Anonymous Multi-Hop Locks for Blockchain Scalability and Interoperability

Giulio Malavolta[*§], Pedro Moreno-Sanchez[*¶†], Clara Schneidewind[†], Aniket Kate[‡], Matteo Maffei[†]
[§]Friedrich-Alexander-University Erlangen-Nürnberg, [†]TU Wien, [‡] Purdue University

***Abstract*—Tremendous growth in cryptocurrency usage is exposing the inherent scalability issues with permissionless blockchain technology. *Payment-channel networks* (PCNs) have emerged as the most widely deployed solution to mitigate the scalability issues, allowing the bulk of payments between two users to be carried out off-chain. Unfortunately, as reported in the literature and further demonstrated in this paper, current PCNs do not provide meaningful security and privacy guarantees [32], [42].**

In this work, we study and design secure and privacy-preserving PCNs. We start with a security analysis of existing PCNs, reporting a new attack that applies to all major PCNs, including the Lightning Network, and allows an attacker to steal the fees from honest intermediaries in the same payment path. We then formally define anonymous multi-hop locks (AMHLs), a novel cryptographic primitive that serves as a cornerstone for the design of secure and privacy-preserving PCNs. We present several provably secure cryptographic instantiations that make AMHLs compatible with the vast majority of cryptocurrencies. In particular, we show that (linear) homomorphic one-way functions suffice to construct AMHLs for PCNs supporting a script language (e.g., Ethereum). We also propose a construction based on ECDSA signatures that *does not require scripts*, thus solving a prominent open problem in the field.

AMHLs constitute a generic primitive whose usefulness goes beyond multi-hop payments in a single PCN and we show how to realize atomic swaps and interoperable PCNs from this primitive. Finally, our performance evaluation on a commodity machine finds that AMHL operations can be performed in less than 100 milliseconds and require less than 500 bytes of communication overhead, even in the worst case. In fact, after acknowledging our attack, the Lightning Network developers have implemented our ECDSA-based AMHLs into their PCN. This demonstrates the practicality of our approach and its impact on the security, privacy, interoperability, and scalability of today's cryptocurrencies.

I. Introduction

Cryptocurrencies are growing in popularity and are playing an increasing role in the worldwide financial ecosystem. In fact, the number of Bitcoin transactions grew by approximately 30% in 2017, reaching a peak of more than $420,000$ transactions per day in December 2017 [2]. This striking increase in demand has given rise to scalability issues [20], which go well beyond the rapidly increasing size of the blockchain. For instance, the permissionless nature of the consensus algorithm used in Bitcoin today limits the transaction rate to tens of transactions per second, whereas other payment networks such as Visa support peaks of up to 47,000 transactions per second [9].

Among the various proposals to solve the scalability issue [22], [23], [40], [50], *payment-channels* have emerged as the most widely deployed solution in practice. In a nutshell, two users open a payment channel by committing a single transaction to the blockchain, which locks their bitcoins in a deposit secured by a Bitcoin (smart) contract. These users can then perform several payments between each other without the need for additional blockchain transactions, by simply locally agreeing on the new deposit balance. A transaction is required only at the end in order to close the payment channel and unlock the final balances of the two parties, thereby drastically reducing the transaction load on the blockchain. Further research has proposed the concept of *payment-channel network* [50] (PCN), where two users not sharing a payment channel can still pay each other using a path of open channels between them. Unfortunately, current PCNs fall short of providing adequate security, privacy, and interoperability guarantees.

A. State-of-the-art in PCNs

Several practical deployments of PCNs exist today [5], [8], [10] based on a common reference description for the Lightning Network (LN) [6]. Unfortunately, this proposal is neither privacy-preserving, as shown in recent works [32], [42], nor secure, which stays in contrast to what until now was commonly believed, as we show in this work. In fact, we present a new attack, the wormhole attack, which applies not only to the LN,

[*] Both contributed equally and are considered to be co-first authors.
[¶] This work was done while this author was at Purdue University.

Network and Distributed Systems Security (NDSS) Symposium 2019
24-27 February 2019, San Diego, CA, USA
ISBN 1-891562-55-X
https://dx.doi.org/10.14722/ndss.2019.23330
www.ndss-symposium.org

the most widely deployed PCN, but also other PCNs based on the same cryptographic lock mechanism, such as the Raiden Network [7].

PCNs have attracted plenty of attention also from academia. Malavolta et al. [42] proposed a secure and privacy-preserving protocol for multi-hop payments. However, this solution is expensive as it requires to exchange a non-trivial amount of data (i.e., around 5 MB) between the users in the payment path and it also hinders interoperability as it requires the Hash Time-Lock Contract (HTLC) supported in the cryptocurrency.

Green and Miers presented BOLT, a hub-based privacy-preserving payment for PCNs [32]. BOLT requires cryptographic primitives only available in Zcash and it cannot be seamlessly deployed in Bitcoin. Moreover, this approach is limited to paths with a single intermediary and the extension to support paths of arbitrary length remains an open problem.

The rest of the existing PCN proposals suffer from similar drawbacks. Apart from not formalizing provable privacy guarantees, they are restricted to a setting with a trusted execution environment [38] or with a Turing complete scripting language [25], [26], [35], [44] so that they cannot seamlessly work with prominent cryptocurrencies today (except for Ethereum).

Poelstra introduced the notion of scriptless scripts, a modified version of a digital signature scheme so that a signature can only be created when faithfully fulfilling a cryptographic condition [49]. The resulting signature is verifiable following the unmodified digital signature scheme. When applied to script-based systems like Bitcoin or Ethereum, they are accompanied by core scripts (e.g., script to verify the signature itself). This approach reduces the space required for cryptographic operations in the script, saving thus invaluable bytes on the blockchain. Moreover, it improves upon the fungibility of the cryptocurrency as transactions from payment channels no longer require a script different from other payments.

Although interesting, current proposals [49] lack formal security and privacy treatment and are based only on the Schnorr signature scheme, thus being incompatible with major cryptocurrencies like Bitcoin. Although there exist early proposals for Schnorr adoption in Bitcoin [55], it is unclear whether they will be realized.

In summary, existing proposals are neither generically applicable nor interoperable, since they rely on specific features (e.g., contracts) of individual cryptocurrencies or trusted hardware. Furthermore, there seems to be a gap between secure realization of PCNs and what is developed in practice, as we demonstrate with our attack, which affects virtually all deployed PCNs.

