Verifiable Timed Linkable Ring Signatures for Scalable Payments for Monero

Sri AravindaKrishnan Thyagarajan[1(✉)], Giulio Malavolta[2], Fritz Schmid[3], and Dominique Schröder[3]

[1] Carnegie Mellon University, Pittsburgh, USA
t.srikrishnan@gmail.com
[2] Max Planck Institute for Security and Privacy, Bochum, Germany
[3] Friedrich Alexander Universität Erlangen-Nürnberg, Erlangen, Germany
{fritz.schmid,dominique.schroeder}@fau.edu

Abstract. Decentralized cryptocurrencies still suffer from three interrelated weaknesses: Low transaction rates, high transaction fees, and long confirmation times. Payment Channels promise to be a solution to these issues, and many constructions for cryptocurrencies, such as Bitcoin and Ethereuem, are known. Somewhat surprisingly, no solution is known for Monero, the largest privacy-preserving cryptocurrency, without requiring system-wide changes like a hard-fork of its blockchain like prior solutions.

In this work, we close this gap for Monero by presenting the first provably secure payment channel protocol that is fully compatible with Monero's transaction scheme. Notably, the payment channel related transactions are identical to standard transactions in Monero, therefore not hampering the coins' fungibility. With standard techniques, our payment channels can be extended to support atomic swap of tokens in Monero with tokens of several other major currencies like Bitcoin, Ethereum, Ripple, etc., in a fungible and privacy-preserving manner.

Our main technical contribution is a new cryptographic tool called verifiable timed linkable ring signatures (VTLRS), where linkable ring signatures can be hidden for a pre-determined amount of time in a verifiable way. We present a practically efficient construction of VTLRS which is fully compatible with the transaction scheme of Monero, and allows for users to make timed payments to the future which might be of independent interest to develop other applications on Monero.

Our implementation results show that even with high network latency and with a single CPU core, two regular users can perform up to 93500 payments over 2 min (the block production rate of Monero). This is approximately five orders of magnitude improvement over the current payment rate of Monero.

Keywords: Timed signatures · Time-lock puzzles · Payment channels

1 Introduction

Modern cryptocurrencies, such as Bitcoin or Monero, realize the digital analog of a fiat currency without a trusted central authority. They typically consist of

V. Atluri et al. (Eds.): ESORICS 2022, LNCS 13555, pp. 467–486, 2022.
https://doi.org/10.1007/978-3-031-17146-8_23

two main components: (i) A public ledger that publishes all transactions and (ii) a transaction scheme that describes the structure and validity of transactions. Compared to traditional centralized solutions, decentralized cryptocurrencies suffer from three weaknesses: First, they have a relatively low transaction rate; for example, the current transaction rate of Bitcoin is about four transactions per second while it is 0.1 transactions per second in case of Monero [1]. Second, the transaction fees are relatively high, about 0,60$ per transaction in the case of Bitcoin and about 0.25$ in Monero [2]. Third, the confirmation of a transaction takes (on average) one hour in the case of Bitcoin and 20 min in the case of Monero. Payment Channels (PC) [27], and its generalization Payment Channel Networks (PCN) [15,22,23,27] have emerged as one of the most promising solutions to mitigate these issues and have been widely deployed to scale payment in major cryptocurrencies, such as Bitcoin [5], Ethereum [7] or Ripple [6]. These solutions are commonly referred to as *layer 2* or *off-chain* solutions.

A PC allows a pair of users to perform multiple payments without committing every intermediate payment to the blockchain. Abstractly, a PC consists of three phases: (i) Two users Alice and Bob, open a payment channel or a *joint address* by adding a single transaction to the blockchain. This transaction is a promise from Alice that she 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 may send coins from the joint address to Bob by sending a corresponding transaction to the other user. (iii) The channel closes when one of those payment transactions is posted on the chain, thus spending coins from the joint address. *Bi-directional* payment channels allow for payments to be made from the joint address to either users. While realizing PCs for Bitcoin is an established task due to the functionality available in the Bitcoin scripting language, several challenges arise when considering privacy-preserving cryptocurrencies like Monero or Zcash [11]. Bolt [18] is a PC proposal for Zcash while Moreno-Sanchez et al. [26] developed a PC protocol for Monero. However, their proposal has various shortcomings (see below for more details) and, as a consequence, is unlikely to be integrated into Monero. In this work, we aim to close this gap by constructing a PC protocol that is fully compatible with the transaction scheme of Monero and can be used to make off-chain transactions.

Brief Look into Monero. Monero is the largest privacy-preserving cryptocurrency. The notion of privacy it offers is that: Any external observer cannot learn who the sender or the receiver of a transaction is and the number of coins being transferred. Monero achieves these properties with Ring Confidential Transactions (RingCT) as its cryptographic bedrock. Briefly, a RingCT is a transaction scheme where the sender of the transaction 'hides' his key in an anonymity set (ring). Comparing with Bitcoin, where transactions have typically one source address, the amount in plain, one or two recipient address(es), and a simple signature (ECDSA), Monero transactions contain a ring of addresses, destination addresses, commitments to amounts, related consistency proofs, and a (linkable ring) signature, making the transactions considerably larger. Moreover, the size

of a Monero transaction grows linearly with the size of the anonymity set. This results in a major setback to the scalability of Monero and often requires users to make tough choices between better privacy (high transaction fee) and smaller transactions (lower transaction fees). There has been a line of research [20,36] that proposes new and efficient RingCT constructions that result in smaller transaction sizes. These approaches help to increase privacy because they support larger ring sizes and therefore do not increase the transaction fees. However, the central three issues that PCs are addressing remain open: Increasing the transaction rate, reducing the transaction fees in general, and therefore supporting fast micro-payments, as well as fast verification time. Moreover, all these on-chain solutions require system-wide changes in the Monero protocol, and it is unclear if Monero will fork and adapt to one of these schemes. Unfortunately, layer two solutions (such as PCs) proposed for Bitcoin do not extend to Monero as they require scripting features that are absent in Monero.

1.1 Our Contribution

The contributions of this work can be summarized as follows.

- We propose PayMo, the first payment channel protocol that is fully compatible with the transaction scheme of Monero (Appendix B). A notable feature of our solution is that PC transactions are syntactically identical to standard transactions in Monero, thus retaining the *fungibility*[1] of the Monero coins. Following the work from [23], we can also extend PayMoto have the first secure *scriptless* atomic-swap for Monero with many other major currencies.
- At the heart of our proposal is a new cryptographic primitive, called *verifiable timed linkable ring signatures* (VTLRS), that we define and construct (Sect. 3). Our solution relies on well-established cryptographic assumptions, and VTLRS can be of interest for other applications on Monero.
- We demonstrate the practicality of our approach by benchmarking PayMo (Sect. 4). Our analysis shows that PayMo can be used on today's hardware by Monero's users. In terms of performance, at its full power, PayMo supports close to 93500 payments for 2 min between two regular users with one CPU core each. Here 2 min is the block production rate of Monero. This is a significant increase in payments in Monero, which currently supports only one payment from an address per 2 min.

1.2 Related Work and Discussion

In the following, we compare our approach with existing systems, and we discuss some of the choices behind practical aspects of our design.

The first PC proposal for Monero was recently put forth by Moreno-Sanchez et al. [26], however, their solution requires a hard fork with major changes to the

[1] Fungibility is a property of a currency whereby two units can be substituted in place of one another: no coins are special irrespective of transactions acting on them.

Monero transaction scheme and is *not* backward compatible. Specifically, a joint address of their comprises of two public keys where the secret keys of both keys are shared among the users, and they also require an explicit time-lock script for the joint address. Both of the above requirements are not supported in Monero.

We stress that even assuming that Monero will fork in the near future to integrate their scheme, the adoption procedure still requires one to solve some challenges: Since the tag generation in [26] is different from the currently used algorithm, one needs to perform massive system-wide changes to the Monero protocol itself, requiring *every* Monero user to spend from their existing, unspent keys (with old tag generation) to a new key (with the new tag generation) during a specific time interval. This is highly undesirable as it requires *every* Monero user to be online and make transactions, and any user unable to make this switch during this time interval loses his coins permanently. An additional limitation of their proposal in terms of transaction privacy where the time-lock information which is public can lead to on-chain censorship from miners and other users.

