Verifiable Timed Signatures Made Practical

Sri Aravinda Krishnan Thyagarajan
Friedrich Alexander Universität Erlangen-Nürnberg
thyagarajan@cs.fau.de

Adithya Bhat
Purdue University
bhat24@purdue.edu

Giulio Malavolta
UC Berkeley and CMU
giulio.malavolta@hotmail.it

Nico Döttling
CISPA Helmholtz Center for Information Security
nico.doettling@gmail.com

Aniket Kate
Purdue University
aniket@purdue.edu

Dominique Schröder
Friedrich Alexander Universität Erlangen-Nürnberg
dominique.schroeder@fau.de

ABSTRACT

A verifiable timed signature (VTS) scheme allows one to time-lock a signature on a known message for a given amount of time **T** such that after performing a sequential computation for time **T** anyone can extract the signature from the time-lock. Verifiability ensures that anyone can publicly check if a time-lock contains a valid signature on the message without solving it first, and that the signature can be obtained by solving the same for time **T**.

This work formalizes VTS, presents efficient constructions compatible with BLS, Schnorr, and ECDSA signatures, and experimentally demonstrates that these constructions can be employed in practice. On a technical level, we design an efficient cut-and-choose protocol based on the *homomorphic time-lock puzzles* to prove the validity of a signature encapsulated in a time-lock puzzle. We also present a new efficient *range proof* protocol that significantly improves upon existing proposals in terms of the proof size, and is also of independent interest.

While VTS is a versatile tool with numerous existing applications, we demonstrate VTS's applicability to resolve three novel challenging issues in the space of cryptocurrencies. Specifically, we show how VTS is the cryptographic cornerstone to construct: (i) Payment channel networks with improved on-chain unlinkability of users involved in a transaction, (ii) multi-party signing of transactions for cryptocurrencies without any on-chain notion of time and (iii) cryptocurrency-enabled fair multi-party computation protocol.

CCS CONCEPTS

• **Security and privacy → Digital signatures**.

KEYWORDS

Timed signatures; Time lock puzzles; Payment Channel Network; Multi-party signing

ACM Reference Format:
Sri Aravinda Krishnan Thyagarajan, Adithya Bhat, Giulio Malavolta, Nico Döttling, Aniket Kate, and Dominique Schröder. 2020. Verifiable Timed Signatures Made Practical. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS '20), November 9–13, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3372297.3417263

1 INTRODUCTION

Timed cryptography studies a general class of primitives that allows a sender to send information to the future. After a pre-determined amount of time, anyone (possibly at the end of a sequential computation) can learn the enclosed secret. Time-Lock puzzles [11, 41, 46], Timed Commitment [15], and Timed release of Signatures [24] are prominent primitives in this class with wide-ranging applications [4, 15, 30, 34].

For many applications, it is important that the receiver is convinced that the message of the sender is well-formed (e.g., it contains a valid signature on a certain message) before committing a large amount of time and resources to solve the corresponding puzzle. Therefore, it is natural to augment the above mentioned primitives with the notion of *verifiability.* In this work we formally introduce the notion of *Verifiable Timed Signature Scheme (VTS)*, where a sender commits to a signature σ on a known message in a verifiable and extractable way[1]. Verifiability refers to the property that one can publicly check that a valid signature is contained in the commitment, whereas extractability guarantees that the signature σ can be recovered from the commitment in time **T**.

1.1 Applications of VTS

Although the utility of VTS in classical applications such as fair contract signing is already well known [15, 24], we observe that it can further solve challenging privacy and compatibility problems in the cryptocurrency (or blockchain) space. Concretely, we discuss three new applications of VTS.

Applications I: Privacy-Preserving Payment Channels Networks. Bitcoin [43] and most permissionless blockchains are inherently limited in transaction throughput and typically have large fees associated with each payment. Payment channels [2, 45] have

[1]In [15], the notion of verifiability for the timed signature is implicitly assumed to exist. We explicitly formalize it and propose efficient protocols for real world applications.

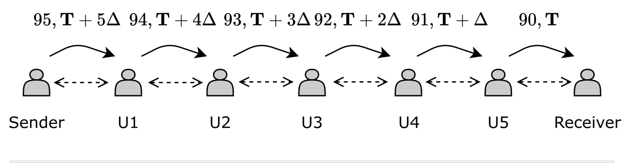

Figure 1: A multi-hop transaction over a payment channel network. Dotted lines with two arrowheads indicate payment channels between successive users. In this example, the Sender pays 90 coins to the Receiver through five intermediate users, each collecting a fee of one coin. Each payment hop is associated with a decreasing expiry time ($\mathbf{T}+c\Delta$, for $c \in \{0, \ldots, 5\}$).

emerged as a prominent scalability solution to mitigate these issues by allowing a pair of users to perform multiple payments without committing every intermediate payment to the blockchain. Abstractly, a payment channel consists of three phases: (i) Two users Alice and Bob open a payment channel by adding a single transaction to the blockchain. Intuitively, this transaction promises that Alice may pay up to a certain amount of coins to Bob, which he must claim before a certain time T; (ii) Within this time window, Alice and Bob may send coins from the joint address to either of them by sending a corresponding transaction to the other user; (iii) The channel is considered closed when the latest of the payment transactions is posted on the chain, thus spending coins from the joint address.

An extension of payment channels is payment channel networks (PCN) [45]. As shown in Figure 1, in a PCN, users can perform *multi-hop* payments, i.e. coins can be transferred to other users in the network without having a common payment channel, routing the payment through a set of intermediate users. For Bitcoin, the atomicity of these payments is ensured using multi-hop locks (in particular, Hash Time Lock Contracts or HTLCs) which guarantee the transfer of v coins if a certain condition is satisfied (e.g., for HTLC, the knowledge of a pre-image x such that $H(x) = y$, where H is a cryptographic hash function) before time $\mathbf{T}$. PCNs are not only well-studied in the academia [6, 19, 20, 39, 40, 49], but also in industry and the Lightning Network (LN) [3, 45] has emerged as the most prominent example.

PCNs are found to be no better than Bitcoin in terms of transaction privacy. By using anonymous multi-hop locks (AMHL) [39, 40], one can make HTLCs unlinkable from the perspective of an on-chain observer, however these proposals do not achieve strong unlinkability of hops as the time-lock information $\mathbf{T}$ is still present in the contract: To avoid race conditions to redeem the coins, the time-lock for the i-th hop is Δ larger than the time-lock for hop $i+1$ (see Figure 1). An attacker observing the on-chain contracts can correlate this time-lock information and detect if certain payments belong to the same multi-hop payment path.

We observe that VTS can solve this privacy issue, by completely *removing* the time-lock information from the payment transactions. At the time of opening a channel between Alice and Bob, Bob signs an additional "steal" transaction for v coins (as in the HTLC) for Alice using a VTS (with time parameter $\mathbf{T}_A$). Alice is then guaranteed that she can redeem these coins after time $\mathbf{T}_A$, by forcing the opening of the VTS: If Bob tried to transfer the coins to his address after time $\mathbf{T}_A$, then Alice would immediately steal them, using the "steal" transaction also signed by Bob. To avoid race conditions, we introduce an artificial delay δ to the payment to Bob. It is important to observe that δ is fixed and in particular is identical for all payment channels. This time delay gives Alice a sufficient window to post the steal transaction with Bob's signature from the VTS (in case of Bitcoin with a relative time-lock using `checkSequenceVerify` OP CODE.

For PCNs, apart from Alice, Bob obtains a steal transaction and a VTS (with timing hardness $\mathbf{T}_B$) from Carol, who in turns is sent a steal transaction and a VTS (with timing hardness $\mathbf{T}_C$) from Dave. The timing hardnesses of these VTS's are structured similarly to the time-locks for HTLC, i.e. $\mathbf{T}_A > \mathbf{T}_B > \mathbf{T}_C$. The important difference is that, even though the time-locks still have the correlation, they are never posted on-chain.

Application II: Multisig Transactions. Computations involving multiple parties in blockchains often rely on transactions with multisig scripts, i.e. conditions that require multiple signatures in order to authenticate transactions. Bitcoin offers t-out of-n multisig scripts that accepts signed transactions from any t-sized subsets of the n users. These have wide ranging applications including [12, 42]. This has motivated a large body of literature on improving security and efficiency of multisig protocols [8, 12, 18, 42] and more efficient constructions of threshold signature schemes [26, 35, 36, 52, 53]. All of these works however (implicitly) assume an expiration time $\mathbf{T}$ for the multisig scripts. This is used to ensure that, even if a large threshold of participants go offline, the coins of the few remaining users are not locked indefinitely. Therefore the scope of multisig-based protocol is limited to those cryptocurrencies that support on-chain notion of time. Those blockchains that do not offer the time-lock functionality are therefore not compatible with these protocols.

We propose to use VTS to bypass this problem. Prior to transferring the funds to the multisig address, all users agree on a default redeem transaction. The redeem transaction transfers the coins from the multisig address back to the respective users. It is signed using a VTS with time parameter $\mathbf{T}$. Once the funds are transferred to the multisig address, users can jointly spend coins by negotiating new refund transactions for which a VTS is given, using a progressively smaller time parameter. If at any point in time, less then t signatures are exchanged by the users, the VTSs exchanged in the previous round make sure that the funds will be redistributed consistently across all participants. Eventually all parties are going to redeem the coins agreed on the previous "stable" state. As an interesting byproduct of our solution, multisig transactions are also indistinguishable from any other kind of transaction, to the eyes of an external observer. This is because the expiration time is never uploaded on-chain.

Application III: Fair Multi-party Computation. In the multi-party computation (MPC) settings, a computation is *fair* if either all parties involved receive the output or none of them does. Recent efforts [10, 31, 32] have proposed leveraging blockchains as a solution to achieve fairness. The general idea is to incentivize users to complete the protocol execution by enforcing some financial penalty in case they fail to do so. More concretely, the participants lock a certain amount of coins in addresses $addr_i$ from which funds can be spent if user U_i reveals a witness to some condition before time **T**. Alternatively, if *all participants* sign the transaction, these coins can be spent and redistributed among the other users after time **T**. Intuitively, an adversary loses coins if he does not reveal the witness, which in turn is crucial to learn the output of the computation. As a compensation, the coins of the adversary are given to the honest users involved in the computation, which incentivizes publishing of such witness, thus ensuring that other users also learn the output of the computation.

This alternate way of spending is negotiated in a payout phase in the form of payout transactions, where all users generate signatures and exchange them with each other. However, these payout transactions are time-locked on chain and are only valid after time **T**. This ensures that other users cannot take the coins and distribute among themselves before the termination of the protocol.

One of the major shortcomings of this proposal (along with similar privacy issues as described above) is that this solution is incompatible with blockchains that do not offer the time-lock functionality, such as Zcash [9] and Monero [33]. VTS[2] can be used to solve such a limitation as follows: All participants sign their payout transaction using a VTS, instead of sending signatures in plain. The privacy of VTS ensures that no party learns the signatures on the payout transaction before time **T**.

1.2 Our Contributions

In summary, in this work we define the notion of *verifiable timed signatures*, propose a number of efficient constructions, and rigorously design and analyze the various applications discussed above. More concretely, our contributions are as follows.

Definitions. We formalize the notion of *Verifiable Timed Signatures (VTS)* (Section 3.2) where the committer creates a commitment to a signature that can be solved and opened after time **T**, along with a proof that certifies that the embedded signature is a valid signature on a message with respect to the correct public key. Anyone can verify this proof and be convinced of the validity of the commitment. In terms of security we require that the commitment and the proof reveal no information about the embedded signature to any PRAM adversary whose running time is bounded by **T** (privacy) and that an adversary should not be able to output a valid proof to a commitment that does not embed a valid signature on a message with respect to a public key (soundness).

Efficient Constructions. We offer three efficient constructions for VTS (Section 4): VT-BLS, VT-Schnorr and VT-ECDSA where the signatures being committed to are BLS, Schnorr and ECDSA signatures, respectively. Our constructions do not require any modification to these signature schemes. Our constructions exploit the group structure of these signature schemes and combine threshold secret-sharing with a cut-and-choose type of argument to achieve practical performance. We also leverage the recently introduced linearly homomorphic time-lock puzzles [41] to reduce the number of puzzles to solve to one (Section 4.4) puzzle. Apart from improving efficiency by decreasing the computational resources needed, this improves security in applications where users may possess different amounts of parallel processors: A user with n processors has no advantage over a user with one processor as they both need to solve only a single puzzle for time **T**. We also present a concretely efficient construction of *Verifiable Timed Commitments (VTC)* (Appendix D), where the signing key is committed instead. Our VTC scheme is applicable to any signature scheme where the secret key is the discrete logarithm of the public key.

Range Proofs. Along the way, we present efficient range proofs (Section 4.5) for proving that the solution of a time-lock puzzle lies within some interval. In contrast with prior works, the protocol batch-proves well-formedness of ℓ time-lock puzzles and the proof size is independent of ℓ. The protocol is generically applicable to all time-lock puzzles/ciphertexts that possess plaintext- and randomness-homomorphism. Such a protocol might be of independent interest.

New Applications. Apart from classical applications such as fair contract signing [15], we identify several applications (as discussed above, and in the full version [50] in formal detail) for VTS where our constructions can be readily used. The primary focus of this paper is on cryptocurrency-based applications where we wish to improve privacy and compatibility of existing solutions. Specifically, (i) we show how to construct privacy-preserving PCNs that prevent de-anonymizing attacks based on on-chain timing correlations, (ii) we construct *single-hop* payment channels without requiring any time-lock functionality from the underlying blockchain, (iii) we present solutions (with different efficiency tradeoffs) to realize blockchain-based *fair computation* without requiring the time-lock functionality from the blockchain, and (iv) we propose a new way to construct multisig contracts from VTS which does not require any time-lock functionality from the corresponding blockchain.

Implementation. We implement our proposed constructions by building an LHTLP library, the range proof, and the other cryptographic primitives. We find that all LHTLP operations are efficient. The homomorphic batching adds a small overhead while outputting a single puzzle to solve. As the most computationally relevant operation, we also estimate the cost of commit and verify operations of our VTS constructions. Results (in an unoptimized implementation) indicate that for practical purposes with a low powered machine, setting the statistical security parameter $n = 40$, our VT-ECDSA verifier takes 9.942s with a soundness error of 7.25×10^{-12}.

2 TECHNICAL OVERVIEW

On a high level, our VTS schemes are built by computing a standard digital signature σ on a message m and *emcoding* it into a time-lock puzzle. Then a non-interactive zero-knowledge (NIZK) proof is used to prove that the puzzle contains a valid signature on m. There are several non-trivial components in our construction, such as encoding the signatures inside the puzzles that is compatible with our efficient instantiation of a non-interactive zero-knowledge proof, novel use of homomorphic operations on the puzzles to ensure better security, all while ensuring that our construction can work with a large class of signature schemes. Throughout the following overview, we describe the VTS as an interactive protocol between a committer and a verifier, which can be made non-interactive using the Fiat-Shamir transformation [22].

[2]In this work we actually solve the problem using a slightly relaxed variant of VTS, i.e. *Verifiable Timed Discrete Logarithm* where, instead of the signature, the signing key of signature scheme is committed to. This makes it compatible with Zcash and Monero.

