Lockable Signatures for Blockchains: Scriptless Scripts for All Signatures

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

Giulio Malavolta
Max Planck Institute for Security and Privacy
giulio.malavolta@hotmail.it

2021 IEEE Symposium on Security and Privacy (SP) | 978-1-7281-8934-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/SP40001.2021.00065

***Abstract*—Payment Channel Networks (PCNs) have given a huge boost to the scalability of blockchain-based cryptocurrencies: Beyond improving the transaction rate, PCNs enabled cheap cross-currency payments and atomic swaps. However, current PCNs proposals either heavily rely on special scripting features of the underlying blockchain (e.g. Hash Time Lock Contracts) or are tailored to a handful of digital signature schemes, such as Schnorr or ECDSA signatures. This leaves us in an unsatisfactory situation where many currencies that are being actively developed and use different signature schemes cannot enjoy the benefits of a PCN.**

In this work, we investigate whether we can construct PCNs assuming the minimal ability of a blockchain to verify a digital signature, for *any signature scheme*. In answering this question in the affirmative, we introduce the notion of *lockable signatures*, which constitutes the cornerstone of our PCN protocols. Our approach is generic and the PCN protocol is compatible with any digital signature scheme, thus inheriting all favorable properties of the underlying scheme that are not offered by Schnorr/ECDSA (e.g. aggregatable signatures or post-quantum security).

While the usage of generic cryptographic machinery makes our generic protocol impractical, we view it as an important feasibility result as it may serve as the basis for constructing optimized protocols for specific signature schemes. To substantiate this claim, we design a highly efficient PCN protocol for the special case of Boneh-Lynn-Shacham (BLS) signatures. BLS signatures enjoy many unique features that make it a viable candidate for a blockchain, e.g. short, unique, and aggregatable signatures. Yet, prior to our work, no PCN was known to be compatible with it (without requiring an advanced scripting language). The cost of our PCN is dominated by a handful of calls to the BLS algorithms. Our concrete evaluation of these basic operations shows that users with commodity hardware can process payments with minimal overhead.

***Index Terms*—Payment Channel Networks, Scriptless Scripts, BLS Signatures**

I. Introduction

Decentralized payments have become a reality with the advent of blockchain based cryptocurrencies like Bitcoin, Ethereum, etc. Users can now make payments among each other without relying on a trusted central bank, but instead rely on a public distributed ledger and a transaction scheme that helps determine the validity of a payment publicly. However, a major drawback of these cryptocurrencies is their low scalability due to a low transaction rate. Traditional payment services like Visa, Mastercard, etc, support order of 3 magnitude more payments than even the most prominent cryptocurrencies like Bitcoin, Ethereum, etc. This is because, a large amount of payments in these currencies are fighting for a few spots on the blockchain, where they are validated and used to update the state of the system.

Off-chain payments were proposed as a solution to this problem where users no longer need to register every payment on the blockchain. Instead, they can make several payments locally and only register their final balances on the chain. *Payment Channels* (PC) [1] and its generalization *Payment Channel Networks* (PCN) [1]–[4] are such off-chain (or layer 2) payment solutions that help improve the scalability of these cryptocurrencies. In a PC protocol, two users Alice and Bob open a payment channel by posting a single transaction on the blockchain. This transaction locks some amount of coins in this channel, referred to as *channel capacity*. Alice and Bob can then make several payments locally by adjusting the balance of the channel, without registering any of them on the chain. Once the payment phase ends, a closing transaction that finalises the balances of Alice and Bob in the channel is posted on the chain, thus closing the channel. Given several payments can be made at the price of only two transactions on the chain, this is a dramatic boost to the number of payments when the number of on-chain payments is limited. Consequently, PCs have been practically deployed in major currencies like Bitcoin [5], Ethereum [6], and Ripple [7].

Payment Channel Networks. A PCN generalizes the notion of a PC and allows payments between users Alice and Bob that do not directly share a PC, through a chain of intermediate users that connects Alice with Bob, referred to as *payment path*. PCNs also enable payments across different currencies in a secure way: As an example, PCNs can be used to perform atomic swaps of coins in different currencies. The first proposals of PCNs were based on a special scripting feature of the blockchain, called *Hash Time-Lock Contracts* (HTLCs) [1], [2]: parties setup payments along the payment path such that each of these payments are successful if a pre-image of a hash value is released before some time t. However, this approach has several major limitations:

1) It suffers from *wormhole attacks* [3], where the adversary can "skip" an honest intermediate user in the payment path, thereby stealing his PCN participation fee.

2) It is compatible only with those blockchains that support advanced scripts and several existing currencies like Monero [8], Mimblewimble [9], Ripple [10], or Zcash [11] (shielded

DOI 10.1109/SP40001.2021.00065

addresses) are therefore left out.

3) It results in larger transactions and consequently larger ledgers.

4) It lacks *on-chain privacy* as PCN transactions of a single payment are linkable with each other,

5) Finally, PCN transactions are clearly distinguishable from standard payments one-to-one payments, thus hampering the *fungibility* of the coins.

To amend this, Malavolta et al. [3] proposed PCN protocols without relying on any special scripts, also referred to as *scriptless scripts* [12], that simultaneously solved all the above issues. They presented two protocols exploiting the algebraic structure of Schnorr and ECDSA signatures. More recently, those techniques have been adapted to the lattice settings [13], barring a few limitations in terms of on-chain privacy, as discussed in [14].

Unfortunately, the techniques from [3] do not seem to extend to PCN protocols for other popular signature schemes that are of great interest to the cryptocurrency space as discussed below. For instance, recently there has been a lot of interest in developing new currencies or adapting existing currencies [15], [16], with transaction schemes that use BLS signatures [17] and its variants [18], that offer useful properties, such as key and signature aggregation, which have the potential to drastically reduce the size of the ledger. Furthermore, the looming threat of quantum computers has motivated a large body of research in lattice-based [13], [19]–[21] and hash-based [22] signatures, and subsequent development of new currencies based on these signatures [23]–[26]. It is plausible to foresee a near future where all blockchains will have to move to a signature scheme with post-quantum security.

Therefore, to enjoy all the benefits of PCNs based on scriptless scripts in all existing (and possibly future) cryptocurrencies, we require a PCN protocol that does not rely on advanced scripts and can handle any signature scheme. This motivates us to ask the following question:

Can we construct a PCN assuming only the bare minimal ability of a blockchain to verify a signature?

In this work we give a positive answer to the above question, for *any signature scheme*. Before delving into the description of our approach, we explain the challenges that arise when trying to construct PCN generically, without assuming any algebraic structure of the signature scheme.

A. Challenges in Constructing PCNs

Let us recall the high-level idea of the PCN protocol from [3]. They consider a setting with n parties $P_1, \ldots, P_n$ where P_1 is the sender and P_n is the receiver. Neighbouring parties P_i and P_{i+1} (for $i \in \{1, \ldots, n-1\}$) in this path have open channels between them. Their PCN protocol consists of two phases: A *lock* and a *release* phase. To make a payment of v coins, P_1, together with each pair of intermediate users, sets up a payment *lock* as follows: For $i \in \{1, \ldots, n-1\}$, P_i and P_{i+1} set up a payment transaction denoted by $tx_{\mathsf{pay},i}$ paying v coins from their channel to P_{i+1}, along with a partial signature $\bar{\sigma}_i$ on this transaction. The partial signature $\bar{\sigma}_i$ is not a valid signature on $tx_{\mathsf{pay},i}$ but can be transformed into a valid signature σ_i provided that a certain release information k_i is available. Generating partial signatures requires exploiting the algebraic structure of Schnorr or ECDSA signatures, where the sender "injects" some randomness in the signing process that causes the signature to be partial or incomplete. Things are set in such a way that if P_{i+1} obtains σ_{i+1}, he can locally recompute k_i and learn the valid signature σ_i.

Once the payment locks are setup until party P_n, the sender P_1 initiates the release phase by revealing the release information for the n-th partial signature to P_n. A cascade of payment release happens until party P_2, which concludes one payment of v coins from P_1 to P_n. There are two main properties that are satisfied by this locking mechanism:

1) *Atomicity*: Even if P_i and P_{i+1} collude, they cannot transform the lock $\bar{\sigma}_i$ into the valid signature σ_i without the help of the sender (P_1). They may of course generate a new valid signature $\sigma_i' \neq \sigma_i$, but only σ_i can help release the $(i-1)$-th lock. Therefore no two colluding intermediate neighbours P_i and P_{i+1} can initiate the payment release ahead of time. This assures the sender that no intermediate user P_{i+1} can steal his v coins, that were intended for P_n.

2) *Balance Security*: Even if all parties but P_i are malicious, P_i is guaranteed that a valid signature on a transaction $tx_{\mathsf{pay},i+1}$ will always allow it to recover a valid signature on $tx_{\mathsf{pay},i}$. This is to ensure that intermediate users never lose coins. The protocol enforces this by having intermediate users checking whether the randomness injected by the sender is *consistent* across different instances of the partial signing protocol.

To ensure both of the properties above, their protocol heavily relies on the ability of the sender to inject consistent randomness in the signing protocol, which in turn leverages the special algebraic structure of Schnorr/ECDSA signatures.

Failed Attempts. We illustrate the pitfalls of extending this approach with the case of BLS signatures, where the signatures are *unique* for a given message under a given key. In this case, two colluding intermediate users P_i and P_{i+1} can always collude to generate the *one and only* valid σ_i on $tx_{\mathsf{pay},i}$. This means P_i and P_{i+1} can release the lock at any time and effectively steal v coins from P_1. Similar problems persist for non-unique signature where there is no way for a third party (in our case the sender) to meaningfully and consistently inject some randomness in the signing process to make the signatures *partial* and at the same time *verifiable*.

One could think of setting up the locks backwards (starting from P_n until P_2) thereby preventing colluding users from initiating the release phase like above. However, such a backwards lock setup strategy suffers from a different issue: Assume that locks up to the i-th one have been setup starting from the n-th lock. If user P_{n-1} and P_n collude and initiate a release phase, the cascade follows until the i-th lock which results in party P_i paying P_{i+1} via $tx_{\mathsf{pay},i}$. If the sender P_1 aborts the lock phase now, an honest intermediate P_i loses v coins and has effectively paid the receiver P_n.

A Fairness Problem. In general, the source of difficulty in designing PCNs seems to be rooted in enforcing *fairness* (a weaker form of guaranteed output delivery) in the locking protocol: We want to ensure that either all parties $P_1, \ldots, P_n$ learn a valid signature on the corresponding transaction, or none does. This makes it especially tricky to design general purpose solution for this problem, as fairness is notoriously difficult to achieve. Even using powerful cryptographic tools, such as general-purpose multi-party computation (MPC), does not seem to trivialize the problem since fairness for MPC is, in general, impossible to achieve [27]. The aim of this work is to evade this fairness barrier and to construct a PCN protocol that is compatible with any signature scheme.

B. Our Contribution

The contributions of this work are summarized as follows.

Generic Solution. We construct the first PCN protocol (Section V-B) that is compatible with *any* signature scheme. The signature scheme shall satisfy the notion of strong Existential Unforgeability under Chosen Message Attacks (sEUF-CMA), which is the de-facto definition of security for digital signatures. Our scheme does not assume any scripting language for the underlying blockchain, besides the ability to verify a signature and the timestamp of a transaction. This enables, in principle, interoperable payments across chains with any signature scheme (unique, aggregatable, post-quantum secure, etc.). Our PCN protocol also guarantees *on-chain privacy* for the users in the payment path of the PCN, that is on par with the state of the art proposals [2], [3]. To achieve this, we introduce and construct *lockable signatures* (Section IV), a new cryptographic tool which may be independent interest.

We wish to clarify that we view our generic protocol as an initial feasibility result and it may not be efficient enough to be deployed in practice, due to the high costs associated with the cryptographic machinery that are used. Apart from the computational overhead, the price to pay for such a generality is that of a slight increase in the number of on-chain transactions per PC. Specifically, in the worst case, parties involved in a payment in our PCN need to post 2 transactions on the chain to close their channels, instead of 1 as in prior works. Nevertheless, we believe that our approach sheds light on the necessary assumptions needed to construct PCNs and can serve as the blueprint to design efficient protocols specific to many signature schemes.

Efficient Protocol for BLS. Following the blueprint of our generic protocol, we construct the first practically efficient PCN protocol fully compatible with the BLS signature scheme (Section V-C), that has unique and aggregatable signatures. Additionally, our BLS-based protocol also inherits the security and privacy properties of our generic protocol. Prior to our work, supporting PCNs for BLS signatures required complex scripts like HTLCs whose disadvantages were discussed above.

Our protocol makes only a handful of calls to the basic BLS algorithms, and does not require any heavy cryptographic machinery. Our efficiency analysis (Section VI) shows that the cost incurred by PCN users is minimal and the protocol can be run on today's low-end devices.

C. Related Work

As off-chain scalability solutions, Payment Channels and Payment Channel Networks [1]–[4] have been proposed and extensively studied. Typical proposals [1], [2] use special scripts like HTLCs, that let a user get paid if he produces a pre-image of a certain hash value before a specific time (payment expiry time). While these protocols are also "universal" (i.e. are compatible with any signature scheme), they fall short in achieving the properties of scriptless scripts, due to their reliance on special scripts and the syntactic difference between standard an PCN transactions. Malavolta et al. [3] propose a PCN protocol that does not rely on HTLC and offers better on-chain privacy using a new tool called Anonymous Multi Hop Locks (AMHL). However, their protocol is tailored for transaction schemes that use Schnorr and ECDSA signatures. Similar techniques were used by Esgin, Esroy and Erkin [13] for a specific lattice based signature scheme, which is a variant of [21]. Egger, Moreno-Sanchez and Maffei [4] propose a PCN protocol compatible with a wide-variety of network topologies. Unfortunately, they lack on-chain privacy of the parties involved and also have increased number of transactions that go on-chain per party in the worst case. As a result, in their proposal it is easy to differentiate between PCN and non-PCN transactions, which heavily affects the fungibility of the coins. Bolt [28] is a payment channel protocol specially tailored for Zcash [11] which uses zk-SNARKs [29]–[31]. A generalisation of PC with complex conditional payments is a *state channel* [32]–[34] that requires highly expressive scripting functionalities from the underlying blockchain (like Ethereum) and are therefore not *scriptless*. Other works in these settings study the notion of non-source routing, where the payment is routed locally by each intermediate node [35]. In contrast, our approach is inherently tied to source routing.

Comparison with Adaptor Signatures. A related (stronger) notion to lockable signatures is that of *adaptor signatures* [36]. Similarly to lockable signatures, adaptor signatures allow one to compute a pre-signature (the analogue of the lock) with respect to some NP-relation, and the witness of such a relation can be then used to recover the full (valid) signature. However, the crucial difference is that the pre-signature can be computed *without* knowing the witness of the given relation, which makes adaptor signatures a more versatile primitive then lockable signatures. On the flip side, adaptor signatures do not seem to be easily realizable for signature schemes without algebraic structure.

In summary, none of these proposals can simultaneously (i) generically handle all signature schemes without using special scripts, (ii) guarantee on-chain privacy and (iii) improve fungibility of the coins, as they either rely on specific signature schemes or leak information on-chain about the parties and the payments made in the PCN.