On the contrary, PayMo does not require any changes to the transaction scheme of Monero, nor it needs to add any functionality to the scripting language. Any interested pair of users can run PayMo without the knowledge of any other user in the Monero system. Furthermore, any PayMo related transaction posted on-chain is *identical* to posting any other regular transaction in Monero. However, PayMousers need to run a background computation in the form of time-lock puzzle solving [28]. We discuss how to mitigate this computational load with batching techniques and outsourcing the solving to a decentralized service [34].

Payment Channels and Payment Channel Networks [22,27] have been proposed as solutions for Bitcoin's scalabiltiy problem. They rely on the special script *Hash Time-Lock Contract (HTLC)* that lets a user get paid if he produces a pre-image of a certain hash value before a specific time, referred to as the time-lock of the payment. Bolt [18] is a payment channel protocol specially tailored for Zcash [11] which uses zk-SNARKs [17], and is not compatible with the transaction scheme of Monero, which is the focus of this work. A generalisation of a payment channel with complex conditional payments is a *state channel* [14] that requires highly expressive scripting functionalities from the underlying blockchain, that are not available in Monero. Recently Gugger [19] proposed a mechanism for atomic swaps between Bitcoin and Monero. However their swap protocol is only semi-scriptless because they require a hash function verification from the bitcoin script. On the other hand, all PayMo requires signature verification from the Bitcoin and Monero and, therefore, improves both the coins' fungibility in their respective chains.

2 Technical Overview

We first introduce the notion of *Verifiable Timed Linkable Ring Signature* (VTLRS) that is fully compatible with the transaction scheme in Monero. Then we describe how to leverage VTLRS to construct payment channels (PCs) for Monero. Finally, we discuss how to extend our protocol to support atomic swaps

with tokens from other currencies and how to integrate our approach in the current implementation of Monero.

For ease of presentation, we consider a simplified view of a transaction in Monero consisting of: A ring of one-time public keys (addresses) $\mathcal{R}$, a linkability tag *tag* (for double-spend protection against same key spending twice), a signature σ and the target public key (recipient). We omit other components of a Monero transaction as our tools and techniques only deal with the above components and it can be naturally extended to the current transaction scheme of Monero with all components in place[2].

2.1 VTLRS for Monero

On a high-level, a VTLRS lets a user create a timed commitment of a linkable ring signature on a message (transaction) such that the recipient of the commitment can force open the commitment and learn the signature only after a pre-specified time $\mathbf{T}$. The recipient also receives a proof that convinces him that the force opening would indeed reveal the valid linkable ring signature on the message. Our construction of VTLRS is compatible with the linkable ring signature transaction scheme that is currently implemented in Monero, where the message is now a Monero transaction.

On a high-level, to commit to a VTLRS, the committer takes a (linkable ring) signature on a transaction *tx* and *encodes* it into a time-lock puzzle [28], which keeps it hidden until time $\mathbf{T}$. To convince the verifier that the puzzle contains a valid signature on the transaction *tx*, the committer also computes a non-interactive zero-knowledge (NIZK) proof for such a statement. The challenge here is to design an efficient NIZK proof that certifies the validity of the encoded signature. General solutions exist [32] only for common signature schemes, like Schnorr, ECDSA, and BLS.

Efficient NIZKs. To design an efficient NIZK, we adopt a cut-and-choose approach. The basic idea of this approach is to encode the signature into many puzzles redundantly. The validity can be checked by revealing the random coins corresponding to a subset of them. If implemented naïvely, this could compromise the privacy of the signature. Instead, we harness the structural properties of signatures in Monero to reveal only isolated components while at the same time keeping the signature hidden. More specifically, the committer computes a t-out-of-n secret sharing of a particular component of the signature. Given a $t-1$ subset of the shares (which are revealed by the cut-and-choose), the verifier can check the validity of the puzzles and that these opened shares are valid shares of the signature component. If the check is successful, then the verifier is convinced that *at least one* of the unopened puzzles contains a valid share, which is enough to reconstruct a valid signature. The scheme is made non-interactive using the Fiat-Shamir transformation [16].

---

[2] A Monero transaction is based on RingCT [20] which additionally consists of commitments to hide the amounts and range proofs to prove that they are well-formed.

Time-Lock Puzzles. We then instantiate the time-lock puzzles with [24,33] and use the homomorphic properties of such a scheme to combine puzzles in such a way that the computation needed to force open is the same as that to force open *a single puzzle*. We stress that the use of homomorphism is crucial for the solver's efficiency (verifier) and also for security. Without the homomorphism, a user with $\tilde{n} = n - (t-1)$ processors can solve $\tilde{n}$ puzzles in parallel and in total time $\mathbf{T}$. On the other hand, users with fewer processors will have to solve the puzzles one after another, thereby spending more time than $\mathbf{T}$ time. This could lead to scenarios in PCs where an adversarial party has an unfair advantage with respect to an honest user and could post a valid transaction ahead of time, effectively stealing coins.

2.2 (Uni-directional) Payment Channels in Monero

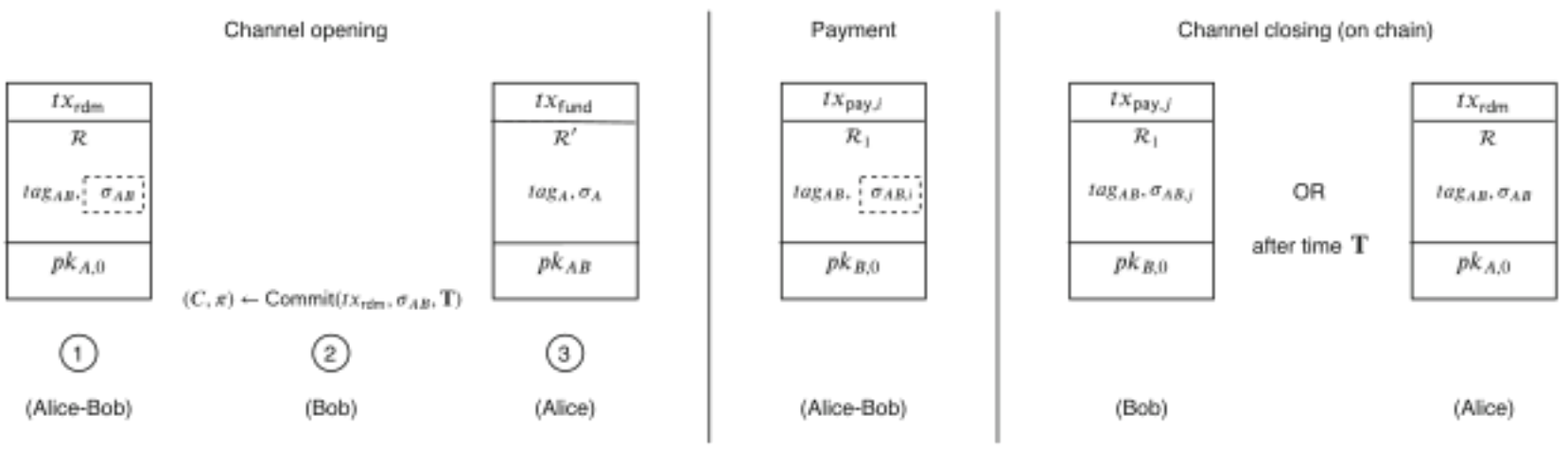

Fig. 1. Three phases to a (uni-directional) payment channel protocol between Alice and Bob in PayMo. The channel opening phase has three steps. Steps run individually and jointly through interaction are denoted with (Alice) or (Bob) and (Alice-Bob) respectively. Signature σ_{AB} inside a dotted box indicates that only Bob learns the signature after interaction with Alice.

Equipped with our efficient VTLRS scheme, we show how Alice and Bob can run a payment channel protocol to make payments (i.e., an address whose secret key is shared between Alice and Bob, and both of them have to agree on a payment). A pictorial description of our PC protocol (PayMo) is given in Fig. 1 and its three main subroutines (channel opening, payment, and channel closing) are discussed below briefly. Due to space constraints, we defer the complete formal description to the full version [35] along with full security analysis.