B. *Our Contributions*

In this work, we contribute to the rigorous understanding of PCNs and present the first interoperable, secure, and privacy-preserving cryptographic construction for multi-hop locks (AMHLs). Specifically,

- We analyze the security of existing PCNs, reporting a new attack (the *wormhole attack*) which allows dishonest users to steal the payment fees from honest users along the path. This attack applies to the LN, as well as any decentralized PCN (following the definition in [42]) where the sender does not know in advance the intermediate users along the path to the receiver. We communicated the attack to the LN developers, who acknowledged the issue.
- In order to construct secure and privacy-preserving PCNs, we introduce a novel cryptographic primitive called anonymous multi-hop lock (AMHL). We model the security of such a primitive in the UC framework [18] to inherit the underlying composability guarantees. Then we show that AMHLs can be generically combined with any blockchain to construct a fully-fledged PCN.
- As a theoretical insight emerging from the wormhole attack, we establish a lower bound on the communication complexity of secure PCNs (Section III) that follow the definition from [42]: Specifically, we show that an extra round of communication to determine the path is necessary to have a secure transaction.
- We show how to realize AMHLs in different settings. In particular, we demonstrate that (linearly) homomorphic operations suffice to build any script-based AMHL. Furthermore, we show how to realize AMHLs in a scriptless setting. This approach is of special interest because it reduces the transaction size, and, consequently, the blockchain load. We give a concrete construction based on the ECDSA signature scheme, solving a prominent problem in the literature [49]. This makes AMHLs compatible with the vast majority of cryptocurrencies (including Bitcoin and Ethereum). In fact, AMHLs have been implemented and tested in the LN [28], [29].
- We implemented our cryptographic constructions and show that they require at most 60 milliseconds to be computed and a communication overhead of less than 500 bytes in the worst case. These results demonstrate that AMHLs are practical and ready to be deployed. In fact, AMHLs can be leveraged to design atomic swaps and interoperable (cross-currency) PCNs.

Organization. Section II shows the background on PCNs. Section III describes the wormhole attack. Section IV formally defines AMHLs. Section V contains our protocols for AMHLs and Section VI analyzes their performance. Section VII describes applications for AMHLs. Section VIII discusses the related work and Section IX concludes this paper.

II. Context: Payment Channel Networks

A. Payment Channels

A payment channel allows two users to exchange bitcoin without committing every single payment to the Bitcoin blockchain. For that, users first publish an on-chain transaction to deposit bitcoin into a multi-signature address controlled by both users. Such deposit also guarantees that all bitcoin are refunded at a possibly different but mutually agreed time if the channel expires. Users can then perform off-chain payments by adjusting the distribution of the deposit (that we will refer to as *balance*) in favor of the payee. When no more off-chain payments are needed (or the capacity of the payment channel is exhausted), the payment channel is closed with a *closing* transaction included in the blockchain. This transaction sends the deposited bitcoin to each user according the most recent balance in the payment channel. We refer the reader to [22], [23], [43], [50] for further details.

B. A Payment Channel Network (PCN)

A PCN can be represented as a directed graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$, where the set $\mathbb{V}$ of vertices represents the Bitcoin accounts and the set $\mathbb{E}$ of weighted edges represents the payment channels. Every vertex $U \in \mathbb{V}$ has associated a non-negative number that denotes the fee it charges for forwarding payments. The weight on a directed edge $(U_1, U_2) \in \mathbb{E}$ denotes the amount of remaining bitcoin that U_1 can pay to U_2.

A PCN is used to perform off-chain payments between two users with no direct payment channel between them but rather connected by a path of open payment channels. For that, assume that S wants to pay α bitcoin to R, which is reachable through a path of the form $S \to U_1 \to \ldots \to U_n \to R$. For their payment to be successful, every link must have a capacity $\gamma_i \geq \alpha'_i$, where $\alpha'_i = \alpha - \sum_{j=1}^{i-1} \mathit{fee}(U_j)$ (i.e., the initial payment value minus the fees charged by intermediate users in the path). If the payment is successful, edges from S to R are decreased by α'_i. Importantly, to ensure that R receives exactly α bitcoin, S must start the payment with a value $\alpha^* = \alpha + \sum_{j=1}^{n} \mathit{fee}(U_j)$. We refer the reader to [32], [42], [43], [50] for further details.

The concepts of payment channels and PCNs have already attracted considerable attention from academia [23], [32], [33], [37], [42]–[44]. In practice, the Lightning Network (LN) [6], [50] has emerged as the most prominent example. Currently, there exist several independent implementations of the LN for Bitcoin [5], [8], [10]. Moreover, the LN is also considered as a scalability solution in other blockchain-based payment systems such as Ethereum [7].

C. Multi-Hop Payments Atomicity

A fundamental property for multi-hop payments is *atomicity*: Either the capacity of all channels in the path is updated or none of the channels is changed. Partial updates can lead to coin losses for the users on the path. For instance, a user could pay a certain amount of bitcoin to the next user in the path but never receive the corresponding bitcoin from the previous neighbour. The LN tackles this challenge by relying on a smart contract called *Hash Time-Lock Contract* (HTLC) [50]. This contract locks x bitcoin that can be released only if the contract's condition is fulfilled. The contract relies on a collision-resistant hash function H and it is defined in terms of a hash value $y := H(R)$, where R is chosen uniformly at random, the amount of bitcoin x, and a timeout t, as follows: (i) If Bob produces the condition R^* such that $H(R^*) = y$ before t days, Alice pays Bob x bitcoin; (ii) If t days elapse, Alice gets back x bitcoin.

Fig. 1 shows an example of the use of HTLC in a payment. For simplicity, we assume that every user charges a fee of one bitcoin and the payment amount is 10 bitcoin. In this payment, Edward first sets up the payment by creating a random value R and sending $H(R)$ to Alice. Then, the *commitment phase* starts by Alice. She first sets on hold 13 bitcoin and then successively every intermediate user sets on hold the received amount minus his/her own fee. After Dave sets 10 coins on hold with Edward, the latter knows that the corresponding payment amount is on hold at each channel and he can start the *releasing phase* (depicted in green). For that, he reveals the value R to Dave allowing him to fulfill the HTLC contract and settle the new capacity at the payment channel. The value R is then passed backwards in the path allowing the settlement of the payment channels.

Privacy Issues in PCNs. Recent works [32], [42] show that the current use of HTLC leaks a common identifier along the payment path (i.e., the condition $H(R)$) that can be used by an adversary to tell who pays to whom. Current solutions to this privacy issue are expensive in terms of computation and communication [42] or incompatible with major cryptocurrencies [32]. This calls for an in-depth study of this cryptographic tool.