High-Level Overview. To illustrate our approach, let us consider the BLS signature scheme [14], the other schemes follow a similar blueprint. Recall that BLS public-secret key pair are of the form (g_0^α, α) and the signature on a message m is $\sigma := H(m)^\alpha$, where $g_0 \in \mathbb{G}_0$ is a generator of $\mathbb{G}_0$, $\alpha \in \mathbb{Z}_q$, and $H : \{0,1\}^* \to \mathbb{G}_1$ is a full domain hash function. The verification algorithm checks if $e(g_0, \sigma) = e(g_0^\alpha, H(m))$. To generate a VTS on a message m, the committer secret shares the signature σ together with the public using a t-out-of-n threshold sharing scheme: The first $t-1$ shares are defined as $\sigma_i := H(m)^{\alpha_i}$ for a uniformly sampled $\alpha_i \in \mathbb{Z}_q$. It is important to observe that such a signature σ_i is a valid BLS signature on m under the public-key $pk_i = g_0^{\alpha_i}$. The rest of the shares are sampled consistently using Lagrange interpolation in the exponent, i.e., for $i \in \{t, t+1, \dots, n\}$ we set

$$\sigma_i = \left(\frac{\sigma}{\prod_{j \in [t-1]} \sigma_j^{\ell_j(0)}} \right)^{\ell_i(0)^{-1}}$$

where $\ell_i(\cdot)$ is the i-th Lagrange polynomial basis. Note that this is a valid signature on m under the corresponding public-key defined as

$$pk_i = \left(\frac{pk}{\prod_{j \in [t-1]} h_j^{\ell_j(0)}} \right)^{\ell_i(0)^{-1}} .$$

This ensures that we can reconstruct (via Lagrange interpolation) the valid signature σ from *any* t-sized set of shares of the signature. Analogously, we can reconstruct the public key pk from any set of shares of size at least t.

The committer then computes a time-lock puzzle Z_i with time parameter $\mathbf{T}$ for each share separately. The first message consists of all puzzles $(Z_1, \dots, Z_n)$ together with all public keys $(pk_1, \dots, pk_n)$ as defined above. The verifier then chooses a random set I of size $(t-1)$. For the challenge set, the committer opens the time-lock puzzles $\{Z_i\}_{i \in I}$ and reveals the underlying message σ_i (together with the corresponding random coins) that it committed to. The verifier accepts the commitment as legitimate if all of the following conditions are satisfied:

(1) All $\{\sigma_i\}_{i \in I}$ are consistent with the corresponding public-key pk, i.e., $e(g_0, \sigma_i) = e(pk_i, H(m))$.
(2) All public keys $\{pk_j\}_{j \notin I}$ reconstruct to the public key of the scheme, i.e., $\prod_{i \in I} pk_i^{\ell_i(0)} \cdot pk_j^{\ell_j(0)} = pk$.

Taken together, these conditions ensure that, as long as at least one of the partial signatures in the *unopened* puzzles is consistent with respect to the corresponding partial public-key, then we can use it to reconstruct σ. This means that a malicious prover would need to guess the set I ahead of time to pass the above checks without actually committing a valid signature σ. Setting t and n appropriately we can guarantee that this happens only with negligible probability.

We exploit similar structural features in the case of Schnorr and ECDSA signatures. In case of Schnorr we additionally secret share the randomness used in signing and in ECDSA we do not secret share the public key but only the randomness and the signature.

Reducing the Work of the Verifier. As described above, our protocol requires the verifier to solve $\tilde{n} = (n - t + 1)$ puzzles to force the opening of a VTS. Ideally, we would like to reduce his workload to the minimal one of solving a single puzzle. If this was not the case, some applications may have users with $\tilde{n}$ processors who can solve $\tilde{n}$ puzzles in parallel and spending time $\mathbf{T}$ in total. While other users with less number of processors will have to solve the puzzles one by one thereby spending more time than $\mathbf{T}$. This could drastically affect security in the case of PCN for instance, where a honest user with less number of processors may be still solving the VTS while his steal transactions becomes invalid on the chain. Our observation is that if the time-lock puzzle has some homomorphic properties, then this can indeed be achieved. Specifically, if we instantiate the time-lock puzzle with a recently introduced linearly homomorphic construction [41], then we can use standard packing techniques to compress $\tilde{n}$ puzzles into a single one Section 4.4. Concretely, the verifier, on input $(Z_1, \dots, Z_{\tilde{n}})$ homomorphically evaluates the linear function

$$f(x_1, \dots, x_{\tilde{n}}) = \sum_{i=1}^{\tilde{n}} 2^{(i-1)\cdot\lambda} \cdot x_i$$

to obtain a single puzzle $\tilde{Z}$, which he can solve in time $\mathbf{T}$. Observe that, once the puzzle is solved, all signatures can be decoded from the bit-representations of the resulting message. However this transformation comes with two caveats:

(1) The message space of the homomorphic time-lock puzzle must be large enough to accommodate for all $\tilde{n}$ signatures.
(2) The signatures σ_i encoded in the the input puzzles must not exceed the maximum size of a signature (say λ bits).

Condition (1) can be satisfied instantiating the linearly homomorphic time-lock puzzles with a large enough message space. On the other hand, condition (2) is enforced by including a range NIZK, which certifies that the message of each time-lock puzzles falls into the range $[0, 2^\lambda]$.

Efficient Range Proofs. What is left to be discussed is how to implement the range NIZK for homomorphic time-lock puzzle. In the following we outline a protocol that allows us to prove the well-formedness of ℓ puzzles with proof size *logarithmic in* ℓ. The proof is generically applicable to any homomorphic time-lock puzzle (or even encryption scheme) that is linearly homomorphic over both the plaintext space and the randomness space, i.e.

$$\mathsf{PGen}(\mathbf{T}, m; r) \cdot \mathsf{PGen}(\mathbf{T}, m'; r') = \mathsf{PGen}(\mathbf{T}, m + m'; r + r').$$

Our proof system uses similar ideas as the range proof system of [37], but we are able to batch range proofs for a large number ℓ of homomorphic time-lock puzzles in a proof which has size independent of ℓ.

For the sake of this overview, let us assume that we want to make sure that plaintexts lie in an interval $[-L, L]$. However, to prove correctness and zero-knowledge we will need to require that honest plaintexts actually lie in a much smaller range $[-B, B]$, where B/L is negligible. This will introduce a slack in the size of the time-lock puzzles, which for practical purposes is roughly 50 bits.

We describe the protocol in its interactive form, although the actual instantiation is going to be made non-interactive via the standard Fiat-Shamir transformation. The prover is given ℓ puzzles $(Z_1, \dots, Z_\ell)$ together with each corresponding plaintext x_i and randomness r_i. The prover samples a drowning term y uniformly from the interval $[-L/4, L/4]$, then computes time-lock puzzles $D = \mathsf{PGen}(\mathbf{T}, y; r')$ for some randomness r'. The verifier is given all puzzles (including the one that contains the drowning term) and returns a random subset I of these puzzles. The prover computes the homomorphic sum of the selected puzzles

$$Z = \prod_{i \in I} Z_i \cdot D = \prod_{i \in I} \mathsf{PGen}(\mathbf{T}, x_i; r_i) \cdot \mathsf{PGen}(\mathbf{T}, y; r').$$

By the plaintext and randomness homomorphism, this is equal to

$$Z = \mathsf{PGen}\left(\mathbf{T}, \sum_{i \in I} x_i + y; \sum_{i \in I} r_i + r'\right).$$

The prover computes the opening for Z, i.e. $\sum_{i \in I} x_i + y$ and $\sum_{i \in I} r_i + r'$, and sends them to the verifier. The verifier accepts if (i) Z is correctly computed (which he can check since he is given the random coins) and if (ii) the plaintext $\sum_{i \in I} x_i + y$ lies within the interval $[-L/2, L/2]$. Given that B is sufficiently smaller than L, specifically $B \leq L/(4\ell)$ the protocol is correct. We can further show that, if any of the input plaintexts is outside the range $[-L, L]$, then the above check fails with constant probability. Negligible soundness is then achieved by repeating the above procedure k times in parallel. For zero-knowledge it suffices to observe that the random term $\tilde{m}$ statistically hides any information about $\sum_{i \in I} x_i$ by a standard drowning argument, given that B/L is negligible.

2.1 Related Work

Notice that VTS can also be seen as a "timed" variant of verifiably encrypted signatures [13, 27], with the difference that no trusted party is needed to recover the signature. Boneh and Naor [15] give an interactive protocol to prove that a time-lock puzzle is well-formed. The verifier is convinced that the sequential squaring is correctly performed. They identify several applications of time-lock puzzles. Garay and Jakobsson [24] and later Garay and Pomerance [25] proposed constructions where they construct special-purpose zero-knowledge proofs to convince a verifier that the time-lock puzzle indeed has a valid signature embedded. However their construction requires both the prover and the verifier to locally store a list of group elements as a "time-line" whose length is equal to the number of timed checkpoints. For instance, the time-line consists of $\mathbf{T}$ group elements if the largest timing hardness is $2^{\mathbf{T}}$. And in a multi-user system, a single user may have to store several time-lines of several other users with whom he has interaction. If they run a one-time setup for the whole system, it needs to be accompanied by a proof of well-formedness of the time-line. To the best of our knowledge, these protocols have never been implemented and in contrast, with our construction, the setup consist of an RSA modulus N and can be shared across all users in the system or sampled by the signer, depending on the application.

Banasik, Dziembowski and Malinowski [7] propose a cut and choose technique to prove that a time-lock puzzle has a valid signing key embedded. The prover sends a puzzles with signing keys for a public keys and the verifier asks to open $a - b$ of them. The verifier checks if the opened puzzles are well-formed and solves the rest of the puzzles. The verifier can finally post a transaction spending from a 'b-out of-$2b - 1$' multisig script where $b - 1$ of the keys are verifier's keys. For a 2^{-48} security they require $b = 8$ which means the spending transaction consists of 8 signatures and 15 public keys. Our VTS and VTC constructions would only require the solver to solve a *single* puzzle after homomorphic evaluation and post a transaction with *single* signature for a corresponding public key. As stated before, given that they require b puzzles to be solved, this could lead problems in applications such as PCN if users have different parallel processing power. Moreover, since signing keys are embedded, parties in their protocol can learn the signing keys of other parties after a given time, contrary to our VTS where parties only learn signatures. There may be scenarios where parties may not wish to share their signing keys: Learning a single signing key could compromise security of the entire wallet of the party [1] (especially in cases of hierarchical wallets).

3 PRELIMINARIES

We denote by $\lambda \in \mathbb{N}$ the security parameter and by $x \leftarrow \mathcal{A}(\mathsf{in})$ the output of the algorithm $\mathcal{A}$ on input in. We denote by $\mathcal{A}(\mathsf{in}; r)$ if algorithm $\mathcal{A}$ is randomized with $r \leftarrow \{0,1\}^*$ as its randomness. We omit this randomness where it is obvious and only mention it explicitly when required. We denote the set $\{1, \ldots, n\}$ by $[n]$.

3.1 Cryptographic Building Blocks

We recall the cryptographic primitives used in our protocol and refer to Appendix A for formal definitions and security.

Digital Signatures. A digital signature scheme consists of the following triple of efficient algorithms: A key generation algorithm $\mathsf{KGen}(1^\lambda)$ that takes as input the security parameter 1^λ and outputs the public/secret key pair (pk, sk). The signing algorithm $\mathsf{Sign}(sk, m)$ inputs a secret key and a message $m \in \{0,1\}^*$ and outputs a signature σ. The verification algorithm $\mathsf{Vf}(pk, m, \sigma)$ outputs 1 if σ is a valid signature on m under the public key pk, and outputs 0 otherwise. We require standard notions of correctness and unforgeability for the signature scheme [29].

Time-Lock Puzzles. A time-lock puzzle $(\mathsf{PGen}, \mathsf{PSolve})$ allows one to conceal a value for a certain amount of time [46]. Intuitively, time-lock puzzles guarantee that a puzzle can be solved in polynomial time, but strictly higher than $\mathbf{T} \in \mathbb{N}$. The only efficient candidate construction of time-lock puzzles was given by Rivest, Shamir, and Wagner and is based on the sequential squaring assumption [46]. The puzzle generation PGen is a probabilistic algorithm that takes as input a hardness-parameter $\mathbf{T}$, a solution $s \in \{0,1\}^*$ and some random coins r, and outputs a puzzle Z. The solving algorithm PSolve takes as input a puzzle Z and outputs a solution s. In this context, we refer to Parallel Random Access Machines (PRAM): which is a model considered for most of the parallel algorithms. Multiple processors are attached to a single block of memory and n number of processors can perform independent operations on n number of data in a particular unit of time. The security requirement is that for every PRAM adversary $\mathcal{A}$ of running time $\leq \mathbf{T}^{\varepsilon}(\lambda)$, and every pair of solutions $(s_0, s_1) \in \{0,1\}^2$, it cannot distinguish a puzzle Z that is generated with solution s_0 from a puzzle generated with solution s_1 where the timing hardness of the puzzle is $\mathbf{T}$ except with negligible probability.

Homomorphic Time-Lock Puzzles. Homomorphic Time-Lock Puzzles (HTLPs) were proposed by Malavolta and Thyagarajan [41]. An HTLP is a tuple of four algorithms $(\mathsf{HTLP.PSetup}, \mathsf{HTLP.PGen}, \mathsf{HTLP.PSolve}, \mathsf{HTLP.PEval})$ that lets one perform homomorphic operations over different time-lock puzzles. Apart from the two algorithms for a time-lock puzzle, HTLPs additionally have a setup algorithm PSetup and a homomorphic evaluation algorithm PEval: PSetup takes as input a security parameter 1^λ and a time hardness parameter $\mathbf{T}$, and outputs public parameters pp, and PEval takes as input a circuit $C : \{0,1\}^n \rightarrow \{0,1\}$, public parameters pp and a set of n puzzles $Z_1, \ldots, Z_n$ and outputs a puzzle Z'. The puzzle generation and solving algorithms also take the public parameters pp as input. The homomorphism property for computing a circuit C states that $\Pr\left[\mathsf{HTLP.PSolve}(pp, Z') \neq C(s_1, \ldots, s_n)\right] \leq \mu(\lambda)$, where $Z' \leftarrow \mathsf{HTLP.PEval}(C, pp, Z_1, \ldots, Z_n)$ and $Z_i \leftarrow \mathsf{HTLP.PGen}(pp, s_i)$ for $(s_1, \ldots, s_n) \in \{0,1\}^n$.

In their work, they show an efficient construction that is linearly homomorphic over the ring $\mathbb{Z}_{N^s}$, where N is an RSA modulus and s is an arbitrary constant. The scheme is perfectly correct and it satisfies the notion of randomness homomorphism, which is needed for our purposes.

Non-Interactive Zero-Knowledge. Let $R : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}$ be a n *NP*-witness-relation with corresponding *NP*-language $\mathcal{L} := \{x : \exists w \text{ s.t. } R(x,w) = 1\}$. A non-interactive zero-knowledge proof (NIZK) [17] system for R is initialized with a setup algorithm $\mathsf{ZKsetup}(1^\lambda)$ that, on input the security parameter, outputs a common reference string *crs*. A prover can show the validity of a statement x with a witness w by invoking $\mathsf{ZKprove}(crs, x, w)$, which outputs a proof π. The proof π can be efficiently checked by the verification algorithm $\mathsf{ZKverify}(crs, x, \pi)$. We require a NIZK system to be (1) zero-knowledge, where the verifier does not learn more than the validity of the statement x, and (2) simulation sound, where it is hard for any prover to convince a verifier of an invalid statement (chosen by the prover) even after having access to polynomially many simulated proofs for statements of his choosing.