Channel Opening. Alice and Bob jointly generate a spending key pk_{AB} (the corresponding secret key is additively shared between Alice and Bob) and a redeem transaction tx_{rdm} (containing the joint tag tag_{AB}) that spends the coins from pk_{AB} (belonging to some ring $\mathcal{R}$) to some address of Alice $pk_{A,0}$. Note that pk_{AB} is an address that is not yet present on the chain. Bob then generates a VTLRS of the signature σ_{AB} on tx_{rdm} with timing hardness $\mathbf{T}$. Bob gives the VTLRS commitment and proof (generated using $\mathsf{Com}(tx_{\mathsf{rdm}}, \sigma_{AB}, \mathbf{T})$ in Fig. 1) to Alice, who then posts the transaction tx_{fund} on the Monero blockchain. Such

a transaction initializes the channel by sending funds from one of her addresses pk_A to the joint key pk_{AB}. The channel is now created and initialized on-chain, and its expiry time is set to $\mathbf{T}$ (via the VTLRS). Note that after time $\mathbf{T}$ Alice will be able to recover the signature tx_{rdm} and redeem the remaining funds in the address pk_{AB} if any.

Payment. When Alice wishes to pay Bob, they jointly generate a transaction $tx_{\mathsf{pay},i}$ for the i-th payment. $tx_{\mathsf{pay},i}$ spends from pk_{AB} (in some ring $\mathcal{R}_1$) and sends it to some address of Bob $pk_{B,0}$. They jointly generate a signature $\sigma_{AB,i}$ on $tx_{\mathsf{pay},i}$ (as the secret key for pk_{AB} was shared among the users), in such a way that only Bob learns the signature $\sigma_{AB,i}$. Importantly, the transaction $tx_{\mathsf{pay},i}$ is not posted on the blockchain. Alice and Bob can continue making further payments this way.

Channel Closing. If Bob wishes to close the channel, he takes the last exchanged transaction payment $tx_{\mathsf{pay},j}$ with Alice and posts it along with $\sigma_{AB,j}$ on the Monero blockchain. In case Bob has not posted any such payment and time $\mathbf{T}$ has passed, Alice by then learns σ_{AB} from the VTLRS on tx_{rdm} (that was given by Bob during channel opening). Alice can now post tx_{rdm} and σ_{AB} on the Monero blockchain and redeem all the coins from the channel. In either case, once a transaction spending from pk_{AB} is posted on-chain, the payment channel is considered closed.

Integration in Monero. An additional challenge stems from the fact that each public key in the ring used in a transaction is associated with a `key_offset` field [3] in the current implementation of Monero. This field stores the index of the key with respect to the global set of public keys as a way to optimise the look up of keys during transaction verification. Recall that, during channel opening, Alice and Bob need to generate the redeem transaction tx_{rdm} that spends from the payment channel key pk_{AB} before tx_{fund} (that spends to pk_{AB}) is posted on the blockchain. This means that, in order to sign a correctly formed transaction tx_{rdm}, one needs to guess ahead of time the offset position of pk_{AB}.

There are two ways to bypass this obstacle. (1) Modify the current implementation (not the transaction scheme) to adopt a different look up strategy for public keys, that allows users to sign transactions that spend from a key that is not posted on the blockchain yet. (2) Instead of generating a VTLRS commitment of a signature on tx_{rdm}, Bob can generate a timed commitment to his share of the joint secret key sk_{AB}, for time $\mathbf{T}$. After force opening the commitment, Alice learns sk_{AB} and can use it to correctly sign tx_{rdm}, since the offset of pk_{AB} is fixed at this point. Clearly, one needs an efficient mechanism to ensure that the timed commitment of Bob indeed contains a valid share of sk_{AB}. This can be realized using a verifiable timed discrete-log (VTDlog) scheme, and an efficient instantiation was recently proposed in [32].

While using VTDlog is a viable option to construct Monero-compatible PCs, we argue that VTLRS is a more desirable solution, since it enables the usage of stealth addresses [20]. Stealth addressing reduces interaction between the sender and the receiver of a payment in the following way: Alice (sender) generates a

one-time public key opk_B for Bob (recipient) given access to Bob's master public key mpk_B, and then sends coins to opk_B. Bob can later spend from opk_B by generating the one time secret key osk_B using his master secret key msk_B. This way the receiver is not required to send a recipient key to the sender every time he wishes to receive funds. Since the VTD-based solution leaks information about the long term secret key of a party, stealth addressing scheme used currently in Monero, is no longer a viable option, as one could link all future transactions of Bob once osk_B is disclosed [30]. Note that this issue does not arise in the VTLRS-based scheme, since neither user learns the secret key shares of the other user involved. More details can be found in [35].

Outsource Computation. Notice that Alice is required to perform persistent computation to open her VTLRS commitments. This could limit the number of channels that Alice can operate simultaneously. However, the persistent computation of opening a VTLRS commitment can be securely outsourced to a decentralized service [34] at a market determined cost. This relieves Alice of any potentially heavy computation related to VTLRS opening, provided she has enough funds to outsource using the service from [34]. Therefore the number of channels Alice operates is no longer limited by her computational power.

Extending to Bi-directional Payments. PayMo supports payment channels with uni-directional payments. A payment channel with bi-directional payments allows both Alice and Bob to make payments to each other using their channel. A recent work [10] proposed *Sleepy Channels*, the first bi-directional payment channel protocol compatible with Monero. However, the key tools that they require to achieve this, are based on our work. Specifically, they crucially rely on VTLRS and the channel operations of PayMo, to realise timed payments and payment revocation, which are essential for bi-directional payment channels.

3 Verifiable Timed Linkable Ring Signature

In the following we define and construct a *Verifiable Timed Linkable Ring Signature* (VTLRS) transaction scheme. We introduce some notation that we use extensively in this paper. We denote by $\lambda \in \mathbb{N}$ the security parameter and by $x \leftarrow \mathcal{A}(\mathsf{in}; r)$ the output of the algorithm $\mathcal{A}$ on input in using $r \leftarrow \{0,1\}^*$ as its randomness. We omit this randomness and only mention it explicitly when required. We denote the set $\{1, \ldots, n\}$ by $[n]$. We model parallel algorithms as Parallel Random Access Machines (PRAM) and *probabilistic polynomial time* (PPT) machines as efficient algorithms.

3.1 Definition

A VTLRS is a linkable ring signature based transaction scheme (see Appendix B) where one commits to such a signature in a *verifiable* and *extractable* way.

Definition 1 (VTLRS). *A Verifiable Timed Linkable Ring Signature Transaction Scheme* Π_{VTLRS}*, for a LRS transaction scheme* $\Sigma := (\mathsf{Setup}, \mathsf{OTKGen}, \mathsf{TgGen}, \mathsf{Spend}, \mathsf{Vf})$ *is a tuple of five algorithms* $(\mathsf{Setup}, \mathsf{Com}, \mathsf{Vfy}, \mathsf{Op}, \mathsf{FOp})$ *where:*

$crs \leftarrow \mathsf{Setup}(1^\lambda)$*: the setup algorithm outputs a common reference string* crs *which is implicitly taken as input in all other algorithms.*

$(C, \pi) \leftarrow \mathsf{Com}(\sigma, tx, \mathbf{T}; r)$*: the commit algorithm takes as input a signature* σ*, the transaction* tx*, a hiding time* $\mathbf{T}$ *and randomness* r*. It outputs a commitment* C *and a proof* π*.*

$0/1 \leftarrow \mathsf{Vfy}(tx, C, \pi)$*: the verify algorithm takes as input a transaction* m*, a commitment* C *of hardness* $\mathbf{T}$ *and a proof* π *and accepts the proof if and only if, the value* σ *embedded in* C *is a valid signature on the transaction* tx *(i.e.,* $\mathsf{LRS.Vf}(tx, \sigma) = 1$*). Else it outputs* 0*.*

$(\sigma, r) \leftarrow \mathsf{Op}(C; r)$*: the opening algorithm is run by the committer that as input a commitment* C *and outputs the committed signature* σ *and the randomness* r *used in generating the commitment* C*.*

$\sigma \leftarrow \mathsf{FOp}(C)$*: the deterministic* FOp *algorithm takes as input the commitment* C *and outputs a signature* σ*.*

We require standard notion of correctness that is formalized in the full version [35]. In terms of security, a VTLRS must satisfy the notions of *timed privacy* and *soundness*, defined below.

