Timed Secret Sharing

Alireza Kavousi, Aydin Abadi, and Philipp Jovanovic

University College London

Abstract. Secret sharing has been a promising tool in cryptographic schemes for decades. It allows a dealer to split a secret into some pieces of shares that carry no sensitive information on their own when being treated individually but lead to the original secret when having a sufficient number of them together. Existing schemes lack considering a *guaranteed* delay prior to secret reconstruction and implicitly assume once the dealer shares the secret, a sufficient number of shareholders will get together and recover the secret at their wish. This, however, may lead to security breaches when a *timely* reconstruction of the secret matters as the early knowledge of a single revealed share is catastrophic assuming a threshold adversary.

This paper presents the notion of *timed secret sharing* (TSS), providing *lower* and *upper* time bounds for secret reconstruction with the use of time-based cryptography. The recent advances in the literature including short-lived proofs [Asiacrypt 2022], enable us to realize an upper time bound shown to be useful in breaking *public goods game*, an inherent issue in secret sharing-based systems. Moreover, we establish an interesting trade-off between time and fault tolerance in a secret sharing scheme by having dealer *gradually* release *additional* shares over time, offering another approach with the same goal. We propose several constructions that offer a range of security properties while maintaining practical efficiency. Our constructions leverage a variety of techniques and state-of-the-art primitives.

1 Introduction

Secret sharing [51] is a vital primitive in cryptography and distributed computing. A threshold secret sharing scheme, which is the focus of this work, lets a dealer distribute a secret s, among n parties such that one can recover the secret given at least a threshold of shares $t + 1$. With any subset of size t shares, no information about s is leaked. This design rationale has been proven to be useful in a wide range of applications from secure multi-party computation [57] and password-protection [6, 34], to vertical federated learning [39], and many more.

Traditionally, the threshold secret sharing schemes have been defined without *explicit* involvement of time bounds for secret reconstruction. That is, once the dealer shares the secret, they may go offline and it is up to a threshold of shareholders to get together and recover the secret at any time. However, there are scenarios where such time bounds are needed. Consider the following example with regard to a voting election. Alice decides to cast her vote in an election

that is due to be held in a decentralized fashion, meaning that there are a set of collectors independently receiving the voting shares. However, she knows that she will not be available at the time of the election while does not want to miss it. It is clear that the shares should be available no sooner than a starting time (*i.e.*, lower time bound), and should be received no later than an ending time (*i.e.*, upper time bound).

Moreover, it is possible to have an independent treatment on the time bounds and argue about their usefulness separately. In some applications like secret management with decentralized trust [18, 25] a *determined* delay prior to the reconstruction may be required which is usually addressed by making honest assumptions on the shareholders to follow a policy. Another example with a reliance on delayed reconstruction is the commit-reveal mechanism for *maximal extractable value* (MEV) prevention in cryptocurrency platforms [32, 42], preventing validators to benefit from ordering manipulation by learning early about a transaction before its inclusion on-chain is guaranteed. In particular, the recent work of [42] has each user encrypt their transaction using a symmetric key and send the ciphertext to the system. They also secret share the key to the validators, allowing them to recover the transaction after it gets committed. We remark that here the event of committing a block is equivalent to a lower time for reconstruction.

At first glance, the underlying honest majority assumption of the threshold secret sharing schemes seems to be addressing such delayed action if having a time hardcoded in the system. In what follows, we argue this cannot provide proper protection when time does matter. **First**, in critical situations the dealer may wish to have a guarantee that nobody learns the secret sooner whatsoever *without* making any honest assumption on the shareholders. Note that such services are always under threat of being compelled by law enforcement to provide access. **Second**, hardcoding time in the system implies making a *synchronized global clock* assumption for shareholders, which is quite challenging to realize [5, 9, 30]. Any violation in achieving this can lead to a serious security violation. Consider a situation where some of the shareholders reveal their shares sooner than the due time because of a mismatch in synchronization. This may enable the threshold adversary already controlling a portion of shares to learn the secret at an earlier time, even before honest parties do. **Third**, when there is a huge incentive in the system for shareholders to perform an earlier reconstruction, like that of MEV for validators, the transition from honest majority to the dishonest majority is likely in practice. This, in turn, is equivalent to the system being centralized and can completely negate the usefulness of the protection mechanism. Observe that here challenging the inherent honest majority assumption makes sense as the main goal is an *early* reconstruction and not avoiding it. In other words, the user (*i.e.*, dealer) is under threat of suffering from a loss by the set of validators (*i.e.*, shareholders) who may not stick to the common honest majority assumption. Such dishonest majority does not violate the possibility of reconstruction as the success of MEV depends on the completion of reconstruction.

Although an upper time bound is implicitly assumed in practice for the sake of *termination* [36], we explicitly formalize it with the additional aim of breaking *public goods game*, an inherent aspect of systems with fault tolerance. In fact, the underlying fault tolerance of the threshold secret sharing schemes makes them a public goods game [8], as only a threshold of shareholders is sufficient for reconstruction. One of the known techniques in the literature to break this is via harnessing the financial capabilities of the blockchain systems to reward the participants if they publish their shares sooner [4,37]. In this work we introduce two other *off-chain* methods to address this issue.

1.1 Our Approaches

We base our approaches on combining threshold secret sharing schemes and time-based cryptographic primitives with concretely efficient instantiations, in particular, *time-lock puzzle* (TLP) [1,41,48], *verifiable timed commitment* (VTC) [54], and *verifiable delay function* (VDF) [46,58]. In the remainder of this section, we outline how each of our proposed construction works.

Timed Secret Sharing (TSS). This is our basic construction, where the dealer locks the shares by putting them into TLPs [41, 48] to realize the lower time bound T_1. This treatment offers two features. First, no computationally bounded shareholder can learn its share before T_1. So, even if all the shareholders collude, they cannot recover the secret. Second, working through TLPs that resist parallel computing provides a consistent relative timing (*i.e.*, computational timing) for the participating parties, eliminating the reliance on a shared global clock to preserve security (*i.e.*, no early release of shares). The reconstruction can occur at any time after T_1 by at least a threshold of parties.

Verifiable Timed Secret Sharing (VTSS). To provide security against an active adversary, we first need to protect against a malicious dealer distributing *malformed puzzles*, *i.e.*, puzzles that either are not extractable or contain invalid shares. Such checks must be performed before shareholders try to retrieve the shares. Moreover, we need to protect against a malicious shareholder sending wrong share. To achieve it, we make use of verifiable timed commitment (VTC) [54] and verifiable secret sharing (VSS) in a novel way to provide verifiability against the dealer in the distribution phase and against shareholders in the reconstruction phase.

Publicly Verifiable Timed Secret Sharing (PVTSS). The first step to make the scheme publicly verifiable is to deploy a publicly verifiable secret sharing (PVSS) scheme [16, 50]. To simultaneously ensure the validity of embedded encrypted shares and extractability of puzzles in a public manner, we construct an efficient non-interactive zero-knowledge (NIZK) protocol and utilize the cut-and-choose technique [38]. The latter forces the dealer to randomly open a set of t puzzles to show they are constructed correctly. Moreover, we observe that public verifiability allows for the use of short-lived proofs (SLPs) [3] to realize/strengthen the notion of upper time bound by binding the *correctness* of the protocol to time. More precisely, we utilize SLPs as proofs with time-sensitive

soundness as part of the reconstruction phase of PVTSS. So, given that the secret (and therefore each share) is distributed uniformly at random, the associated SLP cannot vouch for the validity of a submitted share after some time. This essentially puts an upper time bound for PVTSS by making the system usable up to some time T_2, *i.e.*, the correct reconstruction is only guaranteed before T_2. We then argue that this can provide an off-chain solution for breaking public goods game.

Decrementing-threshold Timed Secret Sharing (DTSS). We observe that having the dealer publish *additional* time-locked shares *gradually* over time is another technique to realize an upper time bound T_2 and of course breaking public goods game. In fact, the consequence of this gradual release of shares is the gradual reduction of the fault tolerance, forcing the shareholders to perform the reconstruction before the resilience of the system against a threshold adversary decays. We design a protocol which lets the dealer release additional time-locked shares recovered gradually at different points in time. The dealer can go offline right after distributing the locked shares. On the other hand, this idea is useful in scenarios where there is not a sufficient number of (honest) shareholders available[1] for reconstruction in the period between T_1 and T_2. This way, the remaining parties can use the additional shares to reconstruct the secret, offering robustness to the system. To construct an efficient protocol for this purpose, we use multi-instance TLP [1].