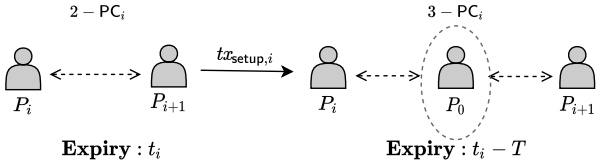

Fig. 1. Payment Setup phase of our PCN protocol. Transaction $tx_{\mathsf{Setup},i}$ transfers (all or part of) the funds from 2-PC_i to 3-PC_i. While 2-PC_i needs P_i and P_{i+1} to sign, 3-PC_i requires additionally a special party P_0 to sign. Channel 3-PC_i expires T times units earlier than 2-PC_i, meaning that after time $t_i - T$, the funds from 3-PC_i are essentially transferred back to 2-PC_i (which is in the control of P_i and P_{i+1}).

II. Technical Overview

In this section we give a brief overview of the techniques we employ to construct a generic scriptless-script PCN protocol compatible with any signature scheme. At the core of our solution we have two main techniques: A local 3-party channel (at the transaction layer) and lockable signatures, a new cryptographic abstraction that we introduce (at the cryptographic layer). We detail each of our technique individually first, then describe how they are put together in our PCN protocol. Finally, we discuss how to construct a highly efficient protocol for the case of BLS signatures, bypassing the need for heavy cryptographic machinery.

A. Transaction Layer: Local 3-Party Channel

Recall that the fairness problem arose in the lock phase because of colluding neighbours P_i and P_{i+1} in a payment path $(P_1, \ldots, P_n)$. Since P_i and P_{i+1} share a payment channel, collectively they possess all the information necessary to sign any transaction. The way prior works [3] bypassed this obstacle, was to allow the sender P_1 to contribute to the randomness of the lock $\bar{\sigma}_i$ between P_i and P_{i+1}. The lock is carefully set such that the input of the sender is hard to compute given the lock, and the valid signature σ_i can only be recovered with the help of the sender. We would like to mimic this approach but by only making generic use of the signature scheme, without relying on any particular feature of the signature scheme.

Local Channel. Our idea is to let parties locally convert their 2-party payment channels into a 3-party payment channel. As shown in Figure 1, parties P_i and P_{i+1} share a channel denoted by 2-PC_i (a public key shared between P_i and P_{i+1}), with a capacity of v coins and channel expiry time t_i. The sender in this case plays the role of a special party P_0 who, along with P_i and P_{i+1}, generates a 3-party payment channel 3-PC_i (a public key shared between P_0, P_i and P_{i+1}). To do this, Parties P_i and P_{i+1} generate a special setup transaction $tx_{\mathsf{Setup},i}$ that sends $v' \leq v$ coins from 2-PC_i to 3-PC_i, while the extra amount $v - v'$ is sent to some channel between P_i and P_{i+1} and is omitted for clarity. Transaction $tx_{\mathsf{Setup},i}$ expires at $t_i - T$ (where T is a system parameter) and once $tx_{\mathsf{Setup},i}$ expires, v' coins are transferred back to 2-PC_i, the 2-party channel between P_i and P_{i+1}.

Notice that, to spend from 3-PC_i, all of P_0, P_i and P_{i+1} have to agree and generate a signature together on the spending transaction. This way the sender (P_0) has an input on spending from 3-PC_i that is hard to compute by a coalition of P_i and P_{i+1}. We stress that $tx_{\mathsf{Setup},i}$ is only kept locally by the parties P_0, P_i and P_{i+1} and not posted on the chain, hence the name *local 3-party channel*. An added advantage of this approach is that intermediate parties need not lock all the (v) coins from their payment channel for the purposes of PCN payments. They can decide how many coins ($v' \leq v$) they want to lock for payments in the form of a 3-party channel with the sender P_0. This way the remaining coins $v - v'$ are available for the intermediate parties which they can use in any way they want.

The notion of a local 3-party channel is reminiscent of the concept of virtual channels [37], however their purposes are very different: A virtual channel allows two parties to exchange coins via a (trustless) intermediary without communicating with it for every payment. On the other hand, a local 3-party channel requires (and enforces) the active interaction of all 3 participants.

Preventing Denial of Service Attacks. A malicious P_0 (the sender) could try a denial of service (DoS) attack, by refusing to spend from 3-PC_i, and locking P_i and P_{i+1}'s channel funds. To prevent this, 3-PC_i is set to expire before 2-PC_i expires. So, if P_0 goes offline and 3-PC_i expires at $t_i - T$, the entire fund of v coins is still available in 2-PC_i up to time T, meaning that parties P_i and P_{i+1} can make payments using 2-PC_i, before 2-PC_i itself expires at time t_i.

On-Chain Privacy. We remark that the creation of these local channels happens completely off-chain and essentially consists of the joint generation of a public key. On-chain transactions from and to such a (shared) public key are identical to any other payment of any other public key. Therefore, our PCN transactions are indistinguishable from other regular transactions on-chain.

B. Cryptographic Layer: Lockable Signatures

The one remaining piece of the puzzle for constructing a generic PCN is the functionality of party P_i being able to obtain a valid signature on a transaction, provided party P_{i+1} releases a valid signature on another transaction. To capture this functionality, we introduce the notion of *lockable signatures*, a new cryptographic abstraction. Intuitively, lockable signatures enable a user to generate a lock ℓk using the Lock algorithm, that hides a signature σ (*the locked signature*) with the help of another signature $\tilde{\sigma}$ (*the locking signature*). Here σ is a signature on a message m signed using the secret key sk, while $\tilde{\sigma}$ is a signature on a message $\tilde{m}$ signed using the secret key $\tilde{sk}$. Note that the locked and the locking signatures may even be from different signature schemes.

In terms of security we want that the lock ℓk is *hiding*, which intuitively says that no information about the locked signature σ is revealed via the lock (if $\tilde{\sigma}$ is not revealed). We also want a guarantee that, given a correctly generated lock ℓk and the valid signature $\tilde{\sigma}$ on the message $\tilde{m}$ under the public key $\tilde{pk}$ (whose corresponding secret key is $\tilde{sk}$), one should be able to run a efficient Unlock procedure that outputs σ. We refer to this security notion as *unlockability*.

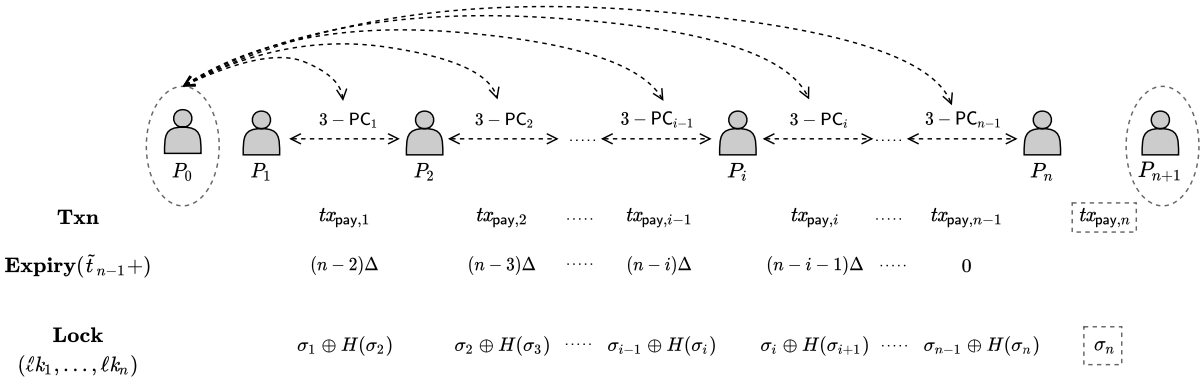

Fig. 2. PCN protocol status after the payment lock phase. Sender P_1 plays the role of the special parties P_0 and P_{n+1}. Transaction $tx_{\mathsf{pay},i}$ spends from $tx_{\mathsf{Setup},i}$ (Figure 1). Transaction $tx_{\mathsf{pay},n}$ inside the dotted box is an arbitrary message known to parties P_0, P_{n+1} and P_n. The expiry times are for payments $tx_{\mathsf{pay},i}, i \in [n-1]$. The lock $\ell k_n := \sigma_n$, in the dotted box is just a regular signature on $tx_{\mathsf{pay},n}$.

We give a simple and generic construction for lockable signatures, where the lock is set as $\ell k := \sigma \oplus H(\tilde{\sigma})$, where H is a hash function and the Unlock procedure simply computes $\sigma := \ell k \oplus H(\tilde{\sigma})$. The hash function H is modelled as a random oracle in the security analysis.

BLS-Based Lockable Signatures. We also present a BLS-based construction, where we have the two signatures as $\sigma \leftarrow \Pi_{\mathsf{BLS}}.\mathsf{Sign}(sk, m)$ and $\tilde{\sigma} \leftarrow \Pi_{\mathsf{BLS}}.\mathsf{Sign}(\tilde{sk}, \tilde{m})$, where $sk, \tilde{sk}$ are two independent secret keys and $m, \tilde{m}$ are two different messages. The lock is simply computed as the aggregate of the two signatures, $\ell k := \sigma \cdot \tilde{\sigma} = \Pi_{\mathsf{BLS}}.\mathsf{Agg}(\sigma, \tilde{\sigma})$ leveraging the well-known properties of the scheme [38]. The Unlock procedure on input a signature $\tilde{\sigma}$ simply computes $\sigma := \ell k / \tilde{\sigma}$. It is well known that extracting aggregated signatures is hard (if no individual aggregated signature is already known). For our purposes, we will need a slightly stronger property: We want that is should be hard to extract an individual signature from a *chain* of aggregate signatures $(\sigma_0 \cdot \sigma_1, \sigma_1 \cdot \sigma_2, \ldots)$. We show that extracting an individual signature from such a chain of aggregates, is as hard as solving the computational Diffie-Hellman (CDH) problem.

The reason why we introduce this special-purpose construction for BLS is that it allows anyone to verify that the lock is well-formed, by simply checking the validity of the BLS aggregate signature. I.e., BLS lockable signatures have a built-in verification to ensure unlockability of (potentially malformed) locks. This feature will greatly increase the practicality of our BLS-based protocol.

Applications of Lockable Signatures. To build up some intuition on how we are going to use lockable signatures for our main result, we provide a toy example for the applicability of this primitive. Consider the scenario where we want a transaction tx_0 to take place only if another transaction tx_1 is posted on the blockchain. We can then generate a lock ℓk on σ_0 (a valid signature for tx_0) conditioned on σ_1 (a valid signature for tx_1). Publishing ℓk guarantees that anyone can recover a valid signature on tx_0 only if tx_1 appears on the blockchain (and not before that). This can be naturally extended to more complex scenarios where the finalization of a contract is conditioned on the finalization of another (possibly independent) contract.

C. Putting Things Together

We now show how to construct a PCN protocol using the above tools. Our payment protocol starts with a one-time *setup* phase that is run by the parties $P_1, \ldots, P_n$, before the payment lock phase. In this phase, the sender P_1 plays the role of a special party P_0 and creates a local 3-party channel 3-PC_i (using $tx_{\mathsf{Setup},i}$) with parties P_i and P_{i+1} for $i \in \{1, \ldots, n-1\}$, as shown in Figure 2. To do this, parties P_0, P_i and P_{i+1} run an MPC protocol that computes a shared public key pk_i and each party receives shares of the secret key sk_i, namely $sk_i^{(0)}, sk_i^{(i)}$ and $sk_i^{(i+1)}$, respectively. Here the shared public key pk_i is set as the 3-party channel 3-PC_i.

In the payment *lock* phase, parties P_0, P_{i-1}, P_i and P_{i+1}, for $i \in \{2, \ldots, n\}$, run a MPC protocol that computes

$$\ell k_{i-1} := \sigma_{i-1} \oplus H(\sigma_i) \leftarrow \mathsf{Lock}(sk_{i-1}, tx_{\mathsf{pay},i-1}, sk_i, tx_{\mathsf{pay},i})$$

and is returned to party P_i. Here, we have:

- The keys sk_{i-1} and sk_i are the shared secret key between (P_0, P_{i-1}, P_i) and (P_0, P_i, P_{i+1}), respectively, established during the payment setup phase.
- The transaction $tx_{\mathsf{pay},i-1}$ (analogously $tx_{\mathsf{pay},i}$) spends v coins from pk_{i-1} or 3-PC_{i-1} (pk_i or 3-PC_i) to some key pk_{i-1}^* (pk_i^*) of party P_i (P_{i+1}).
- The signature σ_{i-1} (analogously σ_i) is the signature on the transaction $tx_{\mathsf{pay},i-1}$ ($tx_{\mathsf{pay},i}$) under the public key pk_{i-1} or 3-PC_{i-1} (pk_i or 3-PC_i).

The recipient finally obtains ℓk_{n-1} generated with the help of P_0, P_{n-1} and P_{n+1} (which is again impersonated by the sender). Notice the expiry times of the payment transactions are set such that $tx_{\mathsf{pay},i-1}$ expires Δ time units after $tx_{pay,i}$ expires. This is to ensure P_i has sufficient time (Δ units) to get paid from $tx_{\mathsf{pay},i-1}$ after it pays P_{i+1} using $tx_{\mathsf{pay},i}$.

The payment release phase is triggered when the sender (P_0, P_{n+1}) jointly generates σ_n along with the receiver P_n. The receiver can unlock ℓk_{n-1} using Unlock to obtain σ_{n-1} and returns it to P_{n-1}. This procedure continues until party

P_2 who unlocks ℓk_1 and learns σ_1. Every party P_i (except the sender) has two transactions $tx_{\mathsf{Setup},i-1}$ and $tx_{\mathsf{pay},i-1}$ (that spends from $tx_{\mathsf{Setup},i-1}$) along with valid signatures on both, concluding one payment of v coins from P_1 to P_n.

For subsequent payments between the same sender and the receiver via the same payment path $P_1, \ldots, P_n$, the parties can re-use the setup phase and only overwrite the lock information by re-running the lock phase.

Security. Intuitively, the unlockability of the lockable signatures guarantees that no adversary can return a valid signature σ_i' on $tx_{\mathsf{pay},i}$ under pk_i to an honest party P_i, such that when P_i tries to unlock ℓk_{i-1} with σ_i', he gets an invalid signature. This ensures the adversary cannot steal the funds of an honest P_i, no matter how many other parties it corrupts.

The hiding property of the lockable signatures ensures that no adversary can unlock ℓk_i (for any i) to reveal a valid signature σ_i on $tx_{\mathsf{pay},i}$, before the sender initiates the release phase. This ensures that no adversary corrupting an intermediate party in the payment path can steal the funds of the sender. Our construction is secure against *wormhole attacks* [3] following a similar argument.

It is important to notice that we consider the standard notion of MPC where the adversary can abort at any time and deny the honest parties from seeing their output (i.e., we do not assume fairness). This means that some extra care is needed to ensure that adversarial aborts do not cause the honest parties to lose coins. To convey some intuition, consider two cases.

- *Abort before all locks are established*: the participants will also abort the execution. The hiding property of the locks guarantees that the signatures on the transactions remain hidden and the honest parties do not lose money.
- *Abort after all locks are established*: the honest parties can "unlock" their signatures by just accessing the blockchain (and using some local information). The unlockability property guarantees that this process never fails.