Timed Privacy. This notion requires that 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. The adversary is given the spending public key and the tag as input, and gets access to a spending oracle. The challenge for the adversary here is to distinguish (within time $\mathbf{T}$ even with parallelism) a commitment from being a commitment to a valid LRS signature with the above attributes, to a simulated commitment.

Definition 2 (Timed Privacy). *A VTLRS scheme* $\Pi_{\mathsf{VTLRS}} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Vfy}, \mathsf{Op}, \mathsf{FOp})$ *for a LRS transaction scheme* Σ *is* timed 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 algorithms* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *where* $\mathcal{A}_1$ *is PPT and* $\mathcal{A}_2$ *is a PRAM whose running time is at most* $t < \mathbf{T}$*, and all* $\lambda \in \mathbb{N}$ *it holds that*

$$\Pr\left[b = b' \;\middle|\; \begin{array}{c} (pk, sk) \leftarrow \mathsf{LRS.OTKGen}(pp) \\ tag \leftarrow \mathsf{TgGen}(sk) \\ (\mathcal{R}, \mathcal{O}, \mu) \leftarrow \mathcal{A}_1^{\mathsf{Spend}\mathcal{O}}(pk, tag, pp) \\ s.t.\ pk = pk_{|\mathcal{R}|} \ and\ tx := \{\mathcal{R}, tag, \mathcal{O}, \mu\} \\ b \leftarrow \{0,1\},\ b' \leftarrow \mathcal{A}_2^{\mathsf{Spend}\mathcal{O}}(tx, C_b, \pi_b) \end{array} \right] \leq \mathsf{negl}(\lambda)$$

where, $pp \leftarrow \mathsf{LRS.Setup}(1^\lambda)$, $crs \leftarrow \mathsf{Setup}(1^\lambda)$ *and if* $b = 0$*, then* $(C_0, \pi_0) \leftarrow \mathsf{Com}(\sigma, tx, \mathbf{T})$ *where* $\sigma \leftarrow \mathsf{LRS.Spend}(\mathcal{R}, (|\mathcal{R}|, sk, tag), \mathcal{O}, \mu)$ *and if* $b = 1$*,* $(C_1, \pi_1) \leftarrow \mathcal{S}(pk, tx, \mathbf{T})$*.*

Soundness. This says that the accepting verifier is convinced that given C, the FOp algorithm will return a valid signature σ on transaction tx in time $\mathbf{T}$. A VTLRS 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.

Definition 3 (Soundness). *A VTLRS scheme* $\Pi_{\mathsf{VTLRS}} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Vfy}, \mathsf{Op}, \mathsf{FOp})$ *for a LRS transaction scheme* Σ *is* sound *if there is a negligible function* negl *such that for all PPT adversaries* $\mathcal{A}$ *and all* $\lambda \in \mathbb{N}$*, we have:*

$$\Pr\left[b_1 = 1 \wedge b_2 = 0 \;\middle|\; \begin{array}{c} crs \leftarrow \mathsf{Setup}(1^\lambda) \\ (tx, C, \pi, \mathbf{T}) \leftarrow \mathcal{A}(crs) \\ (\sigma, r) \leftarrow \mathsf{FOp}(C) \\ b_1 := \mathsf{Vfy}(tx, C, \pi) \\ b_2 := \mathsf{LRS.Vf}(tx, \sigma) \end{array}\right] \leq \mathsf{negl}\,(\lambda)\,.$$

3.2 Our VTLRS Construction

We give a construction of VTLRS transaction scheme for the LRS-TS transaction scheme used in Monero.

LRS-TS Construction in Monero. We give a formal description of the LRS-TS scheme deployed in Monero. We do not consider the "confidential transaction" part, and only focus on the signature of the transaction scheme, for conceptual simplicity. The scheme (Fig. 2) is defined over a cyclic group $\mathbb{G}$ of prime order q with generator G and uses two different hash functions $\mathsf{H_P} : \mathbb{G} \to \mathbb{G}, \mathsf{H_S} : \{0,1\}^* \to \mathbb{Z}_q^*$. The private-public key pair is the tuple $(x, G^x) \in \mathbb{Z}_q^* \times \mathbb{G}$. Each secret key is associated with a unique linkability tag that is set as $tag := \mathsf{H_P}(pk)^{sk}$. For ease of understanding we make the assumption that the spending public key is always $pk_{|\mathcal{R}|}$ (as shown in Fig. 2). The spend algorithm samples $(s'_0, s_1, \ldots, s_D) \leftarrow \mathbb{Z}_q^*$ and computes L_0, R_0, h_0 and L_i, R_i, h_i for each index $i \in [D]$ (as shown in Fig. 2). The algorithm finally sets $s_0 := s'_0 - h_D \cdot sk$ and the signature consists of $\sigma := (s_0, s_1, \ldots, s_D, h_0)$. Note that s_0 is reminiscent of how signing is done in Schnorr signatures. The verification algorithm runs the same loop as in the spend algorithm (except that it now ranges over the full ring) to obtain $h_{|\mathcal{R}|}$ and it accepts only if $h_0 = h_{|\mathcal{R}|}$. For brevity, we denote $D = |\mathcal{R}| - 1$.

Throughout the following overview, we describe the VTLRS as an interactive protocol between a committer and a verifier, which can be made non-interactive using the Fiat-Shamir transformation [16]. A formal description of our VTLRS is given in Figs. 3 and 4, where hash function $\mathsf{H}' : \{0,1\}^* \to J$, with J being a set of indices in $[n]$ such that $|J| = t - 1$, is used used to implement the Fiat-Shamir transformation.

High-Level Overview. The commit algorithm proceeds as follows: Consider a signature $\sigma := (s_0, s_1, \ldots, s_D, h_0)$ (where $D = |\mathcal{R}| - 1$) generated by Spend algorithm of Fig. 2 on a transaction $tx := \left(\{pk_i\}_{i=1}^{|\mathcal{R}|}, tag, \mathcal{O}, \mu\right)$. Let $pk_{|\mathcal{R}|}$ be

$\underline{\mathsf{Setup}(1^\lambda, 1^\alpha)}$

$\mathsf{H_P} : \mathbb{G} \to \mathbb{G}$
$\mathsf{H_S} : \{0,1\}^* \to \mathbb{Z}_q^*$
$pp := (\mathbb{G}, q, G, \mathsf{H_P}, \mathsf{H_S})$
return pp

$\underline{\mathsf{OTKGen}(pp)}$

$x \leftarrow \mathbb{Z}_q^*$
$sk := x$
$pk := G^x$
return (pk, sk)

$\underline{\mathsf{TgGen}(sk)}$

$tag := \mathsf{H_P}\left(G^{sk}\right)^{sk}$
return tag

$\underline{\mathsf{Spend}(\mathcal{R}, \mathcal{I}, \mathcal{O}, \mu)}$

$\mathcal{R} := (pk_1, \ldots, pk_{|\mathcal{R}|})$
$\mathcal{I} := (j, sk, tag),\ s.t.$
$\quad j = |\mathcal{R}|$ and $pk_{|\mathcal{R}|} = G^{sk}$
$tx := tx(\mathcal{R}, \mathcal{I}, \mathcal{O}, \mu)$
$(s'_0, s_1, \ldots, s_D) \leftarrow \mathbb{Z}_q^*$
$L_0 := G^{s'_0}, R_0 := \mathsf{H_P}(pk_{|\mathcal{R}|})^{s'_0}$
$h_0 := \mathsf{H_S}(tx||L_0||R_0)$
for $i \in [D]$ **do**
$\quad L_i := G^{s_i} pk_i^{h_{i-1}},$
$\quad R_i := \mathsf{H_P}(pk_i)^{s_i} tag^{h_{i-1}}$
$\quad h_i := \mathsf{H_S}(tx||L_i||R_i)$
endfor
$s_0 := s'_0 - h_D \cdot sk$
$\sigma := (s_0, s_1, \ldots, s_D, h_0)$
return (tx, σ)