Threshold Secret Sharing. Secret sharing is a method of creating shares of a given secret and later reconstructing the secret itself only if given a threshold number of shares. Shamir [48] proposed a threshold secret sharing scheme where the SS.share algorithm takes a secret $s \in \mathbb{Z}_q$ and generates shares $(s_1, \ldots, s_n)$ each belonging to $\mathbb{Z}_q$. The SS.reconstruct algorithm takes as input at least t shares and outputs a secret s. The security of the secret sharing scheme demands that knowing only a set of shares smaller than the threshold size does *not* help in learning any information about the choice of the secret s.

3.2 Verifiable Timed Signatures

A timed signature [15] is a scheme when a committer commits to a signature on a message and shares it with some user. After some time $\mathbf{T}$ has passed, the committer reveals the committed signature to the user. If he fails to reveal the signature, then the user is guaranteed to forcibly retrieve the signature from the timed commitment given initially. We explicitly state the notion of verifiability for a timed signature, and therefore refer to it as a *Verifiable Timed Signature* (VTS), which lets the user verify if the signature σ committed to in C can be obtained by ForceOp in time $\mathbf{T}$ and is indeed a valid signature on the message m, that is, if $\mathsf{Vf}(pk, m, \sigma) = 1$ in a *non-interactive* manner. This verifiability ensures that the user is guaranteed to obtain a valid signature from the commitment C which he can retrieve using ForceOp. For the sake of clarity, we let Commit additionally output a proof π for the embedded signature to be a valid signature on the message m with respect to pk and we have a Vrfy algorithm that is defined below.

Definition 1 (Verifiable Timed Signatures). *A VTS for a signature scheme* $\Pi = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ *is a tuple of four algorithms* $(\mathsf{Commit}, \mathsf{Vrfy}, \mathsf{Open}, \mathsf{ForceOp})$ *where:*

- $(C, \pi) \leftarrow \mathsf{Commit}(\sigma, \mathbf{T})$*: the commit algorithm (randomized) takes as input a signature* σ *(generated using* $\Pi.\mathsf{Sign}(sk, m)$*) and a hiding time* $\mathbf{T}$ *and outputs a commitment* C *and a proof* π*.*
- $0/1 \leftarrow \mathsf{Vrfy}(pk, m, C, \pi)$*: the verify algorithm takes as input a public key* pk*, a message* m*, a commitment* C *of hardness* $\mathbf{T}$ *and a proof* π *and accepts the proof by outputting* 1 *if and only if, the value* σ *embedded in* c *is a valid signature on the message* m *with respect to the public key* pk *(i.e.,* $\Pi.\mathsf{Vf}(pk, m, \sigma) = 1$*). Otherwise it outputs* 0*.*
- $(\sigma, r) \leftarrow \mathsf{Open}(C)$*: the open phase where the committer takes as input a commitment* C *and outputs the committed signature* σ *and the randomness* r *used in generating* C*.*
- $\sigma \leftarrow \mathsf{ForceOp}(C)$*: the force open algorithm takes as input the commitment* C *and outputs a signature* σ*.*

The security requirements for a VTS are that (*soundness*) the user is convinced that, given C, the ForceOp algorithm will produce the committed signature σ in time $\mathbf{T}$ and that (*privacy*) all PRAM algorithms whose running time is at most t (where $t < \mathbf{T}$) succeed in extracting σ from the commitment C and π with at most negligible probability. We formalize the definition of soundness below.

Definition 2 (Soundness). *A VTS scheme* $\mathsf{VTS} = (\mathsf{Commit}, \mathsf{Vrfy}, \mathsf{Open}, \mathsf{ForceOp})$ *for a signature scheme* $\Pi = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ *is* sound *if there is a negligible function negl such that for all probabilistic polynomial time adversaries* $\mathcal{A}$ *and all* $\lambda \in \mathbb{N}$*, we have:*

$$\Pr\left[b_1 = 1 \land b_2 = 0 : \begin{array}{r} (pk, m, C, \pi, \mathbf{T}) \leftarrow \mathcal{A}(1^\lambda) \\ (\sigma, r) \leftarrow \mathsf{ForceOp}(C) \\ b_1 := \mathsf{Vrfy}(pk, m, C, \pi) \\ b_2 := \Pi.\mathsf{Vf}(pk, m, \sigma) \end{array}\right] \leq negl(\lambda).$$

We say that a VTS is *simulation-sound* if it is sound even when the prover has access to simulated proofs for (possibly false) statements of his choice; i.e., the prover must not be able to compute a valid proof for a fresh false statement of his choice. In the following definition we present the definition of privacy.

Definition 3 (Privacy). *A VTS scheme* $\mathsf{VTS} = (\mathsf{Commit}, \mathsf{Vrfy}, \mathsf{Open}, \mathsf{ForceOp})$ *for a signature scheme* $\Pi = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ *is* private *if there exists a PPT simulator* $\mathcal{S}$*, a negligible function negl, and a polynomial* $\tilde{\mathbf{T}}$ *such that for all polynomials* $\mathbf{T} > \tilde{\mathbf{T}}$*, all PRAM algorithms* $\mathcal{A}$ *whose running time is at most* $t < \mathbf{T}$*, all messages* $m \in \{0,1\}^*$*, and all* $\lambda \in \mathbb{N}$ *it holds that*

$$\left|\begin{array}{l} \Pr\left[\mathcal{A}(pk, m, C, \pi) = 1 : \begin{array}{r} (pk, sk) \leftarrow \Pi.\mathsf{KGen}(1^\lambda) \\ \sigma \leftarrow \Pi.\mathsf{Sign}(sk, m) \\ (C, \pi) \leftarrow \mathsf{Commit}(\sigma, \mathbf{T}) \end{array}\right] \\ -\Pr\left[\mathcal{A}(pk, m, C, \pi) = 1 : \begin{array}{r} (pk, sk) \leftarrow \Pi.\mathsf{KGen}(1^\lambda) \\ (C, \pi, m) \leftarrow \mathcal{S}(pk, \mathbf{T}) \end{array}\right] \end{array}\right| \leq negl(\lambda).$$

Setup: On input 1^λ the setup algorithm does the following.
- Run $\mathsf{ZKsetup}(1^\lambda)$ to generate crs_{range}
- Generate the public parameters $pp \leftarrow \mathsf{LHTLP.PSetup}(1^\lambda, \mathbf{T})$
- Output $crs := (crs_{\text{range}}, pp)$

Commit and Prove: On input (crs, wit) the Commit algorithm does the following.
- Parse $\mathsf{wit} := \sigma$, $crs := (crs_{\text{range}}, pp)$, pk as the BLS public key, and m as the message to be signed
- For all $i \in [t-1]$ sample a uniform $\alpha_i \leftarrow \mathbb{Z}_q$ and set $\sigma_i = H(m)^{\alpha_i}\ h_i := g_0^{\alpha_i}$
- For all $i \in \{t, \dots, n\}$ compute
$$\sigma_i = \left(\frac{\sigma}{\prod_{j\in[t-1]} \sigma_j^{\ell_j(0)}}\right)^{\ell_i(0)^{-1}}, h_i = \left(\frac{pk}{\prod_{j\in[t-1]} h_j^{\ell_j(0)}}\right)^{\ell_i(0)^{-1}}$$
where $\ell_i(\cdot)$ is the i-th Lagrange polynomial basis.
- For $i \in [n]$, generate puzzles with corresponding range proofs as shown below
$$r_i \leftarrow \{0,1\}^\lambda, Z_i \leftarrow \mathsf{LHTLP.PGen}(pp, \sigma_i; r_i)$$
$$\pi_{\text{range},i} \leftarrow \mathsf{ZKprove}(crs_{\text{range}}, (Z_i, 0, 2^\lambda, \mathbf{T}), (\sigma_i, r_i))$$
- Compute $I \leftarrow H'\left(pk, (h_1, Z_1, \pi_{\text{range},1}), \dots, (h_n, Z_n, \pi_{\text{range},n})\right)$
- Output $C := (Z_1, \dots, Z_n, \mathbf{T})$ and $\pi := (\{h_i, \pi_{\text{range},i}\}_{i\in[n]}, I, \{\sigma_i, r_i\}_{i\in I})$

Verification: On input (crs, pk, m, C, π) the Vrfy algorithm does the following.
- Parse $C := (Z_1, \dots, Z_n, \mathbf{T})$, $\pi := (\{h_i, \pi_{\text{range},i}\}_{i\in[n]}, I, \{\sigma_i, r_i\}_{i\in I})$ and $crs := (crs_{\text{range}}, pp)$
- If any of the following conditions is satisfied output 0, else return 1:
  (1) There exists some $j \notin I$ such that $\prod_{i\in I} h_i^{\ell_i(0)} \cdot h_j^{\ell_j(0)} \neq pk$
  (2) There exists some $i \in [n]$ such that $\mathsf{ZKverify}(crs_{\text{range}}, (Z_i, 0, 2^\lambda, \mathbf{T}), \pi_{\text{range},i}) \neq 1$
  (3) There exists some $i \in I$ such that $Z_i \neq \mathsf{LHTLP.PGen}(pp, \sigma_i; r_i)$ or $e(g_0, \sigma_i) \neq e(h_i, H(m))$
  (4) $I \neq H'\left(pk, (h_1, Z_1, \pi_{\text{range},1}), \dots, (h_n, Z_n, \pi_{\text{range},n})\right)$

Open: The Open algorithm outputs $(\sigma, \{r_i\}_{i\in[n]})$.
Force Open: The ForceOp algorithm take as input $C := (Z_1, \dots, Z_n, \mathbf{T})$ and works as follows:
- Runs $\sigma_i \leftarrow \mathsf{LHTLP.PSolve}(pp, Z_i)$ for $i \in [n]$ to obtain all signatures. Notice that since $t-1$ puzzles are already opened by the committer, this only means that ForceOp has to solve only $(n-t+1)$ puzzles.
- Output $\sigma := \prod_{j\in[t]} (\sigma_j)^{\ell_j(0)}$ where wlog., the first t signatures are valid shares.

Figure 2: VT-BLS Signatures

4 EFFICIENT VTS CONSTRUCTIONS

In the following sections we construct VTS for BLS, Schnorr and ECDSA signatures. The key ingredients for constructing VTS are time-lock puzzles, specifically we consider the *Linearly*-HTLP [41] ($\mathsf{LHTLP.PSetup}, \mathsf{LHTLP.PGen}, \mathsf{LHTLP.PSolve}, \mathsf{LHTLP.PEval}$) and public coin interactive zero-knowledge proofs for the language $\mathcal{L}$ described as follows.
$$\mathcal{L} := \begin{Bmatrix} \mathsf{stmt} = (pk, m, Z, \mathbf{T}) : \exists \mathsf{wit} = (\sigma, r) \text{ s.t.} \\ (\mathsf{Vf}(pk, m, \sigma) = 1) \ \wedge (Z \leftarrow \mathsf{LHTLP.PGen}(\mathbf{T}, \sigma; r)) \end{Bmatrix}$$
The Commit algorithm embeds the signatures inside time-lock puzzles and uses the zero-knowledge proof system for $\mathcal{L}$ to prove the validity of the time-locked signature. In practice all of the schemes will be made non-interactive using the Fiat-Shamir transformation [22]. We additionally make use of a zero-knowledge proof system ($\mathsf{ZKsetup}, \mathsf{ZKprove}, \mathsf{ZKverify}$) for the language $\mathcal{L}_{\text{range}}$ as defined below. Intuitively, the language consists of all puzzles whose solution lies in some range $[a, b]$. We give an efficient instantiation of this proof system in Section 4.5.
$$\mathcal{L}_{\text{range}} := \begin{Bmatrix} \mathsf{stmt} = (Z, a, b, \mathbf{T}) : \exists \mathsf{wit} = (\sigma, r) \text{ s.t.} \\ (Z \leftarrow \mathsf{LHTLP.PGen}(\mathbf{T}, \sigma; r)) \ \wedge (\sigma \in [a, b]) \end{Bmatrix}$$
In all protocols described in Figures 2 to 4 we let n be a statistical security parameter and set $t := n/2 + 1$. We let $|\sigma| = \lambda$ is the max number of bits of the signature σ. Define a hash function $H' : \{0,1\}^* \to I \subset [n]$ with $|I| = t - 1$ modeled as a random oracle. Throughout the following description, we make the simplifying assumption that the ForceOp algorithm solves $\tilde{n} = (n - t + 1)$ puzzles in parallel. In Section 4.4 we show how to reduce the number of puzzles to solve to a *single* puzzle exploiting the (linear) homomorphic evaluation algorithm of time-lock puzzles.

4.1 Verifiable Timed BLS Signatures (VT-BLS)

Let $(\mathbb{G}_0, \mathbb{G}_1, \mathbb{G}_t)$ be a bilinear group of prime order q, where q is a λ bit prime. Let e be an efficiently computable bilinear pairing $e : \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_T$, where g_0 and g_1 are generators of $\mathbb{G}_0$ and $\mathbb{G}_1$ respectively. Let H be a hash function $H : \{0,1\}^* \to \mathbb{G}_1$ modeled as a random oracle. We briefly recall here the BLS construction [14] and our VT-BLS protocol is described in Figure 2.

- $(pk, sk) \leftarrow \mathsf{KGen}(1^\lambda)$: Choose $\alpha \leftarrow \mathbb{Z}_q$, set $h \leftarrow g_0^\alpha \in \mathbb{G}_0$ and output $pk := h$ and $sk := \alpha$.
- $\sigma \leftarrow \mathsf{Sign}(sk, m)$: Output $\sigma := H(m)^{sk} \in \mathbb{G}_1$.
- $0/1 \leftarrow \mathsf{Vf}(pk, m, \sigma)$: If $e(g_0, \sigma) = e(pk, H(m))$, then output 1 and otherwise output 0.

The following theorems show that our construction from Figure 2 satisfies privacy and soundness. The formal proofs are deferred to Appendix B.1.

Theorem 1 (Privacy). *Let* ($\mathsf{ZKsetup}, \mathsf{ZKprove}, \mathsf{ZKverify}$) *be a NIZK for* $\mathcal{L}_{\text{range}}$ *and let* LHTLP. *be a secure time-lock puzzle. Then the protocol as described in Figure 2 satisfies privacy as in Definition 3 in the random oracle model.*

Theorem 2 (Soundness). *Let* ($\mathsf{ZKsetup}, \mathsf{ZKprove}, \mathsf{ZKverify}$) *be a NIZK for* $\mathcal{L}_{\text{range}}$ *and let* LHTLP. *be a time-lock puzzle with perfect correctness. Then the protocol as described in Figure 2 satisfies soundness as in Definition 2 in the random oracle model.*

4.2 Verifiable Timed Schnorr Signatures (VT-Schnorr)

The Schnorr signature scheme [47] is defined over a cyclic group $\mathbb{G}$ of prime order q with generator g, and use a hash function H modeled as a random oracle. We briefly recall the construction here and VT-Schnorr protocol is given in Figure 3.

- $(pk, sk) \leftarrow \mathsf{KGen}(1^\lambda)$: Choose $x \leftarrow \mathbb{Z}_q$ and set $sk := x$ and $pk := g^x$.
- $\sigma \leftarrow \mathsf{Sign}(sk, m; r)$: Sample a randomness $r \leftarrow \mathbb{Z}_q$ to compute $R := g^r, c := H(g^x, R, m), s := r + cx$ and output $\sigma := (R, s)$.
- $0/1 \leftarrow \mathsf{Vf}(pk, m, \sigma)$: Parse $\sigma := (R, s)$ and then compute $c := H(pk, R, m)$ and if $g^s = R \cdot pk^c$ output 1, otherwise output 0.