Our contributions. In this paper, we make the following contributions:

- We present and formally define *timed secret sharing* (TSS) scheme, allowing a dealer to share a secret to a set of n shareholders with a timely reconstruction by a threshold of them in $[T_1, T_2]$.
- To provide verifiability, we present and formally define *verifiable timed secret sharing* (VTSS) scheme, protecting against a malicious dealer in distribution and malicious shareholders in reconstruction.
- To provide public verifiability, we present and formally define *publicly verifiable timed secret sharing* (PVTSS) scheme.
- We present and formally define *decrementing-threshold timed secret sharing* (DTSS) scheme, establishing a trade-off between time and fault tolerance in a threshold secret sharing scheme.
- Following the above, we introduce two ideas regarding breaking public goods game in threshold secret sharing-based systems. One leverages the recently introduced notion of *short-lived proofs* [3], and the other is based on the novel technique of *gradual release of additional shares*.
- We propose efficient constructions for all the aforementioned schemes.

2 Related Work

There is a large body of works on the combination of computational timing and cryptographic primitives such as commitment [2, 12, 24, 43, 55], encryp-

[1] A large number of the (honest) shareholders may go offline in a burglary, fire, etc.

tion [15, 21, 40], signature [7, 23, 28, 54], 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. Boneh and Naor [12] put forth the notion of timed commitment where, before solving the puzzle, the receiver gets convinced that it is well-formed by running an interactive protocol with the prover. Recently, Manevich and Akavia [43] presented a primitive called attribute verifiable timed commitments (AVTC) as an enhancement on [12], enabling the sender to convince the receiver that the committed secret possesses some *arbitrary* attribute. They do so by augmenting the original timed commitment with generic zero-knowledge proofs.

The work of [54] 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 [41]. 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. With a focus on reducing the interaction in MPC protocols with limited-time secrecy, the authors in [2] developed a gage time capsule (GaTC), allowing a sender to commit to a value that others can obtain it 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 is similar to DTSS in the sense that over time it gradually decays, as the adversary can invest more and more computational resources. Doweck and Eyal [24] constructed a multi-party timed commitment (MPTC) that enables a group of parties to jointly commit to a secret to be opened by an aggregator later on via brute-force computation. As performing sequential computations might be beyond the capacity of some users, [56] developed a system to allow users to outsource their tasks to some servers in a privacy-preserving manner. Recently, the work of [53] 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 only of theoretical interest and does not have practical efficiency, it enables a party to solve many puzzle instances simultaneously at the cost of solving one puzzle (*i.e.*, batch solving). One of the motivating reasons for TLP of [53] is to enable a party solves the puzzles of others (via batch solving) in case a large number of parties abort. We refer the reader to [45] for a more detailed overview of relevant works.

3 Preliminaries

3.1 Threat Model

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 communication 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 schemes, we assume the existence of broadcast channels. For a publicly verifiable scheme, we assume the existence of an authenticated bulletin board which once a message is posted, it becomes available to everyone. We assume a static adversary that may corrupt up to t out of n parties before the start of protocol execution. The dealer D may also be corrupted. Corruption occurs in two forms of semi-honest or malicious. 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 behavior 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

The core building block of our constructions is a secret sharing scheme that allows D to distribute a secret s among a set of n parties $\mathcal{P}$. The scheme typically consists of two main phases; namely, *distribution* and *reconstruction*. In the former, the dealer sends each party its corresponding share, and in the latter, any proper set of parties can uniquely reconstruct the secret by pooling their shares. A (t, n) threshold secret sharing guarantees that the secret is reconstructed by any subset of at least $t + 1$ shares (*i.e.*, correctness) while no information is revealed about the secret by gathering any fewer shares (*i.e.*, t-security). In this work, we focus on Shamir secret sharing [51] for proposing concrete constructions. However, our proposed definitions are generic and capture any (linear) secret sharing.

Verifiable Secret Sharing (VSS). The basic (t, n) threshold secret sharing scheme of [51] only provides security against *passive* adversary, meaning that its security is guaranteed as long as the participating parties run the protocol as specified by the scheme. When considering malicious adversary, it is required to have the dealer ensure parties about the validity of sharing and parties ensure the reconstructor about the validity of their submitted shares. This is the motivation behind developing VSS schemes [26, 35]. Moreover, apart from satisfying the t-security condition likewise the basic setting, here the scheme requires an additional assumption of $t + 1$-robustness that guarantees there is enough number of shares to reconstruct the secret in the case some parties refuse to take part and give back their shares.

Publicly Verifiable Secret Sharing (PVSS). To extend the scope of verifiability to the public and not only participating parties, PVSS schemes [16, 17, 50] deploy cryptographic primitives such as encryption and NIZK proofs. PVSS enables anyone to verify the distribution phase (by the dealer) and reconstruction phase (by the shareholders). The public verifiability feature is important in various applications, such as generating public randomness [16].

SCRAPE PVSS. The work of [16] introduced a PVSS scheme that is an improvement over [50] enjoying the luxury of a new method for doing the verification regarding the equivalence of Shamir secret sharing and Reed Solomon error-correcting code [44]. In a nutshell, the SCRAPE protocol works as follows. The dealer D chooses a random value $s \xleftarrow{\$} \mathbb{Z}_q$, 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 $\{pk_i\}_{i\in[n]}$. The dealer also 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 in the exponent to reconstruct the secret S. SCRAPE PVSS is proposed in two versions, one in the random oracle model under Decisional Diffie-Hellman (DDH) assumption and the other one in the plain model under Decisional Bilinear Squaring (DBS) assumption [31]. We use the non-pairing variant of the protocol which offers *knowledge soundness*. This is vital for security reasons, ensuring the secret chosen by the adversary is independent of those of honest parties. Moreover, we require the knowledge soundness property for deploying short-live proofs [3].

3.3 Time-lock Puzzles

The idea of TLPs was initially introduced in [48]. TLP locks a secret such that it can only be retrieved after a predefined amount of sequential computation. It consists of two algorithms: $\mathsf{TLP.Gen}$, taking a time parameter T and a secret s to generate a puzzle Z, and $\mathsf{TLP.Solve}$, taking the puzzle Z to retrieve the 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.

Homomorphic Time-lock Puzzles (HTLP). Malavolta and Thyagarajan [41] proposed homomorphic TLP, enabling one to homomorphically combine many instances of TLPs into one compact 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 a set of puzzles to output in 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 [41,48] 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 [12] 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 [12] efficiently verifiable, the recent work of [54] proposed verifiable timed commitment (VTC), enabling the sender to verifiably[2] commit to signing keys of form $pk = g^{sk}$, $sk \in \{0,1\}^{\lambda}$. The VTC primitive consists of a tuple of algorithms (VTC.Setup, VTC.Commit, VTC.Verify, VTC.Solve). Manevich and Akavia in [43] augment the timed commitment of Boneh and Naor [12] with zero-knowledge proofs, enabling the sender to prove *any* arbitrary attribute regarding the committed value. Intuitively, given a statement and the corresponding witness $(v; w) \in R$, they follow the MPC in the head framework [33] to construct a predicate $\mathcal{F} : \{0,1\}^* \rightarrow \{0,1\}$ that verifies the committed value (*i.e.*, witness) indeed satisfies the relation R with respect to the public statement v. To do so, they introduce attribute verifiable timed commitment (AVTC) primitive with the tuple of algorithms (AVTC.Setup, AVTC.Commit, AVTC.Solve). This primitive allows a sender to commit to a witness w while guaranteeing that $(v; w) \in R$. Note that we deploy (A)VTC in a black box manner 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 a rich body of 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 [27] in the random oracle model, one can make the protocol non-interactive with public verifiability. A Sigma protocol should satisfy two security properties including *soundness*, ensuring the verifier about the validity of the statement v, and *zero-knowledge*, ensuring the prover about the secrecy of the witness w.

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

Zero Knowledge proof of equality of discrete logarithm. One of the common Sigma protocols is discrete logarithm equality (DLEQ) proof. That is, for 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, respectively, the 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 actually 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 [19].

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

In Asiacrypt 2022, Arun et al. [3] put forth the notion of *short-lived proofs* (SLPs) which can be roughly defined as types of proofs with expiration such that their soundness will disappear after some time. In fact, they are only sound if being observed before a determined time, afterwards, they may be forgery indistinguishable from 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 after performing some sequential computation for 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* [22] which outputs unpredictable values b periodically.

An SLP scheme consists of a tuple of four algorithms (SLP.Setup, SLP.Gen, SLP.Forge, SLP.Verify) with the following behaviour. SLP.Setup generates public parameters pp on input the security parameter and time parameter T. SLP.Eval takes pp, an input x, a random beacon value b, and generates a proof π. SLP.Forge takes pp, x, b, and produces a proof π. Lastly, 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, *indistinguishability*, making the real and forge proofs indistinguishable.

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. Our definition follows the original definition of a threshold secret sharing scheme.