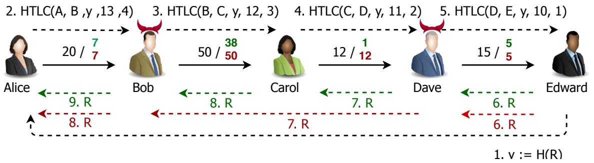

Fig. 1: Payment (with and without wormhole attack) from Alice to Edward for value 10 using HTLC contract. The honest (attacked) releasing phase is depicted in green (red). Non-bold (bold) numbers show the capacity of payment channels before (after) the payment. We assume a common fee of 1 coin.

III. Wormhole Attack in Existing PCNs

In a nutshell, the wormhole attack allows two colliding users on a payment path to exclude intermediate users from participating in the successful completion of a payment, thereby stealing the payment fees which were intended for honest path nodes.

In more detail, assume a payment path $(U_0, \ldots, U_i, \ldots, U_j, \ldots, U_n)$ used by U_0 to pay an amount $\alpha + \sum_k \gamma_k$ to U_n, where $\gamma_k = \mathit{fee}(U_k)$ denotes the fee charged by the intermediate user U_k as a reward for enabling the payment. Further assume that U_i and U_j are two adversarial users that may deviate from the protocol if some economic benefit is at stake. The adversarial strategy is as follows.

In the commitment phase, every user behaves honestly. This, in particular, implies that every honest user has locked a certain amount of coins in the hope of getting rewarded for this. In the releasing phase, honest users $U_{j+1}, \ldots, U_n$ correctly fulfill their HTLC contracts and settle the balances and rewards in their corresponding payment channels.

The user U_j behaves honestly with U_{j+1} effectively settling the balance in their payment channel. On the other hand, U_j waits until the timeout set in the HTLC with U_{j-1} is about to expire and then agrees with U_{j-1} to cancel the HTLC and set the balance in their payment channel back to the last agreed one. Note that from U_{j-1}'s point of view, this is a legitimate situation (e.g., there might not be enough coins in a payment channel at some user after U_j and the payment had to be canceled). Moreover, the channel between U_{j-1} and U_j does not need to be closed, it is just rolled back to a previous balance, a feature present in the Lightning Network.

As U_{j-1} believes that the payment did not go through, she also cancels the HTLC with U_{j-2}, who in turns cancels the HTLC with U_{j-3} and so on. This process continues until U_i is approached by U_{i+1}. Here, U_i cancels the HTLC with U_{i+1}. However, U_i gets the releasing condition R from U_j and can use it to fulfill the HTLC with U_{i-1} and therefore settle the new balance in that payment channel. Therefore, from the point of view of users $U_1, \ldots, U_{i-1}$, the payment has been successfully carried out. An illustrative example of this attack is shown in Fig. 1 with the attacked releasing phase depicted in red.

Discussion. An adversary controlling users U_i and U_j in a payment path that carries out the attack described in this section gets an overall benefit of $\sum_{k=i+1}^{j} \gamma_k$ bitcoins instead of only $\gamma_i + \gamma_j$ bitcoins in the case he behaves honestly. We make several observations here. First, the impact of this attack grows with the number of intermediate users between U_i and U_j as well as the number of payments that take both U_i and U_j in their path. While the Lightning Network is at its infancy, other well-established networks such as Ripple use paths with multiple intermediaries. For instance, in the Ripple network, more than 27% of the payments use more than two intermediaries [45]. Actually, paths with three intermediaries (e.g., sender → bank → currency-exchange → bank → receiver) are essential for currency exchanges, a key use case in LN itself [1]. When the LN grows to the scale of the Internet, routes may consist of several intermediaries as in the Internet today. Given these evidences, we expect long paths in the LN.

Second, honest intermediate users cannot trivially distinguish the situation in which they are under attack from the situation where the payment is simply unsuccessful (e.g., there are not enough coins in one of the channels or one of the users is offline). In both cases, the view for the honest users is that the timeout established in the HTLC is reached, the payment failed and they get their initially committed coins reimbursed. In short, the wormhole attack allows an adversary to steal the fees from intermediate honest users without leaving a inculpatory trace to them.

Third, fees are the main incentive for intermediary users. The wormhole attack takes away this crucial benefit. In fact, this attack not only makes honest users lose their fees, but also incur collateral costs: Coins locked for the payment under attack cannot be used for another (possibly successful) payment simultaneously.

Responsible Disclosure. We notified this attack to the LN developers and they have acknowledged this issue. Additionally, they have implemented our ECDSA-based construction (see Section V-D) and tested it for its integration in the LN, having thereby a fix for the wormhole attack and leveraging its privacy and practical benefits [28], [29].

(In)evitability of the Wormhole Attack. The wormhole attack is not restricted to the LN, but generally applies to PCNs with multi-hop payments that involve only two rounds of communication. We assume a communication round to consist of traversing the payment path once, either forth (e.g., for setting up the payment) or back (e.g., for releasing the coins). Additionally, we assume that in PCNs the communication between nodes is restricted to their direct neighbors, so in particular, there is no broadcast.[1] Consequently, using two rounds of communication for a payment implies that the payment is not preceded by a routing phase in which path-specific information is sent to nodes in the path. Under these assumptions, we state the lower bound informally in Theorem 1 and defer the formal theorem and the proof to Appendix A.

[1] This is the case in the setting of off-chain protocols where users not sharing a payment channel do not communicate with each other.

Theorem 1 (Informal)**.** *For all two-round (without broadcast channels) multi-hop payment protocols there exists a path prone to the wormhole attack.*

In this work we show that adding an additional round of communication suffices to overcome this impossibility result.[2] In particular, with one additional round of communication, the sender of a payment can communicate path-specific secret information to the intermediate nodes. This information can then be used to make the release keys unforgeable for an attacker. The cryptographic protocols we introduce in the remainder of this paper adopt this approach.