In the following theorems we show that our construction of VT-Schnorr from Figure 3 satisfies privacy and soundness. The formal proofs are deferred to Appendix B.2.

Theorem 3 (Privacy). *Let* (ZKsetup, ZKprove, ZKverify) *be a NIZK for* $\mathcal{L}_{\mathsf{range}}$ *and let* LHTLP. *be a secure time-lock puzzle. Then the protocol as described in Figure 3 satisfies privacy as in Definition 3 in the random oracle model.*

Theorem 4 (Soundness). *Let* (ZKsetup, ZKprove, ZKverify) *be a NIZK for* $\mathcal{L}_{\mathsf{range}}$ *and let* LHTLP. *be a time-lock puzzle with perfect correctness. Then the protocol as described in Figure 3 satisfies soundness as in Definition 2 in the random oracle model.*

4.3 Verifiable Timed ECDSA Signatures (VT-ECDSA)

The ECDSA signature scheme [28] is defined over an elliptic curve group $\mathbb{G}$ of prime order q with base point (generator) g. The construction assumes the existence of a hash function $H : \{0, 1\}^* \to \mathbb{Z}_q$ and is given in the following. Our VT-ECDSA protocol is given in Figure 4.

- $(pk, sk) \leftarrow \mathsf{KGen}(1^\lambda)$: Choose $x \leftarrow \mathbb{Z}_q$ and set $sk := x$ and $pk := g^x$.
- $\sigma \leftarrow \mathsf{Sign}(sk, m; r)$: Sample an integer $k \leftarrow \mathbb{Z}_q$ and compute $c \leftarrow H(m)$. Let $(r_x, r_y) := R = g^k$, then set $r := r_x \mod q$ and $s := (c + rx)/k \mod q$. Output $\sigma := (r, s)$.
- $0/1 \leftarrow \mathsf{Vf}(pk, m, \sigma)$: Parse $\sigma := (r, s)$ and compute $c := H(m)$ and return 1 if and only if $(x, y) = (g^c \cdot h^r)^{s^{-1}}$ and $x = r \mod q$. Otherwise output 0.

Notice that ECDSA signature has a non-linear verification unlike in Schnorr. Consequently, notice that unlike VT-BLS and VT-Schnorr, the public key is not secret shared in VT-ECDSA.

The theorems below are for privacy and soundness of our VT-ECDSA protocol. The formal proofs are deferred to Appendix B.3.

Theorem 5 (Privacy). *Let* (ZKsetup, ZKprove, ZKverify) *be a NIZK for* $\mathcal{L}_{\mathsf{range}}$ *and let* LHTLP. *be a secure time-lock puzzle. Then the protocol as described in Figure 4 satisfies privacy as in Definition 3 in the random oracle model.*

Theorem 6 (Soundness). *Let* (ZKsetup, ZKprove, ZKverify) *be a NIZK for* $\mathcal{L}_{\mathsf{range}}$ *and let* LHTLP. *be a time-lock puzzle with perfect correctness. Then the protocol as described in Figure 4 satisfies soundness as in Definition 2 in the random oracle model.*

Setup: Same as Figure 2.

Commit and Prove: On input (crs, wit) the Commit algorithm does the following.

- Parse $\mathsf{wit} := \sigma = (R, s)$, $crs := (crs_{\mathsf{range}}, pp)$, pk as the Schnorr public key, and m as the message to be signed
- For all $i \in [t-1]$ sample a uniform pair $(x_i, k_i) \leftarrow \mathbb{Z}_q$ and set $h_i := g^{x_i}, R_i := g^{k_i}$, and $s_i := k_i + cx_i$ where $c = H(pk, R, m)$
- For all $i \in \{t, \dots, n\}$ compute

$$s_i = \left(s - \sum_{j \in [t-1]} s_j \cdot \ell_j(0)\right) \cdot \ell_i(0)^{-1}, \; h_i = \left(\frac{pk}{\prod_{j \in [t-1]} h_j^{\ell_j(0)}}\right)^{\ell_i(0)^{-1}}$$

$$R_i = \left(\frac{R}{\prod_{j \in [t-1]} R_j^{\ell_j(0)}}\right)^{\ell_i(0)^{-1}}$$

  where $\ell_i(\cdot)$ is the i-th Lagrange polynomial basis
- For $i \in [n]$, generate puzzles with corresponding range proofs as shown below ($|\sigma| = \lambda$ is the max number of bits of σ)

$$r_i \leftarrow \{0, 1\}^\lambda, Z_i \leftarrow \mathsf{LHTLP.PGen}(pp, s_i; r_i)$$

$$\pi_{\mathsf{range},i} \leftarrow \mathsf{ZKprove}(crs_{\mathsf{range}}, (Z_i, 0, 2^\lambda, \mathbf{T}), (s_i, r_i))$$

- Compute $I \leftarrow H'\left(pk, R, (h_1, R_1, Z_1, \pi_{\mathsf{range},1}), \dots, (h_n, R_n, Z_n, \pi_{\mathsf{range},n})\right)$
- Output $C := (R, Z_1, \dots, Z_n, \mathbf{T})$ and $\pi := (\{h_i, R_i, \pi_{\mathsf{range},i}\}_{i \in [n]}, I, \{s_i, r_i\}_{i \in I})$

Verification: On input (crs, pk, m, C, π) the Vrfy algorithm does the following.

- Parse $C := (R, Z_1, \dots, Z_n, \mathbf{T})$, $\pi := (\{h_i, R_i, \pi_{\mathsf{range},i}\}_{i \in [n]}, I, \{s_i, r_i\}_{i \in I})$, and $crs := (crs_{\mathsf{range}}, pp)$
- If any of the following conditions is satisfied output 0, else return 1:
  1. There exists some $j \notin I$ such that $\prod_{i \in I} h_i^{\ell_i(0)} \cdot h_j^{\ell_j(0)} \neq pk$ or $\prod_{i \in I} R_i^{\ell_i(0)} \cdot R_j^{\ell_j(0)} \neq R$
  2. There exists some $i \in [n]$ such that $\mathsf{ZKverify}(crs_{\mathsf{range}}, (Z_i, 0, 2^\lambda, \mathbf{T}), \pi_{\mathsf{range},i}) \neq 1$
  3. There exists some $i \in I$ such that $Z_i \neq \mathsf{LHTLP.PGen}(pp, s_i; r_i)$ or $g^{s_i} \neq R_i \cdot h_i^c$
  4. $I \neq H'\left(pk, R, (h_1, R_1, Z_1, \pi_{\mathsf{range},1}), \dots, (h_n, R_n, Z_n, \pi_{\mathsf{range},n})\right)$

Open: The Open algorithm outputs $((R, s), \{r_i\}_{i \in [n]})$.

Force Open: The ForceOp algorithm take as input $C := (R, Z_1, \dots, Z_n, \mathbf{T})$ and works as follows:

- Runs $s_i \leftarrow \mathsf{LHTLP.PSolve}(pp, Z_i)$ for $i \in [n]$ to obtain all signatures. ForceOp has to solve only $(n - t + 1)$ puzzles, as $t - 1$ puzzles are already opened.
- Output $(R, s := \prod_{j \in [t]} (s_j)^{\ell_j(0)})$ where wlog., the first t are valid shares.

Figure 3: VT-Schnorr Signatures

4.4 Batching Puzzle Solving

As described above, our protocols require the verifier to solve $\tilde{n} = (n - t + 1)$ puzzles in the forced opening phase. In the following we show how to leverage the homomorphic properties of the time-lock puzzles to ensure that the computation is reduced to the solution of a *single* puzzle, regardless of the parameters n and t. This is crucial for our real world applications as without HTLP, users with different degrees of parallelisms can solve $\tilde{n}$ puzzles in different times: A user with several computers can solve $\tilde{n}$ puzzles in parallel

Setup: Same as Figure 2.
Commit and Prove: On input (crs, wit) the Commit algorithm does the following.

- Parse $\text{wit} := \sigma = (r, s)$, $crs := (crs_{\text{range}}, pp)$, pk as the ECDSA public key, and m as the message to be signed
- Define $R := (x, y) = (g^c \cdot h^r)^{s^{-1}}$ and $B = g^c \cdot h^r$, where $c = H(m)$
- For all $i \in [t-1]$ sample a uniform pair $s_i \leftarrow \mathbb{Z}_q$ and set $R_i := B^{s_i}$
- For all $i \in \{t, \ldots, n\}$ compute
$$s_i = \left(s^{-1} - \sum_{j\in[t-1]} s_j \cdot \ell_j(0)\right) \cdot \ell_i(0)^{-1}, \text{ and}$$
$$R_i = \left(\frac{R}{\prod_{j\in[t-1]} R_j^{\ell_j(0)}}\right)^{\ell_i(0)^{-1}}$$
where $\ell_i(\cdot)$ is the i-th Lagrange polynomial basis
- For $i \in [n]$, generate puzzles with corresponding range proofs as shown below ($|\sigma| = \lambda$ is the max number of bits of σ)
$$r_i \leftarrow \{0,1\}^\lambda, Z_i \leftarrow \mathsf{LHTLP.PGen}(pp, s_i; r_i)$$
$$\pi_{\text{range},i} \leftarrow \mathsf{ZKprove}(crs_{\text{range}}, (Z_i, 0, 2^\lambda, \mathbf{T}), (s_i, r_i))$$
- Compute $I \leftarrow H'\left(pk, r, R, (R_1, Z_1, \pi_{\text{range},1}), \ldots, (R_n, Z_n, \pi_{\text{range},n})\right)$
- Output $C := (r, R, Z_1, \ldots, Z_n, \mathbf{T})$ and $\pi := (\{R_i, \pi_{\text{range},i}\}_{i\in[n]}, I, \{s_i, r_i\}_{i\in I})$

Verification: On input (crs, pk, m, C, π) the Vrfy algorithm does the following.

- Parse $C := (r, R, Z_1, \ldots, Z_n, \mathbf{T})$, $\pi := (\{R_i, \pi_{\text{range},i}\}_{i\in[n]}, I, \{s_i, r_i\}_{i\in I})$, and $crs := (crs_{\text{range}}, pp)$
- If any of the following conditions is satisfied output 0, else return 1:

(1) It holds that $x \neq r \mod q$ where $(x, y) := R$
(2) There exists some $j \notin I$ such that $\prod_{i\in I} R_i^{\ell_i(0)} \cdot R_j^{\ell_j(0)} \neq R$
(3) There exists some $i \in [n]$ such that $\mathsf{ZKverify}(crs_{\text{range}}, (Z_i, 0, 2^\lambda, \mathbf{T}), \pi_{\text{range},i}) \neq 1$
(4) There exists some $i \in I$ such that $Z_i \neq \mathsf{LHTLP.PGen}(pp, s_i; r_i)$ or $R_i \neq (g^c \cdot h^r)^{s_i}$
(5) $I \neq H'\left(pk, r, R, (R_1, Z_1, \pi_{\text{range},1}), \ldots, (R_n, Z_n, \pi_{\text{range},n})\right)$

Open: The Open algorithm outputs $((r, s), \{r_i\}_{i\in[n]})$.
Force Open: The ForceOp algorithm take as input $C := (r, R, Z_1, \ldots, Z_n, \mathbf{T})$ and works as follows:

- Obtain $s_i \leftarrow \mathsf{LHTLP.PSolve}(pp, Z_i)$ for $i \in [n]$ same as in Figure 3.
- Output $(r, s := \prod_{j\in[t]}(s_j)^{\ell_j(0)})$ where wlog., the first t are valid shares.

Figure 4: VT-ECDSA Signatures

effectively spending time $\mathbf{T}$, and a user with a single computer solves one puzzle after the other sequentially thus spending $\tilde{n} \cdot \mathbf{T}$.

The high-level idea of exploiting the homomorphism of LHTLP is to pack all partial signatures into a single puzzle, provided that the message space is large enough.

Concretely, the solver, given $\tilde{n}$ puzzles $Z_1, \ldots, Z_{\tilde{n}}$ encoding λ-bits signatures, homomorphically evaluates the linear function
$$f(x_1, \ldots, x_{\tilde{n}}) = \sum_{i=1}^{\tilde{n}} 2^{(i-1)\cdot\lambda} \cdot x_i$$
to obtain the puzzle $\tilde{Z}$, which can be solved in time $\mathbf{T}$. Observe that, once the puzzle is solved, all signatures can be decoded from the bit-representations of the resulting plaintext. Note that in order for this transformation to work we need two conditions to be satisfied:

(1) The signatures σ_i encoded in the the input puzzles must not exceed the maximum size of a signature (which we fix to λ bits)
(2) The message space of the homomorphic time-lock puzzle must be large enough to accommodate for all $\tilde{n}$ signatures.

Condition (1) is enforced by including a range NIZK, which certifies that the message of each time-lock puzzles falls into the range $[0, 2^\lambda]$. On the other hand we can satisfy condition (2) by instantiating the linearly homomorphic time-lock puzzles with modulus N^s, instead of N^2, for a large enough s. This is reminiscent of the Damgård-Jurik [16] extension of Paillier's cryptosystem [44] and was already suggested in [41].

We stress that, even though we can increase the message space arbitrarily, the squaring operations are still performed modulo N, which is important to reduce the gap between the honest and the malicious solver: While squaring is conjectured to be a sequential operation (in groups of unknown order), the computation of a *single squaring operation* can be internally parallelized to boot the overall efficiency of the algorithm. For this reason it is important to keep the modulus as small as possible, at least as far as squaring is concerned. For further details, we refer the reader to [41].

4.5 Range Proof for Homomorphic Time-Lock Puzzles

In this Section we will provide a protocol which allows a prover to convince a verifier in zero-knowledge that a list of linearly homomorphic time-lock puzzles are well-formed. This allows us to homomorphically pack them into a single time-lock puzzle.

Our protocol follows the Fiat-Shamir heuristic and we prove soundness and zero-knowledge in the random oracle model. For our construction we require a linearly homomorphic time-lock puzzle which is also homomorphic in the random coins. The construction of [41] satisfies this property.

In this Section we will always assume that plaintexts in a ring $\mathbb{Z}_q$ are represented via the central representation in $[-q/2, q/2]$.

Our protocol ensures that every plaintext is in the interval $[-L, L]$, given that $2L$ is smaller than the modulus of the plaintext space. We remark that this protocol can be readily used to prove that plaintexts are in a non-centered interval $[a, b]$ via homomorphically shifting plaintexts by $-(a+b)/2$, mapping the interval $[a, b]$ to $[-(b-a)/2, (b-a)/2]$. Consequently, for the sake of simplicity we will only discuss the case of centered intervals. In order to achieve zero-knowledge, we actually need that the plaintexts come form a smaller interval $[-B, B]$, where $B < L$. For our protocols, this means that we need to use slightly looser intervals when batching time-lock puzzles, but the efficiency of the schemes is otherwise

Setup: An RSA modulus N, public parameters pp for HTLP, interval parameters L and B with $B < L$. In this protocol we use k as a statistical security parameter.
Common input: Time-lock puzzles $Z_1, \ldots, Z_\ell$.
Prover: On input wit, where wit $:= ((x_1, r_1), \ldots, (x_\ell, r_\ell))$ and $x_i \in [-B, B]$ such that for all i it holds $Z_i \leftarrow \mathsf{HTLP.PGen}(\mathsf{pp}, x_i; r_i)$, the prover algorithm ZKprove does the following.