Efficient BLS-Based PCN. We obtain an efficient BLS-based PCN protocol by exploiting the structure of the keys and the signature. In the setup phase, we substitute the usage of a general purpose MPC with a non-interactive protocol where the shared public key is set to be the aggregate of individual public keys of the parties. That is,

$$pk_i := pk_i^{(0)} \cdot pk_i^{(i)} \cdot pk_i^{(i+1)}$$

where $pk_i^{(0)}, pk_i^{(i)}$ and $pk_i^{(i+1)}$ are the keys of P_0, P_i and P_{i+1}, respectively. Each key has an associated NIZK proof, proving knowledge of the corresponding secret key, to prevent rogue key attacks [39].

Since BLS signatures are aggregatable, we have that the signature σ_i on $tx_{\mathsf{pay},i}$ under pk_i is just an aggregate of $\sigma_i^{(0)}, \sigma_i^{(i)}$ and $\sigma_i^{(i+1)}$, where each of these are themselves signatures on $tx_{\mathsf{pay},i}$ generated by the respective parties using their respective secret key shares. As discussed above, signatures are locked by simply computing a chain of aggregates

$$(\sigma_1 \cdot \sigma_2, \sigma_2 \cdot \sigma_3, \ldots, \sigma_{n-1} \cdot \sigma_n)$$

i.e., $\ell k_i := \sigma_i \cdot \sigma_{i+1}, (i \in [n-1])$, whose well-formedness can be readily tested using the aggregate verification algorithm of BLS. This allows us to forego entirely the usage of general purpose MPC, thus obtaining a concretely efficient protocol.

III. Preliminaries

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 often omit this randomness and only mention it explicitly when required. The notation $[n]$ denotes a set $\{1, \ldots, n\}$ and $[i, j]$ denotes the set $\{i, i+1, \ldots, j\}$. We consider *probabilistic polynomial time* (PPT) machines as efficient algorithms.

A. Universal Composability

To model security and privacy in the presence of concurrent executions we resort to the *universal composability* framework from Canetti [40] extended to support a global setup [41]. We refer the reader to [40] for a comprehensive discussion. We consider the setting of *static* corruptions, where the adversary must declare ahead of time which parties he wish to corrupt. We denote the environment by $\mathcal{E}$. For a real protocol Π and an adversary $\mathcal{A}$ we write $EXEC_{\tau,\mathcal{A},\mathcal{E}}$ to denote the ensemble corresponding to the protocol execution. For an ideal functionality $\mathcal{F}$ and an adversary $\mathcal{S}$ we write $EXEC_{\mathcal{F},\mathcal{S},\mathcal{E}}$ to denote the distribution ensemble of the ideal world execution.

Definition 1 (Universal Composability): A protocol τ UC-realizes an ideal functionality $\mathcal{F}$ if for any PPT adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$ such that for any environment $\mathcal{E}$ the ensembles $EXEC_{\tau,\mathcal{A},\mathcal{E}}$ and $EXEC_{\mathcal{F},\mathcal{S},\mathcal{E}}$ are computationally indistinguishable.

We require the cryptographic primitives with UC security.

Digital Signatures. A digital signature scheme allows one to authenticate a message with respect to a public key. Formally, it 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 [42]. Unforgeability is formally referred to as *existential unforgeability under chosen message attack* (EUF-CMA), says that a PPT adversary cannot forge a fresh signature on a fresh message of its choice given only the public key and access to a signing oracle (that returns a valid signature on a message of his choice). A stronger notion is that of strong unforgeability (sEUF-CMA) that says the forgery can consist only of a fresh signature, irrespective of whether the message was previously queried to the signing oracle or not. Such a notion is known to be equivalent to the UC formulation of security [43].

In this work we only consider signature schemes with a *deterministic* signing algorithm. This is however without loss of generality, as shown by the following (well known) lemma.

Lemma 1: Let $(\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ be a signature scheme with probabilistic Sign algorithm. Then there exists a signature scheme $(\mathsf{KGen}, \mathsf{Sign}^*, \mathsf{Vf})$ where Sign^* is deterministic.

Proof 1 (Sketch): The new signing algorithm Sign^* is defined to be $\mathsf{Sign}(sk, m; \mathsf{PRF}(sk, m))$, where PRF is a cryptographic pseudorandom function.

Commitment Schemes. A commitment scheme is a digital analogue of sealing a message inside an envelope. Formally, it consists of the following tuple of efficient algorithms: A commitment generation algorithm $\mathsf{Commit}(1^\lambda, m)$ that takes as input a security parameter and a message m to commit to, and outputs a commitment c and a corresponding opening information d. The opening algorithm $\mathsf{Open}(c, d)$ takes as input a commitment c and a opening information d and outputs the committed message m or outputs a special symbol $\perp$ if d is not the valid opening information for the commitment c. In addition to the standard binding and hiding properties (who's UC formalization can be found in [40]), we require that the commitment scheme has unique openings. I.e. for all commitments there exists a single valid message that causes the Open algorithm to accept.

Non-Interactive Zero Knowledge Proofs. Let $R : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$ 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) [44] system for the relation R is initialized with a setup algorithm $\mathsf{Setup}(1^\lambda)$ that, on input the security parameter, outputs a common reference string crs and a trapdoor td. A prover can show the validity of a statement x with a witness w by invoking $\mathcal{P}(crs, x, w)$, which outputs a proof π. The proof π can be efficiently checked by the verification algorithm $\mathcal{V}(crs, x, \pi)$. 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 extractable*, if there exists an extractor algorithm $\mathcal{E}$ that on input the common reference string crs, a trapdoor information td, the statement x and the proof π, and outputs a witness w such that $(x, w) \in R$ with high probability. For formal UC-style definition of security we refer the reader to [45].

Multi-Party Computation. The aim of a secure multi-party computation (MPC) protocol is for the participating parties to securely compute some function f of their private inputs. In terms of properties, we require *correctness* that states that the parties' output is that which is defined by the function. We then require *privacy* that states that the only information learned by the parties in the computation is that specified by the function output. For a comprehensive treatment of the UC definition of MPC we refer the reader to [46]. In this work we require the existence of MPC protocols that are:

- Secure against an *active* adversary corrupting up to $n-1$ of the n players, where the set of corrupted parties is determined ahead of time (static corruption).
- Secure with aborts, i.e. the definition allows the adversary to decide which of the honest parties obtain the output of the computation.

The latter requirement implies that we do not assume that the MPC protocol achieves any form of fairness or guaranteed output delivery. These is arguably the most common flavor of MPC, for which many general-purpose efficient protocols are known (see e.g. [47] and follow-up works).

Synchrony and Communication. We assume synchronous communication between users, where the execution of the protocol happens in rounds. We model this via an ideal functionality $\mathcal{F}_{\mathsf{clock}}$ as it is done in [37], [48], where all honest parties are required to indicate that are ready to proceed to the next round before the clock proceeds. The clock functionality that we consider is fully described in [41]. This means that all entities are always aware of the given round. We also assume the existence of secure message transmission channels between users modelled by $\mathcal{F}_{\mathsf{smt}}$.

Blockchain. We assume the existence of a blockchain $\mathbb{B}$ (just as in [2], [3], [36]) that we model as a trusted append-only bulletin board: The corresponding ideal functionality $\mathcal{F}_{\mathbb{B}}$ maintains the chain $\mathbb{B}$ locally and updates it according to the transactions between users. The functionality is also parameterized by a transaction scheme which also specifies a signature scheme, that lets any user generate key pairs and can post a signed transaction transferring coins from one user to another. We use the notation $tx := tx(A, B, v)$ to denote a transaction that sends v coins from address A to address B. At any point in the execution, any user U can send a distinguished message read to $\mathcal{F}_{\mathbb{B}}$, who sends the whole transcript of $\mathbb{B}$ to U. We refer the reader to [36] for a formal definition of this functionality. Our $\mathcal{F}_{\mathbb{B}}$ functionality also offers a *timelock* interface, where a particular transaction can be assigned a expiry time t. If the global clock is past the expiry time t, it means that the transaction has expired and its effect on user balances, is revoked.

B. Bilinear Maps

Let $(\mathbb{G}_0, \mathbb{G}_1, \mathbb{G}_t)$ be three groups of order q, where q is a λ bit prime. A pairing function e is an efficiently computable function $e : \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_T$, where g_0 and g_1 are generators of $\mathbb{G}_0$ and $\mathbb{G}_1$ respectively. The pairing operation is *bilinear* if for all $u \in \mathbb{G}_0, v \in \mathbb{G}_1, a, b \in \mathbb{Z}$, we have $e(u^a, v^b) = e(u, v)^{ab}$. The pairing operation is *non-degenerate* if $e(g_1, g_2) \neq 1$.

BLS Signature Scheme. Let H be a hash function $H : \{0,1\}^* \to \mathbb{G}_0$. We briefly recall the BLS aggregatable signature [17] in Figure 3. It is well known that the scheme has unique signature and it is strongly unforgeable in the random oracle model if the computational Diffie-Hellman (CDH) problem is hard over the bilinear group. In Appendix B, we recall the aggregate extraction assumption from [38] (Definition 4.3), which in a later work, Coron and Naccache [49] showed is equivalent to the CDH assumption.

We introduce a new hard problem that we call *chained aggregate extraction problem*, that says that an adversary who is given a chain of aggregate values, cannot extract any of the aggregated values in polynomial time. This assumption significantly simplifies our security analysis of our BLS based PCN protocol. We formally define the problem below.

$\underline{\mathsf{KGen}(1^\lambda)}$
$\alpha \leftarrow \mathbb{Z}_q$
$h \leftarrow g_1^\alpha \in \mathbb{G}_1$
$pk := h, sk := \alpha$
return (pk, sk)

$\underline{\mathsf{Sign}(sk, m)}$
$\sigma := H(m)^{sk}$
return σ

$\underline{\mathsf{Agg}(\{\sigma_i\}_{i\in[t]})}$
$\sigma \leftarrow \prod_{i\in[t]} \sigma_i$
return σ

$\underline{\mathsf{Vf}(pk, m, \sigma)}$
if $(e(\sigma, g_1) = e(H(m), pk))$
  return 1
else return 0

$\underline{\mathsf{VfAgg}(\{pk_i\}_{i\in[t]}, \{m_i\}_{i\in[t]}, \sigma)}$
if $(e(\sigma, g_1) = \prod_{i\in[t]} e(H(m_i), pk_i))$
  return 1
else return 0

Fig. 3: BLS signature scheme

$\underline{\mathsf{ExpChAgExt}_{\mathcal{A},\mathbb{G}_0,\mathbb{G}_1,\mathbb{G}_t}(1^\lambda, n)}$
$x_i, y_i \leftarrow \mathbb{Z}_q, \forall i \in [n]$
$\sigma_i \leftarrow g_0^{(x_i y_i + x_{i+1} y_{i+1})}, \forall i \in [n-1]$
$(\sigma, j) \leftarrow \mathcal{A}(\{g_1^{x_i}, g_0^{y_i}\}_{i\in[n]}, \{\sigma_i\}_{i\in[n-1]})$
$b_0 := (j \in [n])$
$b_1 := (\sigma = g_0^{x_j y_j})$
return $b_0 \wedge b_1$

Fig. 4: Chained Aggregate Extraction experiment

Definition 2 (Chained Aggregate Extraction Problem): The *chained aggregate extraction problem* for a bilinear pairing group $\mathbb{G}_0, \mathbb{G}_1, \mathbb{G}_t$ of order q is said to be hard if there exists a negligible function negl, for all $\lambda \in \mathbb{N}$ and $n = \mathsf{poly}(\lambda)$ for some polynomial poly, and all PPT adversaries $\mathcal{A}$, it holds that,

$$\Pr\left[\mathsf{ExpChAgExt}_{\mathcal{A},\mathbb{G}_0,\mathbb{G}_1,\mathbb{G}_t}(1^\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$$

where ExpChAgExt is defined in Figure 4.

In Theorem 3.1 we show that the chained aggregate extraction problem is equivalent to the aggregate extraction problem, when $k = 2$, where k denotes the number of signatures aggregated with each other into a single value. Formal security proof of the theorem can be found in Appendix B.

Theorem 3.1: For $k = 2$, the aggregate extraction assumption (Definition 8) is equivalent to the Chained Aggregate Extraction assumption (Definition 2).

Corollary 1: The Chained Aggregate Extraction assumption is equivalent to the CDH assumption.

IV. Lockable Signatures

A lockable signature scheme allows one to lock a signature (referred to as the *locked* signature) on a message with respect to another signature (referred to as the *locking* signature), possibly on a different message. The two signatures can verify against two different public keys. To learn the locked signature, one has to know the locking signature and otherwise the locked signature is computationally hidden.

A. Definitions

Below we formally define the interfaces, notion of correctness and the properties of interest of a lockable signature.

Definition 3 (Lockable Signatures): A Lockable signature scheme LS is defined with respect to a signature scheme $\Pi_{\mathcal{DS}}$ and consists of PPT algorithms $(\mathsf{Lock}, \mathsf{Unlock})$ defined below.

$\ell k \leftarrow \mathsf{Lock}(sk, m, \tilde{sk}, \tilde{m})$: The lock algorithm takes as input two secret keys $(sk, \tilde{sk})$ and two messages $(m, \tilde{m})$ and returns a lock ℓk.

$\sigma \leftarrow \mathsf{Unlock}(pk, m, \tilde{pk}, \tilde{m}, \tilde{\sigma}, \ell k)$: The unlock algorithm takes as input two public keys $(pk, \tilde{pk})$, a pair of messages $(m, \tilde{m})$, a signature $\tilde{\sigma}$, and a lock ℓk, and outputs a signature σ.

Definition 4 (Correctness): A lockable signature scheme LS is *correct* if for all $\lambda \in \mathbb{N}$, all pairs of messages $(m, \tilde{m}) \in \{0,1\}^\lambda$, for all key pairs (pk, sk) and $(\tilde{pk}, \tilde{sk})$ in the image of KGen it holds that

$$\mathsf{Vf}(pk, m, \mathsf{Unlock}(pk, m, \tilde{pk}, \tilde{m}, \tilde{\sigma}, \ell k)) = 1$$

where $\ell k \leftarrow \mathsf{Lock}(sk, m, \tilde{sk}, \tilde{m})$ and $\tilde{\sigma} \leftarrow \mathsf{Sign}(\tilde{sk}, \tilde{m})$.

Below we define the security notions for lockable signatures. As discussed before, we consider without loss of generality only signatures where the signing algorithm is deterministic. In all of the security notions, we assume that the Lock algorithm is executed honestly. Looking ahead to our main protocol, this assumption is justified by the fact that the algorithm is always executed as part of an MPC protocol where at least one of the participants is honest.

Unlockability. The unlockability property ensures that a correctly generated lock can be unlocked to reveal a valid locked signature, when provided with a valid locking signature. We capture this intuition in the form of an experiment EUn-CMA in Figure 5. Here the adversary gets to choose the messages $(m, \tilde{m})$ while it has access to a signing oracle for the key $\tilde{sk}$. The experiment generates a lock ℓk honestly by running $\mathsf{Lock}(sk, m, \tilde{sk}, \tilde{m})$ and gives the lock to the adversary. The adversary returns a candidate locking signature σ^* which is used to unlock the lock ℓk and obtain a candidate locked signature σ'. The adversary wins the experiment if σ^* is a valid signature on the message $\tilde{m}$ under $\tilde{pk}$ (condition b_0) while σ' is an invalid signature on the message m under pk (condition b_1). A lockable signature scheme is said to satisfy unlockability if the adversary wins the above experiment at most with negligible probability in the security parameter.

