,
   - Compute $\Sigma.\mathsf{Announce}(v, w) \rightarrow a$
   - Compute $c = H(v \parallel b \parallel a)$
   - Set sub-challenge $c_2 = b^* \oplus b$
   - Compute sub-challenge $c_1 = c \oplus c_2$
   - Compute $\Sigma.\mathsf{Response}(v, w, a, c_1) \rightarrow r$
   - Output $\pi =: \{a, c_1, r, c_2, y^*, \pi^*_{VDF}\}$
3. Forgery: $\mathsf{SLP.Forge}(pp, v, b) \rightarrow \tilde{\pi}$,
   - Compute $\Sigma.\mathsf{Simulator}(v) \rightarrow (\tilde{a}, \tilde{c_1}, \tilde{r})$
   - Compute $c = H(v \parallel b \parallel \tilde{a})$
   - Set sub-challenge $c_2 = c \oplus \tilde{c_1}$
   - Compute $\mathsf{VDF.Eval}(pp, b \oplus c_2) \rightarrow \{y, \pi_{VDF}\}$
   - Output $\tilde{\pi} =: \{\tilde{a}, \tilde{c_1}, \tilde{r}, c_2, y, \pi_{VDF}\}$
4. Proof verification: $\mathsf{SLP.Verify}(pp, v, \pi/\tilde{\pi}, b) \rightarrow \{0, 1\}$
   - Compute $c = H(v \parallel b \parallel a)$
   - Accept if:
     - $c = c_1 \oplus c_2$
     - $\Sigma.\mathsf{Verify}(v, a, c_1, r) = 1$
     - $\mathsf{VDF.Verify}(pp, b \oplus c_2, y, \pi_{VDF}) = 1$

Fig. 6: Short-lived proof for a relation $R = \{(v; w)\}$ using pre-computed VDFs [5]

As an optimization, some alternative ways for generating a VDF solution by the honest prover instead of pre-computing a VDF from scratch have been proposed that we refer the reader to [5] for more details.