- Choose $y_1, \ldots, y_k \leftarrow [-L/4, L/4]$ and random coins $r'_1, \ldots, r'_k$ from their corresponding ring.
- For $i = 1, \ldots, k$ compute $D_i \leftarrow \mathsf{HTLP.PGen}(\mathsf{pp}, y_i; r'_i)$
- Compute $(\mathbf{t}_1, \ldots, \mathbf{t}_k) \leftarrow H(Z_1, \ldots, Z_\ell, D_1, \ldots, D_k)$, where the $\mathbf{t}_i \in \{0,1\}^\ell$.
- For $i = 1, \ldots k$ compute $v_i \leftarrow y_i + \sum_{j=1}^{\ell} t_{i,j} \cdot x_j$ and $w_i \leftarrow r'_i + \sum_{j=1}^{\ell} t_{i,j} \cdot r_j$
- Set $\pi \leftarrow (D_i, v_i, w_i)_{i \in [k]}$ and output π

Verifier: On input $\pi = (D_i, v_i, w_i)_{i \in [k]}$ the do the following.

- Compute $(\mathbf{t}_1, \ldots, \mathbf{t}_k) \leftarrow H(Z_1, \ldots, Z_\ell, D_1, \ldots, D_k)$
- For $i = 1, \ldots k$ check if $v_i \in [-L/2, L/2]$, compute $F_i \leftarrow D_i \cdot \prod_{j=1}^{\ell} Z_j^{t_{i,j}}$ and check if $F_i = \mathsf{HTLP.PGen}(\mathsf{pp}, v_i; w_i)$.
- If all checks pass output 1, otherwise 0.

Figure 5: NIZK protocol for well-formedness of a vector of homomorphic time-lock puzzles

unaffected. Formal analysis of soundness and zero-knowledge of our protocol is deferred to Appendix C due to space constraints.
Correctness Correctness of the protocol can be established given that $B \leq L/(4\ell)$. Assuming that $x_1, \ldots, x_\ell \in [-B, B]$, it follows that $|y_i + \sum t_{i,j} x_j| \leq \ell \cdot B + L/4 \leq L/2$, as $B \leq L/(4\ell)$. Consequently the verifier's checks will pass.

4.6 On The Setup Assumption

Our VTS protocols require a one-time setup that is computed once and for all by a trusted party. A careful analysis of the structure of our protocol reveals that the setup consists of the common reference string crs_{range} for the range proof and the public parameters pp of the homomorphic time-lock puzzles. In our instantiations, crs_{range} consists of sampling a random oracle and pp is a (uniformly sampled) RSA integer $N = p \cdot q$, so the problem boils down to securely sampling N, which is then made available to all parties. In general, this can be resolved by sampling N via a multi-party computation protocol, for which many ad-hoc solutions exist [23].

However, when looking at specific applications of VTS, we do not always need to resort to the power of multi-party computation. As an example, for applications where VTS are exchanged only among pairs of users (such as payment channel networks or claim-and-refund) it suffices to enforce that the verifier does not learn the factorization of N and therefore we can sample N in key generation algorithm of the signer.

5 PERFORMANCE EVALUATION

In this section we present empirical results for VTS. In this regard, we first survey AWS machines and benchmark squaring operations on various machines. Then we implement and evaluate key components: the standard signature schemes, BLS, Schnorr and ECDSA, and Linearly Homomorphic Time-Lock Puzzles on the weakest AWS machine to justify practicality of the construction. Then we estimate the cost of the VTS operations: Commit and Vrfy. As part of these operations, we also implement the range proofs Section 4.5. Our implementation is in C programming language and does not use any optimizations (logical and others) or concurrency. Our numbers are proof-of-concept and can be significantly improved in production.

We estimate the HTLCs that are posted on the Bitcoin blockchain as a payment closing step in a PCN heuristically. We estimate the number of outputs present in the closing transaction to get a real-world estimate on the number of times the worst-case, i.e. a channel closes with pending payments.

We also measure the size of transactions (in bytes) for our VTS based solution for the current implementation in Lightning Network (the PCN in Bitcoin). Our VTS solution does not leak any information about the time-lock on-chain with a negligible overhead in terms of on-chain space (steal transaction in the worst case).

5.1 Setup and Preliminaries

System Specifications. We use the following versions of the software and libraries for our experiments: Bitcoin-client *bitcoin-cli*, Bitcoin daemon *bitcoind* (v0.18.0.0-g2472733), Bitcoin blockchain parser *blocksci* (master branch with commit hash 49e97ad) and Lightning client *lnd* (0.5.1-beta commit=v0.5.1-beta-579-gb7387a) to analyze the Bitcoin blockchain and the Lightning network. We employ OpenSSL Library *openssl* (1.1.1.d-2), GNU Multi-Precision library *gmp* (6.1.2-3) and Pairing Based Cryptography Library *pbc* (0.5.14-1) to estimate the cryptographic computations. The machine used for data capture has the following hardware configuration: CPU (`Intel(R) Xeon(R) Silver 4116 CPU @ 2.10GHz with 20 cores`) and RAM (128GB).
Parameters. For all the experiments, unless otherwise specified, we use RSA 1024-bit modulus, random messages m with size $|m| =$ 100 bits and threshold $t = \lceil n/2 \rceil$.
Squarings in AWS. Given that practical constructions of time-lock puzzles are based on sequential squaring in an RSA group [46], we use different AWS machines to perform squarings to capture the role played by hardware. We use 6 different types of AWS machines whose configurations are detailed in Table 1.

First we ran the squaring experiment on the various AWS machines (presented in Table 1). As expected, we observe that the RAM and number of cores do not help in improving the number of squarings performed. However, we note that AWS' *compute optimized* machines perform better than the regular machines. We also observe that the SSD equipped machines seem to perform more squarings than the EBS counterparts.

For the subsequent experiments, we implement using *t3a.large*, the weakest (and the cheapest) of the AWS machines in Table 1 to show the efficiency and practicality of our constructions.
Benchmarks. Since, we use BLS, ECDSA and Schnorr, we first measure the cost of basic operations for these signature schemes. For VT-BLS, we use the Type A curves using PBC (pairing based

Table 1: AWS Machine Types and Squaring Performance

Machine	RAM	vCPU	Disk	Special Features	Cost($/Hr)	$\|N\|$ and Squarings	
						1024	2048
t2.xlarge	16.0	4	EBS	-	0.1856	5.853×10^9	1.881×10^9
t3a.large	8.0	2	EBS	-	0.0752	5.685×10^9	1.806×10^9
c5n.large	5.25	2	EBS	Compute Optimized	0.108	8.069×10^9	2.600×10^9
d2.xlarge	30.5	4	HDD	Storage Optimized	0.69	5.025×10^9	1.525×10^9
m5ad.large	8.0	2	SSD	1×75 NVMe	0.103	6.600×10^9	2.097×10^9
r5ad.large	16.0	2	SSD	1×75 NVMe and Memory Optimized	0.131	6.626×10^9	2.108×10^9

Table 2: A summary of costs for KGen, Sign **and** Vf

Operation	Schnorr	ECDSA	BLS
Keygen	1.69 ms	1.70 ms	0.024 s
Sign	1.63 ms	1.67 ms	0.023 s
Verify	1.55 ms	1.57 ms	0.047 s

cryptography) library [38] for pairing implementation. For VT-ECDSA, we use the implementation and the *secp256k1* curve present in OpenSSL [21]. For VT-Schnorr, we use the proposed BIP schnorr standard [51] to instantiate Schnorr in *secp256k1* elliptic curves. A summary of the cost of key generation, signing and verification for these signature schemes is presented in Table 2.

5.2 Performance Evaluation

Linearly Homomorphic Time-Lock Puzzles. We first implement a library for LHTLP variant proposed in [41] and use the library to estimate the cost (time) of various cryptographic operations needed to be performed in LHTLP.PSetup, LHTLP.PGen, LHTLP.PEval and LHTLP.PSolve.

We ran each phase of LHTLP 10 times and present the average. We used an RSA modulus of 1024 bits and $\mathbf{T} = 1.0 \times 10^6$, on average:

- LHTLP.PSetup takes **5.521** s,
- LHTLP.PGen takes **9.93** ms,
- LHTLP.PSolve takes **0.692** s.
- Batching using LHTLP.PEval is very efficient (linear in number of puzzles merged) and this is presented in Figure 6.

Verifiable Timed Signatures. We estimate the time required to perform *Commit and Prove* and *Verification* algorithms for VT-BLS (Figure 2), VT-Schnorr (Figure 3) and VT-ECDSA (Figure 4), and present them in Table 3. We observe that the curves used for BLS are not optimized and therefore lead to much slower computations. All three implementations can be significantly improved using concurrency and other efficient data structures. We also observe that despite this, the operations are still practical for use in the real-world.

5.3 VTS and Lightning Network

In order to get an idea for the hiding time parameter **T** and understand how the Commit transactions are employed in PCNs today, we study the Bitcoin PCN - the Lightning network.

Bitcoin uses elliptic curve *secp256k1* for signature generation. There is a proposal to use Schnorr signatures [51]. We study the graph statistics for the Lightning Network (LN) [45] and analyze

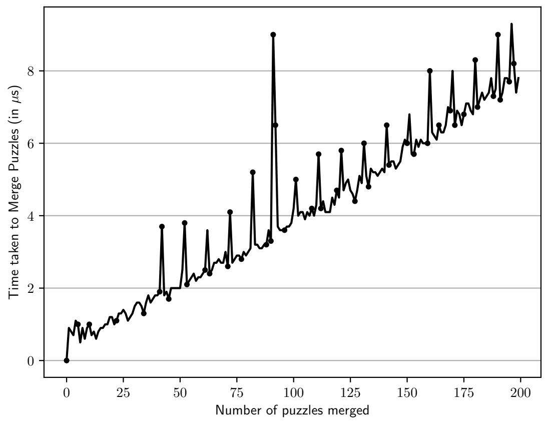

Figure 6: The time to batch puzzles vs the number of puzzles batched.

Table 3: The cost (time) of *commit and prove*, and *verification* steps for Schnorr, ECDSA and BLS for different values of n.

Operation		Parameter n (Soundness error)	
		30 (6.44×10^{-9})	40 (7.25×10^{-12})
VT-BLS	Commit	20.44 s	32.19 s
	Verify	33.24 s	41.37 s
VT-ECDSA	Commit	7.77 s	10.41 s
	Verify	7.53 s	9.94 s
VT-Schnorr	Commit	7.93 s	10.71 s
	Verify	7.93 s	10.72 s

the number of outputs from the Bitcoin blockchain for our analysis. The number of outputs indicate the number of times a channel is closed before completion of PCN payment.

Lightning Graph Statistics. We scanned their network using the *describegraph* command of the lightning client. We consolidated the information into Table 4. We observe that 80.05% of the payment channels are disabled, i.e do not allow the channel to be a part of a PCN for a payment. We also observe that the time lock duration is 85.53 blocks which is equivalent to 14.25 hours giving an estimate to **T**.

Estimating Channel Closures. When a channel between Alice and Bob is closed, there are two primary outputs. The first output

Table 4: Lightning Graph Data (as of November 23, 2019)

Parameter	Value	Unit
Nodes	4,692	-
Channels	30,665	-
Percentage Disabled	80.05	%
Avg. Channel Capacity	2,673,295.67	sat
Avg. Minimum HTLC Amount	1,237.42	10^{-3} sat
Avg. Base Fee	1,008.51	10^{-3} sat
Avg. Fee Rate	683,536.66	10^{-6} sat
Avg. Time Lock Delta	85.53	blocks

is the *to_local* output which settles the money to the address going on-chain after a delay using P2WSH. This delay ensures that if Alice closes the channel using a stale transaction, Bob can use a revocation key to penalize Alice by stealing the money from the output. The second output *to_remote* pays money (using P2WPKH) directly to the other user in the channel. Other outputs can exist if Alice and Bob were a part of a payment network, but one of them decides to close the channel. In this case, there are extra outputs which contain HTLC -Claim or Refund Scripts or a new channel creation. Since, we cannot know what these outputs until the witness is produced on chain, we conservatively estimate them as HTLC-Claim or Refund outputs.

Therefore estimating the number of closure transactions on the blockchain whose outputs are more than 2 gives an estimate of how likely the closure of a channel with an ongoing payment is in the real-world.

From the main chain, we observe 116,502 closing transactions by examining opened outputs with one output matching a *to_local* output. These are transactions that are used to close the payment channel. Among them, *66.84 %* (77,867) of transactions contain more than one *P2WSH* outputs. We present the number of *P2WSH* outputs in these closing transactions in Table 5.

Table 5: A histogram of the number of outputs in a lightning closing transaction. There is one standard output called the *to_local* which pays self after a delay giving the remote user time to penalize the party if a stale transaction is published. Any output more than 1, indicates that the user was involved as a intermediate in a payment channel but decided to close the channel.

# of P2WSH Outputs	# of Closing Tx
1 (No HTLC output)	38,635
2 (1 HTLC output)	47,172
3 − 10	20,634
11 − 100	9,245
≥ 100	10

From Table 4, we observe that the average base-fee is 1 satoshi which equates to 0.000087 USD (as of today) and the average fee-rate is 683 satoshis which amounts to 0.06 USD. From Table 1, we observe that a faster machine can perform a lot more (2×) squarings than the cheaper machines. However, this is not beneficial for the attacker. Lighting network consists of micro-transactions, whose cost to exploit outweigh the gain. For privacy-critical applications, VTS provides an efficient alternative.

6 CONCLUSION AND FUTURE WORK

We theoretically analyze and present secure constructions for Verifiable Timed Signatures compatible with standard signatures such as BLS, Schnorr and ECDSA. Our constructions are efficient in terms of cost for verifying the timed commitments if they indeed encapsulate a valid signature on a message and if it can be obtained after the given time **T**. Our constructions are readily usable in several current day systems for achieving improved privacy and compatibility across several settings. We present several cryptocurrency-related applications for VTS and a variant of VTS where the signing key is committed instead of the signature. We experimentally evaluate our approach to show that our constructions are practical. In terms of future work, the next step is developing VTS-based solutions for other real world systems that can benefit from the timed nature of the primitive. Our verifiability techniques leaves open the question of whether we can improve the efficiency of the verifier and the size of the proof even further. Developing efficient range proofs with smaller slack for HTLP is of independent interest.

ACKNOWLEDGMENTS

This work was partially supported by the National Science Foundation under grants CNS 1719196 and CNS 1846316, and the IARPA HECTOR program. The work was also partially supported by the German research foundation (DFG) through the collaborative research center 1223, and by the state of Bavaria at the Nuremberg Campus of Technology (NCT). NCT is a research cooperation between FAU and the Technische Hochschule Nürnberg Georg Simon Ohm (THN). We would also like to thank the anonymous reviewers for their valuable comments for improving the paper.