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*, publishes public parameters* pp*. The algorithm is run by the dealer* D*.*
2. **Distribution:**
   - Sharing: $\mathsf{TSS.Sharing}(pp, s) \rightarrow \{C_i\}_{i\in[n]}$*, on input* pp *and secret* $s \in S_\lambda$*, generates a locked share* C_i *with time parameter* T_1 *for each party* P_i *in the set* $\mathcal{P}$*. The algorithm is run by the dealer* D*.*
3. **Reconstruction:**
   - Recovering: $\mathsf{TSS.Recover}(pp, C_i) \rightarrow s_i$*, on input* pp *and* C_i*, recovers* s_i*. The algorithm is run by each party* P_i *in* $\mathcal{P}$*.*
   - Pooling: $\mathsf{TSS.Pool}(pp, \mathcal{S}, T_2) \rightarrow s$*, 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 prior to T_1 and *security*, ensuring any set of shares less than a threshold reveals no information about the secret prior to T_2. Note that we care about T_1 security-wise due to our main motivation of guaranteeing a lower time bound for reconstruction. 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 to realize. We later show how to relax this assumption using computational timing.

Definition 1.1 (Correctness)**.** A TSS satisfies correctness if for all secret $s \in S_\lambda$ it holds

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

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

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

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

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

4.2 TSS Construction

We present an instantiation of TSS in Figure 1. To enforce a lower time bound T_1, the dealer uses TLPs [41, 48] as they allow the dealer to lock the shares into puzzles and enforce a computational delay for each party to recover its corresponding share. TLP guarantees that no party P_i can obtain its share s_i sooner than T_1.

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

The proof of the theorem can be found in Appendix B.1.

5 Verifiable Timed Secret Sharing (VTSS)

So far we assumed all participating parties, including the dealer, follow the protocol faithfully, providing passive security. In this section, we present verifiable timed secret sharing (VTSS), an enhanced TSS which considers active adversaries who may arbitrarily deviate from the protocol's description. It protects against a malicious dealer who may send incorrect or even no shares during the distribution phase. It also protects against a malicious shareholder who may send an incorrect share during the reconstruction phase.

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 its 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 time T_2, the reconstructor (a party in $\mathcal{P}$) reconstructs the secret s using Lagrange interpolation at $f(0)$; otherwise, it returns $\perp$.

Fig. 1. TSS protocol Π_{TSS}

5.1 VTSS Definition

In this section, we present a formal definition of VTSS. Our definition builds upon the verifiable secret sharing (VSS) of Feldman [26] which is the basis for all the existing VSS schemes.

Definition 2 (Verifiable Timed Secret Sharing). *A verifiable timed secret sharing (VTSS) scheme includes the following algorithms.*

1. **Initialization:**
   - Setup: $\mathsf{VTSS.Setup}(1^\lambda, T_1, T_2) \to pp$, *on input security parameter* λ*, lower time bound* T_1 *and upper time bound* T_2*, generates public parameters* pp*.*
2. **Distribution:**
   - Sharing: $\mathsf{VTSS.Sharing}(pp, s) \to \{C_i, \pi_i\}_{i\in[n]}$, *on input* pp *and a secret* s*, generates locked share* C_i *with time parameter* T_1 *for each party* P_i *for* $i \in [n]$*. It also generates a proof of validity* π_i *for each party's locked share* C_i*.*
   - Share verification: $\mathsf{VTSS.Verify}_1(pp, C_i, \pi_i) \to \{0, 1\}$, *on input* pp*,* C_i*, and* π_i*, checks the validity of shares to ensure the published locked share* C_i *is well-formed and contains a valid share of secret* s*. The algorithm is run by each party* P_i *and returns* 1 *if both checks pass. Otherwise, it returns* 0*.*
3. **Reconstruction:**
   - Recovering: $\mathsf{VTSS.Recover}(pp, C_i) \to s_i$, *on input* pp *and* C_i*, forcibly outputs a share* s_i*. The algorithm is run by each party* P_i*.*
   - Recovery verification: $\mathsf{VTSS.Verify}_2(pp, s_i, \pi_i) \to \{0, 1\}$, *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. Otherwise, it returns* $\perp$*. The algorithm is run by the verifier* V*.*

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

Definition 2.1 (Correctness)**.** A VTSS satisfies correctness if for all secret $s \in S_\lambda$, all $\lambda \in \mathbb{N}$, and all $i \in [n]$ it holds

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

Definition 2.2 (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

$$\Pr\left[b_1 = 1 \vee b_2 = 1 : \begin{matrix}\mathsf{VTSS.Setup}(1^\lambda, T_1, T_2) \to pp, \\ \mathcal{A}(pp) \to (\{C_i, \pi_i\}_{i\in[n]}, s_i), \\ b_1 := \mathsf{VTSS.Verify}_1(pp, C_i, \pi_i) \wedge \nexists s \text{ s.t.} \\ \mathsf{VTSS.Sharing}(pp, s) \to \{\{C_i\}_{i\in[n]}, \cdot\}, \\ b_2 := \mathsf{VTSS.Verify}_2(pp, s_i, \pi_i) \wedge \nexists C_i \text{ s.t.} \\ \mathsf{VTSS.Recover}(pp, C_i) \to s_i\end{matrix}\right] \leq \mu(\lambda)$$

Definition 2.3 (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

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

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

Definition 2.4 (Security)**.** A VTSS satisfies security if there exists a negligible function μ such that for an adversary controlling a subset $\mathcal{S}'$ of parties/shares, where $|\mathcal{S}'| \leq t$ and $s \in S_\lambda$, it holds

$$\Pr\left[\mathcal{A}(pp, \mathcal{S}', T_2) \to s : \begin{matrix}\mathsf{VTSS.Setup}(1^\lambda, T_1, T_2) \to pp, \\ \mathsf{VTSS.Sharing}(pp, s) \to \{C_i, \pi_i\}, \\ \mathsf{VTSS.Recover}(pp, C_i) \to s_i\end{matrix}\right] \leq \mu(\lambda)$$

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 its sharing $\{C_i, \pi_i'\}$.
   - Share verification: $\mathsf{VTSS.Verify}_1(pp, C_i, \pi_i) \to \{0, 1\}$, party P_i runs $\mathsf{VTC.Verify}$ (pp, v_i, C_i, π_i') to check the locked share C_i is well-formed and embeds the share s_i corresponding to v_i. It then validates the consistency of the shares by sampling a code word $\mathbf{y}^\perp \in \mathcal{C}^\perp$, where $\mathbf{y}^\perp = \{y_1^\perp, \ldots, y_n^\perp\}$, and checking if $\prod_{j=1}^n {v_j}^{y_j^\perp} = 1$.
   - 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 a proof does not verify (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: $\mathsf{VTSS.Verify}_2(pp, s_i, \pi_i) \to \{0, 1\}$, 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 time T_2, the reconstrctor (a party in $\mathcal{P}$) reconstructs the secret s using Lagrange interpolation at $f(0)$ or aborts otherwise.

Fig. 2. VTSS protocol Π_{VTSS}

5.2 VTSS Construction

We present a protocol for VTSS in Figure 2. Following Feldman VSS [26], we make a crucial change in the protocol to adapt it for VTSS. 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 [47]. This is due to the equivalency of the Shamir secret sharing with Reed-Solomon encoding observed by [44]. We restate the basic fact of linear error correcting code in Lemma 1. We refer the reader to [16] for a detailed description of the verification procedure. 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. So, it is needed to use the following fact of linear error correcting code to ensure sharing has been done 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 $\mathcal{C}^\perp$, then the probability that $\langle \boldsymbol{x}, \boldsymbol{y}^\perp \rangle = 0$ is exactly $1/q$.*

Second, it enables us to make a black box use of VTC primitive [54] to non-interactively ensure each party P_i that it indeed obtains its correct share s_i at T_1. More precisely, 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 $\{s_i\}_{i \in n}$. Below, we present the VTSS protocol's security theorem.

Theorem 2. *If the verifiable timed commitments* VTC *and Feldman verifiable secret sharing are secure, then verifiable timed secret sharing protocol Π_{VTSS} presented in Figure 2 satisfies soundness, privacy, and security, w.r.t. definitions 2.2, 2.3, and 2.4 respectively.*

The proof of the theorem can be found in Appendix B.2.

6 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. Such a feature in verification enables removing a possible *complaint round* as everyone can validate the correctness of sharing by the dealer in the distribution phase. To this end, the main building block to deploy is a publicly verifiable secret sharing (PVSS) scheme that enforces parties to behave correctly by *non-interactively* proving the validity of the messages sent at distribution and reconstruction phases.

6.1 PVTSS Definition

We now present a formal definition of PVTSS. It is based on the models already provided in the literature for PVSS like [16, 17, 50].