IV. Definition

Here we introduce a new cryptographic primitive called anonymous multi-hop lock (AMHL). This primitive generalizes the locking mechanism used for payments in state-of-the-art PCNs such as the LN. In Section VII we show that AMHL is the main cryptographic component required to construct fully-fledged PCNs. As motivated in the previous section, we model the primitive such that it allows for an initial setup phase where the first node of the path provides the other nodes with some secret (path-specific) state. Formally, an AMHL is defined with respect to a universe of users $\mathbb{U}$ and it is a five-tuple of PPT algorithms and protocols $\mathbb{L} = (\mathsf{KGen}, \mathsf{Setup}, \mathsf{Lock}, \mathsf{Rel}, \mathsf{Vf})$ defined as follows:

Definition 1. *An* AMHL $\mathbb{L} = (\mathsf{KGen}, \mathsf{Setup}, \mathsf{Lock}, \mathsf{Rel}, \mathsf{Vf})$ *consists of the following efficient algorithms:*

$\{(\mathsf{sk}_i, \mathsf{pk}), (\mathsf{sk}_j, \mathsf{pk})\} \leftarrow \langle \mathsf{KGen}_{U_i}(1^\lambda), \mathsf{KGen}_{U_j}(1^\lambda)\rangle$: *On input the security parameter 1^λ the key generation protocol returns a shared public key* pk *and a secret key* sk_i *(*sk_j*, respectively) to U_i and U_j.*

$\{s_0^I, \ldots, (s_n^I, k_n)\} \leftarrow \langle \mathsf{Setup}_{U_0}(1^\lambda, U_1, \ldots, U_n), \mathsf{Setup}_{U_1}(1^\lambda), \ldots, \mathsf{Setup}_{U_n}(1^\lambda)\rangle$: *On input a vector of identities $(U_1, \ldots, U_n)$ and the security parameter 1^λ, the setup protocol returns, for $i \in [0, n]$, a state s_i^I to user U_i. The user U_n additionally receives a key k_n.*

$\{(\ell, s_i^R), (\ell, s_{i+1}^L)\} \leftarrow \langle \mathsf{Lock}_{U_i}(s_i^I, \mathsf{sk}_i, \mathsf{pk}), \mathsf{Lock}_{U_{i+1}}(s_{i+1}^I, \mathsf{sk}_{i+1}, \mathsf{pk})\rangle$: *On input two initial states s_i^I and s_{i+1}^I, two secret keys* sk_i *and* sk_{i+1}*, and a public key* pk*, the locking protocol is executed between two users (U_i, U_{i+1}) and returns a lock ℓ and a right state s_i^R to U_i and the same lock ℓ and a left state s_{i+1}^L to U_{i+1}.*

$k' \leftarrow \mathsf{Rel}(k, (s^I, s^L, s^R))$: *On input an opening key k and a triple of states (s^I, s^L, s^R), the release algorithm returns a new opening key k'.*

$\{0, 1\} \leftarrow \mathsf{Vf}(\ell, k)$: *On input a lock ℓ and a key k the verification algorithm returns a bit $b \in \{0, 1\}$.*

Correctness. An AMHL is *correct* if the verification algorithm Vf always accepts an honestly generated lock-key pair. For a more detailed and formal correctness definition, we refer the reader to Appendix B.

Key Ideas. Fig. 2 illustrates the usage of the different protocols underlying the AMHL primitive. First, we assume an (interactive) KGen phase that emulates the opening of payment channels that compose the PCN.

In the setup phase (green arrows), the introduction of the initial state at each intermediate user is crucial for security and privacy. Intuitively, we can use this initial state as "rerandomization factor" to ensure that locks in the same path are unlinkable for the adversary.

Next, in the locking phase, each pair of users jointly executes the Lock protocol to generate a lock ℓ_i. The creation of this lock represents the commitment from U_i to perform an application-dependent action if a cryptographic problem is solved by U_{i+1}. In the case of LN, this operation represents the commitment of U_i to pay a certain amount of coins to U_{i+1} if U_{i+1} solves the cryptographic condition. Each user also learns some extra state s_i^R (resp. s_{i+1}^L) that will be needed for releasing the lock later on. While these extra states are not present in the LN (i.e., every lock is based on the same cryptographic puzzle $H(R)$), they are crucial for security. They make the releasing of different locks in the path independent and thus ensure that a lock ℓ_i can only be released if ℓ_{i+1} has been released before.

Finally, after the entire path is locked, the receiver U_n can generate a key for releasing its left lock. Then, each intermediate node can derive a valid key for its left lock from a valid key for its right lock using the Rel algorithm. This last phase resembles the opening phase of the LN where each pair of users settles the new balances for their deposit at each payment channel in the payment path.

A. Security and Privacy Definition

To model security and privacy in the presence of concurrent executions we resort to the universal composability framework from Canetti [18]. We allow thereby the composition of AMHLs with other application-dependent protocols while maintaining security and privacy guarantees.

Attacker Model. We model the players in our protocol as interactive Turing machines that communicate with a trusted functionality $\mathcal{F}$ via secure and authenticated channels. We model the attacker $\mathcal{A}$ as a PPT machine that has access to an interface $\mathsf{corrupt}(\cdot)$ that takes as

[2] A malicious sender can still bypass intermediate nodes, but he has no incentive as it implies stealing coins from himself.

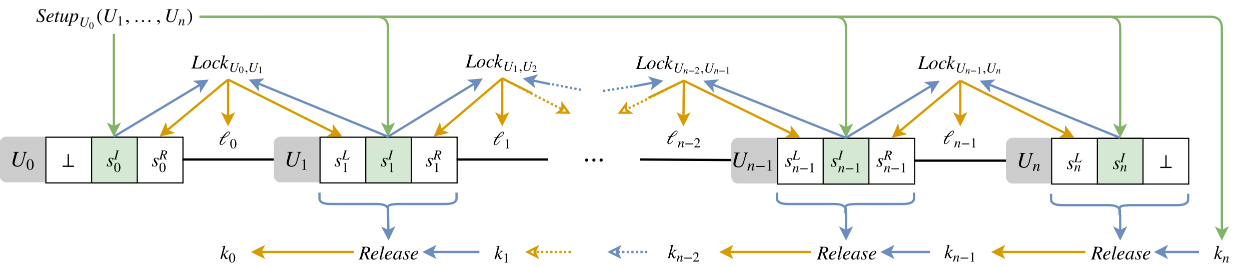

Fig. 2: Usage of the AMHL primitive. It is assumed that links between the users on the path have been created upfront (using KGen) and that the resulting public and secret keys are implicitly given as argument to the corresponding executions of Lock. Otherwise, the inputs (outputs) to (from) the Lock protocol and the Rel algorithm are indicated by blue (orange) arrows.

input a user identifier U and provides the attacker with the internal state of U. All the subsequent incoming and outgoing communication of U are then routed through $\mathcal{A}$. We consider the static corruption model, that is, the attacker is required to commit to the identifiers of the users he wishes to corrupt ahead of time.[3]