Definition 5 (Unlockability): A lockable signature LS is *unlockable* if there exists a negligible function negl such that for all $\lambda \in \mathbb{N}$ and for all PPT adversaries $\mathcal{A}$ it holds that

$$\Pr[\mathsf{EUn\text{-}CMA}_{\mathcal{A},\mathsf{LS}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$$

where the experiment EUn-CMA is defined in Figure 5.

Hiding. The hiding property ensures that a correctly generated lock reveals no information about the locked or the locking signature. We capture this intuition in the form of an experiment EHi-CMA in Figure 5. Here the adversary gets to choose two messages $(m, \tilde{m})$ while having access to signing oracles with respect to both keys sk and $\tilde{sk}$. The experiment chooses a bit b uniformly random and if $b = 0$, it generates a lock correctly using $\mathsf{Lock}(sk, m, \tilde{sk}, \tilde{m})$. If $b = 1$, the experiment uses a simulator Sim that only takes as input the public keys $pk, \tilde{pk}$

$\mathsf{EUn\text{-}CMA}_{\mathcal{A},\mathsf{LS}}(\lambda)$

$Q := \emptyset$
$(pk, sk) \leftarrow \Pi_{\mathcal{DS}}.\mathsf{KGen}(1^\lambda)$
$(\tilde{pk}, \tilde{sk}) \leftarrow \Pi_{\mathcal{DS}}.\mathsf{KGen}(1^\lambda)$
$(m, \tilde{m}, \mathsf{state}) \leftarrow \mathcal{A}^{\mathsf{Sign}\mathcal{O}(\tilde{sk},\cdot)}(sk, \tilde{pk})$
$\ell k \leftarrow \mathsf{Lock}(sk, m, \tilde{sk}, \tilde{m})$
$\sigma^* \leftarrow \mathcal{A}^{\mathsf{Sign}\mathcal{O}(\tilde{sk},\cdot)}(\mathsf{state}, \ell k)$
$\sigma' \leftarrow \mathsf{Unlock}(pk, m, \tilde{pk}, \tilde{m}, \sigma^*, \ell k)$
$b_0 := \Pi_{\mathcal{DS}}.\mathsf{Vf}(\tilde{pk}, \tilde{m}, \sigma^*) = 1$
$b_1 := \Pi_{\mathcal{DS}}.\mathsf{Vf}(pk, m, \sigma') \neq 1$
return $b_0 \wedge b_1$

$\mathsf{EHi\text{-}CMA}_{\mathcal{A},\mathsf{LS}}(\lambda)$

$Q := \emptyset$
$(pk, sk) \leftarrow \Pi_{\mathcal{DS}}.\mathsf{KGen}(1^\lambda)$
$(\tilde{pk}, \tilde{sk}) \leftarrow \Pi_{\mathcal{DS}}.\mathsf{KGen}(1^\lambda)$
$\mathbb{O} := \{\mathsf{Sign}\mathcal{O}(sk, \cdot), \mathsf{Sign}\mathcal{O}(\tilde{sk}, \cdot)\}$
$(m, \tilde{m}, \mathsf{state}) \leftarrow \mathcal{A}^{\mathbb{O}}(pk, \tilde{pk})$
$b \leftarrow \{0, 1\}$
if $b = 0$ **then**
  $\ell k \leftarrow \mathsf{Lock}(sk, m, \tilde{sk}, \tilde{m})$
else $\ell k \leftarrow \mathsf{Sim}(pk, \tilde{pk})$
$b^* \leftarrow \mathcal{A}^{\mathbb{O}}(\mathsf{state}, \ell k)$
$b_0 := (b = b^*)$
$b_1 := (\tilde{m} \notin Q)$
return $b_0 \wedge b_1$

$\mathsf{Sign}\mathcal{O}(\tilde{sk}, m)$

$\sigma \leftarrow \Pi_{\mathcal{DS}}.\mathsf{Sign}(\tilde{sk}, m)$
$Q := Q \cup \{\tilde{m}\}$
return σ

$\mathsf{Sign}\mathcal{O}(sk, m)$

$\sigma \leftarrow \Pi_{\mathcal{DS}}.\mathsf{Sign}(sk, m)$
return σ

Fig. 5: Experiments for unlockability and hiding of lockable signatures

(corresponding to sk and $\tilde{sk}$, respectively) and outputs a lock ℓk. The adversary is given ℓk and outputs a guess b^*. The adversary wins the experiment if the guess was correct, i.e. $b = b^*$ (condition b_0) and if $\tilde{m}$ was never queried before to the signing oracle with respect to $\tilde{sk}$. The latter condition is necessary to avoid trivial attacks where the adversary uses a signature on $\tilde{m}$ to run the unlocking algorithm. A lockable signature is said to satisfy hiding if the adversary wins the above experiment with probability negligibly close to $1/2$.

Definition 6 (Hiding): A lockable signature LS is *hiding* if there exists a negligible function negl and a simulator Sim such that for all $\lambda \in \mathbb{N}$ and all PPT adversaries $\mathcal{A}$ it holds that

$$\Pr[\mathsf{EHi\text{-}CMA}_{\mathcal{A},\mathsf{LS}}(\lambda) = 1] \leq 1/2 + \mathsf{negl}(\lambda)$$

where the experiment $\mathsf{EHi\text{-}CMA}$ is defined in Figure 5.

B. Constructions of Lockable Signatures

In the following we describe our generic and BLS-based construction of a lockable signatures.

$\mathsf{Lock}(sk, m, \tilde{sk}, \tilde{m})$

$\sigma \leftarrow \Pi_{\mathcal{DS}}.\mathsf{Sign}(sk, m)$
$\tilde{\sigma} \leftarrow \Pi_{\mathcal{DS}}.\mathsf{Sign}(\tilde{sk}, \tilde{m})$
$\ell k := \sigma \oplus H(\tilde{\sigma})$
return ℓk

$\mathsf{Unlock}(pk, m, \tilde{pk}, \tilde{m}, \tilde{\sigma}, \ell k)$

return $\ell k \oplus H(\tilde{\sigma})$

$\mathsf{Lock}(sk, m, \tilde{sk}, \tilde{m})$

$\sigma \leftarrow \Pi_{\mathsf{BLS}}.\mathsf{Sign}(sk, m)$
$\tilde{\sigma} \leftarrow \Pi_{\mathsf{BLS}}.\mathsf{Sign}(\tilde{sk}, \tilde{m})$
$\ell k := \Pi_{\mathsf{BLS}}.\mathsf{Agg}(\sigma, \tilde{\sigma})$
return ℓk

$\mathsf{Unlock}(pk, m, \tilde{pk}, \tilde{m}, \tilde{\sigma}, \ell k)$

return $\ell k / \tilde{\sigma}$

Fig. 6: Generic (left) and BLS-based (right) constructions of Lockable Signatures

Generic Construction. Let $\Pi_{\mathcal{DS}} := (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ be a digital signature scheme (with deterministic signing algorithm) and let $H : \{0,1\}^\lambda \to \{0,1\}^\lambda$ be a hash function. Our generic construction of lockable signature is specified in Figure 6 (left). In our construction, a lock is an xor of the locked signature σ and the hash of the locking signature $\tilde{\sigma}$. Provided the hash of the locking signature $H(\tilde{\sigma})$ is a random string with high entropy, we can see that intuitively the locked signature σ is hidden inside the lock, similar to a one-time pad. The release procedure is an xor of $H(\tilde{\sigma})$ and the lock, similar to the decryption of a one-time pad.

In the theorem below, we prove that our generic construction satisfies unlockability (Definition 5) and hiding (Definition 6), as defined before. In favor of a simpler analysis, we model the hash function H as a random oracle, however we note that one could also set H to be a hardcore function [50] (for polynomially-many bits) and security would follow with a similar argument. We defer the proof to the full version due to space constraints.

Theorem 4.1 (Unlockability & Hiding): Let $\Pi_{\mathcal{DS}} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ be a strongly unforgeable digital signature scheme. Then $\mathsf{LS} := (\mathsf{Lock}, \mathsf{Unlock})$ is unlockable and hiding in the random oracle model.

BLS-Based Construction. We present an alternative construction of lockable signatures $\mathsf{LS} := (\mathsf{Lock}, \mathsf{Unlock})$ for the BLS signature scheme $\Pi_{\mathsf{BLS}} := (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf}, \mathsf{Agg}, \mathsf{VfAgg})$ in Figure 6 (right). While this scheme satisfies a weaker notion of hiding (as defined in Appendix A), it will allow us to construct a much more efficient PCN, without the need to resort to complex cryptographic machinery. Intuitively, a lockable signature consists of an aggregate of two signatures, and the security comes from the hardness of extracting signatures from aggregates.

Unlockability follows unconditionally from the uniqueness of BLS signatures. The scheme only satisfies a weaker notion of hiding (i.e. only the search version of the problem is hard), which is captured by the chained aggregate extraction assumption (Definition 2). Loosely speaking, weak hiding ensures that an adversarial party cannot extract a valid signature from a lock, before the corresponding locking signature is revealed. In our protocol (see Section V-C), this is sufficient to ensure that an adversarial party in the payment path cannot steal the funds of honest parties in the same path.

V. Payment Channel Networks

In this section we describe two constructions for Payment Channel Networks (PCNs) from lockable signatures: The first construction is generic, i.e. individual payment channels in the payment path can be based on any signature scheme. Our second construction is an efficient instantiation of the generic construction, tailored for the case where the individual payment channels are based on BLS signatures.

A. Definition of PCN

We recall the notion of a PCN as an ideal functionality $\mathcal{F}_{PCN}$ as proposed in [2]. Payment channels in the chain $\mathbb{B}$ are of the form $(c_{\langle u_0,u_1\rangle}, v, t, f)$, where $c_{\langle u_0,u_1\rangle}$ is a unique channel identifier for the channel between users u_0 and u_1, v is the capacity of the channel, t is the expiration time of

the channel and f is the associated fee. Note that any two users may have multiple channels open simultaneously. The functionality maintains two additional internal lists $\mathcal{L}$ (to keep track of the closed channels) and $\mathcal{P}$ (to record the off-chain payments in an open channel). Entries in $\mathcal{P}$ are of the form $(c_{\langle u_0,u_1\rangle}, v, t, h)$, where $c_{\langle u_0,u_1\rangle}$ is the corresponding identifier, v is the amount of credit used, t is the expiration time of the channel, and h is the identifier of the entry.

The functionality provides the users with interfaces `open`, `close`, and `pay`, which are used to open a channel, close the channel, and make payments using the channel, respectively. $\mathcal{F}_{PCN}$ initializes a pair of empty lists $\mathcal{P}, \mathcal{L}$. Users can query $\mathcal{F}_{PCN}$ to open channels and close them to any valid state in $\mathcal{P}$. On input a value v and a set of payment channels $c_{\langle u_0,u_1\rangle}, \ldots, c_{\langle u_n,u_{n+1}\rangle}$ from some user u_0, $\mathcal{F}_{PCN}$ checks whether the path has enough capacity (step 1) and initiates the payment. Each intermediate user can either allow the payment or deny it. Once the payment has reached the receiver, each user can again decide to interrupt the flow of the payment (step 2), i.e. pay instead of the sender. Finally $\mathcal{F}_{PCN}$ informs the involved nodes of the success of the operation (step 3) and adds the updated state to $\mathcal{P}$ for the corresponding channels.

In terms of off-chain privacy, we model a slightly weaker notion than the functionality from [2]: Each user u_i, in addition to learning the channel identifiers $c_{\langle u_{i-1},u_i\rangle}$ and $c_{\langle u_i,u_{i+1}\rangle}$ of its neighbours u_{i-1} and u_{i+1}, also learns the channel identifiers $c_{\langle u_{i-2},u_{i-1}\rangle}$ and $c_{\langle u_{i+1},u_{i+2}\rangle}$ of its neighbours' neighbours u_{i-2} and u_{i+2}. In other words, any intermediate user u_i learns the identity of the sender u_0 and of the two nodes that precede and succeed him in the payment path. For completeness, we describe the modified ideal functionality in Figure 9.

B. Construction I: PCNs for all Signatures

We now describe our generic construction of PCN using lockable signatures. Consider the setting where party P_1 wants to send v coins to party P_n via off-chain payments. We denote by j-PC a payment channel involving j-many parties, i.e. where the secret key is shared among j users. We assume that neighbouring parties in $P_1, \ldots, P_n$ have payment channels between them, namely, P_i has a payment channel 2-PC_i with P_{i+1} for $i \in [n-1]$. Using this network of payment channels, P_1 wishes to route v coins to P_n.

For ease of understanding, we make the following simplifying assumptions: First, we assume the participation fee for each party in the payment path $f_i := 0$. Second, we assume that every payment channel 2-PC_i (for $i \in [n-1]$) in the path from P_1 to P_n is based on the same signature scheme $\Pi_{\mathcal{DS}}$. Our construction can be easily tweaked to support a different signature scheme for each payment channel in the payment path, provided that the signature schemes satisfy the requirements necessary to construct a lockable signature (namely, strong unforgeability and deterministic signing).

1) Overview: Our protocol assumes the existence of the following cryptographic tools: (i) A digital signature scheme $\Pi_{\mathcal{DS}}$ with the corresponding lockable signature algorithms LS, (ii) a UC-secure commitment scheme $(\mathsf{Commit}, \mathsf{Open})$ with unique openings, and (iii) a UC-secure MPC protocol Γ for general functions.

We also assume the transaction expiry functionality from the underlying blockchain that invalidates a transaction if the transaction is unspent and a pre-specified time has passed. In Section V-D we discuss how this can be realized in practice.

Our generic PCN protocol formally described in Figure 7, proceeds in three phases, namely, a (one-time) setup phase, a lock phase and a release phase.

Setup Phase. The sender P_1 plays the role of a (special) party P_0 and sets up a *local* 3-party payment channel with each successive pair of users. To do this, party P_0 along with parties P_i and P_{i+1} (for $i \in [n-1]$), run an MPC protocol Γ to generate (i) a new joint public key pk_i, (ii) a commitment c_i to the corresponding secret key sk_i, (iii) a 3-out-of-3 share of the secret key sk_i and the opening information d_i (corresponding to the commitment c_i). The commitment c_i will force the parties to use consistent inputs (i.e. shares of the secret key) in the MPC run during the locking phase. Once the MPC successfully terminates, party P_i and P_{i+1} jointly generate a transaction $tx_{\mathsf{Setup},i}$ that transfers v coins from 2-PC_i (sending address) to the newly generated joint key pk_i, referred to as 3-PC_i (recipient address).

Note that in an honest run of the protocol, this setup transaction $tx_{\mathsf{Setup},i}$ is *never* published on the chain and only kept among the parties P_0, P_i, P_{i+1} locally. It acts as a safeguard for the sender P_1 who wants to ensure that no two intermediate parties P_i, P_{i+1} can release the payments without his permission. Additionally, if the payment channel 2-PC_i's expiry time was t_i, the setup transaction $tx_{\mathsf{Setup},i}$ is set to have a expiry time $t_{\mathsf{Setup},i} = t_i - T$, where T is some system parameter. This is to ensure that even if the sender P_1 (or P_0) goes offline and coins from $tx_{\mathsf{Setup},i}$ remain unspent, parties P_i and P_{i+1} can still get back the locked coins from $tx_{\mathsf{Setup},i}$. Once they get the coins back to 2-PC_i from 3-PC_i, they can make payments in the remaining time T after which the channel 2-PC_i itself expires.