Definition 3 (Publicly Verifiable Timed Secret Sharing). *A PVTSS scheme involves the following algorithms.*

1. **Initialization:**
   - Setup: $\mathsf{PVTSS.Setup}(1^\lambda, T_1, T_2) \rightarrow pp$, *on input security parameter λ, lower time bound T_1 and upper time bound T_2, generates public parameters pp. Each party P_i announces a registered public key pk_i which its secret key sk_i is only known to them.*

2. **Distribution:**
   - Sharing: $\mathsf{PVTSS.Sharing}(pp, s, \{pk_i\}_{i\in[n]}) \rightarrow \{\{C_i\}_{i\in[n]}, \pi_D\}$, *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 *for* $i \in [n]$. *It also generates a proof* π_D *for the correctness of shares. The algorithm is run by the dealer* D.
   - Share verification: $\mathsf{PVTSS.Verify_1}(pp, \{pk_i, C_i\}_{i\in[n]}, \pi_D) \rightarrow \{0, 1\}$, *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: $\mathsf{PVTSS.Recover}(pp, C_i, pk_i, sk_i) \rightarrow \{\tilde{s}_i, \pi_i\}$, *on input* pp, C_i, pk_i, *and* sk_i, *forcibly outputs a decrypted share* $\tilde{s}_i$ *together with proof* π_i *of valid decryption. The algorithm is run by each party* P_i.
   - Recovery verification: $\mathsf{PVTSS.Verify_2}(pp, C_i, \tilde{s}_i, \pi_i) \rightarrow \{0, 1\}$, *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: $\mathsf{PVTSS.Pool}(pp, \mathcal{S}, T_2) \rightarrow S$, *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. The algorithm is run by* V.

Formally, A PVTSS scheme must satisfy the following properties.

Definition 3.1 (Correctness)**.** PVTSS satisfies correctness if for all secret $s \in S_\lambda$ and all $i \in [n]$ it holds that

$$\Pr\left[\begin{array}{ll} \mathsf{PVTSS.Verify_1}(pp, \{C_i\}_{i\in[n]}, & \mathsf{PVTSS.Setup}(1^\lambda, T_1, T_2) \rightarrow pp, \\ \pi_D, \{pk_i\}_{i\in[n]}) = 1 & \mathsf{PVTSS.Sharing}(pp, s, \{pk_i\}_{i\in[n]}) \\ \mathsf{PVTSS.Verify_2}(pp, C_i, \tilde{s}_i, \pi_i) = 1 : & \rightarrow \{\{C_i\}_{i\in[n]}, \pi_D\}, \\ \mathsf{PVTSS.Pool}(pp, \mathcal{S}, T_2) \rightarrow S & \mathsf{PVTSS.Recover}(pp, C_i, pk_i, sk_i) \rightarrow \{\tilde{s}_i, \pi_i\} \end{array}\right] = 1$$

Definition 3.2 (Soundness)**.** 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

$$\Pr\left[b_1 = 1 \vee b_2 = 1 : \begin{array}{l} \mathsf{PVTSS.Setup}(1^\lambda, T_1, T_2) \rightarrow pp, \\ \mathcal{A}(pp) \rightarrow (\{pk_i, C_i\}_{i\in[n]}, \pi_D, \tilde{s}, \pi), \\ b_1 := \mathsf{PVTSS.Verify_1}(pp, \{pk_i, C_i\}_{i\in[n]}, \pi_D) \wedge \nexists s \text{ s.t.} \\ \mathsf{PVTSS.Sharing}(pp, s, \{pk_i\}_{i\in[n]}) \rightarrow \{\{C_i\}_{i\in[n]}, \cdot\}, \\ b_2 := \mathsf{PVTSS.Verify_2}(pp, C, \tilde{s}, \pi), \wedge \nexists sk \text{ s.t.} \\ \mathsf{PVTSS.Recover}(pp, C, pk, sk) \rightarrow \{\tilde{s}, \cdot\}, \end{array}\right] \leq \mu(\lambda)$$

Definition 3.3 (t-Privacy)**.** PVTSS satisfies t-privacy if for all parallel algorithms $\mathcal{A}$ whose running time is at most T_1, and a 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 $\forall\lambda \in \mathbb{N}$ it holds that

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

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

Definition 3.4 (Security)**.** A PVTSS satisfies security if there exists a negligible function μ such that for an adversary controlling a set $\mathcal{S}'$ of parties/shares, where $|\mathcal{S}'| \leq t$ and $s \in S_\lambda$, together with the public information, denoted by PI, it holds that [3]

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

An indistinguishability definition given in [31,49] and adopted by [16] formalizes this. We refer to Appendix B.4 for more details.

It is worth mentioning that similar to [16], the security requirement here does not capture any privacy for the secret after the reconstruction phase. In other words, the privacy of the secret matters as long as it is reconstructed by an eligible set of parties after which anyone (even an external party) can learn the secret. Moreover, one may notice that *before* time T_1 the privacy requirement implies security. That is, even if the adversary corrupts *all* the parties involved in the protocol, it cannot learn any information about the secret.

6.2 PVTSS Construction

We present the complete PVTSS protocol in Figure 3. In what follows, we elaborate on a number of 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.

Handling a malicious dealer. What makes the protection mechanism challenging for PVTSS is the fact that *anyone*, before going through sequential computation, should be able to check the correctness of sharing 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, (2) locked encrypted shares contain the committed shares, and (3) shares are obtainable in time T_1, all with respect to some public information. Thus, this essentially takes away deploying VTC primitive that we used for VTSS construction. We first explore how to guarantee consistency and verifiability followed by our approach regarding extractability.

[3] This property is presented as IND1-Secrecy in [31, 49].

Blinded DLEQ. Our solution to address the first two aforementioned guarantees 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 5.2). To do so, we slightly modify the DLEQ proof (Section 3.5) and construct a Sigma protocol we call *blinded* DLEQ proof[4]. In fact, 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.
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 3. *The protocol Π_{BDLEQ} is a public-coin honest-verifier zero-knowledge argument of knowledge corresponding to the language L_{BDLEQ}.*

The proof of the theorem can be found in Appendix B.3.

Cut-and-choose technique. What is left to discuss is how the dealer convinces the parties they can obtain their shares in time T_1. This is equivalent to saying the TLP Z_i has indeed the β_i embedded. A promising way to show the correctness of puzzle generation is utilizing the cut-and-choose technique that has been explored in previous works [7, 53]. At a high level, this technique forces a sender to behave correctly by randomly opening a (fixed) subset of puzzles it has already sent to the receiver based on the receiver's choice. We show that it is possible to deploy the cut-and-choose technique in our construction without sacrificing security according to two observations. First, as we already assume the existence of a randomness beacon for establishing short-lived proofs, we can utilize a beacon output b' as a random challenge for the dealer to sample a set of t locked encrypted shares[5]. Second, since opening just reveals a set of size t of encrypted shares, we are still guaranteed that the secret remains hidden up to time T_1, no matter what. Moreover, there is no need to deploy a range proof as each party is supposed to open its corresponding locked encrypted share, which is not among the opened ones by the dealer.

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

[4] We are not aware of the existence of such protocol under different name in the literature.

[5] Instead of using b', the dealer can compute a hash function, modeled as a random oracle, on public sharing material and use the output as a random challenge.

be reconstructed/used by any external party. This leads us to deploy *short-lived proofs* (SLPs) [3] for realizing an upper time bound in the construction. 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 being 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 our PVTSS construction by piggybacking on the *proof of decryptions* π_i generated by each party P_i as part of the Reconstruction phase, turning them into short-lived proofs where their expiration time matches the upper time bound T_2. So, given the properties of short-lived proofs and also the fact that the secret is *randomly distributed* in SCRAPE PVSS[6], 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. As shown in [3], a short-lived proof for any arbitrary relation R for which there exists a Sigma protocol can be efficiently constructed. For completeness, we present the short-lived proof for a relation R using pre-computed VDFs in Figure 5. 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 1. Recently, there has been a number of works focusing on the notion of forgeability over time, particularly for developing short-lived signature or forward-forgeable signature [3, 52]. To the best of our knowledge, the work of [3] is the only one exploring the time-based forgeability in proof systems. This in turn enables us to make use of their primitive to provide the upper time bound for PVTSS in a way, relating the correctness of the secret reconstruction to time.

Remark 2. We make no assumption on the availability of an *online* verifier who observes the protocol over time. In fact, due to the characteristic of SLPs, their use is 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 afterwards, negating the forgeability property. Moreover, as pointed out in [3], convincingly timestamping the messages published on the bulletin board is also opposed to the usability of SLPs.

[6] 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 [10].

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 pointed out in [3], T does not need to be hardcoded when $\mathsf{PVTSS.Setup}$ is run. This allows using VDFs with any time parameter $T' > T$, while still generating short-lived proofs with respect to time T. That is, even if different shareholders use different time parameters with $T' > T$ for their VDF evaluations, only those proofs observed before time T are convincing.

Theorem 4. *If the time-lock puzzle* TLP*, short-lived proofs* SLP*, and Scrape PVSS are secure, then publicly verifiable timed secret sharing protocol* Π_{PVTSS} *presented in Figure 3 satisfies soundness, t-privacy, and security, w.r.t. definitions 3.2, 3.3, and 3.4 respectively.*

For a proof of theorem see Appendix B.5.