$\underline{\mathsf{Vf}(tx, \sigma)}$

$tx := (\mathcal{R}, tag, \mathcal{O}, \mu)$
$\mathcal{R} := (pk_1, \ldots, pk_{|\mathcal{R}|})$
$\sigma := (s_0, \ldots, s_D, h_0)$
set $s_{|\mathcal{R}|} := s_0$
for $i \in [|\mathcal{R}|]$ **do**
$\quad L_i := G^{s_i} pk_i^{h_{i-1}}$
$\quad R_i := \mathsf{H_P}(pk_i)^{s_i} tag^{h_{i-1}}$
$\quad h_i := \mathsf{H_S}(tx||L_i||R_i)$
endfor
return $(h_0 = h_{|\mathcal{R}|})$

Fig. 2. LRS-TS $\Sigma := (\mathsf{Setup}, \mathsf{OTKGen}, \mathsf{TgGen}, \mathsf{Spend}, \mathsf{Vf})$ used in Monero.

the spending key. The commit algorithm takes as input this transaction tx, signature σ and the hiding time $\mathbf{T}$. To generate a VTLRS on transaction tx, the committer secret shares the values in $sc := (s_0, G^{s_0}, \mathsf{H_P}(pk_{|\mathcal{R}|})^{s_0})$ using a t-out-of-n threshold sharing scheme:

1. For the first $t-1$ shares, choose $\alpha_i \in \mathbb{Z}_q$ uniformly at random and set $K_i := G^{\alpha_i}$ and $Y_i := \mathsf{H_P}(pk_{|\mathcal{R}|})^{\alpha_i}$, respectively.
2. For the remaining $n-(t-1)$ shares, use Lagrange interpolation in the exponent, i.e., for $i \in \{t, t+1, \ldots, n\}$ set α_i, K_i, Y_i as

$$\left(s_0 - \sum_{j\in[t-1]} \alpha_j^{\ell_j(0)}\right)^{\ell_i(0)^{-1}}, \left(\frac{G^{s_0}}{\prod_{j\in[t-1]} K_j^{\ell_j(0)}}\right)^{\ell_i(0)^{-1}},$$
$$\left(\frac{\mathsf{H_P}(pk_{|\mathcal{R}|})^{s_0}}{\prod_{j\in[t-1]} Y_j^{\ell_j(0)}}\right)^{\ell_i(0)^{-1}}$$

respectively, where $\ell_i(\cdot)$ is the i-th Lagrange polynomial basis.

The K and Y elements ensure verifiability, that is we can indeed reconstruct (via Lagrange interpolation) the value s_0 that is part of the signature σ from *any* t-sized set of shares of sc.

The committer then computes a time-lock puzzle Z_i (using $\mathsf{LHTLP.PGen}$) with time parameter $\mathbf{T}$ for each share α_i separately. The first message consists of all puzzles $(Z_1, \ldots, Z_n)$ together with $G^{s_0}, \mathsf{H_P}(pk_{|\mathcal{R}|})^{s_0}$ and all (K_i, Y_i) as defined above.

After receiving the above first message, the verifier chooses a random set I of size $(t-1)$ as the challenge set. For this set, the committer opens the time-lock puzzles $\{Z_i\}_{i\in I}$ and reveals the underlying value α_i (together with the corresponding random coins) that it committed to. The verifier wants to ensure that, (i) the puzzles are indeed generated for the correct timing hardness $\mathbf{T}$ and can be successfully solved in that time and (ii) as long as at least one of the shares in the *unopened* puzzles $(\{Z_i\}_{i\in[n]/I})$ is consistent with respect to the corresponding partial commitments (K_i, Y_i), then we can use it to reconstruct s_0 and therefore a valid σ. To do this, the verifier performs the following checks and accepts the commitment as legitimate only if they are all successful: (1) All puzzles $\{Z_i\}_{i\in I}$ are correctly generated using α_i and the corresponding randomness (which was also revealed above) with timing hardness $\mathbf{T}$, (2) All $\{\alpha_i\}_{i\in I}$ are consistent with the corresponding K_i, Y_i, i.e., $K_i = G^{\alpha_i}, Y_i = \mathsf{H_P}(pk_{|\mathcal{R}|})^{\alpha_i}$ and (3) All K_i, Y_i are valid shares of G^{s_0} and $\mathsf{H_P}(pk_{|\mathcal{R}|})^{s_0}$ respectively, i.e., $K_i^{\ell_i(0)} \cdot \prod_{j\in I} K_j^{\ell_j(0)} = G^{s_0}$ and $Y_i^{\ell_i(0)} \cdot \prod_{j\in I} Y_j^{\ell_j(0)} = \mathsf{H_P}(pk_{|\mathcal{R}|})^{s_0}$.

Consequently, to fool a verifier, a malicious prover has to guess the challenge set I ahead of time to pass the above checks without actually committing a valid s_0 (signature σ). Setting the parameters appropriately, we can guarantee that this happens only with negligible probability.

Signature Recovery via Homomorphic Packing. To recover s_0 and the valid signature, the verifier has to solve $\tilde{n} = (n - t + 1)$ puzzles to force the opening of a VTLRS. To close the gap between honest and malicious verifiers, we would like to reduce his workload to the minimal one of solving a single puzzle. To achieve this goal, we use the linearly homomorphic time-lock puzzle construction from [24] or [33] (for a transparent setup), combined with standard packing techniques to compress $\tilde{n}$ puzzles into a single one. Concretely, the verifier, on input $(Z_1, \ldots, Z_{\tilde{n}})$ 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 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 we need to ensure that: (1) The message space of the homomorphic time-lock puzzle must be large enough to accommodate all $\tilde{n}$ signatures and (2) The values α_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 $(\mathcal{P}_{\mathsf{NIZK},\mathcal{L}_{\mathsf{rng}}}, \mathcal{V}_{\mathsf{NIZK},\mathcal{L}_{\mathsf{rng}}})$ for the language $\mathcal{L}_{\mathsf{rng}} := \{(Z, 0, 2^\lambda, \mathbf{T}) : \exists w = (\alpha, r),\ s.t.,\ (Z = \mathsf{LHTLP.PGen}(pp, \alpha; r)) \ \wedge (\alpha \in [0, 2^\lambda])\}$, which certifies that the message of each time-lock puzzles falls into the range $[0, 2^\lambda]$. We instantiate the range proof with the recently introduced protocol [32].

The following theorem states the security of our VTLRS construction and the formal proof is deferred to the full version [35].

Theorem 1 (Timed Privacy, Soundness). *Let* $(\mathsf{Setup}_{\mathsf{NIZK},\mathcal{L}_{\mathsf{rng}}}, \mathcal{P}_{\mathsf{NIZK},\mathcal{L}_{\mathsf{rng}}}, \mathcal{V}_{\mathsf{NIZK},\mathcal{L}_{\mathsf{rng}}})$ *be a NIZK for* $\mathcal{L}_{\mathsf{rng}}$ *and let* LHTLP *be a secure time-lock puzzle with perfect correctness. Then the protocol satisfies timed privacy (Definition 2) and soundness (Definition 3) in the ROM.*