Communication Model. Communication happens through the secure message transmission functionality $\mathcal{F}_{\mathsf{smt}}$ that informs the attacker whenever some communication happens between two users and the attacker can delay the delivery of the message arbitrarily (for a concrete functionality see [18]). We also assume the existence of a functionality $\mathcal{F}_{\mathsf{anon}}$ (see [17] for an example), which provides user with an anonymous communication channel. In its simplest form, $\mathcal{F}_{\mathsf{anon}}$ is identical to $\mathcal{F}_{\mathsf{smt}}$, except that it omits the identifier of the sender from the message sent to the receiver. We assume a synchronous communication network, where the execution of the protocol happens in discrete rounds. The parties are always aware of the current round and if a message is created at round i, then it is delivered at the beginning of the $(i + 1)$-th round. Our model assumes that computation is instantaneous. In the real world, this is justified by setting a maximum time bound for message transmission, which is known by all users. If no message is delivered by the expiration time, then the message is set to be $\perp$. We remark that such an assumption is standard in the literature [25] and for an example of the corresponding ideal functionality $\mathcal{F}_{\mathsf{syn}}$ we refer the reader to [18], [34].

Universal Composability. Let $\mathsf{EXEC}_{\tau,\mathcal{A},\mathcal{E}}$ be the ensemble of the outputs of the environment $\mathcal{E}$ when interacting with the attacker $\mathcal{A}$ and users running protocol τ (over the random coins of all the involved machines).

Definition 2 (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 $\mathsf{EXEC}_{\tau,\mathcal{A},\mathcal{E}}$ and $\mathsf{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{E}}$ are computationally indistinguishable.*

Ideal Functionality. We formally define the ideal world functionality $\mathcal{F}$ for AMHLs in the following. For a more modular treatment, our UC definition models only the cryptographic lock functionality, rather than aiming at a comprehensive characterization of PCNs. In Section VII we show how one can construct a full PCN (e.g., as defined in [42]) by composing this functionality with time locks, balance updates, and on-chain channel management. For ease of exposition we assume that each pair of users establishes only a single link per direction. The model can be easily extended to handle the more generic case. $\mathcal{F}$ works in interaction with a universe of users $\mathbb{U}$ and initializes two empty lists $(\mathcal{U}, \mathcal{L}) := \emptyset$, which are used to track the users and the locks, respectively. The list $\mathcal{L}$ represents a set of lock chains. The entries are of the form $(lid_i, U_i, U_{i+1}, f, lid_{i+1})$ where lid_i is a lock identifier that is unique even among other lock chains in $\mathcal{L}$, U_i and U_{i+1} are the users connected by the lock, $f \in \{\mathsf{Init}, \mathsf{Lock}, \mathsf{Rel}\}$ is a flag that represents the status of the lock, and lid_{i+1} is the identifier of the next lock in the path. For sake of better readability, we define functions operating on $\mathcal{L}$ extracting lock-specific information given the lock's identifier, such as the lock's status ($getStatus(\cdot)$), the nodes it is connecting ($getLeft(\cdot)$, $getRight(\cdot)$), and the next lock's identifier ($getNextLock(\cdot)$). In addition we define an update function $updateStatus(\cdot, \cdot)$ that changes the status of a lock to a new flag.

The interfaces of the functionality $\mathcal{F}$ are specified in Fig. 3. The KeyGen interface allows a user to establish a link with another user (specifying whether it wants to be the left or the right part of the link). The Setup interface allows a user U_0 to setup a path (starting from U_0) along previously established links. The Lock interface allows a user to create a lock with

[3] Extending our protocol to support adaptive corruption queries is an interesting open problem.

```
**KeyGen**($sid, U_j, \{L, R\}$)
Upon invocation by $U_i$:
send_s $(sid, U_i, \{L, R\})$ to $U_j$
receive_s $(sid, b)$ from $U_j$
if $b = \bot$ send $\bot$ to $U_i$ and abort
if $L$ insert $(U_i, U_j)$ into $\mathcal{U}$ and send_s $(sid, U_i, U_j)$ to $U_i$
if $R$ insert $(U_j, U_i)$ into $\mathcal{U}$ and send_s $(sid, U_j, U_i)$ to $U_i$

**Lock**($sid, lid$)
Upon invocation by $U_i$:
if $getStatus(lid) \neq \mathsf{Init}$ or $getLeft(lid) \neq U_i$ then abort
send_s $(sid, lid, \mathsf{Lock})$ to $getRight(lid)$
receive_s $(sid, b)$ from $getRight(lid)$
if $b = \bot$ send $\bot$ to $U_i$ and abort
$updateStatus(lid, \mathsf{Lock})$
send_s $(sid, lid, \mathsf{Lock})$ to $U_i$

**GetStatus**($sid, lid$)
Upon invocation by $U_i$:
return $(sid, lid, getStatus(lid))$ to $U_i$

**Setup**($sid, U_0, \ldots, U_n$)
Upon invocation by $U_0$:
if $\forall i \in [0, n-1] : (U_i, U_{i+1}) \notin \mathcal{U}$ then abort
$\forall i \in [0, n-1]$: $lid_i \leftarrow_\$ \{0,1\}^\lambda$
insert $(lid_0, U_0, U_1, \mathsf{Init}, lid_1), (lid_{n-1}, U_{n-1}, U_n, \mathsf{Init}, \bot)$ into $\mathcal{L}$
send_an $(sid, \bot, lid_0, \bot, U_1, \mathsf{Init})$ to $U_0$
send_an $(sid, lid_{n-1}, \bot, U_{n-1}, \bot, \mathsf{Init})$ to $U_n$
$\forall i \in [1, n-1]$: insert $(lid_i, U_i, U_{i+1}, \mathsf{Init}, lid_{i+1})$ into $\mathcal{L}$
        send_an $(sid, lid_{i-1}, lid_i, U_{i-1}, U_{i+1}, \mathsf{Init})$ to $U_i$

**Release**($sid, lid$)
Upon invocation by $U_i$:
if $getRight(lid) \neq U_i$ or $getStatus(lid) \neq \mathsf{Lock}$ or
  $getStatus(getNextLock(lid)) \neq \mathsf{Rel}$
  and $getNextLock(lid) \neq \bot$ then abort
$updateStatus(lid, \mathsf{Rel})$
send_s $(sid, lid, \mathsf{Rel})$ to $getLeft(lid)$
```

Fig. 3: Ideal functionality for cryptographic locks (AMHLs)

its right neighbor on a previously created path and the Release algorithm allows a user to release the lock with its left neighbor, in case that the user is either the receiver or its right lock has been released before. Finally, the GetStatus interface allows one to check the current status of a lock, i.e., whether it is initialized, locked or released. Internally, the locks are assigned identifiers that are unique across all paths. We define the interfaces $\mathsf{send_s}$ and $\mathsf{receive_s}$ to exchange messages through the $\mathcal{F}_{\mathsf{smt}}$ functionality and the interface $\mathsf{send_{an}}$ to send messages via $\mathcal{F}_{\mathsf{anon}}$.