7 Decrementing-threshold Timed Secret Sharing (DTSS)

7.1 Secret Sharing with Additional Shares

A threshold secret sharing scheme guarantees *t-security*, preventing an adversary controlling any set of parties of size $\leq t$ from learning the secret. There is also $t+1$-*robustness* assumption[7], 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 availability). To be concrete, a possible scenario that may lead to having less than a threshold of honest parties available is explored in [54] known as *denial of spending* (DoSp) attack. Here, the adversary can carry out an attack against honest parties to control, say 51% of parties, while the reconstruction threshold to spend a multisig transaction is, say 52%. Consequently, the set of available honest parties cannot reach the threshold and their possible investment will be remained locked. We can extend this to a secret sharing setting where honest parties may not reach the threshold by themselves, at least for some time (*e.g.*, the upper time bound).

Our goal is to mitigate this *liveness* assumption 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 that each party already receives during the distribution phase of the protocol. For instance, using Shamir secret sharing, the additional shares are computed by evaluating the sharing polynomial $f(\cdot)$ at some distinct publicly known points. So, even if there is less than a threshold

[7] Note that this is the case in a malicious setting where parties may not take part in the reconstruction phase.

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 $\{pk_i\}_{i\in[n]}$. The dealer 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 $c_i = \hat{s}_i g_3^{\beta_i}$. It 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, the dealer runs Π_{BDLEQ} resulting in proof $\pi_D =: (v_i, e, r_{1,i}, r_{2,i})$ for $i \in [n]$. Finally, the dealer publishes the locked encrypted shares $\{C_i\}_{i\in[n]}$ and proof π_D on a public bulletin board.
   - Share verification: $\mathsf{PVTSS.Verify_1}(pp, \{C_i\}_{i\in[n]}, \pi_D, b', \{pk_i\}_{i\in[n]}) \to \{0,1\}$, 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 $\prod_{j=1}^n {v_j}^{y_j^\perp} = 1$. It then checks the proof π_D is valid. Using a beacon value b' released after the completion of the sharing step, the dealer randomly opens a set $H(b') \to I$ of locked encrypted shares by publishing their corresponding randomness β_i and encrypted share $\hat{s}_i$. So, the verifier V can check 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 its share, unlocks the blinding factor β_i by running $\mathsf{TLP.Solve}(pp, Z_i)$, and obtains its 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 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: $\mathsf{PVTSS.Verify_2}(pp, C_i, \tilde{s}_i, \pi_i, b, T_2) \to \{0,1\}$, any (external) verifier V can check the validity of published share $\tilde{s}_i$ via $\mathsf{DLEQ.SLV}$ $(\pi_i, g_1, pk_i, \tilde{s}_i, \hat{s}_i)$. This involves calling $\mathsf{SLP.Verify}$. Note that having C_i, the verifier first obtains $\hat{s}_i$ using 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. 3. PVTSS protocol Π_{PVTSS}

of parties available at the reconstruction period (*i.e.*, $[T_1, T_2]$), the remaining parties 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 schemes assumes all the participating parties, not only those interacting in the reconstruction phase, learn the secret [16, 35]. Given this, we argue that the availability of a threshold number of additional time-lock shares at the proper time (*i.e.*, T_2) does not violate the security of the system since it enables all the parties eventually learn the secret at the same point via sequential computation if they have not already learned it[8].

To focus on the core problem, which is preserving the robustness of the system in case of unavailability of a threshold of honest parties, we assume the additional time-locked shares are honestly generated. However, should a malicious dealer attempt to misbehave, this assumption can be lifted by using mechanisms similar to the techniques used in the previous sections.

Pessimistic condition. The setting we are considering can be referred to as pessimistic condition, where the number of *honest* parties drops lower than the required robustness threshold of $n/2$ in the synchronous model [20]. Note that this is rather opposite the *optimistic condition* in the literature [11], demanding the number of honest parties to be the super majority (*i.e.*, more than $3n/4$) for some property (*i.e.*, responsiveness) to hold.

Releasing additional shares in one go. We provide a simple extension to the VTSS protocol Π_{VTSS} presented in Figure 2 by having the dealer embed t additional shares concatenated in a TLP, offering robustness even if all but one honest party goes offline during the reconstruction period. We present the extension in Appendix D.

7.2 Decrementing-threshold Timed Secret Sharing (DTSS)

It is possible to derive a *trade-off* between time and fault tolerance by having the dealer release each additional time-locked share periodically at different points in time. The consequence of this *gradual* release is twofold. Firstly, if necessary, 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 of time *i.e.*, $[T_0, T_2]$. In fact, they can stop working up to a point where a *sufficient* number of additional shares is gained. Secondly, as time goes by and the reconstruction is not initiated, the adversary can get more additional shares by investing computational effort, leading to a gradual lessening of the fault tolerance of the system. Interestingly, this feature happens to be useful to break 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.

[8] We stress that our security argument relies on the conjecture that the adversary does not have an advantage over honest parties with respect to performing a certain number of steps of sequential computation [48].

7.3 DTSS Definition

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

Definition 4 (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:**
   - Setup: $\mathsf{DTSS.Setup}(1^\lambda, T_1, t) \rightarrow \{pp, pk, sk\}$, *on input security parameter λ, lower time bound T_1, and a value t, generates public parameters pp and key pair (pk, sk) to be used for generating additional locked shares by the dealer D.*
2. **Distribution:**
   - Sharing: $\mathsf{DTSS.Sharing}(pp, s, pk, sk) \rightarrow \{C_i, \mathbf{v}, \{O_j\}_{j\in[t]}\}$, *on input pp, a secret s, and a key pair (pk, sk), generates locked share C_i with time parameter T_1 together with commitment to shares $\boldsymbol{v}$ for each party P_i for $i \in [n]$. Moreover, it outputs t additional locked shares $\{O_j\}_{j\in[t]}$, with O_j being locked with time parameter $(j+1)T_1$. The algorithm us run by D.*
3. **Reconstruction:**
   - Share recovery: $\mathsf{DTSS.ShaRecover}(pp, C_i) \rightarrow s_i$, *on input pp, C_i, and pk, outputs a share s_i. The algorithm is run by each party P_i.*
   - Recovery verification: $\mathsf{DTSS.Verify}(pp, s_i, \mathbf{v}) \rightarrow \{0, 1\}$, *on input pp, s_i, and $\boldsymbol{v}$, checks the validity of the received share. The algorithm is run by a verifier $V \in \mathcal{P}$.*
   - Additional share recovery: $\mathsf{DTSS.AddRecover}(pp, pk, \{O_j\}_{j\in[t]}) \rightarrow \{s'_j\}$, *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 some additional shares.*
   - Pooling: $\mathsf{DTSS.Pool}(pp, \mathcal{S}, T_2) \rightarrow s$, *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. The algorithm is run by V.*

We could also include a verification algorithm $\mathsf{VTSS.Verify_2}$ for a verifier to check the validity of the presented additional share by a participating party. We refrain from formalizing this algorithm since we implicitly assume all the parties involved in reconstruction retrieve the additional time-locked shares, negating the verification. However, such a verification algorithm can introduce efficiency as it allows reconstruction of the secret while having *only one* party solve the puzzles containing additional time-locked shares and prove their correctness to others.

Formally, a DTSS scheme must provide the following properties.

Definition 4.1 (Correctness)**.** A DTSS satisfies correctness if for all secret $s \in S_\lambda$ and all $i \in [n]$ it holds that

$$\Pr\left[\begin{array}{l}\mathsf{DTSS.Verify}(pp, C_i, \tilde{s}_i, \pi_i) = 1,\\ \mathsf{DTSS.Pool}(pp, \mathcal{S}, T_2) \to s\end{array} : \begin{array}{l}\mathsf{DTSS.Setup}(1^\lambda, T_1, t) \to pp,\\ \mathsf{DTSS.Sharing}(pp, s)\\ \to \{\{Z_i\}_{i\in[n]}, \{O_i\}_{i\in[t]}\},\\ \mathsf{DTSS.ShaRecover}(pp, C_i, pk_i, sk_i)\\ \to \{\tilde{s}_i, \pi_i\},\\ \mathsf{DTSS.AddRecover}(pp, \{O_i\}_{i\in t})\\ \to \{s'_i\}_{i\in[j], 1\le j\le t}\end{array}\right] = 1$$

Definition 4.2 (Verifiability)**.** A DTSS scheme is verifiable if there exists a negligible function μ such that for all PPT adversaries $\mathcal{A}$, all $\lambda \in \mathbb{N}$, and $i \in [n]$, it holds that

$$\Pr\left[b = 1 : \begin{array}{l}\mathsf{DTSS.Setup}(1^\lambda, T_1, t) \to \{pp, pk, sk\},\\ \mathcal{A}(pp) \to (s_i, \mathbf{v}, \cdot),\\ b := \mathsf{DTSS.Verify}(pp, s_i, \mathbf{v}), \wedge\, \nexists s \text{ s.t.}\\ \mathsf{DTSS.Sharing}(pp, s, pk, sk) \to \{\mathbf{v}, \cdot\}\end{array}\right] \le \mu(\lambda)$$