REFERENCES

[1] [n.d.]. bip32. ([n. d.]). https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki.
[2] [n.d.]. Bitcoin Wiki: Payment Channels. https://en.bitcoin.it/wiki/Payment_channels.
[3] [n.d.]. BOLT #3: Bitcoin Transaction and Script Formats. https://github.com/lightningnetwork/lightning-rfc/blob/master/03-transactions.md#offered-htlc-outputs.
[4] [n.d.]. Self Decrypting Files. https://www.gwern.net/Self-decrypting-files.
[5] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. 2011. How to Garble Arithmetic Circuits. In *52nd FOCS*, Rafail Ostrovsky (Ed.). IEEE Computer Society Press, Palm Springs, CA, USA, 120–129. https://doi.org/10.1109/FOCS.2011.40
[6] Vivek Kumar Bagaria, Joachim Neu, and David Tse. 2020. Boomerang: Redundancy Improves Latency and Throughput in Payment-Channel Networks. In *24th International Conference on Financial Cryptography and Data Security FC 2020*. 304–324.
[7] Waclaw Banasik, Stefan Dziembowski, and Daniel Malinowski. 2016. Efficient Zero-Knowledge Contingent Payments in Cryptocurrencies Without Scripts. In *ESORICS 2016, Part II (LNCS, Vol. 9879)*, Ioannis G. Askoxylakis, Sotiris Ioannidis, Sokratis K. Katsikas, and Catherine A. Meadows (Eds.). Springer, Heidelberg, Germany, Heraklion, Greece, 261–280. https://doi.org/10.1007/978-3-319-45741-3_14
[8] Rachid El Bansarkhani and Jan Sturm. 2016. An Efficient Lattice-Based Multisignature Scheme with Applications to Bitcoins. In *CANS 16 (LNCS, Vol. 10052)*, Sara Foresti and Giuseppe Persiano (Eds.). Springer, Heidelberg, Germany, Milan, Italy, 140–155. https://doi.org/10.1007/978-3-319-48965-0_9
[9] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. 2014. Zerocash: Decentralized Anonymous Payments from Bitcoin. In *2014 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, Berkeley, CA, USA, 459–474. https://doi.org/10.1109/SP.2014.36

[10] Iddo Bentov and Ranjit Kumaresan. 2014. How to Use Bitcoin to Design Fair Protocols. In *CRYPTO 2014, Part II (LNCS, Vol. 8617)*, Juan A. Garay and Rosario Gennaro (Eds.). Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 421–439. https://doi.org/10.1007/978-3-662-44381-1_24

[11] Nir Bitansky, Shafi Goldwasser, Abhishek Jain, Omer Paneth, Vinod Vaikuntanathan, and Brent Waters. 2016. Time-Lock Puzzles from Randomized Encodings. In *ITCS 2016*, Madhu Sudan (Ed.). ACM, Cambridge, MA, USA, 345–356. https://doi.org/10.1145/2840728.2840745

[12] Dan Boneh, Manu Drijvers, and Gregory Neven. 2018. Compact Multi-signatures for Smaller Blockchains. In *ASIACRYPT 2018, Part II (LNCS, Vol. 11273)*, Thomas Peyrin and Steven Galbraith (Eds.). Springer, Heidelberg, Germany, Brisbane, Queensland, Australia, 435–464. https://doi.org/10.1007/978-3-030-03329-3_15

[13] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. 2003. Aggregate and Verifiably Encrypted Signatures from Bilinear Maps. In *EUROCRYPT 2003 (LNCS, Vol. 2656)*, Eli Biham (Ed.). Springer, Heidelberg, Germany, Warsaw, Poland, 416–432. https://doi.org/10.1007/3-540-39200-9_26

[14] Dan Boneh, Ben Lynn, and Hovav Shacham. 2001. Short Signatures from the Weil Pairing. In *ASIACRYPT 2001 (LNCS, Vol. 2248)*, Colin Boyd (Ed.). Springer, Heidelberg, Germany, Gold Coast, Australia, 514–532. https://doi.org/10.1007/3-540-45682-1_30

[15] Dan Boneh and Moni Naor. 2000. Timed Commitments. In *CRYPTO 2000 (LNCS, Vol. 1880)*, Mihir Bellare (Ed.). Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 236–254. https://doi.org/10.1007/3-540-44598-6_15

[16] Ivan Damgård and Mats Jurik. 2001. A Generalisation, a Simplification and Some Applications of Paillier's Probabilistic Public-Key System. In *PKC 2001 (LNCS, Vol. 1992)*, Kwangjo Kim (Ed.). Springer, Heidelberg, Germany, Cheju Island, South Korea, 119–136. https://doi.org/10.1007/3-540-44586-2_9

[17] Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. 1987. Non-interactive zero-knowledge proof systems. In *Conference on the Theory and Application of Cryptographic Techniques*. Springer, 52–72.

[18] Manu Drijvers, Kasra Edalatnejad, Bryan Ford, Eike Kiltz, Julian Loss, Gregory Neven, and Igors Stepanovs. 2019. On the security of two-round multi-signatures. In *On the Security of Two-Round Multi-Signatures*. IEEE, 0.

[19] Lisa Eckey, Sebastian Faust, Kristina Hostáková, and Stefanie Roos. 2020. Splitting Payments Locally While Routing Interdimensionally. *IACR Cryptol. ePrint Arch.* 2020 (2020), 555.

[20] Christoph Egger, Pedro Moreno-Sanchez, and Matteo Maffei. 2019. Atomic Multi-Channel Updates with Constant Collateral in Bitcoin-Compatible Payment-Channel Networks. In *ACM CCS 2019*. ACM Press, 801–815. https://doi.org/10.1145/3319535.3345666

[21] Ralf S Engelschall. 2001. Openssl: The open source toolkit for SSL/TLS. *URL: http://www. openssl. org* (2001), 2001–04.

[22] Amos Fiat and Adi Shamir. 1987. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *CRYPTO'86 (LNCS, Vol. 263)*, Andrew M. Odlyzko (Ed.). Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 186–194. https://doi.org/10.1007/3-540-47721-7_12

[23] Tore Kasper Frederiksen, Yehuda Lindell, Valery Osheter, and Benny Pinkas. 2018. Fast distributed RSA key generation for semi-honest and malicious adversaries. In *Annual International Cryptology Conference*. Springer, 331–361.

[24] Juan A. Garay and Markus Jakobsson. 2003. Timed Release of Standard Digital Signatures. In *FC 2002 (LNCS, Vol. 2357)*, Matt Blaze (Ed.). Springer, Heidelberg, Germany, Southampton, Bermuda, 168–182.

[25] Juan A. Garay and Carl Pomerance. 2003. Timed Fair Exchange of Standard Signatures: [Extended Abstract]. In *FC 2003 (LNCS, Vol. 2742)*, Rebecca Wright (Ed.). Springer, Heidelberg, Germany, Guadeloupe, French West Indies, 190–207.

[26] Rosario Gennaro and Steven Goldfeder. 2018. Fast multiparty threshold ecdsa with fast trustless setup. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1179–1194.

[27] Christian Hanser, Max Rabkin, and Dominique Schröder. 2015. Verifiably Encrypted Signatures: Security Revisited and a New Construction. In *ESORICS 2015, Part I (LNCS, Vol. 9326)*, Günther Pernul, Peter Y. A. Ryan, and Edgar R. Weippl (Eds.). Springer, Heidelberg, Germany, Vienna, Austria, 146–164. https://doi.org/10.1007/978-3-319-24174-6_8

[28] Don Johnson, Alfred Menezes, and Scott Vanstone. 2001. The Elliptic Curve Digital Signature Algorithm (ECDSA). *International Journal of Information Security* 1, 1 (01 Aug 2001), 36–63. https://doi.org/10.1007/s102070100002

[29] Jonathan Katz. 2010. *Digital signatures*. Springer Science & Business Media.

[30] Jonathan Katz, Andrew Miller, and Elaine Shi. 2014. Pseudonymous secure computation from time-lock puzzles. (2014).

[31] Ranjit Kumaresan and Iddo Bentov. 2014. How to Use Bitcoin to Incentivize Correct Computations. In *ACM CCS 2014*, Gail-Joon Ahn, Moti Yung, and Ninghui Li (Eds.). ACM Press, Scottsdale, AZ, USA, 30–41. https://doi.org/10.1145/2660267.2660380

[32] Ranjit Kumaresan, Tal Moran, and Iddo Bentov. 2015. How to Use Bitcoin to Play Decentralized Poker. In *ACM CCS 2015*, Indrajit Ray, Ninghui Li, and Christopher Kruegel (Eds.). ACM Press, Denver, CO, USA, 195–206. https://doi.org/10.1145/2810103.2813712

[33] Russell W. F. Lai, Viktoria Ronge, Tim Ruffing, Dominique Schröder, Sri Aravinda Krishnan Thyagarajan, and Jiafan Wang. 2019. Omniring: Scaling Private Payments Without Trusted Setup. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security* (London, United Kingdom) *(CCS '19)*. Association for Computing Machinery, New York, NY, USA, 31–48. https://doi.org/10.1145/3319535.3345655

[34] Huijia Lin, Rafael Pass, and Pratik Soni. 2017. Two-Round and Non-Interactive Concurrent Non-Malleable Commitments from Time-Lock Puzzles. In *58th FOCS*, Chris Umans (Ed.). IEEE Computer Society Press, Berkeley, CA, USA, 576–587. https://doi.org/10.1109/FOCS.2017.59

[35] Yehuda Lindell. 2017. Fast secure two-party ECDSA signing. In *Annual International Cryptology Conference*. Springer, 613–644.

[36] Yehuda Lindell and Ariel Nof. 2018. Fast secure multiparty ecdsa with practical distributed key generation and applications to cryptocurrency custody. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1837–1854.

[37] Yehuda Lindell and Ariel Nof. 2018. Fast Secure Multiparty ECDSA with Practical Distributed Key Generation and Applications to Cryptocurrency Custody. In *ACM CCS 2018*, David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang (Eds.). ACM Press, Toronto, ON, Canada, 1837–1854. https://doi.org/10.1145/3243734.3243788

[38] Ben Lynn et al. 2006. PBC library. *Online: http://crypto. stanford. edu/pbc* 59 (2006), 76–99.

[39] Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, Matteo Maffei, and Srivatsan Ravi. 2017. Concurrency and Privacy with Payment-Channel Networks. In *ACM CCS 2017*, Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM Press, Dallas, TX, USA, 455–471. https://doi.org/10.1145/3133956.3134096

[40] Giulio Malavolta, Pedro Moreno-Sanchez, Clara Schneidewind, Aniket Kate, and Matteo Maffei. 2019. Anonymous Multi-Hop Locks for Blockchain Scalability and Interoperability. In *NDSS 2019*. The Internet Society, San Diego, CA, USA.

[41] Giulio Malavolta and Sri Aravinda Krishnan Thyagarajan. 2019. Homomorphic Time-Lock Puzzles and Applications. In *CRYPTO 2019, Part I (LNCS)*, Hovav Shacham and Alexandra Boldyreva (Eds.). Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 620–649. https://doi.org/10.1007/978-3-030-26948-7_22

[42] Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. 2018. Simple Schnorr Multi-Signatures with Applications to Bitcoin. Cryptology ePrint Archive, Report 2018/068. https://eprint.iacr.org/2018/068.

[43] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system.

[44] Pascal Paillier. 1999. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *EUROCRYPT'99 (LNCS, Vol. 1592)*, Jacques Stern (Ed.). Springer, Heidelberg, Germany, Prague, Czech Republic, 223–238. https://doi.org/10.1007/3-540-48910-X_16

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

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

[47] Claus-Peter Schnorr. 1990. Efficient Identification and Signatures for Smart Cards. In *CRYPTO'89 (LNCS, Vol. 435)*, Gilles Brassard (Ed.). Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 239–252. https://doi.org/10.1007/0-387-34805-0_22

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

[49] Vibhaalakshmi Sivaraman, Shaileshh Bojja Venkatakrishnan, Kathleen Ruan, Parimarjan Negi, Lei Yang, Radhika Mittal, Giulia Fanti, and Mohammad Alizadeh. 2020. High Throughput Cryptocurrency Routing in Payment Channel Networks. In *17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20)*. 777–796.

[50] Sri Aravinda Krishnan Thyagarajan, Adithya Bhat, Giulio Malavolta, Nico Döttling, Aniket Kate, and Schröder Dominique. [n.d.]. Verifiable Timed Signatures Project Page. ([n. d.]). https://github.com/verifiable-timed-signatures/web/.

[51] P Wuille. 2018. Schnorr's bip.

[52] Jan Henrik Ziegeldorf, Fred Grossmann, Martin Henze, Nicolas Inden, and Klaus Wehrle. 2015. Coinparty: Secure multi-party mixing of bitcoins. In *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*. ACM, 75–86.

[53] Jan Henrik Ziegeldorf, Roman Matzutt, Martin Henze, Fred Grossmann, and Klaus Wehrle. 2018. Secure and anonymous decentralized Bitcoin mixing. *Future Generation Computer Systems* 80 (2018), 448–466.

A CRYPTOGRAPHIC BUILDING BLOCKS

A.1 Digital Signatures

Definition 4 (Digital Signatures). *A (digital) signature scheme consists of three probabilistic polynomial time algorithms* (KGen, Sign, Vf) *such that:*

- $(pk, sk) \leftarrow \mathsf{KGen}(1^\lambda)$: *the key generation algorithm takes as input a security parameter 1^λ and outputs a pair of keys (pk, sk). We assume that pk and sk each has length at least λ, and that λ can be determined from pk or sk.*
- $\sigma \leftarrow \mathsf{Sign}(sk, m)$: *the signing algorithm takes as input a private key sk and a message m from some message space (that may depend on pk). It outputs a signature σ.*
- $0/1 \leftarrow \mathsf{Vf}(pk, m, \sigma)$: *the deterministic verification algorithm* Vf *takes as input a public key pk, a message m, and a signature σ. It outputs a bit b, with $b = 1$ meaning valid and $b = 0$ meaning invalid.*

It is required that except with negligible probability over (pk, sk) output by $\mathsf{KGen}(1^\lambda)$, *it holds that* $\mathsf{Vf}(pk, m, \mathsf{Sign}(sk, m)) = 1$ *for every (legal) message m.*

Definition 5. *A signature scheme* $\mathsf{DS} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ *is* existentially unforgeable under an adaptive chosen-message attack, *or just secure, if for all probabilistic polynomial-time adversaries $\mathcal{A}$, there is a negligible function negl such that:*

$$\Pr\left[\mathsf{ExpEUFCMA}_{\mathcal{A},\mathsf{DS}}(\lambda) = 1\right] \le negl$$

```
$\mathsf{ExpEUFCMA}_{\mathcal{A},\mathsf{DS}}(\lambda)$
$Q := \emptyset$
$(pk, sk) \leftarrow \mathsf{KGen}(1^\lambda)$
$(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}O}(pk)$
$b^* = \mathsf{Vf}(pk, m^*, \sigma^*) \wedge (m^* \notin Q)$
return $b^*$

Oracle $\mathsf{Sign}O(m)$
$\sigma \leftarrow \mathsf{Sign}(sk, m)$
$Q := Q \cup \{\sigma\}$
return $\sigma$
```

Figure 7: Experiment for unforgeability of the signature scheme DS

A.2 Time-Lock Puzzles

We recall the definition of standard time-lock puzzles [11]. For conceptual simplicity we consider only schemes with binary solutions.