B. Discussion

We discuss how the security and privacy notions of interest for AMHLs are captured by functionality $\mathcal{F}$.

Atomicity. Loosely speaking, atomicity means that every user in a path is able to release its left lock in case that his right lock was already released. This is enforced by $\mathcal{F}$ as i) it is keeping track of the chain of locks and their current status in the list $\mathcal{L}$ and ii) the Release interface of $\mathcal{F}$ allows one to release a lock *lid* (changing the flag to Rel) if *lid* is locked and the follow-up lock ($getNextLock(lid)$) was already released.

Consistency. An AMHL is consistent if no attacker can release his left lock without its right lock being released before. This prevents scenarios where some AMHL is released before the receiver is reached and, more generically, the wormhole attack described in Section III. To see why our ideal functionality models this property, observe that the Release interface allows a user to release the left lock only if the right lock has already been released or the user itself is the receiver. In this context, no wormhole attack is possible as intermediate nodes cannot be bypassed.

Relationship Anonymity. Relationship anonymity [12] requires that each intermediate node does not learn any information about the set of users in an AMHL beyond its direct neighbors. This property is satisfied by $\mathcal{F}$ as the lock identifiers are sampled at random and during the locking phase a user only learns the identifiers of its left and right lock as well as its left and right neighbor. We discuss this further in Appendix D.

V. CONSTRUCTIONS

A. Cryptographic Building Blocks

Throughout this work we denote by $1^\lambda \in \mathbb{N}^+$ the security parameter. Given a set S, we denote by $x \leftarrow_\$ S$ the sampling of an element uniformly at random from S, and we denote by $x \leftarrow A(in)$ the output of the algorithm A on input *in*. We denote by $\mathsf{min}(a, b)$ the function that takes as input two integers and returns the smaller of the two. To favor readability, we omit session identifiers from the description of the protocols. In the following we briefly recall the cryptographic building blocks of our schemes.

Homomorphic One-Way Functions. A function $g : \mathcal{D} \rightarrow \mathcal{R}$ is one-way if, given a random element $x \in \mathcal{R}$, it is hard to compute a $y \in \mathcal{D}$ such that $g(y) = x$. We say that a function g is homomorphic if $\mathcal{D}$ and $\mathcal{R}$ define two abelian groups and for each pair $(a, b) \in \mathcal{D}^2$ it holds that $g(a \circ b) = g(a) \circ g(b)$, where $\circ$ denotes the group operation. Throughout this work we denote the corresponding arithmetic group additively.

Commitment Scheme. A commitment scheme COM consists of a commitment algorithm $(\mathsf{decom}, \mathsf{com}) \leftarrow \mathsf{Commit}(1^\lambda, m)$ and a verification algorithm $\{0, 1\} \leftarrow \mathsf{V}_{\mathsf{com}}(\mathsf{com}, \mathsf{decom}, m)$. The commitment algorithm allows a prover to commit to a message m without revealing it. In a second phase, the prover can convince a verifier that the message m was indeed committed by showing the unveil information decom. The security of a commitment scheme is captured by the standard ideal functionality $\mathcal{F}_{\mathsf{com}}$ [18].

Non-Interactive Zero-Knowledge. Let R be an NP relation and let L be the set of positive instances, i.e., $L := \{x \mid \exists w \text{ s.t. } R(x, w) = 1\}$. A non-interactive zero-knowledge proof [15] scheme NIZK consists of an efficient prover algorithm $\pi \leftarrow \mathsf{P}_{\mathsf{NIZK}}(w, x)$ and an efficient verifier $\{0, 1\} \leftarrow \mathsf{V}_{\mathsf{NIZK}}(x, \pi)$. A NIZK scheme allows the prover to convince the verifier about the existence of a witness w for a certain statement x without revealing any additional information. The security of a NIZK scheme is modeled by the following ideal functionality $\mathcal{F}_{\mathsf{NIZK}}$: On input $(\mathsf{prove}, \mathit{sid}, x, w)$ by the prover, check if $R(x, w) = 1$ and send $(\mathsf{proof}, \mathit{sid}, x)$ to the verifier if this is the case.

Homomorphic Encryption. One of the building blocks of our work is the additive homomorphic encryption scheme $\mathsf{HE} := (\mathsf{KGen}_{\mathsf{HE}}, \mathsf{Enc}_{\mathsf{HE}}, \mathsf{Dec}_{\mathsf{HE}})$ from Paillier [46]. The scheme supports homomorphic operation over the ciphertexts of the form $\mathsf{Enc}_{\mathsf{HE}}(\mathsf{pk}, m) \cdot \mathsf{Enc}_{\mathsf{HE}}(\mathsf{pk}, m') = \mathsf{Enc}_{\mathsf{HE}}(\mathsf{pk}, m + m')$. We assume that Paillier's encryption scheme satisfies the notion of ecCPA security, as defined in the work of Lindell [39].

ECDSA Signatures. Let $\mathbb{G}$ be an elliptic curve group of order q with base point G and let $H : \{0, 1\}^* \rightarrow \{0, 1\}^{|q|}$ be a collision resistant hash function. The key generation algorithm $\mathsf{KGen}_{\mathsf{ECDSA}}(1^\lambda)$ samples a private key as a random value $x \xleftarrow{\$} \mathbb{Z}_q$ and sets the corresponding public key as $Q := x \cdot G$. To sign a message m, the signing algorithm $\mathsf{Sig}_{\mathsf{ECDSA}}(\mathsf{sk}, m)$ samples some $k \xleftarrow{\$} \mathbb{Z}_q$ and computes $e := H(m)$. Let $(r_x, r_y) := R = k \cdot G$, then the signing algorithm computes $r := r_x \bmod q$ and $s := \frac{e+rx}{k} \bmod q$. The signature consists of (r, s). The verification algorithm $\mathsf{Vf}_{\mathsf{ECDSA}}(\mathsf{pk}, \sigma, m)$ recomputes $e = H(m)$ and returns 1 if and only if $(x, y) = \frac{e}{s} \cdot G + \frac{r}{s} \cdot Q$ and $r = x \bmod q$. It is a well known fact that for every valid signature (r, s), also the pair $(r, -s)$ is a valid signature. To make the signature *strongly* unforgeable we augment the verification equation with a check that $s \leq \frac{q-1}{2}$. We assume the existence of an interactive protocol $\Pi^{\mathsf{ECDSA}}_{\mathsf{KGen}}$ executed between two users where the one receives (x_0, Q, sk), where sk is a Paillier secret key and $Q = x_0 \cdot x_1 \cdot G$, whereas the other obtains $(x_1, Q, \mathsf{Enc}_{\mathsf{HE}}(\mathsf{pk}, x_0))$, where pk is the corresponding Paillier public-key. For correctness, we require that the Paillier modulus is $N = O(q^4)$. We assume that the parties have access to an ideal functionality $\mathcal{F}^{\mathsf{ECDSA}}_{\mathsf{kgen}}$ (refer to Appendix E for a precise definition) that securely computes the tuples for both parties. An efficient protocol has been recently proposed by Lindell [39].