Definition 4.3 (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 $\forall\lambda \in \mathbb{N}$ it holds that

$$\left|\Pr\left[\mathcal{A}(pp, pk, s, \{C_i\}_{i\in[n]}, \mathbf{v}, \{O_j\}_{j\in[t]}) = 1 : \begin{array}{l}\mathsf{DTSS.Setup}(1^\lambda, T_1) \to \{pp, pk, sk\},\\ \mathcal{A}(1^\lambda, pp) \to s\\ \mathsf{DTSS.Sharing}(pp, s) \to \{C_i, \mathbf{v}, \{O_j\}_{j\in[t]}\}\end{array}\right] -\right.$$

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

Definition 4.4 (Security)**.** Let $2T_1, \ldots, (t+1)T_1$ be times at which each additional time-locked share is forcibly obtained. A DTSS is secure if prior to $(j+1)T_1$, where $1 \le j \le t$, the adversary controlling at most $\le t - (j-1)$ parties learns no information about $s \in S_\lambda$ in a computational sense. Thus, it holds:

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

Definition 4.5 (Robustness)**.** A DTSS is robust if each party in $\mathcal{P}$ can *eventually* reconstruct the secret s, either after receiving a sufficient number of other parties' shares and/or obtaining the additional time-locked shares.

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

7.4 DTSS Construction

We present a construction of DTSS in Figure 4. As mentioned, we would like a protocol in which anyone is able to obtain each additional share s'_j at time $(j+1)T_1$ given that dealer's role must end with the distribution phase[9]. 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 $T_1\sum_{j=1}^{t} j$ number of squaring. To get away with this issue, we use multi-instance time-lock puzzle (MTLP) [1], a primitive allowing sequential release of solutions where the overall computation cost of solving t puzzles is actually equal to that of solving only the last one.

Theorem 5. *If the multi-instance time-lock puzzle* MTLP *and verifiable timed secret sharing* VTSS *are secure, then our DTSS protocol* Π_{DTSS} *presented in Figure 4 satisfies the properties described in Section 7.3.*

For a proof of theorem see Appendix B.6.

Remark 3. By looking closely, we see the trade-off between time and fault tolerance can be added as a property to any threshold secret sharing scheme. More precisely, the gradual release of fault tolerance over time is implied by the release of the shares and not the type of underlying secret sharing scheme. In addition, the use of additional time-locked shares implicitly provides an upper time bound. This is because by the time T_2 – which the last puzzle is supposed to open – the secret is revealed to all parties in $\mathcal{P}$.

8 Discussion

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

TSS as a generalization of secret sharing. A timed secret sharing scheme can be thought of as a time-based generalization of a normal secret sharing scheme. That is, theoretically, if we set $T_1 = 0$ and $T_2 = \infty$, then the resulting scheme is a normal secret sharing. Also, the *independency* of the methods used to realize the lower and upper time bounds makes it possible to consider them separately depending on applications.

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

1. **Initialization:**
   - Setup: $\mathsf{DTSS.Setup}(1^\lambda, T_1, t) \to \{pp, pk, sk\}$, the dealer D invokes two algorithms of $\mathsf{VTSS.Setup}(1^\lambda, T_1)$ and $\mathsf{MTLP.Setup}(1^\lambda, T_1, t+1)$, and publishes the set of public parameters pp, pk.
2. **Distribution:**
   - Sharing: $\mathsf{DTSS.Sharing}(pp, s, pk, sk) \to \{C_i, \mathbf{v}, \{O_j\}_{j\in[t]}\}$, the dealer D first picks a secret $s \xleftarrow{\$} \mathbb{Z}_q$ and invokes $\mathsf{VTSS.Sharing}(pp, s)$ to generate n locked shares $\{C_i\}_{i\in[n]}$ and $\mathbf{v}$. Moreover, it computes t additional shares $f(a_j) = s'_j$ for $j \in [t]$, where $f(0) = s$ and $\{a_1, \ldots, a_t\}$ are some known distinct points. Finally, it invokes $\mathsf{MTLP.Gen}(\mathbf{m}, pk, sk)$, where $\mathbf{m} = \{\perp, s'_1, \ldots, s'_t\}$ to generate an MTLP containing $\{s'_j\}_{j\in[t]}$.
3. **Reconstruction:**
   - Share recovery: $\mathsf{DTSS.ShaRecover}(pp, C_i) \to s_i$, each party P_i runs $\mathsf{VTSS.Recover}(pp, C_i)$ to recover its share s_i.
   - Recovery verification: $\mathsf{DTSS.Verify}(pp, s_i, \mathbf{v}) \to \{0, 1\}$, any reconstructor V runs $\mathsf{VTSS.Verify}_2(pp, s_i, \mathbf{v})$ to check the validity of the published share s_i.
   - Additional share recovery: $\mathsf{DTSS.AddRecover}(pp, pk, \{O_j\}_{j\in[t]}) \to \{s'_j\}_{j\in[t]}$, anyone wishing to obtain additional time-locked shares $\{s'_j\}_{j\in[t]}$ runs $\mathsf{MTLP.Solve}(pp, \{O_j\}_{j\in[t]})$.
   - Pooling: $\mathsf{DTSS.Pool}(pp, \mathcal{S}, T_2) \to s$, 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. 4. DTSS protocol Π_{DTSS}

On the setup phase. In all of our schemes, Setup algorithm is responsible to generate 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 2 requires a trusted setup to generate the parameters for the underlying VTC primitive. This is due to the linearly homomorphic TLP of [41] deployed in VTC construction. In fact, the functionality of the primitive heavily 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 [14] 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 [41]. Observe that deploying the class groups solely does not eliminate the need for a trusted setup as it is still feasible that malicious sender fools a receiver into accepting locked shares that will never be opened. We propose another construction for VTSS in Figure 6 based on AVTC primitive [43] that has a transparent setup. Moreover, the VDF primitive used in SLPs can be instantiated efficiently via class groups [58] without making any trusted setup assumption.

On the use of SLPs. As already mentioned, by deploying SLPs in PVTSS protocol we strengthen the notion of upper time bound as the system is *safely* usable

until T_2. In fact, as we show next, after T_2 any reconstruction fails with overwhelming probability. This necessitates the availability of a reconstructor during the protocol execution for a correct reconstruction. Moreover, we deploy short-lived proofs using precomputed VDFs [3] 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 single-use.

Failure probability. As a result of deploying SLPs, here we briefly analyze the probability of a reconstruction failure after T_2. 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. As mentioned, one of the promising ways to break public goods game is to reward those parties who publish their shares sooner [4, 37]. That is, the shareholder receives some *reward* if its submitted share is among the first $t + 1$ shares published on the chain. This in turn motivates each shareholder to show up sooner. We believe our two ideas, namely using short-lived proofs and gradual release of additional shares, can be considered as orthogonal methods for this purpose. More precisely, using SLPs forces parties to publish their shares before some time, otherwise, they may not be able to recover the correct secret. Moreover, using a gradual release of additional shares can also play the same role; however, by causing the threat of security reduction over time. Consequently, shareholders are motivated to act as soon as possible to avoid any pitfalls.

Acknowledgements. The authors would like to thank Dan Ristea for initiating the idea of secret sharing with additional shares. 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. 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. *Cryptology ePrint Archive*, 2021.
3. 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.
4. 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.
5. 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.
6. 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.
7. 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.
8. 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.
9. 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.
10. 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.
11. A. Bhat, A. Kate, K. Nayak, and N. Shrestha. Optrand: Optimistically responsive distributed random beacons. *Cryptology ePrint Archive*, 2022.
12. D. Boneh and M. Naor. Timed commitments. In *Annual international cryptology conference*, pages 236–254. Springer, 2000.
13. J. Bonneau, J. Clark, and S. Goldfeder. On bitcoin as a public randomness source. *Cryptology ePrint Archive*, 2015.
14. J. Buchmann and H. C. Williams. A key-exchange system based on imaginary quadratic fields. *Journal of Cryptology*, 1(2):107–118, 1988.
15. 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.
16. 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.
17. 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.
18. M. Chase, H. Davis, E. Ghosh, and K. Laine. Acsesor: A new framework for auditable custodial secret storage and recovery. *Cryptology ePrint Archive*, 2022.