Definition 6 (Time-Lock Puzzles). *A time-lock puzzle is a tuple of two algorithms* (PGen, PSolve) *defined as follows.*

- $Z \leftarrow \mathsf{PGen}(\mathbf{T}, s)$ *a probabilistic algorithm that takes as input a hardness-parameter* $\mathbf{T}$ *and a solution $s \in \{0, 1\}$, and outputs a puzzle Z.*
- $s \leftarrow \mathsf{PSolve}(Z)$ *a deterministic algorithm that takes as input a puzzle Z and outputs a solution s.*

Definition 7 (Correctness). *For all $\lambda \in \mathbb{N}$, for all polynomials* $\mathbf{T}$ *in λ, and for all $s \in \{0, 1\}$, it holds that* $s = \mathsf{PSolve}(\mathsf{PGen}(\mathbf{T}, s))$.

Definition 8 (Security). *A scheme* (PGen, PSolve) *is secure with gap $\varepsilon < 1$ if there exists a polynomial $\tilde{\mathbf{T}}(\cdot)$ such that for all polynomials $\mathbf{T}(\cdot) \ge \tilde{\mathbf{T}}(\cdot)$ and every polynomial-size adversary $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$ of depth $\le \mathbf{T}^\varepsilon(\lambda)$, there exists a negligible function $\mu(\cdot)$, such that for all $\lambda \in \mathbb{N}$ it holds that*

$$\Pr\left[b \leftarrow \mathcal{A}(Z) : \ Z \leftarrow \mathsf{PGen}(\mathbf{T}(\lambda), b)\ \right] \le \frac{1}{2} + \mu(\lambda).$$

A.3 Homomorphic Time-Lock Puzzles

Definition 9 (Homomorphic Time-Lock Puzzles [41]). *Let $C = \{C_\lambda\}_{\lambda\in\mathbb{N}}$ be a class of circuits and let S be a finite domain. A homomorphic time-lock puzzle (HTLP) with respect to C and with solution space S is tuple of four algorithms* (HTLP.PSetup, HTLP.PGen, HTLP.PSolve, HTLP.PEval) *defined as follows.*

- $\mathsf{pp} \leftarrow \mathsf{HTLP.PSetup}(1^\lambda, \mathbf{T})$ *a probabilistic algorithm that takes as input a security parameter 1^λ and a time hardness parameter* $\mathbf{T}$, *and outputs public parameters* pp.
- $Z \leftarrow \mathsf{HTLP.PGen}(\mathsf{pp}, s)$ *a probabilistic algorithm that takes as input public parameters* pp, *and a solution $s \in S$, and outputs a puzzle Z.*
- $s \leftarrow \mathsf{HTLP.PSolve}(\mathsf{pp}, Z)$ *a deterministic algorithm that takes as input public parameters* pp *and a puzzle Z and outputs a solution s.*
- $Z' \leftarrow \mathsf{HTLP.PEval}(C, \mathsf{pp}, Z_1, \ldots, Z_n)$ *a probabilistic algorithm that takes as input a circuit $C \in C_\lambda$, public parameters* pp *and a set of n puzzles $(Z_1, \ldots, Z_n)$ and outputs a puzzle Z'.*

Security requires that the solution of the puzzles is hidden for all adversaries that run in (parallel) time less than $\mathbf{T}$. Here we consider a basic version where the time is counted from the moment the public parameters are published.

Definition 10 (Security of HTLP). *An HTLP scheme consisting of* HTLP.PSetup, HTLP.PGen, HTLP.PSolve, HTLP.PEval, *is secure with gap $\varepsilon < 1$ if there exists a polynomial $\tilde{\mathbf{T}}(\cdot)$ such that for all polynomials $\mathbf{T}(\cdot) \ge \tilde{\mathbf{T}}(\cdot)$ and every polynomial-size adversary $(\mathcal{A}_1, \mathcal{A}_2) = \{(\mathcal{A}_1, \mathcal{A}_2)_\lambda\}_{\lambda\in\mathbb{N}}$ where the depth of $\mathcal{A}_2$ is bounded from above by $\mathbf{T}^\varepsilon(\lambda)$, there exists a negligible function $\mu(\cdot)$, such that for all $\lambda \in \mathbb{N}$ it holds that*

$$\Pr\left[b \leftarrow \mathcal{A}_2(\mathsf{pp}, Z, \tau) : \begin{array}{l} (\tau, s_0, s_1) \leftarrow \mathcal{A}_1(1^\lambda) \\ \mathsf{pp} \leftarrow \mathsf{HTLP.PSetup}(1^\lambda, \mathbf{T}(\lambda)) \\ b \leftarrow_\$ \{0, 1\} \\ Z \leftarrow \mathsf{HTLP.PGen}(\mathsf{pp}, s_b) \end{array} \right]$$

$$\le \frac{1}{2} + \mu(\lambda)$$

and $(s_0, s_1) \in S^2$.

Definition 11 (Compactness). *Let $C = \{C_\lambda\}_{\lambda\in\mathbb{N}}$ be a class of circuits (along with their respective representations). An HTLP scheme* (HTLP.PSetup, HTLP.PGen, HTLP.PSolve, HTLP.PEval) *is compact (for the class C) if for all $\lambda \in \mathbb{N}$, all polynomials* $\mathbf{T}$ *in λ, all circuits $C \in C_\lambda$ and respective inputs $(s_1, \ldots, s_n) \in S^n$, all* pp *in the support of* $\mathsf{HTLP.PSetup}(1^\lambda, \mathbf{T})$, *and all Z_i in the support of* $\mathsf{HTLP.PGen}(\mathsf{pp}, s_i)$, *the following two conditions are satisfied:*

- *There exists a fixed polynomial $p(\cdot)$ such that $|Z| = p(\lambda, |C(s_1, \ldots, s_n)|)$, where $Z \leftarrow \mathsf{HTLP.PEval}(C, \mathsf{pp}, Z_1, \ldots, Z_n)$.*
- *There exists a fixed polynomial $\tilde{p}(\cdot)$ such that the runtime of* $\mathsf{HTLP.PEval}(C, \mathsf{pp}, Z_1, \ldots, Z_n)$ *is bounded by $\tilde{p}(\lambda, |C|)$.*

B SECURITY ANALYSIS OF VTS CONSTRUCTIONS

B.1 Proof of Theorem 1 and Theorem 2

Proof. We show that the protocol (Figure 2) is private against an adversary of depth bounded by $\mathbf{T}^{\varepsilon}$, for some non-negative $\varepsilon < 1$. We now gradually change the simulation through a series of hybrids and then we argue about the proximity of neighbouring experiments.

Hybrid $\mathcal{H}_0$: This is the original execution.

Hybrid $\mathcal{H}_1$: This is identical to the previous hybrid except that the random oracle is simulated by lazy sampling. In addition a random set I^* (where $|I^*| = t-1$) is sampled ahead of time, and the output of the random oracle on the cut-and-choose instance is programmed to I^*. Note that the changes of this hybrid are only syntactical and therefore the distribution is unchanged.

Hybrid $\mathcal{H}_2$: In this hybrid we sample a simulated common reference string crs_{range}. By the zero-knowledge property of (ZKsetup, ZKprove, ZKverify) this change is computationally indistinguishable.

Hybrid $\mathcal{H}_3 \dots \mathcal{H}_{3+n}$: In the hybrid $\mathcal{H}_{3+i}$, for all $i \in [n]$, the proof $\pi_{\text{range},i}$ is computed via the simulator provided by the underlying NIZK proof. By the zero-knowledge property of (ZKsetup, ZKprove, ZKverify), the distance between neighbouring hybrids is bounded by a negligible function in the security parameter.

Hybrid $\mathcal{H}_{3+n} \dots \mathcal{H}_{3+2n-t+1}$: In the i-th hybrid $\mathcal{H}_{3+i}$, for all $i \in [n-(t-1)]$, the puzzle corresponding to the i-th element of the set $\bar{I}^*$ is computed as LHTLP.PGen(pp, $0^{\lambda}; r_i$), where $\bar{I}^*$ is the complement of I^*. Since the distinguisher is depth-bounded, indistinguishability follows from an invocation of the security of LHTLP..

Hybrid $\mathcal{H}_{3+2n-t+2}$: In this hybrid the prover behaves as follows. For all $i \in I^*$ it samples a uniform $\alpha_i \leftarrow \mathbb{Z}_q$ and sets $h_i = g_0^{\alpha_i}$ and computes the puzzle as described in the protocol. On the other hand, for all $i \notin I^*$ it computes h_i as

$$h_i = \left(\frac{pk}{\prod_{j\in I^*} h_j^{\ell_j(0)}}\right)^{\ell_i(0)^{-1}}.$$

The rest of the execution is unchanged. Note that for all $i \notin I^*$ we have that

$$\prod_{j\in I^*} h_j^{\ell_j(0)} \cdot h_i^{\ell_i(0)} = pk$$

as expected. It follows that the changes in this hybrid are only syntactical and the distribution induced is identical to that of the previous hybrid.

Simulator $\mathcal{S}$: The simulator is defined to be identical to the last hybrid. Note that no information about the witness is used to compute the proof. This concludes our proof. □

We now show that our protocol (Figure 2) is sound and the proof of Theorem 2.

Proof. We analyze the protocol in its interactive version and the soundness of non-interactive protocol follows from the Fiat-Shamir transformation [22] for constant-round protocols. Let $\mathcal{A}$ be an adversary that efficiently breaks the soundness of the protocol. In particular this means that the adversary produces a commitment $(Z_1, \dots, Z_n)$ such that for all $Z_i \notin I$ it holds that LHTLP.PSolve(pp, $Z_i) = \tilde{\sigma}_i$ such that

$$e(g_0, \tilde{\sigma}_i) \neq e(h_i, H(m)).$$

Assume the contrary, then we could recover a valid signature on m by interpolating $\tilde{\sigma}_i$ with $\{\sigma_i\}_{i\in I}$, which satisfy the above relation by definition of the verification algorithm. Further observe that all puzzles $(Z_1, \dots, Z_n)$ are well-formed, i.e., the solving algorithm always outputs some well-defined value, except with negligible probability, by the soundness of the range NIZK.

It follows that, given $(Z_1, \dots, Z_n)$ we can recover some set I' in polynomial time by solving the puzzles and checking which of the signatures satisfy the above relation. In order for the verifier to accept, it must be the case that $I' = I$ which means that the prover correctly guesses a random n-bit string uniformly chosen from the set of strings with exactly $n/2$-many 0's. This happens with probability exactly $\frac{(n/2!)^2}{n!}$.

Observe that, in the non-interactive variant of the protocol, the above argument holds even in the presence of an arbitrary (polynomial) number of simulated proofs, as long as the range NIZK is simulation-sound. Therefore, if we instantiate the range NIZK with a simulation-sound scheme, then so is the resulting VTS. □

B.2 Proof of Theorem 3 and Theorem 4

Proof. We show that the protocol (Figure 3) is private against an adversary of depth bounded by $\mathbf{T}^{\varepsilon}$, for some non-negative $\varepsilon < 1$. Consider the following sequence of hybrids.

Hybrid $\mathcal{H}_0 \dots \mathcal{H}_{3+2n-t+1}$: Defined as in the proof of Theorem 1.

Hybrid $\mathcal{H}_{3+2n-t+2}$: In this hybrid the prover behaves as follows. For all $i \in I^*$ it samples a uniform $(x_i, k_i) \leftarrow \mathbb{Z}_q$ and sets $h_i = g^{x_i}$, $R_i = g^{k_i}$, and $s_i = k_i + cx_i$ and computes the corresponding puzzle as described in the protocol. On the other hand, for all $i \notin I^*$ it computes h_i as

$$h_i = \left(\frac{pk}{\prod_{j\in I^*} h_j^{\ell_j(0)}}\right)^{\ell_i(0)^{-1}} \quad \text{and}$$

$$R_i = \left(\frac{R}{\prod_{j\in I^*} R_j^{\ell_j(0)}}\right)^{\ell_i(0)^{-1}}.$$

The rest of the execution is unchanged. Note that for all $i \notin I^*$ we have that

$$\prod_{j\in I^*} h_j^{\ell_j(0)} \cdot h_i^{\ell_i(0)} = pk \text{ and}$$

$$\prod_{j\in I^*} R_j^{\ell_j(0)} \cdot R_i^{\ell_i(0)} = R$$

as expected. It follows that the changes in this hybrid are only syntactical and the distribution induced by this hybrid is identical to that of the previous hybrid.

Hybrid $\mathcal{H}_{3+2n-t+3}$: Defined as in the previous hybrid except that R is sampled uniformly over $\mathbb{G}$. Note that this does not change the distribution observed by the distinguisher.

Simulator $\mathcal{S}$: The simulator is defined to be identical to the last hybrid. Note that no information about the witness is used to compute the proof. This concludes our proof. □

We show that the protocol (Figure 3) satisfies soundness which is the proof for Theorem 4.

Proof. As for the proof of Theorem 2 we assume that the challenge set is sampled interactively by the verifier. The soundness of the non-interactive version follows by a standard argument. Consider an adversary that can efficiently violate the soundness of the protocol. This implies that such and adversary produces a commitment $(R, Z_1, \dots, Z_n)$ such that for all $Z_i \notin I$ it holds that $\mathsf{LHTLP.PSolve}(\mathsf{pp}, Z_i) = \tilde{s}_i$ where

$$g^{\tilde{s}_i} \neq R_i \cdot h_i^c.$$

Assume the contrary, then we could recover a valid signature on m by interpolating $\tilde{s}_i$ with $\{s_i\}_{i \in I}$, which gives us a valid signature (R, s) on m, by linearity. Further observe that all puzzles $(Z_1, \dots, Z_n)$ are well-formed, i.e., the solving algorithm always outputs some well-defined value, except with negligible probability, by the soundness of the range NIZK.

It follows that, given $(Z_1, \dots, Z_n)$ we can define some set I' in polynomial time by solving the puzzles and checking which of the resulting $\tilde{s}_i$ satisfy the above relation. In order for the verifier to accept, it must be the case that $I' = I$ which means that the prover correctly guesses a random n-bit string uniformly chosen from the set of strings with exactly $n/2$-many 0's. This happens with probability exactly $\frac{(n/2!)^2}{n!}$.

As discussed in the proof of Theorem 2, the non-interactive variant of the protocol can be shown to be simulation sound with the same argument, assuming a simulation-sound range NIZK. □

B.3 Proof of Theorem 5 and Theorem 6

Proof. We show that the protocol (Figure 4) is private against an adversary of depth bounded by $\mathbf{T}^\varepsilon$, for some non-negative $\varepsilon < 1$. We do this by defining a series of hybrids.

Hybrid $\mathcal{H}_0 \dots \mathcal{H}_{3+2n-t+1}$: Defined as in the proof of Theorem 1.

Hybrid $\mathcal{H}_{3+2n-t+2}$: In this hybrid the prover does the following. For all $i \in I^*$ it samples a uniform $s_i \leftarrow \mathbb{Z}_q$ and sets $R_i = B^{s_i} = (g^c \cdot h^r)^{s_i}$ and computes the corresponding puzzle as described in the protocol. On the other hand, for all $i \notin I^*$ it computes R_i as

$$R_i = \left(\frac{R}{\prod_{j \in I^*} R_j^{\ell_j(0)}} \right)^{\ell_i(0)^{-1}}.$$

The rest of the execution is unchanged. Note that for all $i \notin I^*$ we have that

$$\prod_{j \in I^*} R_j^{\ell_j(0)} \cdot R_i^{\ell_i(0)} = R$$

as expected. It follows that the changes in this hybrid are only syntactical and the distribution induced by this hybrid is identical to that of the previous hybrid.