Lock Phase. Parties $P_0, P_{i-1}, P_i, P_{i+1}$ (for $i \in [2, n-1]$), run the MPC Γ to set up the lock ℓk_{i-1} that is received by P_i. The locked signature in ℓk_{i-1} is σ_{i-1}, which is a signature on the transaction $tx_{\mathsf{pay},i-1}$ under the key pk_{i-1} (or 3-PC_{i-1}). Here the transaction $tx_{\mathsf{pay},i-1}$ pays v coins from 3-PC_{i-1} to a key pk_i^* of party P_i. A critical requirement of the MPC protocol is that it must ensure that the locked signature σ_{i-1} in the lock ℓk_{i-1} is actually the locking signature for lock ℓk_{i-2}, and so on. This is done by checking that the parties provide as input valid shares of the secret key contained in the commitment c_i, which is verified inside of the MPC protocol.

For the final lock, party P_0, P_{n-1}, P_n and (a special) party P_{n+1} (again played by the sender P_1) setup the lock ℓk_{n-1} whose locking signature σ_n is a signature on some (arbitrary) message $tx_{\mathsf{pay},n}$ under the public key pk_n (shared among P_0, P_{n+1} and P_n).

Notice that $tx_{\mathsf{pay},i-1}$ expires Δ time units after $tx_{\mathsf{pay},i}$. This is to ensure that party P_i has enough time in the release phase

(described below), to learn σ_{i-1} and get paid by $tx_{\mathsf{pay},i-1}$ after P_{i+1} gets paid by $tx_{\mathsf{pay},i}$. This requirement is the same as what is required in [2], [3].

Release Phase. Once the payment locks have been established until the recipient P_n, the sender P_1 along with the receiver P_n, jointly unlock ℓk_{n-1}. This is done by running the MPC protocol Γ to compute σ_n on $tx_{\mathsf{pay},n}$ using their (P_0, P_n, P_{n+1}) shares of the secret key sk_n.

P_n learns σ_n using which it reveals the signature σ_{n-1} on $tx_{\mathsf{pay},n-1}$, by executing the Unlock algorithm. Party P_n then sends σ_{n-1} to party P_{n-1} using which P_{n-1} can unlock ℓk_{n-2}, and so on. The cascade ends with party P_2 learning σ_1, thus concluding one payment of v coins from P_1 to P_n.

We stress that in case all parties are honest, then no information about the payment is posted on the blockchain. Thus subsequent payments can happen by simply overwriting this information (i.e. the locks). On the other hand, if any party P_i wants to close the channel with P_{i-1}, it can post the transactions $tx_{\mathsf{Setup},i-1}$ and $tx_{\mathsf{pay},i-1}$ along with valid signatures for both transactions before either of them expire.

Security Analysis. The following theorem states the security of our protocol and we defer the proof to the full version.

Theorem 5.1: Let LS be a hiding and an unlockable lockable signature scheme with respect to the signature scheme $\Pi_{\mathcal{DS}}$ that is strongly unforgeable. Let $(\mathsf{Commit}, \mathsf{Open})$ be a UC-secure commitment scheme and Γ be a UC-secure MPC for general functions. Then, the PCN described in Figure 7 with access to $(\mathcal{F}_{\mathbb{B}}, \mathcal{F}_{\mathsf{smt}}, \mathcal{F}_{\mathsf{clock}})$ UC-realizes the functionality $\mathcal{F}_{PCN}$ (Figure 9).

C. Construction II: PCNs for BLS Signatures

We now describe our BLS-based PCN protocol where the settings are identical to Section V-B. However, the structure and properties of BLS signatures will allow us to design a PCN with significant efficiency improvements. For example, the joint generation of a shared public key can be done "non-interactively" by simply setting $pk = pk_0 \cdot pk_1$ and $sk = sk_0 + sk_1$. Then a valid signature under pk is an aggregate of a signature under pk_0 and a signature under pk_1.

1) Overview: As before, we describe the payment protocol in three (setup, locking, and release) phases. A formal description of our scheme can be found in Figure 11.

Setup Phase. During this phase, parties P_0, P_i and P_{i+1} (for $i \in [n-1]$), jointly generate pk_i such that each party has a 3-out of-3 share of the corresponding secret key sk_i. Similar to the generic construction, P_0 and P_{n+1} are special parties whose role is played by the sender P_1. To prevent rogue key attacks [39], each party proves the knowledge of its secret share using a NIZK for the language, $\mathcal{L}_{\mathsf{DL}} := \left\{stmt = pk : \exists sk \;\; s.t. \;\; (pk := g_1^{sk})\right\}$. Since BLS signatures are unique to the message and the public key, we do not need an explicit commitment to the secret key.

Lock Phase. Our interactive payment lock phase between $P_0, P_{i-1}, P_i, P_{i+1}$ (for $i \in [2, n-1]$), proceeds in rounds. At the end P_i obtains a lock $\ell k_{i-1} := \sigma_{i-1} \cdot \sigma_i$ where σ_{i-1} is a valid signature on $tx_{\mathsf{pay},i-1}$ under 3-PC_{i-1} and σ_i is a valid signature on $tx_{\mathsf{pay},i}$ under 3-PC_i. The interactions are designed to ensure that it is infeasible for P_{i+1} to produce σ_i without colluding with P_0 and P_i. Similarly, P_{i-1} cannot output σ_{i-1} without colluding with P_0 and P_i. At the end of the locking phase, parties P_0, P_{n-1}, P_n and P_{n+1} establish a lock $\ell k_{n-1} := \sigma_{n-1} \cdot \sigma_n$, where σ_n is a signature on some (arbitrary) message $tx_{\mathsf{pay},n}$ generated by P_0 and P_n.

Release Phase. This phase is initiated by P_0 by computing σ_n. Starting from P_n, all parties can progressively start unlocking their signatures: For $i \in [n, 2]$ party P_i learns σ_{i-1} by simply computing $\ell k_{i-1}/\sigma_i$, i.e. by extracting σ_{i-1} from the aggregate of the two signatures. Note that the uniqueness of the signature allows us to ensure that the extraction is always successful for intermediate users.

Security Analysis. The following theorem establishes the security of our BLS-based PCN protocol and we defer the proof to the full version.

Theorem 5.2: Let $(\mathcal{P}_{\mathsf{NIZK},\mathcal{L}_{\mathsf{DL}}}, \mathcal{V}_{\mathsf{NIZK},\mathcal{L}_{\mathsf{DL}}})$ be a UC secure NIZK and let $(\mathbb{G}_0, \mathbb{G}_1, \mathbb{G}_t)$ be a CDH-hard bilinear group. Then the PCN protocol described in Figure 11 with access to $(\mathcal{F}_{\mathbb{B}}, \mathcal{F}_{\mathsf{smt}}, \mathcal{F}_{\mathsf{clock}})$ UC-realizes the functionality $\mathcal{F}_{PCN}$ (Figure 9) in the random oracle model.

D. On the Transaction Expiry Functionality

For our PCN protocols we assume that the blockchain offers an expiration functionality, where transactions can be set to expire after some time. In practice, the standard way to realise such an expiry time of a transaction tx_A is to set a `nLockTime` field in the script of another transaction tx_B that spends from tx_A. Transaction tx_B is accepted on the chain only if the number in its `nLockTime` field is larger than or equal to the current chain height. Therefore, if tx_A is on the chain and remains unspent until the expiry time (the number in `nLockTime` field of tx_B), transaction tx_B can be posted on the chain thereby spending from tx_A.

Although the setup transaction $tx_{\mathsf{Setup},i}$ is only kept locally and never posted on the chain, a malicious sender P_1 may post $tx_{\mathsf{Setup},i}$ on the blockchain and go offline. Parties P_i and P_{i+1} would end up locking their funds in 3-PC_i forever. To prevent such a DoS attack, the parties may generate a revoke transaction $tx_{\mathsf{revoke},i} := tx(3\text{-}\mathsf{PC}_i, 2\text{-}\mathsf{PC}_i, v_i)$ with its `nLockTime` set such that it is enforceable only after time $t_{\mathsf{Setup},i}$. This feature is available in most scripting languages. The revoke transaction $tx_{\mathsf{revoke},i}$ in this case can be posted on the blockchain after time $t_{\mathsf{Setup},i}$ and the parties P_i and P_{i+1} can get their funds back.

We could also adapt our protocol according to a recent work [51], which would let us eliminate the need for the `nLockTime` script entirely, while using time-lock puzzles [52] and a script to delay the spending of the output of a transaction, relative to the time of posting that transaction on chain[1].

[1]Bitcoin scripting language offers this feature with the script `CHECKSEQUENCEVERIFY`.

Input: For $i \in [n-1]$, parties P_i and P_{i+1} share a payment channel 2-PC_i with channel expiry t_i. For each channel 2-PC_i, party P_i has a secret signing key $sk_{p,i}^{(i)}$ and party P_{i+1} has a secret signing key $sk_{p,i}^{(i+1)}$. All parties are aware of the system parameters $\Delta \in \mathbb{N}$ and $T \in \mathbb{N}$. We consider party P_1 the sender of the payment, to play the role of a P_0 in the below protocol.

Payment Setup Phase

Parties P_0, P_i, P_{i+1} for $i \in [n-1]$ do the following:

1) They engage in the execution of the MPC protocol Γ for the following functionality:
   - Compute $(pk_i, sk_i) \leftarrow \Pi_{\mathcal{DS}}.\mathsf{KGen}(1^\lambda)$ and $(c_i, d_i) \leftarrow \Pi_{\mathsf{C}}.\mathsf{Commit}(1^\lambda, sk_i)$.
   - Sample uniformly at random $\left(sk_i^{(0)}, sk_i^{(i)}, sk_i^{(i+1)}\right)$ and $\left(d_i^{(0)}, d_i^{(i)}, d_i^{(i+1)}\right)$ such that $sk_i = sk_i^{(0)} \oplus sk_i^{(i)} \oplus sk_i^{(i+1)}$ and $d_i = d_i^{(0)} \oplus d_i^{(i)} \oplus d_i^{(i+1)}$.
   - Return $\left(pk_i, c_i, sk_i^{(0)}, d_i^{(0)}\right), \left(pk_i, c_i, sk_i^{(i)}, d_i^{(i)}\right), \left(pk_i, c_i, sk_i^{(i+1)}, d_i^{(i+1)}\right)$ to P_0, P_i and P_{i+1}, respectively.
2) Parties P_i and P_{i+1} do the following:
   - Generate $tx_{\mathsf{Setup},i} = tx(2\text{-}\mathsf{PC}_i, 3\text{-}\mathsf{PC}_i, v_i)$ with transaction expiry time $t_{\mathsf{Setup},i} = t_i - T$, where 3-$\mathsf{PC}_i := pk_i$
   - Run the MPC protocol Γ to generate $\sigma_{s,i} \leftarrow \Pi_{\mathcal{DS}}.\mathsf{Sign}\left(sk_{p,i}^{(i)} \oplus sk_{p,i}^{(i+1)}, tx_{\mathsf{Setup},i}\right)$
   - Send $(tx_{\mathsf{Setup},i}, \sigma_{s,i}, t_{\mathsf{Setup},i})$ to P_0

Payment Lock Phase

Parties $P_0, P_{i-1}, P_i, P_{i+1}$ for $i \in [2, n-1]$ generate a payment lock for party P_i by doing the following:

1) Parties P_0, P_{i-1}, P_i generate $tx_{\mathsf{pay},i-1} = tx(3\text{-}\mathsf{PC}_{i-1}, pk_i^*, v)$ with payment expiry time $\tilde{t}_{i-1}$ and parties P_0, P_i, P_{i+1} generate $tx_{\mathsf{pay},i} = tx\left(3\text{-}\mathsf{PC}_i, pk_{i+1}^*, v\right)$ with payment expiry time $\tilde{t}_i$, such that $\tilde{t}_{i-1} = \tilde{t}_i + \Delta$.
2) Party P_0 sends $(3\text{-}\mathsf{PC}_{i-1}, c_{i-1})$ to party P_{i+1} and sends $(3\text{-}\mathsf{PC}_i, c_i)$ to party P_{i-1}
3) Parties P_0, P_{i-1}, P_i and P_{i+1} engage in the MPC protocol Γ, to compute $\ell k_{i-1} \leftarrow \mathsf{Lock}\left(sk_{i-1}, tx_{\mathsf{pay},i-1}, sk_i, tx_{\mathsf{pay},i}\right)$. Specifically, the MPC takes as input $(pk_{i-1}, c_{i-1}, pk_i, c_i)$ from all parties these parties and as private inputs $\left\{\left(sk_{i-1}^{(0)}, d_{i-1}^{(0)}\right), \left(sk_i^{(0)}, d_i^{(0)}\right)\right\}$ from P_0, $\left(sk_{i-1}^{(i-1)}, d_{i-1}^{(i-1)}\right)$ from P_{i-1}, $\left\{\left(sk_{i-1}^{(i)}, d_{i-1}^{(i)}\right), \left(sk_i^{(i)}, d_i^{(i)}\right)\right\}$ from P_i and $\left(sk_i^{(i+1)}, d_i^{(i+1)}\right)$ from P_{i+1}. The MPC computes the following functionality:
   - Check whether $sk_{i-1}^{(0)} \oplus sk_{i-1}^{(i-1)} \oplus sk_{i-1}^{(i)} = \Pi_{\mathsf{C}}.\mathsf{Open}\left(d_{i-1}^{(0)} \oplus d_{i-1}^{(i-1)} \oplus d_{i-1}^{(i)}, c_{i-1}\right)$ and $sk_i^{(0)} \oplus sk_i^{(i)} \oplus sk_i^{(i+1)} = \Pi_{\mathsf{C}}.\mathsf{Open}\left(d_i^{(0)} \oplus d_i^{(i)} \oplus d_i^{(i+1)}, c_i\right)$ and abort otherwise.
   - Return $\ell k_{i-1} = \mathsf{Lock}\left(sk_{i-1}^{(0)} \oplus sk_{i-1}^{(i-1)} \oplus sk_{i-1}^{(i)}, tx_{\mathsf{pay},i-1}, sk_i^{(0)} \oplus sk_i^{(i)} \oplus sk_i^{(i+1)}, tx_{\mathsf{pay},i}\right)$ to P_i.
4) Parties $P_0, P_{n-1}, P_n, P_{n+1}$ where P_{n+1} is party P_1, run the above steps that returns $\ell k_{n-1} := \mathsf{Lock}(sk_{n-1}, tx_{\mathsf{pay},n-1}, sk_n, tx_{\mathsf{pay},n})$ to P_n, where $tx_{\mathsf{pay},n}$ is some message known to P_0 and P_n and sk_n is a secret key shared among P_0, P_{n+1} and P_n.