Anonymous Communication. We assume an anonymous communication channel Π_{anon} available among users in the network, which is modelled by the ideal functionality $\mathcal{F}_{\mathsf{anon}}$. It anonymously delivers messages to users in the network (e.g., see [17]).

B. Generic Construction

An interesting question related to AMHLs is under which class of hard problems such a primitive exists. A generic construction using trapdoor permutations was given (implicitly) in [42]. Here we propose a scheme from any homomorphic one-way function. Examples of homomorphic one-way functions include discrete logarithm and the learning with errors problem [51]. Let $g : \mathcal{D} \rightarrow \mathcal{R}$ be a homomorphic one-way function, and let $\mathcal{F}_{\mathsf{anon}}$ be the ideal functionality for an anonymous communication channel. The algorithms of our construction are given in Fig. 4. Note that KeyGen simply returns the users identities and thus it is omitted.

In the setup algorithm, the user U_0 initializes the AMHL by sampling n values $(y_0, \ldots, y_{n-1})$ from the domain of g. Then it sends (via $\mathcal{F}_{\mathsf{anon}}$) a triple $(g(\sum_{j=0}^{i-1} y_j), g(\sum_{j=0}^{i} y_j), y_i)$ to each intermediate user. The intermediate user U_i can then check that the triple is well formed using the homomorphic properties of g. Two contiguous users U_i and U_{i+1} can agree on the shared value of $\ell_i := Y_i = g(\sum_{j=0}^{i} y_j)$ by simply comparing the second and first element of their triple, respectively. Note that publishing a valid opening key k such that $g(k) = \ell$ corresponds to inverting the one-way function g. The opening of the locks can be triggered by the last node in the chain U_n: The initial key $k_n := \sum_{i=0}^{n-1} y_i$ consists of a valid pre-image of $\ell_{n-1} := Y_{n-1}$. As soon as the "right" lock is released, each intermediate user U_i has enough information to release its "left" lock. To see this, observe that $g(k_{i+1} - y_i) = g(\sum_{j=0}^{i} y_i - y_i) = g(\sum_{j=0}^{i-1} y_i) = Y_{i-1}$. For the security of the construction, we state the following theorem. Due to space constraints, the proof is deferred to Appendix E.

$\underline{\mathsf{Setup}_{U_i}(1^\lambda)}$		$\underline{\mathsf{Setup}_{U_0}(1^\lambda, U_1, \ldots, U_n)}$		$\underline{\mathsf{Setup}_{U_n}(1^\lambda)}$
		$y_0 \leftarrow_\$ \mathcal{D}$		
		$Y_0 := g(y_0)$		
		$\forall i \in [1, n-1] : y_i \leftarrow_\$ \mathcal{D}$		
if $Y_i \neq Y_{i-1} + g(y_i)$ then abort	$\xleftarrow{(Y_{i-1}, Y_i, y_i)}$	$Y_i := Y_{i-1} + g(y_i)$	$\xrightarrow{(Y_{n-1}, k_n := \sum_{i=0}^{n-1} y_i)}$	
return (Y_{i-1}, Y_i, y_i)		return y_0		return $((Y_{n-1}, 0, 0), k_n)$

$\underline{\mathsf{Lock}_{U_i}(s_i^I, \mathsf{sk}_i, \mathsf{pk})}$		$\underline{\mathsf{Lock}_{U_{i+1}}(s_{i+1}^I, \mathsf{sk}_{i+1}, \mathsf{pk})}$
parse s_i^I as (Y_i', Y_i, y_i)	$\xrightarrow{Y_i}$	parse s_{i+1} as $(Y_{i+1}', Y_{i+1}, y_{i+1})$
		if $Y_i \neq Y_{i+1}'$ then abort
return $(Y_i, \bot)$		return $(Y_i, \bot)$

```
$\underline{\mathsf{Rel}(k, (s^I, s^L, s^R))}$
parse $s^I$ as $(Y', Y, y)$
return $k - y$
```

```
$\underline{\mathsf{Vf}(\ell, k)}$
return $g(k) = \ell$
```

Fig. 4: Algorithms and protocols for the generic construction

Theorem 2. *Let g be a homomorphic one-way function, then the construction in Fig. 4 UC-realizes the ideal functionality $\mathcal{F}$ in the $(\mathcal{F}_{\mathsf{syn}}, \mathcal{F}_{\mathsf{smt}}, \mathcal{F}_{\mathsf{anon}})$-hybrid model.*

The generic construction presented here requires a cryptocurrency supporting scripts that define (linearly) homomorphic operations. This construction is therefore of special interest in blockchain technologies such as Ethereum [4] and Hyperledger Fabric [11], where any user can freely deploy a smart contract without restrictions in the cryptographic operations available. We stress that any function with homomorphic properties is suitable to implement our construction. For instance, lattice-based functions (e.g., from the learning with errors problem) can be used for applications where post-quantum cryptography is required. However, many cryptocurrencies, led by Bitcoin, do not support unrestricted scripts and the deployment of generic AMHLs requires non-trivial changes (i.e., a hard fork). To overcome this challenge, we turn our attention to scriptless AMHLs, where a signature scheme can simultaneously be used for authorization and locking.

C. Scriptless Schnorr-based Construction

The crux of a scriptless locking mechanism is that the lock can consist only of a message m and a public key pk of a given signature scheme and can be released only with a valid signature σ of m under pk. Scriptless locks stem from an idea of Poelstra [48], who proposed a way to embed contracts into Schnorr signatures. In this work we cast Poelstra's informal idea in our framework and we formally characterize its security and privacy guarantees. We further optimize this scheme in order to save one round of communication.