19. D. Chaum and T. P. Pedersen. Wallet databases with observers. In *Annual international cryptology conference*, pages 89–105. Springer, 1992.
20. B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*, pages 383–395. IEEE, 1985.
21. 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.
22. J. Clark and U. Hengartner. On the use of financial data as a random beacon. *Evt/wote*, 89, 2010.
23. Y. Dodis and D. H. Yum. Time capsule signature. In *International Conference on Financial Cryptography and Data Security*, pages 57–71. Springer, 2005.
24. Y. Doweck and I. Eyal. Multi-party timed commitments. *arXiv preprint arXiv:2005.04883*, 2020.
25. 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.
26. 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.
27. 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.
28. J. A. Garay and M. Jakobsson. Timed release of standard digital signatures. In *International Conference on Financial Cryptography*, pages 168–182. Springer, 2002.
29. J. Groth. On the size of pairing-based non-interactive arguments. In *Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35*, pages 305–326. Springer, 2016.
30. 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.
31. 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.
32. L. Heimbach and R. Wattenhofer. Sok: Preventing transaction reordering manipulations in decentralized finance. *arXiv preprint arXiv:2203.11520*, 2022.
33. Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Zero-knowledge from secure multiparty computation. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 21–30, 2007.
34. 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.
35. 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.
36. J. Knapp and E. A. Quaglia. Fair and sound secret sharing from homomorphic time-lock puzzles. In *Provable and Practical Security: 14th International Conference, ProvSec 2020, Singapore, November 29–December 1, 2020, Proceedings 14*, pages 341–360. Springer, 2020.
37. 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.
38. Y. Lindell. Fast cut-and-choose-based protocols for malicious and covert adversaries. *Journal of Cryptology*, 29(2):456–490, 2016.
39. Y. Liu, Y. Kang, T. Zou, Y. Pu, Y. He, X. Ye, Y. Ouyang, Y. Zhang, and Q. Yang. Vertical federated learning. *CoRR*, 2022.
40. 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.
41. 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.
42. D. Malkhi and P. Szalachowski. Maximal extractable value (mev) protection on a dag. *arXiv preprint arXiv:2208.00940*, 2022.
43. 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.
44. R. J. McEliece and D. V. Sarwate. On sharing secrets and reed-solomon codes. *Communications of the ACM*, 24(9):583–584, 1981.
45. L. Medley, A. F. Loe, and E. A. Quaglia. Sok: Delay-based cryptography. *Cryptology ePrint Archive*, 2023.
46. K. Pietrzak. Simple verifiable delay functions. In *10th innovations in theoretical computer science conference (itcs 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.
47. 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.
48. R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock puzzles and timed-release crypto. 1996.
49. 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.
50. 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.
51. A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.
52. 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.
53. S. Srinivasan, J. Loss, G. Malavolta, K. Nayak, C. Papamanthou, and S. A. Thyagarajan. Transparent batchable time-lock puzzles and applications to byzantine consensus. *Cryptology ePrint Archive*, 2022.

54. 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.
55. 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.
56. 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.
57. V. Vaikuntanathan, A. Narayanan, K. Srinathan, C. P. Rangan, and K. Kim. On the power of computational secret sharing. In *Progress in Cryptology - INDOCRYPT 2003, 4th International Conference on Cryptology in India, New Delhi, India, December 8-10, 2003, Proceedings*, 2003.
58. B. Wesolowski. Efficient verifiable delay functions. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 379–407. Springer, 2019.

A Cryptographic Building Blocks

A.1 Time-lock Puzzles (TLP)

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

1. $\mathsf{TLP.Gen}(1^\lambda, T, s) \to Z$, *a probabilistic algorithm that takes time parameter* T *and a secret* s*, and generates a puzzle* Z*.*
2. $\mathsf{TLP.Solve}(T, Z) \to s$, *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 [41]**.** 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

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

Security [41]**.** 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

$$\Pr\left[\mathcal{A}(Z) \to b : \mathsf{TLP.Gen}(1^\lambda, T(\lambda), s_b) \to Z, b \xleftarrow{\$} \{0,1\}\right] \leq \frac{1}{2} + \mu(\lambda)$$

In particular, the seminal work of [48] 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 [41]**).** *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. $\mathsf{HTLP.Setup}(1^\lambda, T) \to pp$*, a probabilistic algorithm that takes a security parameter* 1^λ *and time parameter* T*, and generates public parameters* pp*.*
2. $\mathsf{HTLP.Gen}(pp, s) \to Z$*, a probabilistic algorithm that takes public parameters* pp *and a solution* $s \in S_\lambda$*, and generates a puzzle* Z*.*
3. $\mathsf{HTLP.Solve}(pp, Z) \to s$*, a deterministic algorithm that takes public parameters* pp *and puzzle* Z*, and retrieves a secret* s*.*
4. $\mathsf{HTLP.Eval}(C, pp, Z_1, \ldots, Z_n) \to Z'$*, 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 [41]**.** 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 [10] 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 [41] 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. $\mathsf{MTLP.Setup}(1^\lambda, T, z) \to \{pk, sk, \vec{\mathbf{d}}\}$, *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 $\vec{\boldsymbol{d}}$.*
2. $\mathsf{MTLP.Gen}(\vec{\mathbf{m}}, pk, sk, \vec{\mathbf{d}}) \to \{\vec{\mathbf{o}}, \vec{\mathbf{h}}\}$, *a probabilistic algorithm that takes a message vector $\vec{\boldsymbol{m}}$, the public-private key (pk, sk), secret witness vector $\vec{\boldsymbol{d}}$, and outputs a puzzle vector $\vec{\boldsymbol{o}}$ and a commitment vector $\vec{\boldsymbol{h}}$.*
3. $\mathsf{MTLP.Solve}(pk, \vec{\mathbf{o}}) \to \vec{\mathbf{s}}$, *a deterministic algorithm that takes the public key pk and the puzzle vector $\vec{\boldsymbol{o}}$, and outputs a solution vector $\vec{\boldsymbol{s}}$, 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

[10] 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

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

A.4 Verifiable Delay Function

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

1. $\mathsf{VDF.Setup}(1^\lambda, T) \to pp$, *a probabilistic algorithm that takes security parameter* λ *and time parameter* T*, and generates system parameters* pp*.*
2. $\mathsf{VDF.Eval}(pp, x) \to \{y, \pi\}$, *a deterministic algorithm that given system parameters* pp *and a randomly chosen input* x*, computes a unique output* y *and a proof* π*.*
3. $\mathsf{VDF.Verify}(pp, x, y, \pi) \to \{0, 1\}$, *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 [46, 58] 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 [58] 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 [14] which do not require a trusted setup assumption.

A.5 Verifiable Timed Commitment

Definition 9 (Verifiable Timed Commitment [54]**).** *A verifiable timed commitment consists of the following algorithms:*

1. $\mathsf{VTC.Setup}(1^\lambda, T) \rightarrow pp$, *a probabilistic algorithm that takes a security parameter* 1^λ *and time parameter* T*, and generates public parameters* pp*.*
2. $\mathsf{VTC.Commit}(pp, s) \rightarrow \{C, \pi\}$*, a probabilistic algorithm that takes public parameters* pp *and a secret* s*, and generates a commitment* C *and proof* π*.*
3. $\mathsf{VTC.Verify}(pp, pk, C, \pi) \rightarrow \{0, 1\}$*, 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. $\mathsf{VTC.Solve}(pp, C) \rightarrow s$*, 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 the 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. $\Sigma.\mathsf{Ann}(v, w) \rightarrow a$, runs by P and outputs a message a to V.
2. $\Sigma.\mathsf{Cha}(v) \rightarrow c$, runs by V and outputs a message c to P.
3. $\Sigma.\mathsf{Res}(v, w, c) \rightarrow r$, runs by P and outputs a message r to V.
4. $\Sigma.\mathsf{Ver}(v, a, c, r) \rightarrow \{0, 1\}$, 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 formalised 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 formalised 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 [27] in the random oracle model, making it publicly verifiable with no honest verifier assumption.

A.7 Short-lived Proofs

Definition 10 (Short-lived Proofs). *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. $\mathsf{SLP.Verify}(pp, v, \pi, b) \to \{0, 1\}$, *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 [13], financial market [22], and more. Such an assumption is necessary to eliminate the need for having a shared global clock (*i.e.*, timestamping). In fact, 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 [3], 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. In fact, the modification is done in such a way to let the honest prover creates 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 [3] 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.

1. Initialization: On input a random value b^*, computes $\mathsf{VDF.Eval}(pp, b^*) \rightarrow \{y^*, \pi^*_{VDF}\}$
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. 5. Short-lived proof for a relation $R = \{(v; w)\}$ using pre-computed VDFs [3]

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 by Arun et al. that we refer the reader to [3] for more details.

B Proofs

We now describe the proofs that were skipped in the main body of the paper due to the limitation in space.

B.1 Proof of Theorem 1

Proof (sketch). 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'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.

B.2 Proof of Theorem 2

Proof (sketch). 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 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 DL 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; afterwards, the security follows directly from that of Feldman VSS due to the security of the commitment $\mathbf{v}$.