Payment Release Phase

The sender (parties P_0, P_{n+1}) initiates the payment release by jointly generating $\sigma_n \leftarrow \Pi_{\mathcal{DS}}.\mathsf{Sign}(sk_n, tx_{\mathsf{pay},n})$ with P_n via the MPC protocol Γ. Parties P_i for $i \in [n, 2]$ do the following:

1) Release the locks by computing $\sigma_{i-1} \leftarrow \mathsf{Unlock}\left(3\text{-}\mathsf{PC}_i, tx_{\mathsf{pay},i}, \sigma_i, 3\text{-}\mathsf{PC}_{i-1}, tx_{\mathsf{pay},i-1}\ell k_{i-1}\right)$.
2) Store $(tx_{\mathsf{Setup},i-1}, \sigma_{s,i-1}, tx_{\mathsf{pay},i-1}, \sigma_{i-1})$ to post on the blockchain if the case arises.

Fig. 7: Generic Payment Channel Network Protocol run between parties $P_1, \ldots, P_n$

E. On the Privacy our PCN Protocols

In both of our PCN protocols, the keys involved in the same payment path are not correlated to the eyes of an external observer. Thus, they achieve the same notion of on-chain privacy as in existing proposals based on scriptless scripts [3]. For off-chain privacy, our generic protocol reveals to the intermediate node P_i the identity of (i) the sender P_0, (ii) its neighbours P_{i-1} and P_{i+1}, and (iii) the neighbours' neighbours P_{i-2} and P_{i+2}, in that session. On the other hand, in our BLS-based protocol the corrupted intermediate users only learn the identities of the sender and of the direct neighbours (same as in [3]).

VI. Implementation and Efficiency Analysis

We implemented our BLS-based PCN protocol and we compare its performance against the Schnorr and ECDSA based PCN protocols from [3].

Instantiations. The three PCN protocols under consideration make use of a NIZK $(\mathcal{P}_{\mathsf{NIZK},\mathcal{L}_{\mathsf{DL}}}, \mathcal{V}_{\mathsf{NIZK},\mathcal{L}_{\mathsf{DL}}})$ for $\mathcal{L}_{\mathsf{DL}}$ which we instantiate with the Schnorr identification scheme [53] and make non-interactive using the Fiat-Shamir transformation [54]. While this scheme is not known to be UC-secure (the classical proof [55] is in the stand-alone model) we view this as a reasonable compromise in favor of a more efficient protocol.

TABLE I
NUMBER OF CALLS TO THE ALGORITHMS OF THE NIZK FOR $\mathcal{L}_{\mathsf{DL}}$ AND OF THE BLS SIGNATURE SCHEME FOR EACH PARTY DURING PAYMENT SETUP PHASE (FIGURE 11). HERE P_1'S OPERATIONS INCLUDE THOSE OF P_0 AND P_{n+1}.

Party	$\mathcal{P}_{\mathsf{NIZK},\mathcal{L}_{\mathsf{DL}}}$	$\mathcal{V}_{\mathsf{NIZK},\mathcal{L}_{\mathsf{DL}}}$	$\Pi_{\mathsf{BLS}}.\mathsf{KGen}$	$\Pi_{\mathsf{BLS}}.\mathsf{Sign}$	$\Pi_{\mathsf{BLS}}.\mathsf{Vf}$	$\Pi_{\mathsf{BLS}}.\mathsf{Agg}$	$\Pi_{\mathsf{BLS}}.\mathsf{VfAgg}$
P_1	$n+1$	$2n$	$n+1$	n	$n-1$	0	0
$P_i, i \in [2,n]$	2	4	2	4	2	2	0

We further note that the same trade-off is common to other works in the literature (e.g. [3]).

Efficiency and Comparison. We consider an n party payment path for the PCN. Given that the PCN protocols of [3] also follows a similar structure, we compare the timing cost of the operations performed in each phase: The payment setup, payment lock, and payment release. We measured the costs on a local machine with `1,8 GHz Octa-Core Intel Core i7-8550U` processor and `16 GB 2133 MHz LPDDR3` memory. We used the python libraries `charm` [56] and BLS signature library `blspy` [57] for evaluating the ECDSA/Schnorr-based PCN protocol [3] and our BLS-based PCN protocol, respectively. We ran the operations 10K times and the measurements (taken as the average) are reported in Table II. From [2], [3], we can also see that multi-hop HTLC based payments where $n = 5$, takes about 5 seconds to complete with a communication cost of 17 MB. This is much more expensive than the scriptless variants. For extensive comparisons with non-private PCN protocols and HTLC based protocols we refer the reader to [3].

In measuring the efficiency of the payment setup phase we also consider the time to sign a revoke transaction $tx_{\mathsf{revoke},i}$ for every $tx_{\mathsf{Setup},i}$ (recall that $tx_{\mathsf{revoke},i}$ was used to realize the expiry of the 3-party channel 3-PC_i). Notice that the payment lock phase is interactive, with 3 rounds of messages being sent to party P_i who sets the lock ℓk_{i-1}. While we do not consider the network latency in our measurements, this does not constitute the efficiency bottleneck of our approach.

TABLE II
COMPARISON OF THE COMPUTATIONAL TIME AND COMMUNICATION COSTS ACROSS PCN PROTOCOLS, FOR A PATH LENGTH OF n. TIMES ARE REPORTED IN MILLISECONDS (MS) AND COMMUNICATION COSTS ARE REPORTED IN BYTES.

Phase	**Resource**	**Schnorr** [3]	**ECDSA** [3]	**Our BLS**
Setup	Time	1.75 n	1.91 n	19.43 n
	Comm.	128 n	128 n	242 n
Lock	Time	4.58 $(n-1)$	38.81 $(n-1)$	13.78 $(n-1)$
	Comm.	256 $(n-1)$	416 $(n-1)$	196 $(n-1)$
Release	Time	0.0009 n	1.95 n	0.39 n
	Comm.	0	0	0

The more expensive setup phase for our BLS-based PCN protocol comes from the generation of a 3-party local channel. We provide a more detailed overview of the operations that constitute the setup phase and their computational costs in Table I. For the lock phase, our BLS-based protocol performs significantly better when compared to the ECDSA/Schnorr variants: The signature and key aggregation property of BLS signatures help significantly reduce the communication costs, while the uniqueness property of the signatures help in saving computation, as we discuss below.

Optimizations. We observe that, in the execution of two consequent locking phase, each intermediate party P_i has to generate two pairs signatures $\sigma_{i-1}^{(i)}$ on $tx_{\mathsf{Pay},i-1}$ and $\sigma_i^{(i)}$ on $tx_{\mathsf{Pay},i}$, respectively. Since BLS signatures are unique, we can safely generate each signature exactly once. Furthermore, reduce the cost of the sender by half, leveraging a similar observation. A naive implementation of our protocol (Figure 11) would require the sender to run $\Pi_{\mathsf{BLS}}.\mathsf{Sign}$ and $\Pi_{\mathsf{BLS}}.\mathsf{Agg}$, $2n$ times each, twice for each lock, during the lock phase. However, since the sender has to send signatures on the same transaction twice with respect to the same secret key, he only needs to run the signing algorithm once and re-send the same signature. For additional clarity, we report the computation cost in the payment lock phase in terms of the number of signature operations of a BLS signature in Table III.

TABLE III
NUMBER OF CALLS TO THE ALGORITHMS OF THE BLS SIGNATURE SCHEME FOR EACH PARTY DURING PAYMENT LOCK PHASE (FIGURE 11). HERE P_1'S OPERATIONS INCLUDE THOSE OF P_0 AND P_{n+1}.

Party	$\Pi_{\mathsf{BLS}}.\mathsf{Sign}$	$\Pi_{\mathsf{BLS}}.\mathsf{Vf}$	$\Pi_{\mathsf{BLS}}.\mathsf{Agg}$	$\Pi_{\mathsf{BLS}}.\mathsf{VfAgg}$
P_1	$n+1$	0	$n-1$	0
$P_i, i \in [2,n]$	2	2	2	1

VII. CONCLUSION

In this work we introduced the notion of lockable signatures as the cornerstone to construct PCNs. As a result we obtain a PCN protocol that does not require any special scripts and is compatible with *any signature* scheme. Our approach expands the scope of PCNs to signature schemes with extra properties (e.g. aggregatable, post-quantum secure, etc.) and facilitates payments across different chains. Our BLS-based PCN protocol is the first that is fully compatible with the BLS signature scheme and makes lightweight use of cryptographic tools, thus offering competitive performances. As an exciting next step, we plan to explore the large scale adoption of our BLS-based construction as to study the benefits offered by the signature aggregation in the context of PCNs.

ACKNOWLEDGMENTS

The work was also partially supported by the Deutsche Forschungsgemeinschaft (DFG – German Research Foundation) under SCHR 1415/4-1, and by the state of Bavaria at the Nuremberg Campus of Technology (NCT). The authors wish to thank Sebastian Faust, Kristina Hostakova and Siavash Riahi for insightful comments on an earlier draft of this work.

REFERENCES

[1] J. Poon and T. Dryja, *The bitcoin lightning network: Scalable off-chain instant payments*, 2016.

[2] G. Malavolta, P. Moreno-Sanchez, A. Kate, M. Maffei, and S. Ravi, "Concurrency and privacy with payment-channel networks," in *ACM CCS 2017*, B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, Eds., ACM Press, 2017, pp. 455–471. DOI: 10.1145/3133956.3134096.

[3] G. Malavolta, P. Moreno-Sanchez, C. Schneidewind, A. Kate, and M. Maffei, "Anonymous multi-hop locks for blockchain scalability and interoperability," in *NDSS 2019*, The Internet Society, Feb. 2019.

[4] C. Egger, P. Moreno-Sanchez, and M. Maffei, "Atomic multi-channel updates with constant collateral in bitcoin-compatible payment-channel networks," in *ACM CCS 2019*, L. Cavallaro, J. Kinder, X. Wang, and J. Katz, Eds., ACM Press, Nov. 2019, pp. 801–815. DOI: 10.1145/3319535.3345666.

[5] *Lightning network*, https://lightning.network/.

[6] *Raiden network*, https://raiden.network/.

[7] *Payment channels in ripple*, https://xrpl.org/use-payment-channels.html.

[8] R. W. F. Lai, V. Ronge, T. Ruffing, D. Schröder, S. A. K. Thyagarajan, and J. Wang, "Omniring: Scaling private payments without trusted setup," in *ACM CCS 2019*, L. Cavallaro, J. Kinder, X. Wang, and J. Katz, Eds., ACM Press, Nov. 2019, pp. 31–48. DOI: 10.1145/3319535.3345655.

[9] A. Poelstra, *Mimblewimble*, https://download.wpsoftware.net/bitcoin/wizardry/mimblewimble.pdf.

[10] D. Schwartz, N. Youngs, A. Britto, *et al.*, "The ripple protocol consensus algorithm," *Ripple Labs Inc White Paper*, vol. 5, no. 8, 2014.

[11] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in *2014 IEEE Symposium on Security and Privacy*, IEEE Computer Society Press, May 2014, pp. 459–474. DOI: 10.1109/SP.2014.36.

[12] *Scriptless scripts*, https://tlu.tarilabs.com/cryptography/scriptless-scripts/introduction-to-scriptless-scripts.html.

[13] M. F. Esgin, O. Ersoy, and Z. Erkin, *Post-quantum adaptor signatures and payment channel networks*, Cryptology ePrint Archive, Report 2020/845, https://eprint.iacr.org/2020/845, 2020.

[14] E. Tairi, P. Moreno-Sanchez, and M. Maffei, *Post-quantum adaptor signature for privacy-preserving off-chain payments*, Cryptology ePrint Archive, Report 2020/1345, https://eprint.iacr.org/2020/1345, 2020.

[15] *Chia network faq*, https://www.chia.net/faq/.

[16] *Pragmatic signature aggregation with bls*, https://ethresear.ch/t/pragmatic-signature-aggregation-with-bls/2105.

[17] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the Weil pairing," in *ASIACRYPT 2001*, C. Boyd, Ed., ser. LNCS, vol. 2248, Springer, Heidelberg, Dec. 2001, pp. 514–532. DOI: 10.1007/3-540-45682-1_30.

[18] D. Boneh, M. Drijvers, and G. Neven, "Compact multi-signatures for smaller blockchains," in *ASIACRYPT 2018, Part II*, T. Peyrin and S. Galbraith, Eds., ser. LNCS, vol. 11273, Springer, Heidelberg, Dec. 2018, pp. 435–464. DOI: 10.1007/978-3-030-03329-3_15.

[19] L. Ducas, A. Durmus, T. Lepoint, and V. Lyubashevsky, "Lattice signatures and bimodal Gaussians," in *CRYPTO 2013, Part I*, R. Canetti and J. A. Garay, Eds., ser. LNCS, vol. 8042, Springer, Heidelberg, Aug. 2013, pp. 40–56. DOI: 10.1007/978-3-642-40041-4_3.

[20] W. A. A. Torres, R. Steinfeld, A. Sakzad, J. K. Liu, V. Kuchta, N. Bhattacharjee, M. H. Au, and J. Cheng, "Post-quantum one-time linkable ring signature and application to ring confidential transactions in blockchain (lattice ringct v1. 0)," in *Australasian Conference on Information Security and Privacy*, Springer, 2018, pp. 558–576.

[21] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, "CRYSTALS-Dilithium: A lattice-based digital signature scheme," *IACR TCHES*, vol. 2018, no. 1, pp. 238–268, 2018, https://tches.iacr.org/index.php/TCHES/article/view/839, ISSN: 2569-2925. DOI: 10.13154/tches.v2018.i1.238-268.

[22] D. J. Bernstein, A. Hülsing, S. Kölbl, R. Niederhagen, J. Rijneveld, and P. Schwabe, "The sphincs+ signature framework," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 2129–2146.

[23] *From post-quantum cryptography to post-quantum blockchains and cryptocurrencies: An introduction*, https://medium.com/abelian/from-post-quantum-cryptography-to-post-quantum-blockchains-and-cryptocurrencies-an-introduction-eb0b50ed129a.

[24] *Understanding serenity*, https://blog.ethereum.org/2015/12/24/understanding-serenity-part-i-abstraction/.

[25] *Hcash*, https://h.cash/.

[26] *Zcash faq*, https://z.cash/support/faq/.

[27] R. Cleve, "Limits on the security of coin flips when half the processors are faulty (extended abstract)," in *18th ACM STOC*, ACM Press, May 1986, pp. 364–369. DOI: 10.1145/12130.12168.

[28] M. Green and I. Miers, "Bolt: Anonymous payment channels for decentralized currencies," in *ACM CCS 2017*, B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, Eds., ACM Press, 2017, pp. 473–489. DOI: 10.1145/3133956.3134093.

[29] J. Kilian, "A note on efficient zero-knowledge proofs and arguments (extended abstract)," in *24th ACM STOC*, ACM Press, May 1992, pp. 723–732. DOI: 10.1145/129712.129782.

[30] S. Micali, "CS proofs (extended abstracts)," in *35th FOCS*, IEEE Computer Society Press, Nov. 1994, pp. 436–453. DOI: 10.1109/SFCS.1994.365746.

[31] R. Gennaro, C. Gentry, B. Parno, and M. Raykova, "Quadratic span programs and succinct NIZKs without PCPs," in *EUROCRYPT 2013*, T. Johansson and P. Q. Nguyen, Eds., ser. LNCS, vol. 7881, Springer, Heidelberg, May 2013, pp. 626–645. DOI: 10.1007/978-3-642-38348-9_37.