$\mathsf{Setup}(1^\lambda)$

$crs_{\mathsf{rng}} \leftarrow \mathsf{ZK.Setup}(1^\lambda)$

$pp_{\mathsf{LRS}} \leftarrow \mathsf{LRS.Setup}(1^\lambda, 1^\alpha)$

$pp_{\mathsf{LHTLP}} \leftarrow \mathsf{LHTLP.Setup}(1^\lambda, \mathbf{T})$

$crs := (crs_{\mathsf{rng}}, pp_{\mathsf{LRS}}, pp_{\mathsf{LHTLP}})$

return crs

$\mathsf{FOp}(C)$

$C := (\tilde{G}, \tilde{H}, \{s_i\}_{i\in[D]}, h_0, \{Z_i\}_{i\in[n]})$

$\forall_{i\in[n]}\ \alpha_i \leftarrow \mathsf{LHTLP.PSolve}(pp, Z_i)$

$s_0 := \sum_{j\in[t]} (\alpha_j) \cdot \ell_j(0)$

$\sigma := (s_0, \ldots, s_D, h_0)$

return σ

$\mathsf{Op}(C, \{r_i\}_{i\in[n]})$

return $(\sigma, \{r_i\}_{i\in[n]})$

$\mathsf{Com}(\sigma, tx, \mathbf{T})$

$crs := (crs_{\mathsf{rng}}, pp_{\mathsf{LRS}}, pp_{\mathsf{LHTLP}})$

$tx := \left(\{pk_i\}_{i=1}^{|\mathcal{R}|}, tag, \mathcal{O}, \mu\right)$

$\sigma := (s_0, s_1, \ldots, s_D, h_0)$

$\forall i \in [t-1]\ \alpha_i \leftarrow \mathbb{Z}_q^*,$

$K_i := G^{\alpha_i}, Y_i := \mathsf{H_P}(pk_{|\mathcal{R}|})^{\alpha_i}$

for $i \in \{t, \ldots, n\}$ **do**

$$\alpha_i = \left(s_0 - \sum_{j\in[t-1]} \alpha_j \cdot \ell_j(0)\right) \cdot \ell_i(0)^{-1}$$

$$K_i = \left(\frac{G^{s_0}}{\prod_{j\in[t-1]} K_j^{\ell_j(0)}}\right)^{\ell_i(0)^{-1}}$$

$$Y_i = \left(\frac{\mathsf{H_P}(pk_{|\mathcal{R}|})^{s_0}}{\prod_{j\in[t-1]} Y_j^{\ell_j(0)}}\right)^{\ell_i(0)^{-1}}$$

endfor

for $i \in [n]$ **do**

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

$\pi_{\mathsf{rng},i} \leftarrow \mathcal{P}_{\mathsf{NIZK},\mathcal{L}_{\mathsf{rng}}}(crs_{\mathsf{rng}}, (Z_i, 0, 2^\lambda, \mathbf{T}), (\alpha_i, r_i))$

endfor

$I \leftarrow \mathsf{H}'\big(G^{s_0}, \mathsf{H_P}(pk_{|\mathcal{R}|})^{s_0}, \{(K_i, Y_i, Z_i, \pi_{\mathsf{rng},i})\}_{i\in[n]}\big)$

$C := (G^{s_0}, \mathsf{H_P}(pk_{|\mathcal{R}|})^{s_0}, \{s_i\}_{i\in[D]}, h_0, \{Z_i\}_{i\in[n]})$

$\pi := (\{K_i, Y_i, \pi_{\mathsf{rng},i}\}_{i\in[n]}, I, \{\alpha_i, r_i\}_{i\in I})$

return (C, π)

Fig. 3. Algorithms of our VTLRS Scheme. Here $D = |\mathcal{R}| - 1$.

Instantiating LHTLP**.** We can instantiate the LHTLP with the RSA based construction from [24] or the class group based construction from [33]. In the

former we can let the committer run the setup while additionally proving the well-formedness, and in the latter we have a transparent setup which can be run by any party. Notice that one-time setup for VTLRS other than LHTLP, can be instantiated in the ROM without any trust.

Batch Force-Opening of VTLRS Commitments. We observe that assuming (i) a large enough message space of the time-lock puzzles and (ii) global public parameters pp, one can batch the solutions of different puzzles into a single one using known constructions from [24]. This can be done by homomorphically packing the messages in each puzzle into a single puzzle with a linear function $f(x_1, \ldots, x_{\tilde{n}}) = \sum_{i=1}^{\tilde{n}} 2^{(i-1)\cdot\lambda} \cdot x_i$ as discussed before. Therefore opening a VTLRS commitment involves solving a single puzzle, and infact, we can potentially open many VTLRS commitments by solving a single puzzle.

$\mathsf{Vfy}(tx, C, \pi)$

$crs := (crs_{\mathsf{rng}}, pp_{\mathsf{LRS}}, pp_{\mathsf{LHTLP}}),\ tx := \left(\{pk_i\}_{i=1}^{|\mathcal{R}|}, tag, \mathcal{O}, \mu\right)$

$C := (\tilde{G}, \tilde{H}, \{s_i\}_{i\in[D]}, h_0, \{Z_i\}_{i\in[n]})$

$\pi := (\{K_i, Y_i, \pi_{\mathsf{rng},i}\}_{i\in[n]}, I, \{\alpha_i, r_i\}_{i\in I})$

$\forall i \in [D],\quad L_i := G^{s_i} pk_i^{h_{i-1}},\ R_i := \mathsf{H_P}(pk_i)^{s_i}\, tag^{h_{i-1}}, h_i := \mathsf{H_S}(tx||L_i||R_i)$

$L_{|\mathcal{R}|} := \tilde{G} \cdot pk_{|\mathcal{R}|}^{h_D},\quad R_{|\mathcal{R}|} := \tilde{H} \cdot tag^{h_D},\ h_{|\mathcal{R}|} := \mathsf{H_S}(tx||L_{|\mathcal{R}|}||R_{|\mathcal{R}|})$

$b_1 := (h_0 \neq h_{|\mathcal{R}|})$

$b_2 := \exists j \notin I \left(K_j^{\ell_j(0)} \cdot \prod_{i\in I} K_i^{\ell_i(0)} \neq \tilde{G}\right)$

$b_3 := \exists j \notin I \left(Y_j^{\ell_j(0)} \cdot \prod_{i\in I} Y_i^{\ell_i(0)} \neq \tilde{H}\right)$

$b_4 := \exists i \in [n] \left(\mathcal{V}_{\mathsf{NIZK},\mathcal{L}_{\mathsf{rng}}}(crs_{\mathsf{rng}}, (Z_i, 0, 2^\lambda, \mathbf{T}), \pi_{\mathsf{rng},i}) \neq 1\right)$

$b_5 := \exists i \in I\, (Z_i \neq \mathsf{LHTLP.PGen}(pp, \alpha_i; r_i))$

$b_6 := \exists i \in I\ (K_i \neq G^{\alpha_i})$

$b_7 := \exists i \in I \left(Y_i \neq \mathsf{H_P}(pk_{|\mathcal{R}|})^{\alpha_i}\right)$

$b_8 := \left(I \neq \mathsf{H}'\left(\tilde{G}, \tilde{H}, \{(K_i, Y_i, Z_i, \pi_{\mathsf{rng},i})\}_{i\in[n]}\right)\right)$

if $\bigvee_{i\in[8]} b_i = 1$ **thenreturn** 0 **else return** 1

Fig. 4. Verification algorithm of our VTLRS scheme. Here $D = |\mathcal{R}| - 1$.

4 Benchmarking

We implement prototypes of our VTLRS construction and PayMo. We build our VTLRS prototype with rust using the curve25519-dalek [8] library with security parameter $\lambda = 128$. All measurements were done on on a single CPU core of an AWS t2 micro instance for easier comparison with the following specifications: 1 core of a Intel Xeon E5-2676 v3 @ 2.40 Ghz, 1 GB of RAM, Ubuntu Linux 18.04.2 LTS (4.15.0-1045-aws) and rust 1.41. Our measurements are reported as a median over 1000 executions.

All hashing operations ($\mathsf{H_S}, \mathsf{H_P}, \mathsf{H}$ and H') are implemented using SHA-512 or the Keccak variant used in Monero. For all Elliptic Curve Operations the Curve25519 implementation from curve25519-dalek [8] in Ristretto form [9] was used. We implemented the NIZK proof from [12] for $\mathcal{L}_{\mathsf{eqdl}}$ and the NIZK proof for $\mathcal{L}_{\mathsf{rng}}$ from [32]. The prover and verifier times for the NIZK proof for $\mathcal{L}_{\mathsf{eqdl}}$ is 0.079 ms and 0.143 ms, respectively. For the NIZK for $\mathcal{L}_{\mathsf{rng}}$, for statistical soundness parameter $k = 64$, the prover and verifier times are 258.66 ms and 289.56 ms, respectively. We implemented the LHTLP construction [24] for time-lock puzzles, with a 1024 bit RSA modulus N. In our benchmark, the time taken for PSetup (including prime generation) is 730.43 ms, and the time taken for PGen is 3.557 ms. And the time taken by PSolve for timing hardness $\mathbf{T} := 1024, 2048$ and 4096 is 2.708 ms, 4.070 ms and 6.795 ms, respectively.