B.3 Blinded DLEQ Proof

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

$$g_1^{r_1} = a_1 x^c,\ g_2^{r_1} g_3^{r_2} = a_2 y^c \ \ ;\ \ g_1^{r_1'} = a_1 x^{c\prime},\ g_2^{r_1'} g_3^{r_2'} = a_2 y^{c\prime}$$

$$g_1^{r_1 - r_1'} = x^{c-c'} \Leftrightarrow x = g_1^{\frac{r_1 - r_1'}{c-c'}}$$

$$g_2^{r_1 - r_1'} g_3^{r_2 - r_2'} = y^{c-c'} \Leftrightarrow y = g_2^{\alpha} g_3^{\frac{r_2 - r_2'}{c-c'}}$$

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 the following two distributions, namely real protocol distribution and simulated distribution, are identically distributed.

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

$$\mathtt{Sim} : \{(a_1, a_2; c; r_1, r_2) : r_1, r_2 \xleftarrow{\$} \mathbb{Z}_q; a_1 = g_1^{r_1} x^{-c}, a_2 = g_2^{r_1} g_3^{r_2} y^{-c}\}$$

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

B.4 Security Game for PVSS

Definition 11 (Indistinguishably of Secrets [16]). *A PVSS is said to be secure if any polynomial time adversary $\mathcal{A}$ corrupting at most t parties has a negligible probability in the following game played against a challenger.*

1. *Playing the role of a dealer, the challenger runs the Setup step of the PVTSS and sends all the public information to $\mathcal{A}$. Moreover, it creates the key-pairs for the honest parties and send the corresponding public keys to $\mathcal{A}$.*
2. *$\mathcal{A}$ creates and sends the public keys of the corrupted parties to the challenger.*
3. *The challenger randomly picks the values s and s' in the space of the secret. It then chooses $b \leftarrow \{0,1\}$ uniformly at random and runs the Sharing step of the protocol with s as secret. It sends $\mathcal{A}$ all public information generated in that phase together with s_b.*
4. *$\mathcal{A}$ outputs a guess b'.*
   The advantage of $\mathcal{A}$ is defined as $|\Pr[b' = b] - 1/2|$.

B.5 Proof of Theorem 4

Proof (sketch). As our protocol follows closely the one in [16], we analyse the security properties with respect the new techniques we apply.

Before T_2, the correctness is straightforward. Afterwards, 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). In fact, 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 [16] to generate $\hat{s_i}$ is not IND-CPA-secure.

B.6 Proof of Theorem 5

Proof (sketch). Correctness is straightforward. Verifiability is implied by the underlying Π_{VTSS} protocol. Privacy follows from that of Π_{VTSS} together with the underlying Π_{MTLP} protocol for additional time-locked shares. Moreover, the commitments to shares $\mathbf{v}$ do not reveal any information about the secret s under the DL assumption. Security is satisfied with respect to 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.

C A New Construction For VTSS

As an alternative to our VTSS protocol presented in Figure 2, here we present a rather similar construction using AVTC primitive. Thanks to a black box use of AVTC primitive, each shareholder P_i can check the validity of its locked share C_i with respect to the committed share v_i to see if $(s_i, v_i) \in R_{DL}$ holds, where R_{DL} is a discrete logarithm relation. Moreover, the reconstructor can also verify the validity of the submitted share s_i by P_i with the knowledge of v_i. The new protocols is presented in Figure 6

The AVTC primitive proposed in [43] uses generic circuit-based MPC computation to build the underlying proof system that can be implemented via zk-SNARKs [29]. So, it might not be a concretely efficient option for proving an algebraic statement like a discrete logarithm as we have in VTSS construction.

Theorem 6. *If the attributed verifiable timed commitments* AVTC *and Feldman's verifiable secret sharing are secure, then our verifiable timed secret sharing protocol* Π_{VTSS} *presented in Figure 2 satisfies soundness, privacy, and security, w.r.t. definitions 2.2, 2.3, and 2.4 respectively.*

The proof is similar to Appendix B.2 except using AVTC instead of VTC.

D Secret Sharing with Additional Shares

Construction. The dealer D performs the $\mathsf{VTSS.Sharing}$ to share a secret s as specified in the description of the protocol Π_{VTSS}. Moreover, it computes t additional shares $\{s'_1, \ldots, s'_t\}$ by evaluating the same Shamir polynomial $f(\cdot)$ containing the secret s at some known distinct points $\{a_1, \ldots, a_t\}$. The dealer then constructs and publishes an RSA-based TLP Z' with time parameter T_2 by running $\mathsf{TLP.Gen}$ embedding the concatenation of the shares $s'_1||\ldots||s'_t$ as its solution. The parties in $\mathcal{P}$ run $\mathsf{VTSS.Recover}$ and $\mathsf{VTSS.Verify}_2$ as specified in the description of the protocol Π_{VTSS}. In addition, any party P_i willing to obtain the additional shares starts solving the puzzle Z' upon receiving it. So, if less than a sufficient number of shares gets communicated by parties during the reconstruction period $[T_1, T_2]$, the shares locked in Z' would allow P_i to reconstruct the secret by the time T_2.

Theorem 7. *The above construction realizes the definition of VTSS from Section 5.1.*

1. **Initialization:**
   - Setup: $\mathsf{VTSS.Setup}(1^\lambda, T_1, T_2) \to pp$, the dealer D runs $\mathsf{AVTC.Setup}(1^\lambda)$ and publishes the public parameters pp. Let g be a generator of a group $\mathbb{G}$ of order q.
2. **Distribution:**
   - Sharing: $\mathsf{VTSS.Sharing}(pp, s) \to \{C_i, \pi_i\}_{i\in[n]}$, the dealer D picks a secret $s \xleftarrow{\$} \mathbb{Z}_q$ 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 then commits to f by computing $v_i = g^{s_i}$ and broadcasting $\mathbf{v} = \{v_i\}_{i\in[n]}$. Then, D runs $\mathsf{AVTC.Commit}(pp, T_1, s_i)$ 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 for all $i \in [n]$. Finally, the dealer D privately sends each party P_i its sharing $\{C_i, \pi_i\}$.
   - Share verification: $\mathsf{VTSS.Verify_1}(pp, C_i, \pi_i) \to \{0, 1\}$, each party P_i first checks that the locked share C_i is well-formed and embeds the share s_i using the proof π_i. Note that the proof ensures that $(s_i, v_i) \in R_{DL}$. It then verifies the consistency of the shares by sampling a code word $\mathbf{y}^\perp \in \mathcal{C}^\perp$, where $\mathbf{y}^\perp = \{y_1^\perp, \ldots, y_n^\perp\}$, and checking if $\prod_{j=1}^{n} {v_j}^{y_j^\perp} = 1$.
   - Complaint round: If a set of parties of size $\geq t+1$ complain about sharing, the dealer D is disqualified. Otherwise, the dealer reveals the corresponding locked shares with proofs by broadcasting $\{C_i, \pi_i\}$. If a proof does not verify (or the dealer does not broadcast), the dealer is disqualified.
3. **Reconstruction:**
   - Recovering: $\mathsf{VTSS.Recover}(pp, C_i) \to s_i$, each party P_i wishing to participate in reconstruction runs $\mathsf{AVTC.Solve}(pp, C_i)$ to obtain its share s_i no sooner than T_1.
   - Recovery verification: $\mathsf{VTSS.Verify_2}(pp, s_i, \pi_i) \to \{0, 1\}$, 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 time T_2, the reconstrctor (a party in $\mathcal{P}$) reconstructs the secret s using Lagrange interpolation at $f(0)$ or aborts otherwise.

Fig. 6. VTSS protocol Π_{VTSS}

Proof (sketch). assuming the additional time-locked shares are contained in a secure TLP, the security properties follow directly from the underlying VTSS scheme. For brevity, we omit a formal treatment but note the following nuance. The reason why we limit the dealer to send(at most) t shares is to allow only those parties involved in the protocol to obtain the secret and not anyone in public. Moreover, the fault tolerance of the system does not reduce, as the T_2 is the upper time bound for the system and the adversary controlling at most t shares does not learn any information about secret s before honest parties do.

Using packed secret sharing. In the packed Shamir secret sharing which is a generalization of Shamir secret sharing, the dealer shares l secrets $(s_1, \ldots, s_l)$ using one single (Shamir) polynomial $f(\cdot)$ of degree $t + l - 1$, where t is the fault tolerance threshold. The resulting throughput by packing more secrets implies

weakening the fault tolerance as $t + l$ shares are now needed for reconstruction, limiting the supported fault tolerance to $n - (t + l)$. To achieve the standard fault tolerance of t in a time-based setting, the dealer can send l additional time-locked shares to make up for the availability of more honest shares in the reconstruction phase. In this way, the scheme can resist up to $n - t$ malicious parties while the honest parties can reconstruct the secret after obtaining the additional shares by the time T_2.