[32] S. Dziembowski, L. Eckey, S. Faust, and D. Malinowski, "Perun: Virtual payment hubs over cryptocurrencies," in *2019 IEEE Symposium on Security and Privacy*, IEEE Computer Society Press, May 2019, pp. 106–123. DOI: 10.1109/SP.2019.00020.

[33] S. Dziembowski, S. Faust, and K. Hostáková, "General state channel networks," in *ACM CCS 2018*, D. Lie, M. Mannan, M. Backes, and X. Wang, Eds., ACM Press, Oct. 2018, pp. 949–966. DOI: 10.1145/3243734.3243856.

[34] A. Miller, I. Bentov, S. Bakshi, R. Kumaresan, and P. McCorry, "Sprites and state channels: Payment networks that go faster than lightning," in *FC 2019*, I. Goldberg and T. Moore, Eds., ser. LNCS, vol. 11598, Springer, Heidelberg, Feb. 2019, pp. 508–526. DOI: 10.1007/978-3-030-32101-7_30.

[35] L. Eckey, S. Faust, K. Hostáková, and S. Roos, *Splitting payments locally while routing interdimensionally*, Cryptology ePrint Archive, Report 2020/555, https://eprint.iacr.org/2020/555, 2020.

[36] L. Aumayr, O. Ersoy, A. Erwig, S. Faust, K. Hostakova, M. Maffei, P. Moreno-Sanchez, and S. Riahi, "Generalized bitcoin-compatible channels.," *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 476, 2020.

[37] S. Dziembowski, L. Eckey, S. Faust, J. Hesse, and K. Hostáková, "Multi-party virtual state channels," in *EUROCRYPT 2019, Part I*, Y. Ishai and V. Rijmen, Eds., ser. LNCS, vol. 11476, Springer, Heidelberg, May 2019, pp. 625–656. DOI: 10.1007/978-3-030-17653-2_21.

[38] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, "Aggregate and verifiably encrypted signatures from bilinear maps," in *EUROCRYPT 2003*, E. Biham, Ed., ser. LNCS, vol. 2656, Springer, Heidelberg, May 2003, pp. 416–432. DOI: 10.1007/3-540-39200-9_26.

[39] *Rogue key attacks*, https://medium.com/@coolcottontail/rogue-key-attack-in-bls-signature-and-harmony-security-eac1ea2370ee.

[40] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, IEEE, 2001, pp. 136–145.

[41] R. Canetti, Y. Dodis, R. Pass, and S. Walfish, "Universally composable security with global setup," in *TCC 2007*, S. P. Vadhan, Ed., ser. LNCS, vol. 4392, Springer, Heidelberg, Feb. 2007, pp. 61–85. DOI: 10.1007/978-3-540-70936-7_4.

[42] S. Goldwasser, S. Micali, and R. L. Rivest, "A digital signature scheme secure against adaptive chosen-message attacks," *SIAM Journal on Computing*, vol. 17, no. 2, pp. 281–308, Apr. 1988.

[43] M. Backes and D. Hofheinz, "How to break and repair a universally composable signature functionality," in *ISC 2004*, K. Zhang and Y. Zheng, Eds., ser. LNCS, vol. 3225, Springer, Heidelberg, Sep. 2004, pp. 61–72.

[44] A. De Santis, S. Micali, and G. Persiano, "Non-interactive zero-knowledge proof systems," in *Conference on the Theory and Application of Cryptographic Techniques*, Springer, 1987, pp. 52–72.

[45] J. Camenisch, S. Krenn, and V. Shoup, "A framework for practical universally composable zero-knowledge protocols," in *ASIACRYPT 2011*, D. H. Lee and X. Wang, Eds., ser. LNCS, vol. 7073, Springer, Heidelberg, Dec. 2011, pp. 449–467. DOI: 10.1007/978-3-642-25385-0_24.

[46] R. Canetti, "Security and composition of multiparty cryptographic protocols," *Journal of Cryptology*, vol. 13, no. 1, pp. 143–202, Jan. 2000. DOI: 10.1007/s001459910006.

[47] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *CRYPTO 2012*, R. Safavi-Naini and R. Canetti, Eds., ser. LNCS, vol. 7417, Springer, Heidelberg, Aug. 2012, pp. 643–662. DOI: 10.1007/978-3-642-32009-5_38.

[48] J. Katz, U. Maurer, B. Tackmann, and V. Zikas, "Universally composable synchronous computation," in *TCC 2013*, A. Sahai, Ed., ser. LNCS, vol. 7785, Springer, Heidelberg, Mar. 2013, pp. 477–498. DOI: 10.1007/978-3-642-36594-2_27.

[49] J.-S. Coron and D. Naccache, "Boneh et al.'s k-element aggregate extraction assumption is equivalent to the Diffie-Hellman assumption," in *ASIACRYPT 2003*, C.-S. Laih, Ed., ser. LNCS, vol. 2894, Springer, Heidelberg, 2003, pp. 392–397. DOI: 10.1007/978-3-540-40061-5_25.

[50] O. Goldreich and L. A. Levin, "A hard-core predicate for all one-way functions," in *21st ACM STOC*, ACM Press, May 1989, pp. 25–32. DOI: 10.1145/73007.73010.

[51] "Personal communication," *To appear at CCS 2020*,

[52] R. L. Rivest, A. Shamir, and D. A. Wagner, "Time-lock puzzles and timed-release crypto," Cambridge, MA, USA, Tech. Rep., 1996.

[53] C.-P. Schnorr, "Efficient identification and signatures for smart cards," in *CRYPTO'89*, G. Brassard, Ed., ser. LNCS, vol. 435, Springer, Heidelberg, Aug. 1990, pp. 239–252. DOI: 10.1007/0-387-34805-0_22.

[54] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *CRYPTO'86*, A. M. Odlyzko, Ed., ser. LNCS, vol. 263, Springer, Heidelberg, Aug. 1987, pp. 186–194. DOI: 10.1007/3-540-47721-7_12.

[55] D. Pointcheval and J. Stern, "Security proofs for signature schemes," in *EUROCRYPT'96*, U. M. Maurer, Ed., ser. LNCS, vol. 1070, Springer, Heidelberg, May 1996, pp. 387–398. DOI: 10.1007/3-540-68339-9_33.

[56] *Charm*, https://jhuisi.github.io/charm/index.html.

$\underline{\mathsf{EWHi\text{-}CMA}_{\mathcal{A},\mathsf{LS}}(\lambda)}$

$Q := \tilde{Q} := \emptyset$
$(pk, sk) \leftarrow \Pi_{\mathcal{DS}}.\mathsf{KGen}(1^\lambda)$
$(\tilde{pk}, \tilde{sk}) \leftarrow \Pi_{\mathcal{DS}}.\mathsf{KGen}(1^\lambda)$
$\mathbb{O} := \{\mathsf{Sign}\mathcal{O}(sk, \cdot)\mathsf{Sign}\mathcal{O}(\tilde{sk}, \cdot)\}$
$(m, \tilde{m}, \mathsf{state}) \leftarrow \mathcal{A}^{\mathbb{O}}(pk, \tilde{pk})$
$\ell k \leftarrow \mathsf{Lock}(sk, m, \tilde{sk}, \tilde{m})$
$\sigma^* \leftarrow \mathcal{A}^{\mathbb{O}}(\mathsf{state}, \ell k)$
$b_0 := \Pi_{\mathcal{DS}}.\mathsf{Vf}(pk, m, \sigma^*) = 1$
$b_1 := (m \notin Q \wedge \tilde{m} \notin \tilde{Q})$
return $b_0 \wedge b_1$

$\underline{\mathsf{Sign}\mathcal{O}(sk, m)}$

$\sigma \leftarrow \Pi_{\mathcal{DS}}.\mathsf{Sign}(sk, m)$
$Q := Q \cup \{m\}$
return σ

$\underline{\mathsf{Sign}\mathcal{O}(\tilde{sk}, \tilde{m})}$

$\tilde{\sigma} \leftarrow \Pi_{\mathcal{DS}}.\mathsf{Sign}(\tilde{sk}, \tilde{m})$
$\tilde{Q} := \tilde{Q} \cup \{\tilde{m}\}$
return $\tilde{\sigma}$

Fig. 8: Experiment for weak hiding of lockable signatures

[57] *Chia-network/bls-signatures*, https://github.com/Chia-Network/bls-signatures/tree/master/python-bindings.

[58] L. Cavallaro, J. Kinder, X. Wang, and J. Katz, Eds., *ACM CCS 2019*, ACM Press, Nov. 2019.

[59] B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, Eds., *ACM CCS 2017*, ACM Press, 2017.

APPENDIX

A. More Formal Definitions - Lockable Signatures

We define a weaker version of the hiding property, which is useful for one of our schemes. Such a property ensures that an adversary, who is given a correctly generated lock, cannot output the locked signature without knowing the locking signature. This is formally captured in the experiment in Figure 8 which is identical to EHi-CMA except now the adversary is always given the correctly generated lock and has to output a valid locked signature to win the experiment. The formal definition is given in the following.

Definition 7 (Weak Hiding): A lockable signature LS is *weakly hiding* if there exists a negligible function negl such that for all $\lambda \in \mathbb{N}$ and for all PPT adversaries $\mathcal{A}$ it holds that

$$\Pr[\mathsf{EWHi\text{-}CMA}_{\mathcal{A},\mathsf{LS}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$$

where the experiment EHi-CMA is defined in Figure 8.

B. More Proofs - Theorem 3.1

We recall the aggregate extraction assumption from [38] (Definition 4.3).

Definition 8 (Aggregate Extraction problem): The *aggregate extraction problem* for a bilinear pairing group $\mathbb{G}_0, \mathbb{G}_1, \mathbb{G}_t$ of order q is said to be hard if there exists a negligible function negl, for all $\lambda, k \in \mathbb{N}$ such that $k = \mathsf{poly}(\lambda)$ for some polynomial poly, and all PPT adversaries $\mathcal{A}$, the following holds,

$$\Pr\left[\mathsf{ExpAgExt}_{\mathcal{A},\mathbb{G}_0,\mathbb{G}_1,\mathbb{G}_t}(1^\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$$

where ExpAgExt is defined in Figure 10.

Proof 2 (Proof of Theorem 3.1): We will prove that the hardness of aggregate extraction assumption holds if and only if the hardness of chained aggregate extraction assumption holds.

Only if. We first show that "only if" condition holds. That is, aggregate extraction assumption for $k = 2$ implies the chained aggregate extraction assumption. Assume towards the contrary that chained aggregate extraction assumption does not hold. This means there exists a PPT adversary $\mathcal{A}$ that wins the ExpChAgExt experiment with non-negligible probability. We construct a reduction algorithm $\mathcal{R}$ that participates in the ExpAgExt experiment while making use of $\mathcal{A}$ as a sub-routine.

The reduction $\mathcal{R}$ gets as input $(\{G_i, H_i\}_{i\in[2]}, \sigma)$. It sets $\sigma_1 := \sigma$ and does the following,

for $\ell \in [2, n-1]$ **do**
$r_{\ell+1}, s_{\ell+1} \leftarrow \mathbb{Z}_q^*$
$G_{\ell+1} := \left(G_{\ell-1} \cdot g_1^{r_{\ell+1}}\right)^{s_{\ell+1}^{-1}}$
$H_{\ell+1} := (H_{\ell-1})^{s_{\ell+1}}$
$\sigma_\ell := \sigma_{\ell-1} \cdot H_{\ell-1}^{r_{\ell+1}}$

It invokes the adversary $\mathcal{A}$ with inputs $(\{G_i, H_i\}_{i\in[n]}, \{\sigma_i\}_{i\in[n-1]})$. It receives a (σ^*, j^*) as output from $\mathcal{A}$. Reduction $\mathcal{R}$ checks if $j^* \in [n]$, if not it aborts.

If j^* is odd, reduction $\mathcal{R}$ first computes computes $F := H_1^{(r_3+s_3r_5+\cdots+s_{j^*-2}r_{j^*})}$. The reduction $\mathcal{R}$ outputs $(\sigma^*/F, 1)$ as its output to its own challenger.

If j^* is even, reduction $\mathcal{R}$ first computes $F := H_2^{(r_4+s_4r_6+\cdots+s_{j^*-2}r_{j^*})}$. The reduction $\mathcal{R}$ outputs $(\sigma^*/F, 2)$ as its output to its own challenger.

To see why the view of $\mathcal{A}$ is correctly simulated, notice that $(r_3, s_3, r_4, s_4, \ldots, r_n, s_n)$ are chosen uniformly at random from $\mathbb{Z}_p^*$, the values $(\{G_i, H_i\}_{i\in[n]})$ are uniformly distributed. By induction, we have the base case $G_1 := g_1^{x_1}, H_1 := g_0^{y_1}, G_2 := g_1^{x_2}, H_2 := g_0^{y_2}, \sigma_1 := g_0^{x_1y_1+x_2y_2}$. By induction hypothesis, we have $G_i := g_1^{x_i}, H_i := g_0^{y_i}, \sigma_{i-1} := g_0^{x_iy_i+x_{i-1}y_{i-1}}$.

$$G_{i+1} := g_1^{\left(\frac{x_{i-1}+r_{i+1}}{s_{i+1}}\right)} := g_1^{x_{i+1}}, \text{ where } x_{i+1} := \frac{x_{i-1}+r_{i+1}}{s_{i+1}}$$

$$H_{i+1} := g_0^{y_{i-1}s_{i+1}} := g_0^{y_{i+1}}, \text{ where } y_{i+1} := y_{i-1}s_{i+1}$$

$$\sigma_i := \sigma_{i-1} \cdot H_{i-1}^{r_{i+1}} := g_0^{x_iy_i+x_{i-1}y_{i-1}+y_{i-1}r_{i+1}}$$
$$:= g_0^{x_iy_i+y_{i-1}s_{i+1}\left(\frac{x_{i-1}+r_{i+1}}{s_{i+1}}\right)} := g_0^{x_iy_i+y_{i+1}x_{i+1}}$$

By induction, we can see that the simulation by the reduction $\mathcal{R}$ is faithful to the execution in the experiment ExpChAgExt.

Let us analyse the case for j^* is odd (the even case of analogous) why the reduction $\mathcal{R}$'s output wins the experiment ExpAgExt. Notice that,

$$\sigma_{j^*} := g_0^{y_1\left(x_1+r_3+\sum_{i=5}^{j^*} s_{i-2}r_i\right)+y_2\left(x_2+r_4+\sum_{i=6}^{j^*-1} s_{i-2}r_i\right)}$$

$$\sigma^* := g_0^{y_1\left(x_1+r_3+\sum_{i=5}^{j^*} s_{i-2}r_i\right)} := g_0^{y_1x_1+y_1\left(r_3+\sum_{i=5}^{j^*} s_{i-2}r_i\right)}$$

$$F := H_1^{\left(r_3+\sum_{i=5}^{j^*} s_{i-2}r_i\right)} := g_0^{y_1\left(r_3+\sum_{i=5}^{j^*} s_{i-2}r_i\right)}$$

Thus, $\sigma^*/F := g_0^{y_1x_1}$. Therefore $(\sigma^*/F, 1)$ is a winning output from $\mathcal{R}$ with respect to ExpAgExt.