Hybrid $\mathcal{H}_{3+2n-t+3}$: Defined as the previous hybrid except that $R = (x, y)$ is sampled as a uniform point in the curve and r is set to x mod q. Again this change is only syntactical since R is uniformly distributed in the previous hybrid.

Simulator $\mathcal{S}$: The simulator is defined to be identical to the last hybrid. Note that no information about the witness is used to compute the proof. This concludes our proof. □

We now give the formal proof of Theorem 6.

Proof. As discussed in the proof of Theorem 2, it suffices to show that we can correctly reconstruct a valid signature as long as at least one of the unopened puzzles contains some s_i such that

$$R_i = (g^c \cdot h^r)^{s_i}.$$

Let I be the set of disclosed puzzles, then we have that

$$R_i^{\ell_i(0)} \cdot \prod_{j \in I} R_j^{\ell_j(0)} = (g^c \cdot h^r)^{s_i \cdot \ell_i(0)} \cdot \prod_{j \in I} (g^c \cdot h^r)^{s_j \cdot \ell_j(0)}$$
$$R = (g^c \cdot h^r)^{s_i \cdot \ell_i(0) + \sum_{i \in I} s_j \cdot \ell_j(0)}$$
$$R = (g^c \cdot h^r)^{\tilde{s}}$$

and $x = r \mod q$, where $R := (x, y)$, by definition of the verification equation. It follows that $(r, \tilde{s})$ is a valid signature on m. Then the proof is completed by observing that a prover that commits invalid s_i on all unopened puzzles must have guessed the challenge set I ahead of time, which happens only with negligible probability.

We again stress that the non-interactive variant of the proof can be shown to be simulation-sound with the same argument, assuming an appropriate instantiation of the range NIZK. □

C PROOF ANALYSIS FOR RANGE PROOFS

Soundness We now establish soundness of our protocol. As usual for Fiat-Shamir protocols, we will consider soundness of the interactive protocol. Thus, fix time-lock puzzles $Z_1, \dots, Z_\ell$. Since the public parameters pp are chosen honestly, each time-lock puzzle Z_j has a unique corresponding plaintext x_j, i.e. the time-lock puzzles are perfectly binding commitments.

Now assume that $Z_1, \dots, Z_\ell$ is a false statement, i.e. there exists an index j^* such that $x_{j^*} \notin [-L, L]$. We now show that the verifier rejects the statement, except with negligible probability.

Let $\mathbf{t}_1, \dots, \mathbf{t}_k$ be the verifier's challenge. We only consider a single index i and show that the verifier accepts for this index with probability at most 1/2. It follows by a standard parallel repetition argument that the verifier accepts with probability at most 2^{-k}. Thus fix an index i.

Further fix all $t_{i,j}$ for $j \neq j^*$ for the moment. We distinguish two cases.

(1) In this case it holds that $y_i + \sum_{j \in [k], j \neq j^*} t_{i,j} x_j \notin [-L/2, L/2]$. Since t_{i,j^*} is uniform in $\{0, 1\}$, it holds that $\Pr[t_{i,j^*} = 0] = \frac{1}{2}$. Thus, it follows that $\Pr[y_i + \sum_{j \in [k]} t_{i,j} x_j \in [-L/2, L/2]] \leq 1/2$.
(2) In this case it holds that $y_i + \sum_{j \in [k], j \neq j^*} t_{i,j} x_j \in [-L/2, L/2]$. It follows that $y_i + x_{j^*} + \sum_{j \in [k], j \neq j^*} t_{i,j} x_j \notin [-L/2, L/2]$ as $x_{j^*} \notin [-L, L]$. Consequently, as t_{i,j^*} is uniform on $\{0, 1\}$, it holds that $\Pr[y_i + \sum_{j \in [k]} t_{i,j} x_j \in [-L/2, L/2]] \leq 1/2$

Applying the law of total probability, i.e. marginalizing over all choices of $t_{i,j}$ for $j \neq j^*$, Pr[Verifier accepts for index i] $\leq 1/2$. It follows that Pr[Verifier accepts] $\leq 2^{-k}$.

Zero-Knowledge We now show that the proof-system is zero-knowledge. We do this by showing that the corresponding interactive proof system is honest-verifier statistical zero-knowledge. We use the following standard lemma, proven e.g. in [5].

Lemma 1. *Let $U_{[-r,r]}$ be the uniform distribution on the interval $[-r,r]$ and $\beta \in \mathbb{Z}$. Then the statistical distance between $U_{[-r,r]}$ and $U_{[-r,r]} + \beta$ is β/r.*

Our simulator $\mathcal{S}$ is now given as follows.

- Input: Statement pp, $Z_1, \dots, Z_\ell$ and challenge $\mathbf{t}_1, \dots, \mathbf{t}_\ell$
- Choose $\tilde{v}_1, \dots, \tilde{v}_k \leftarrow [-L/2, L/2]$ and $\tilde{w}_1, \dots, \tilde{w}_k \leftarrow \mathbb{Z}_N$.
- For $i = 1, \dots, k$ compute $G_i \leftarrow \mathsf{HTLP.PGen}(\mathrm{pp}, \tilde{v}_i; \tilde{w}_i)$.
- For $i = 1, \dots, k$ set $\tilde{D}_i \leftarrow G_i \cdot \left(\prod_{j=1}^{\ell} Z_j^{t_{i,j}}\right)^{-1}$.
- Output $\pi \leftarrow (\tilde{D}_i, \tilde{v}_i, \tilde{w}_i)_{i \in [k]}$

First note that the simulator $\mathcal{S}$ is efficient and produces an accepting proof π if the statement is valid. We argue that the distributions produced by the prover and the simulator $\mathcal{S}$ are statistically close, given that B/L is negligible. A reasonable practical choice of parameters may be $B/L = 2^{-50}$, i.e. B is 50 bits shorter than L.

To prove the ZK property, it is sufficient to argue that the $(D_1, \dots, D_k)$ produced by the prover and $(\tilde{D}_1, \dots, \tilde{D}_k)$ produced by the simulator are statistically close. Note that the $(D_1, \dots, D_k)$ are uniquely specified by $((v_1, w_1), \dots, (v_k, w_k))$ and the $(\tilde{D}_1, \dots, \tilde{D}_k)$ by $((\tilde{v}_1, \tilde{w}_1), \dots, (\tilde{v}_k, \tilde{w}_k))$.

First note that the distributions of the w_i and the $\tilde{w}_i$ are each i.i.d uniformly random. Define the $\tilde{y}_i$ to be the plaintexts corresponding to the time-lock puzzles $\tilde{D}_i$ produced by the simulator $\mathcal{S}$. It holds for all $i \in [k]$ that $\tilde{y}_i = \tilde{v}_i - \sum_{j=1}^{\ell} t_{i,j} x_j$, where as above the $x_1, \dots, x_\ell$ are the plaintexts corresponding to $Z_1, \dots, Z_\ell$. By Lemma 1 and a hybrid argument it follows that $(y_1, \dots, y_k)$ and $(\tilde{y}_1, \dots, \tilde{y}_k)$ have statistical distance at most $k \cdot \frac{B}{L/2} = 2k \cdot B/L$, which is negligible. This also implies that $(D_1, \dots, D_k)$ and $(\tilde{D}_1, \dots, \tilde{D}_k)$ are statistically close.

We remark that since our simulator does not use a trapdoor, one can readily show that our proof-system satisfies simulation-soundness, as is typical for proof-systems constructed via the Fiat-Shamir methodology.

D VERIFIABLE TIMED COMMITMENT

Verifiable Timed Commitments for signing keys are similar to VTS except that now the committer creates a commitment of a discrete log (signing key) instead of a signature. In the context of a signature scheme, formally, the Commit algorithm outputs a timed commitment C to a signing key corresponding to a public key and a proof π for the same. The definitions of privacy and soundness follow same as VTS.

Theorem 7 (Privacy). *Let* (ZKsetup, ZKprove, ZKverify) *be a NIZK for $\mathcal{L}_{\text{range}}$ and let* LHTLP. *be a secure time-lock puzzle. Then the protocol as described in Figure 8 satisfies privacy in the random oracle model.*

Proof. We show that the protocol (Figure 8) is private against an adversary of depth bounded by $\mathbf{T}^{\varepsilon}$, for some non-negative $\varepsilon < 1$. We now gradually change the simulation through a series of hybrids and then we argue about the proximity of neighbouring experiments.

Hybrid $\mathcal{H}_0$: This is the original execution.

Hybrid $\mathcal{H}_1$: This is identical to the previous hybrid except that the random oracle is simulated by lazy sampling. In addition a random set I^* (where $|I^*| = t-1$) is sampled ahead of time, and the output of the random oracle on the cut-and-choose instance is programmed to I^*. Note that the changes of this hybrid are only syntactical and therefore the distribution is unchanged.

Hybrid $\mathcal{H}_2$: In this hybrid we sample a simulated common reference string crs_{range}. By the zero-knowledge property of (ZKsetup, ZKprove, ZKverify) this change is computationally indistinguishable.

Hybrid $\mathcal{H}_3 \dots \mathcal{H}_{3+n}$: In the hybrid $\mathcal{H}_{3+i}$, for all $i \in [n]$, the proof $\pi_{\text{range},i}$ is computed via the simulator provided by the underlying NIZK proof. By the zero-knowledge property of (ZKsetup, ZKprove, ZKverify), the distance between neighbouring hybrids is bounded by a negligible function in the security parameter.

Hybrid $\mathcal{H}_{3+n} \dots \mathcal{H}_{3+2n-t+1}$: In the i-th hybrid $\mathcal{H}_{3+i}$, for all $i \in [n-(t-1)]$, the puzzle corresponding to the i-th element of the set $\bar{I}^*$ is

Setup: Same as Figure 2.

Commit and Prove: On input (crs, wit) the Commit algorithm does the following.

- Parse wit $:= sk$, $crs := (crs_{\text{range}}, \mathrm{pp})$, $pk = h$ as the public key
- For all $i \in [t-1]$ sample a uniform $x_i \leftarrow \mathbb{Z}_q$ and set $h_i := g^{x_i}$
- For all $i \in \{t, \dots, n\}$ compute

$$x_i = \left(sk - \sum_{j \in [t]} x_j \cdot \ell_j(0)\right) \cdot \ell_i(0)^{-1} \text{ and } h_i = \left(\frac{pk}{\prod_{j \in [t]} h_j^{\ell_j(0)}}\right)^{\ell_i(0)^{-1}}$$

  where $\ell_i(\cdot)$ is the i-th Lagrange polynomial basis.
- For $i \in [n]$, generate puzzles with corresponding range proofs as shown below

$$r_i \leftarrow \{0,1\}^{\lambda}, Z_i \leftarrow \mathsf{LHTLP.PGen}(\mathrm{pp}, x_i; r_i)$$
$$\pi_{\text{range},i} \leftarrow \mathsf{ZKprove}(crs_{\text{range}}, (Z_i, a, b, \mathbf{T}), (x_i, r_i))$$

- Compute $I \leftarrow H'\left(pk, (h_1, Z_1, \pi_{\text{range},1}), \dots, (h_n, Z_n, \pi_{\text{range},n})\right)$
- The Commit algorithm outputs $C := (Z_1, \dots, Z_n, \mathbf{T})$ and $\pi := (\{h_i, \pi_{\text{range},i}\}_{i \in [n]}, I, \{\sigma_i, r_i\}_{i \in I})$
- Finally output (pk, C, π)

Verification: On input (crs, pk, C, π) the Vrfy algorithm does the following.

- Parse $C := (Z_1, \dots, Z_n, \mathbf{T})$, $\pi := (\{h_i, \pi_{\text{range},i}\}_{i \in [n]}, I, \{x_i, r_i\}_{i \in I})$ and $crs := (crs_{\text{range}}, \mathrm{pp})$
- If any of the following conditions is satisfied output 0, else return 1:
  (1) There exists some $j \notin I$ such that $\prod_{i \in I} h_i^{\ell_i(0)} \cdot h_j^{\ell_j(0)} \neq pk$
  (2) There exists some $i \in [n]$ such that $\mathsf{ZKverify}(crs_{\text{range}}, (Z_i, a, b, \mathbf{T}), \pi_{\text{range},i}) \neq 1$
  (3) There exists some $i \in I$ such that $Z_i \neq \mathsf{LHTLP.PGen}(\mathrm{pp}, x_i; r_i)$ or $h_i = g^{x_i}$
  (4) $I \neq H'\left(pk, (h_1, Z_1, \pi_{\text{range},1}), \dots, (h_n, Z_n, \pi_{\text{range},n})\right)$

Figure 8: Verifiable Timed commitments for signing keys of the form $pk = g^{sk}$, $sk \in \{0,1\}^{\lambda}$

computed as LHTLP.PGen(pp, $0^\lambda; r_i$), where $\bar{I}^*$ is the complement of I^*. Since the distinguisher is depth-bounded, indistinguishability follows from an invocation of the security of LHTLP..

Hybrid $\mathcal{H}_{3+2n-t+2}$: In this hybrid the prover behaves as follows. For all $i \in I^*$ it samples a uniform $x_i \leftarrow \mathbb{Z}_q$ and sets $h_i = g^{x_i}$ and computes the puzzle as described in the protocol. On the other hand, for all $i \notin I^*$ it computes h_i as

$$h_i = \left(\frac{pk}{\prod_{j \in I^*} h_j^{\ell_j(0)}} \right)^{\ell_i(0)^{-1}}.$$

The rest of the execution is unchanged. Note that for all $i \notin I^*$ we have that

$$\prod_{j \in I^*} h_j^{\ell_j(0)} \cdot h_i^{\ell_i(0)} = pk$$

as expected. It follows that the changes in this hybrid are only syntactical and the distribution induced is identical to that of the previous hybrid.

Simulator $\mathcal{S}$: The simulator is defined to be identical to the last hybrid. Note that no information about the witness is used to compute the proof. This concludes our proof.

□

Theorem 8 (Soundness). *Let* (ZKsetup, ZKprove, ZKverify) *be a NIZK for* $\mathcal{L}_{\text{range}}$ *and let* LHTLP. *be a time-lock puzzle with perfect correctness. Then the protocol as described in Figure 8 satisfies soundness in the random oracle model.*

Proof. We analyze the protocol in its interactive version and the soundness of non-interactive protocol follows from the Fiat-Shamir transformation [22] for constant-round protocols. Let $\mathcal{A}$ be an adversary that efficiently breaks the soundness of the protocol. In particular this means that the adversary produces a commitment $(Z_1, \dots, Z_n)$ and for all $Z_i \notin I$ it holds that LHTLP.PSolve(pp, Z_i) = $\tilde{x}_i$ such that

$$h_i \neq g^{\tilde{x}_i}.$$

Assume the contrary, then we could recover a valid signing key by interpolating $\tilde{x}_i$ with $\{x_i\}_{i \in I}$, which satisfy the above relation of the public key and secret key. Further observe that all puzzles $(Z_1, \dots, Z_n)$ are well-formed, i.e., the solving algorithm always outputs some well-defined value, except with negligible probability, by the soundness of the range NIZK.

It follows that, given $(Z_1, \dots, Z_n)$ we can recover some set I' in polynomial time by solving the puzzles and checking which of the signing keys satisfy the above relation. In order for the verifier to accept, it must be the case that $I' = I$ which means that the prover correctly guesses a random n-bit string uniformly chosen from the set of strings with exactly $n/2$-many 0's. This happens with probability exactly $\frac{(n/2!)^2}{n!}$.

□