VTLRS. We evaluated our VTLRS construction by setting the parameters as $n = 80$ and $t = 40$ (probability of adversary breaking soundness is 9.3×10^{-24}). Our results show that Com and Vfy algorithms of our VTLRS construction take 586.76 ms and 467.84 ms in CPU time, respectively. We implemented the LRS transaction scheme of Monero with a ring size of 10 keys (which is the common size used in Monero today [4]), with one spending key and one recipient.

Evaluation of PayMo. We consider two different measurements: (i) Only the computation operations and not the cost of serialisation and network transmission in PayMo. This shows the performance of the protocol on the sender and receiver side of a PayMo channel. (ii) Total time taken by operations including network operations and latency. To show the impact of network transmission in this measurement, two settings with different network latency are considered. We focus only on LRS-TS of Monero and omit other confidential transactions related operations. A possible future work is to have a complete prototype that is executable on the Monero network.

We consider Alice and Bob who share a payment channel. To evaluate the performance of PayMo, we measure the computation time of both users during the channel opening and payment phase. We also measure the total time taken for PayMo operations that includes network latency between parties. Our results from Table 1 show that the time taken for finishing a single payment is less a third of second even under high latency scenarios.

Table 1. PayMo operations for Alice and Bob excluding network overhead. PC operations for both parties including network latency, with low latency setup **S1** (Alice and Bob) and high latency setup **S2** (Alice and Bob) with Round Trip Times between the two users of 0.3 ms and 144 ms, respectively. All measurements are reported in milliseconds (ms).

	Alice	**Bob**	Setup **S1**	Setup **S2**
Joint key/Tag generation	0.13	0.31	1.85	440.7
PC opening	468.1	588.4	1060	1351
PC payment	1.30	1.28	3.61	297.9

Interpretation. Our results from Table 1 show that by exploiting parallel request processing, the receiver of one or more channel(s) can process around 780 payments per second per CPU core, while the sender of one of more channel(s) can process around 770 payments per second per CPU core. The parties can scale up their processing power if they spawn more PC nodes (or cores) as done in the Lightning Network.

For instance, for a payment service provider (recipient) who has payment channels with several users, it can accept more than 93600 payments per CPU core over a span of 2 min (average block production rate in Monero), from users with PayMo channels with the service provider. In this case, only the receiver's CPU time for payments is considered, excluding the overhead for serialization and network.

Alice and Bob can process close to 93500 payments per CPU core (with acknowledgement of payment) over a span of 2 min even with a round trip latency time of 144 ms per message. This is because during message transmission, parties do not stay idle but instead spawn new payments in parallel. In case the parties only make sequential payments, Alice can still make more than 400 payments over the span of 2 min.

5 Conclusions

We presented verifaible timed linkable ring signatures a new cryptographic tool and PayMo, which is the first payment channel protocol that is fully compatible with Monero, the largest privacy-preserving cryptocurrency. Our results show an increase in the transaction throughput of several orders of magnitudes when compared with the current implementation of Monero. As an exciting next step, we plan to work on large scale adoption of PayMo in Monero.

Acknowledgements. The work was in part supported by THE DAVID AND LUCILLE PACKARD FOUNDATION - Award #202071730, SRI INTERNATIONAL - Award #53978 / Prime: DEFENSE ADVANCED RESEARCH PROJECTS AGENCY - Award #HR00110C0086 and NATIONAL SCIENCE FOUNDATION - Award #2212746. This work is also partially supported by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) as part of the Research

and Training Group 2475 "Cybercrime and Forensic Computing" (grant number 393541319/GRK2475/1-2019), and by the grant 442893093, and by the state of Bavaria at the Nuremberg Campus of Technology (NCT). NCT is a research cooperation between the Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU) and the Technische Hochschule Nürnberg Georg Simon Ohm (THN).

A Preliminaries

Time-Lock Puzzles. Time-lock puzzles [28] allow one to conceal a secret for a certain amount of time $\mathbf{T}$. *Homomorphic Time-Lock Puzzles (HTLPs)* [24] allow one to perform homomorphic computation on honestly generated puzzles. It consists of a setup algorithm (PSetup), that takes as input a time hardness parameter $\mathbf{T}$ and outputs public parameters of the system pp, a puzzle generation algorithm (PGen) that, on input a message, generates the corresponding puzzle. One can then evaluate homomorphically functions over encrypted messages (PEval) and solve the resulting puzzle in time $\mathbf{T}$ (PSolve). The security requirement is that for every PRAM adversary $\mathcal{A}$ of running time $\leq \mathbf{T}^{\varepsilon}(\lambda)$ the messages encrypted are computationally hidden. Malavolta and Thyagarajan [24] show an efficient construction that is linearly homomorphic over the ring $\mathbb{Z}_{N^s}$, where N is an RSA modulus and s is any positive integer. The scheme is perfectly correct and is secure under the sequential squaring assumption [28].

Non-interactive Zero-Knowledge. Let $R : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$ be an NP relation with corresponding NP-language $\mathcal{L} := \{stmt : \exists w \; s.t. \; R(stmt, w) = 1\}$. A non-interactive zero-knowledge proof (NIZK) [13] system for $\mathcal{L}$ is initialized with a setup algorithm $\mathsf{Setup}(1^\lambda)$ that outputs a common reference string crs. A prover can show the validity of a statement $stmt$ with a witness w by invoking $\mathcal{P}_{\mathsf{NIZK},\mathcal{L}}(crs, stmt, w)$, which outputs a proof π. The proof π can be efficiently checked by the verification algorithm $\mathcal{V}_{\mathsf{NIZK},\mathcal{L}}(crs, stmt, \pi)$. A NIZK proof for language $\mathcal{L}$ is simulation extractable if one can extract a valid w from adversarially generated proofs, even if the adversary sees arbitrarily many simulated proofs. A NIZK must also be zero knowledge in the sense that nothing beyond the validity of the statement is leaked to the verifier.

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 [31] proposed a threshold secret sharing scheme where the sharing algorithm takes a secret $s \in \mathbb{Z}_q$ and generates shares $(s_1, \ldots, s_n)$ each in $\mathbb{Z}_q$. The reconstruction algorithm takes as input at least t shares and outputs a secret s. The security demands that knowing only a set of shares smaller than the threshold size does *not*reveal any information about s.

B Transaction Scheme of Monero

We review the basic definitions of Linkable Ring Signatures (LRS) following Lai et al. [20]. In contrast to their work, our definitions do not consider the "confidential transaction" part, and only focus on the signature of the transaction scheme, for conceptual simplicity.

B.1 Definition

A ring signature [29] scheme allows to sign messages such that the signer is anonymous within a set a possible signers, called the ring. The members associated to the ring are chosen "on-the-fly" by the signer using their public-keys. Linkability [21] means that anonymity is retained unless the same user signing key is used to sign twice. This is achieved by associating a unique linkability tag to each signing key that is revealed while generating a signature.

In a transaction scheme, we have a block of data referred to as a transaction, that determines the amount of coins transferred from one user address (source) to another user address (target) and it is accompanied by an authentication token (signature) of the sending user. Since the sending user is represented through the source address in the transaction, the signature is checked for validity with respect to the source account. Combining linkable ring signatures and a transaction scheme, we have a linkable ring signature based transaction scheme (LRS-TS), where the message signed is the transaction which consists of: A ring of addresses (LRS public keys) and their associated coins (out of which one of the addresses is the source account), and one or more target addresses. The authentication token of the transaction is a linkable ring signature on the transaction (as message), with the ring of addresses as the ring, and the secret authentication key of the source address as the signing key of the linkable ring signature scheme. To prevent leakage of the source address it is assumed that each address in the ring of addresses have the same amount of associated coins[3].