Clearly $\mathcal{R}$ is efficient. Therefore, given our initial assumption that $\mathcal{A}$ outputs a winning (σ^*, j^*) with non-negligible

Open Channel: On input $(\mathsf{OpChannel}, c_{\langle u,u'\rangle}, v, u', t, f)$ from a user u, the functionality checks whether $c_{\langle u,u'\rangle}$ is well-formed (contains valid identifiers and it is not a duplicate) and eventually sends $(c_{\langle u,u'\rangle}, v, t, f)$ to u', who can either abort or authorize the operation. In the latter case, the functionality appends the tuple $(c_{\langle u,u'\rangle}, v, t, f)$ to $\mathbb{B}$ and the tuple $(c_{\langle u,u'\rangle}, v, t, h)$ to $\mathcal{P}$, for some random h. The functionality returns h to u and u'.

Close Channel: On input $(\mathsf{ClChannel}, c_{\langle u,u'\rangle}, h)$ from either user u or u', the ideal functionality parses $\mathbb{B}$ for an entry $(c_{\langle u,u'\rangle}, v, t, f)$ and $\mathcal{P}$ for an entry $(c_{\langle u,u'\rangle}, v', t', h)$, for $h \neq \perp$. If $c_{\langle u,u'\rangle} \in \mathcal{L}$ or either t or t' is past the current time, the functionality aborts. Otherwise the functionality adds $(c_{\langle u,u'\rangle}, u', v', t')$ to $\mathbb{B}$ and adds $c_{\langle u,u'\rangle}$ to $\mathcal{L}$. The functionality then notifies other parties with the message $(c_{\langle u,u'\rangle}, \perp, h)$.

Pay: On input $(\mathsf{payChannel}, v, c_{\langle u_0,u_1\rangle}, \ldots, c_{\langle u_n,u_{n+1}\rangle}), (t_0, \ldots, t_n))$ from a user u_0, the functionality executes the following interactive protocol:

- For all $i \in 1, \ldots, (n+1)$, it samples a random h_i and parses $\mathbb{B}$ for an entry of the form $(c_{\langle u_{i-1},u_i'\rangle}, v_i, t_i', f_i)$. If such an entry does exist, it sends the tuple $(u_0, h_{i-1}, h_i, h_{i+1}, h_{i+2}, c_{\langle u_{i-2},u_{i-1}\rangle}, c_{\langle u_{i-1},u_i\rangle}, c_{\langle u_i,u_{i+1}\rangle}, c_{\langle u_{i+1},u_{i+2}\rangle}, v - \sum_{j=i}^{n} f_j, t_{i-2}, t_{i-1}, t_i, t_{i+1})$to the user u_i via an anonymous channel (for the specific case of the receiver the tuple is only $(h_{n+1}, c_{\langle u_n,u_{n+1}\rangle}, v, t_n)$). Then it checks whether for all entries of the form $(c, v', \cdot, \cdot) \in \mathcal{P}$ it holds that $v' \geq (v - \sum_{j=i}^{n} f_j)$ and that $t_{i-1} \geq t_i$. If this is the case it adds $d_i = (c_{\langle u_{i-1},u_i\rangle}, (v_i' - (v - \sum_{j=i}^{n} f_j), t_i, \perp)$ to $\mathcal{P}$, where $(c_{\langle u_{i-1},u_i\rangle}, v_i', \cdot, \cdot) \in \mathcal{P}$ is the entry with the lowest v_i'. If any of the conditions above is not met, it removes from $\mathcal{P}$ all the entries d_i added in this phase and aborts.
- For all $i \in \{(n+1), \ldots, 1\}$, it queries all u_i with (h_i, h_{i+1}), through an anonymous channel. Each user can reply with either $\top$ or $\perp$. Let j be the index of the user that returns $\perp$ such that for all $i > j : u_i$ returned $\top$. If no user returned $\perp$ we set $j = 0$.
- For all $i \in \{j+1, \ldots, n\}$ the ideal functionality updates $d_i \in \mathcal{P}$ (defined as above) to $(-, -, -, h_i)$ and notifies the user of the success of the operation with with some distinguished message $(\mathtt{success}, h_i, h_{i+1})$. For all $i \in \{0 \ldots, j\}$ (if $j \neq 0$), it removes d_i from $\mathcal{P}$ and notifies the user with the message $(\perp, h_i, h_{i+1})$.

Fig. 9: Ideal functionality $\mathcal{F}_{PCN}$ for PCNs (with weaker off-chain privacy).

$\underline{\mathsf{ExpAgExt}_{\mathcal{A},\mathbb{G}_0,\mathbb{G}_1,\mathbb{G}_t}(1^\lambda, k)}$

$x_i, y_i \leftarrow \mathbb{Z}_q, \forall i \in [k]$

$\sigma \leftarrow g_0^{\sum_{i\in[k]} x_i y_i}$

$(\sigma', I') \leftarrow \mathcal{A}(\{g_1^{x_i}, g_0^{y_i}\}_{i\in[k]}, \sigma)$

$b_0 := (\emptyset \neq I' \subset [k])$

$b_1 := \left(\sigma' = g_0^{\sum_{i\in I'} x_i y_i}\right)$

return $b_0 \wedge b_1$

Fig. 10: Aggregate Extraction experiment

probability in ExpAgExt, we have that $\mathcal{R}$ wins the experiment ExpChAgExt with the same non-negligible probability. But this means $\mathcal{R}$ is a PPT algorithm that breaks the aggregate extraction assumption, which is a contradiction. Hence it must be the case that no such PPT $\mathcal{A}$ can exist. This concludes the "only if" direction.

If. Proving this direction is trivial. Here we build a reduction algorithm $\mathcal{R}$ that plays in ExpChAgExt and runs a sub-routine $\mathcal{A}$ that is participating in ExpAgExt with $k = 2$. The reduction $\mathcal{R}$ receives a chained aggregate challenge and simply chooses one of the $n-1$ aggregates at random and gives it to $\mathcal{A}$. Now, whatever signature $\mathcal{A}$ outputs, $\mathcal{R}$ outputs the same signature along with the index of the chosen aggregate. It is easy to see that the simulation by $\mathcal{R}$ for the adversary $\mathcal{A}$ is faithful to ExpAgExt. If $\mathcal{A}$ outputs a valid extraction for the aggregate it was given as input with non-negligible probability, the reduction $\mathcal{R}$ is able to win ExpChAgExt with the same non-negligible probability. By contradiction, this proves the "if" direction.

Input: For $i \in [n-1]$, parties P_i and P_{i+1} share a payment channel 2-PC_i with channel expiry t_i. For each channel 2-PC_i, party P_i has a secret signing key $sk_{p,i}^{(i)}$ and party P_{i+1} has a secret signing key $sk_{p,i}^{(i+1)}$. All parties are aware of the system parameters $\Delta \in \mathbb{N}$ and $T \in \mathbb{N}$.

We consider party P_1 the sender of the payment, to play the role of a P_0 in the below protocol.

Payment Setup Phase

Parties P_0, P_i, P_{i+1} for $i \in [n-1]$ (Party P_{n+1}, is just a copy of party P_n) do the following:

1) Each party $P_j, j \in \{0, i, i+1\}$, executes the following steps:
   - Sample $(pk_i^{(j)}, sk_i^{(j)}) \leftarrow \Pi_{\mathsf{BLS}}.\mathsf{KGen}(1^\lambda)$
   - Generate $\pi_i^{(j)} \leftarrow \mathcal{P}_{\mathsf{NIZK},\mathcal{L}_{\mathsf{DL}}}\left(pk_i^{(j)}, sk_i^{(j)}\right)$ and send $\left(pk_i^{(j)}, \pi_i^{(j)}\right)$ to other parties
   - Verify if the proof $\pi_i^{(k)}, k \in \{0, i, i+1\}/j$ of other parties is correct by checking $\mathcal{V}_{\mathsf{NIZK},\mathcal{L}_{\mathsf{DL}}}\left(pk_i^{(k)}, \pi_i^{(k)}\right) = 1$. If any one of the proof does not verify, the party aborts.
   - The final public key is $pk_i \leftarrow \prod_{j \in \{0,i,i+1\}} pk_i^{(j)}$
   - Return $\left(pk_i, sk_i^{(0)}\right)$, $\left(pk_i, sk_i^{(i)}\right)$ and $\left(pk_i, sk_i^{(i+1)}\right)$ to P_0, P_i and P_{i+1}, respectively.
2) Parties P_i and P_{i+1} do the following:
   - Generate $tx_{\mathsf{Setup},i} = tx\,(2\text{-}\mathsf{PC}_i, 3\text{-}\mathsf{PC}_i, v_i)$ with transaction expiry time $t_{\mathsf{Setup},i} = t_i - T$, where 3-$\mathsf{PC}_i := pk_i$
   - Generate $\sigma_{s,i} \leftarrow \Pi_{\mathsf{BLS}}.\mathsf{Sign}\left(sk_{p,i}^{(i)} + sk_{p,i}^{(i+1)}, tx_{\mathsf{Setup},i}\right)$
   - Send $(tx_{\mathsf{Setup},i}, \sigma_{s,i}, t_{\mathsf{Setup},i})$ to P_0 and P_0 verifies if the transaction is correctly formed and checks if $\Pi_{\mathsf{BLS}}.\mathsf{Vf}(2\text{-}\mathsf{PC}_i, tx_{\mathsf{Setup},i}, \sigma_{s,i}) = 1$, and aborts otherwise.

Payment Lock Phase

Parties $P_0, P_{i-1}, P_i, P_{i+1}$ for $i \in [2, n-1]$ generate a payment lock for party P_i by doing the following:

1) Parties P_0, P_{i-1}, P_i generate $tx_{\mathsf{pay},i-1} = tx(3\text{-}\mathsf{PC}_{i-1}, pk_i^*, v)$ with payment expiry time $\tilde{t}_{i-1}$ and parties P_0, P_i, P_{i+1} generate $tx_{\mathsf{pay},i} = tx(3\text{-}\mathsf{PC}_i, pk_{i+1}^*, v)$ with payment expiry time $\tilde{t}_i$, such that $\tilde{t}_{i-1} = \tilde{t}_i + \Delta$.
2) Party P_0 sends 3-PC_{i-1} to party P_{i+1} and sends 3-PC_i to party P_{i-1}
3) Parties P_0, P_{i-1}, P_i and P_{i+1} engage in the following interactive protocol to compute $\ell k_{i-1} \leftarrow \mathsf{Lock}\,(sk_{i-1}, tx_{\mathsf{pay},i-1}, sk_i, tx_{\mathsf{pay},i})$. The common input all parties is (pk_{i-1}, pk_i) and as private inputs P_0 has $\left(sk_{i-1}^{(0)}, sk_i^{(0)}\right)$, P_{i-1} has $sk_{i-1}^{(i-1)}$, P_i has $\left(sk_{i-1}^{(i)}, sk_i^{(i)}\right)$ and P_{i+1} has $sk_i^{(i+1)}$.
   - (Round 1): P_{i-1} generates $\sigma_{i-1}^{(i-1)} \leftarrow \Pi_{\mathsf{BLS}}.\mathsf{Sign}\left(sk_{i-1}^{(i-1)}, tx_{\mathsf{pay},i-1}\right)$ and sends $\sigma_{i-1}^{(i-1)}$ to P_i. Party P_i aborts if $\Pi_{\mathsf{BLS}}.\mathsf{Vf}\left(pk_{i-1}^{(i-1)}, tx_{\mathsf{pay},i-1}, \sigma_{i-1}^{(i-1)}\right) = 0$ and continues otherwise.
   - (Round 2): P_{i+1} generates $\sigma_i^{(i+1)} \leftarrow \Pi_{\mathsf{BLS}}.\mathsf{Sign}\left(sk_i^{(i+1)}, tx_{\mathsf{pay},i}\right)$ and sends $\sigma_i^{(i+1)}$ to P_i. Party P_i aborts if $\Pi_{\mathsf{BLS}}.\mathsf{Vf}\left(pk_i^{(i+1)}, tx_{\mathsf{pay},i}, \sigma_i^{(i+1)}\right) = 0$ and continue otherwise.
   - (Round 3): P_0 generates $\sigma_{i-1}^{(0)} \leftarrow \Pi_{\mathsf{BLS}}.\mathsf{Sign}\left(sk_{i-1}^{(0)}, tx_{\mathsf{pay},i-1}\right)$ and $\sigma_i^{(0)} \leftarrow \Pi_{\mathsf{BLS}}.\mathsf{Sign}\left(sk_i^{(0)}, tx_{\mathsf{pay},i}\right)$. It computes $\sigma' \leftarrow \Pi_{\mathsf{BLS}}.\mathsf{Agg}(\sigma_{i-1}^{(0)}, \sigma_i^{(0)})$ and sends σ' to P_i. Party P_i aborts if $\Pi_{\mathsf{BLS}}.\mathsf{VfAgg}\left(\left\{pk_{i-1}^{(0)}, pk_i^{(0)}\right\}, \{tx_{\mathsf{pay},i-1}, tx_{\mathsf{pay},i}\}, \sigma'\right) = 0$ and continues otherwise.
   - Party P_i generates signatures $\sigma_{i-1}^{(i)} \leftarrow \Pi_{\mathsf{BLS}}.\mathsf{Sign}\left(sk_{i-1}^{(i)}, tx_{\mathsf{pay},i-1}\right)$ and $\sigma_i^{(i)} \leftarrow \Pi_{\mathsf{BLS}}.\mathsf{Sign}\left(sk_i^{(i)}, tx_{\mathsf{pay},i}\right)$. It then sets $\ell k_{i-1} := \Pi_{\mathsf{BLS}}.\mathsf{Agg}\left(\sigma_{i-1}^{(i-1)}, \sigma_{i-1}^{(i)}, \sigma_i^{(i+1)}, \sigma_i^{(i)}, \sigma'\right)$.
4) Parties $P_0, P_{n-1}, P_n, P_{n+1}$ where P_{n+1} is party P_1, run the above steps that returns $\ell k_{n-1} := l_1 \cdot l_2$ to P_n where $l_1 := \sigma_{n-1}$ and $l_2 := \sigma_n$. Here σ_{n-1} is a signature on transaction $tx_{\mathsf{pay},n-1} := tx(3\text{-}\mathsf{PC}_{n-1}, pk_n^*, v)$ and $\sigma_n \leftarrow \Pi_{\mathsf{BLS}}.\mathsf{Sign}(sk_n, tx_{\mathsf{pay},n})$ is a signature on some message $tx_{\mathsf{pay},n}$ (known to P_0, P_{n+1} and P_n) under public key pk_n.

Payment Release Phase

The sender (parties P_0, P_{n+1}) initiates the payment release by jointly generating σ_n with P_n. Parties P_i for $i \in [n, 2]$ do the following:

1) Check if $\Pi_{\mathsf{BLS}}.\mathsf{Vf}(pk_i, tx_{\mathsf{pay},i}, \sigma_i) = 1$ and if so, release the locks by computing $\sigma_{i-1} := \ell k_{i-1}/\sigma_i := \mathsf{Unlock}(pk_{i-1}, tx_{\mathsf{pay},i-1}, pk_i, tx_{\mathsf{pay},i}, \sigma_i, \ell k_{i-1})$.
2) Store $(tx_{\mathsf{Setup},i-1}, \sigma_{s,i-1}, tx_{\mathsf{pay},i-1}, \sigma_{i-1})$ to post on the blockchain if the case arises.

Fig. 11: BLS based Payment Channel Network Protocol run between parties $P_1, \ldots, P_n$