Definition 4. *A* Linkable Ring Signature (LRS) *transaction scheme* Σ *consists of the* PPT *algorithms* $(\mathsf{Setup}, \mathsf{OTKGen}, \mathsf{TgGen}, \mathsf{Spend}, \mathsf{Vf})$ *which are defined as follows:*
$pp \leftarrow \mathsf{Setup}(1^\lambda)$*: outputs the public parameter* pp*.*
$(pk, sk) \leftarrow \mathsf{OTKGen}(pp)$*: The* one-time key generation *algorithm outputs a public-secret key-pair* (pk, sk)*.*
$tag \leftarrow \mathsf{TgGen}(sk)$*: The* tag-generation *algorithm takes as input a secret key* sk*. It outputs a tag* tag*.*
$(tx, \sigma) \leftarrow \mathsf{Spend}(\mathcal{R}, \mathcal{I}, \mathcal{O}, \mu)$*: The* spend *algorithm takes as input a set* $\mathcal{R}$ *of public keys with each key associated with* c *coins, a tuple* $I = (j, sk, tag)$ *consisting of an index* j*, a secret key* sk*, and a tag* tag*, a set* $\mathcal{O}$ *consisting of target public keys and some metadata* μ*. It outputs a transaction* $tx := (\mathcal{R}, tag, \mathcal{O}, \mu)$ *and a signature* σ*.*
$b \leftarrow \mathsf{Vf}(tx, \sigma)$*: The* verify *algorithm inputs a transaction* tx *and a signature* σ*. It outputs a bit* b *denoting the validity of* σ*.*

Security. We have three properties of LRS-TS, namely (1) *Privacy*: LRS-TS should ensure privacy of the source account, meaning an adversarial observer on the blockchain should not learn any information about the source address from a transaction other than the fact that it is a member of the ring of one-time

[3] This assumption can be relaxed with the use of confidential transactions [25] where an account's associated amount is hidden using commitments.

addresses, (2) *Non-Slanderability (Unforgeability)*: LRS-TS must ensure that an adversarial user cannot steal the coins of an honest user (unforgeability) or spend coins on behalf of an honest user (non-slanderability), and (3) *Linkability*: LRS-TS must ensure that an adversary cannot double spend his coins and any such attempts must be linkable. We refer the reader to [35] for the formal definitions.

References

1. https://www.blockchain.com/en/charts/transactions-per-second
2. https://bitinfocharts.com/comparison/transactionfees-btc-xmr.html
3. https://tinyurl.com/sujsu369
4. https://moneroblocks.info/block/2047966
5. Lightning Network. https://lightning.network/
6. Payment Channels in Ripple. https://xrpl.org/use-payment-channels.html
7. Raiden Network. https://raiden.network/
8. curve25519-dalek (2019). https://tinyurl.com/rb3pnfvm
9. Arcieri, T., de Valence, H., Lovecruft, I.: The ristretto group (2019). https://ristretto.group/ristretto.html
10. Aumayr, L., Thyagarajan, S.A., Malavolta, G., Monero-Sánchez, P., Maffei, M.: Sleepy channels: bitcoin-compatible bi-directional payment channels without watchtowers (2021). (To Appear at ACM CCS 2022)
11. Ben-Sasson, E., et al.: Zerocash: decentralized anonymous payments from bitcoin. In: 2014 S&P, pp. 459–474. IEEE (2014). https://doi.org/10.1109/SP.2014.36
12. Camenisch, J., Stadler, M.: Proof systems for general statements about discrete logarithms. Technical report/Department of Computer Science, ETH Zürich 260 (1997)
13. De Santis, A., Micali, S., Persiano, G.: Non-interactive zero-knowledge proof systems. In: Pomerance, C. (ed.) CRYPTO 1987. LNCS, vol. 293, pp. 52–72. Springer, Heidelberg (1988). https://doi.org/10.1007/3-540-48184-2_5
14. Dziembowski, S., Eckey, L., Faust, S., Malinowski, D.: Perun: virtual payment hubs over cryptocurrencies. In: 2019 S&P, pp. 106–123. IEEE (2019). https://doi.org/10.1109/SP.2019.00020
15. Egger, C., Moreno-Sanchez, P., Maffei, M.: Atomic multi-channel updates with constant collateral in bitcoin-compatible payment-channel networks. In: ACM CCS 2019, pp. 801–815. ACM Press (2019). https://doi.org/10.1145/3319535.3345666
16. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194 (1987). https://doi.org/10.1007/3-540-47721-7_12
17. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct NIZKs without PCPs. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 626–645. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38348-9_37
18. Green, M., Miers, I.: Bolt: anonymous payment channels for decentralized currencies. In: ACM CCS 2017, pp. 473–489. ACM Press (2017). https://doi.org/10.1145/3133956.3134093
19. Gugger, J.: Bitcoin-monero cross-chain atomic swap. Cryptology ePrint Archive, Report 2020/1126 (2020). https://eprint.iacr.org/2020/1126
20. Lai, R.W.F., Ronge, V., Ruffing, T., Schröder, D., Thyagarajan, S.A.K., Wang, J.: Omniring: scaling private payments without trusted setup. In: ACM CCS 2019, pp. 31–48. ACM Press (2019). https://doi.org/10.1145/3319535.3345655

21. Liu, J.K., Wei, V.K., Wong, D.S.: Linkable spontaneous anonymous group signature for Ad Hoc groups. In: Wang, H., Pieprzyk, J., Varadharajan, V. (eds.) ACISP 2004. LNCS, vol. 3108, pp. 325–335. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-27800-9_28
22. Malavolta, G., Moreno-Sanchez, P., Kate, A., Maffei, M., Ravi, S.: Concurrency and privacy with payment-channel networks. In: ACM CCS 2017, pp. 455–471. ACM Press (2017). https://doi.org/10.1145/3133956.3134096
23. Malavolta, G., Moreno-Sanchez, P., Schneidewind, C., Kate, A., Maffei, M.: Anonymous multi-hop locks for blockchain scalability and interoperability. In: NDSS 2019. ISOC (2019)
24. Malavolta, G., Thyagarajan, S.A.K.: Homomorphic time-lock puzzles and applications. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11692, pp. 620–649. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26948-7_22
25. Maxwell, G.: Confidential transactions (2015). https://people.xiph.org/~greg/confidential_values.txt
26. Moreno-Sanchez, P., Le, D.V., Noether, S., Goodell, B., Kate, A.: Dlsag: non-interactive refund transactions for interoperable payment channels in Monero. Tech. rep., Cryptology ePrint Archive, Report 2019/595 (2019)
27. Poon, J., Dryja, T.: The bitcoin lightning network: Scalable off-chain instant payments (2016)
28. Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock puzzles and timed-release crypto. Tech. rep. (1996)
29. Rivest, R.L., Shamir, A., Tauman, Y.: How to leak a secret. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 552–565. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45682-1_32
30. van Saberhagen, N.: Cryptonote v 2.0 (2013)
31. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (1979)
32. Thyagarajan, S.A.K., Bhat, A., Malavolta, G., Döttling, N., Kate, A., Schröder, D.: Verifiable Timed Signatures Made Practical, CCS 2020. Association for Computing Machinery (2020)
33. Thyagarajan, S.A.K., Castagnos, G., Laguillaumie, F., Malavolta, G.: Efficient CCA Timed Commitments in Class Groups. ACM CCS (2021)
34. Thyagarajan, S.A.K., Gong, T., Bhat, A., Kate, A., Schröder, D.: Opensquare: Decentralized Repeated Modular Squaring Service, CCS 2021 (2021)
35. Thyagarajan, S.A.K., Malavolta, G., Schmidt, F., Schröder, D.: Paymo: payment channels for monero. Cryptology ePrint Archive, Report 2020/1441 (2020)
36. Yuen, T.H., et al.: RingCT 3.0 for blockchain confidential transaction: shorter size and stronger security. Cryptology ePrint Archive, Report 2019/508 (2019). https://eprint.iacr.org/2019/508