Recall that a public key in a Schnorr signature consists of an element $Q := x \cdot G$ and a signature $\sigma := (k \cdot G, s)$ on a message m is generated by sampling $k \leftarrow_\$ \mathbb{Z}_q$, computing $e := H(Q \| k \cdot G \| m)$, and setting $s := k - xe$. On a very high level, the locking mechanism consists of an "incomplete" distributed signing of some message m: Two users U_i and U_{i+1} agree on a randomly chosen element $R_0 + R_1$ using a coin tossing protocol, then they set the randomness of the signature to be $R := R_0 + R_1 + Y_i$. Next they jointly compute the value $s := r_0 + r_1 + e \cdot (x_0 + x_1)$ as if Y_i was not part of the randomness, where e is the hash of the transcript so far. The resulting (R, s) is *not* a valid signature on m, since the additive term y^* (where $y^* \cdot G = Y_i$) is missing from the computation of s. However, once the discrete logarithm of Y_i is revealed, a valid signature m can be computed by U_{i+1}. Leveraging this observation, we can enforce an *atomic* opening: The subsequent locking (between U_{i+1} and U_{i+2}) is conditioned on some $Y_{i+1} = Y_i + y_{i+1} \cdot G$. This way, the opening of the right lock reveals the value $y^* + y_{i+1}$ and U_{i+1} can immediately extract y^* and open its left lock with a valid signature on m. We defer the formal description and the analysis of the scheme to Appendix C.

D. Scriptless ECDSA-based Construction

The Schnorr-based scheme is limited to cryptocurrencies that use Schnorr signatures to authorize transactions and thus is not compatible with those systems, prominently Bitcoin, that implement ECDSA signatures. Therefore, an ECDSA-based scriptless AMHL is interesting both from a practical and a theoretical perspective as to whether it can be done at all. Prior to our work, the existence of such a construction was regarded an open question [49]. The core difficulty is that the Schnorr-based construction exploits the linear structure of the signature, whereas the ECDSA signing algorithm completely breaks this linearity feature (e.g., it requires to compute multiplicative shares of a key and the inverse of elements within a group). In the following, we show how to overcome these problems, introducing an ECDSA-based construction for AMHLs: Locks are of the form (pk, m) and can only be opened with an ECDSA signature σ on m under pk.

$\underline{\mathsf{Setup}_{U_i}(1^\lambda)}$		$\underline{\mathsf{Setup}_{U_0}(1^\lambda, U_1, \dots, U_n)}$		$\underline{\mathsf{Setup}_{U_n}(1^\lambda)}$
		$y_0 \leftarrow_{\$} \mathbb{Z}_q;\ Y_0 = y_0 \cdot G$		
		$\forall i \in [1, n-1] : y_i \leftarrow_{\$} \mathbb{Z}_q$		
		$\quad Y_i := Y_{i-1} + y_i \cdot G$		
$\mathrm{stmt}_i := \{\exists y \text{ s.t. } Y_i = y \cdot G\}$		$\quad \mathrm{stmt}_i := \{\exists y \text{ s.t. } Y_i = y \cdot G\}$		
$b \leftarrow \mathsf{V}_{\mathsf{NIZK}}(\mathrm{stmt}_i, \pi_i)$	$\xleftarrow{(Y_{i-1}, Y_i, \pi_i)}$	$\quad \pi_i \leftarrow \mathsf{P}_{\mathsf{NIZK}}\left(\sum_{j=0}^{i} y_j, \mathrm{stmt}_i\right)$	$\xrightarrow{(Y_{n-1}, k_n := \sum_{i=0}^{n-1} y_i)}$	
if $b = 0$ then abort				
$Y_i := Y_{i-1} + y_i \cdot G$				
return (Y_{i-1}, Y_i, y_i)		return y_0		return $((Y_{n-1}, 0, 0), k_n)$

$\underline{\mathsf{Lock}_{U_i}(s_i^I, \mathsf{sk}_i, \mathsf{pk})}$		$\underline{\mathsf{Lock}_{U_{i+1}}(s_{i+1}^I, \mathsf{sk}_{i+1}, \mathsf{pk})}$
parse s_i^I as (Y_0', Y_0, y_0)		parse s_{i+1}^I as (Y_1', Y_1, y_1)
parse sk_i as $(x_0, \mathsf{sk}_{\mathsf{HE}})$		parse sk_{i+1} as (x_1, c)
$r_0 \leftarrow_{\$} \mathbb{Z}_q;\ R_0 := r_0 \cdot G;\ R_0' := r_0 \cdot Y_0$		$r_1 \leftarrow_{\$} \mathbb{Z}_q;\ R_1 := r_1 \cdot G;\ R_1' := r_1 \cdot Y_1'$
$\mathrm{stmt}_0 := \{\exists r_0 \text{ s.t. } R_0 = r_0 \cdot G \text{ and } R_0' = r_0 \cdot Y_0\}$		$\mathrm{stmt}_1 := \{\exists r_1 \text{ s.t. } R_1 = r_1 \cdot G \text{ and } R_1' = r_1 \cdot Y_1'\}$
$\pi_0 \leftarrow \mathsf{P}_{\mathsf{NIZK}}(r_0, \mathrm{stmt}_0)$		$\pi_1 \leftarrow \mathsf{P}_{\mathsf{NIZK}}(r_1, \mathrm{stmt}_1)$
	$\xleftarrow{\mathsf{com}}$	$(\mathsf{decom}, \mathsf{com}) \leftarrow \mathsf{Commit}(1^\lambda, (R_1, R_1', \pi_1))$
	$\xrightarrow{(R_0, R_0', \pi_0)}$	if $\mathsf{V}_{\mathsf{NIZK}}(\mathrm{stmt}_0, \pi_0) \neq 1$ then abort
		$(r_x, r_y) := R = r_1 \cdot R_0';\ \rho \leftarrow_{\$} \mathbb{Z}_{q^2}$
if $\mathsf{V}_{\mathsf{com}}(\mathsf{com}, \mathsf{decom}, (R_1, R_1' \pi_1)) \neq 1$ then abort	$\xleftarrow{(\mathsf{decom}, R_1, R_1', \pi_1, c')}$	$c' := c^{r_x (r_1)^{-1} x_1} \cdot \mathsf{Enc}_{\mathsf{HE}}(\mathsf{pk}, H(m)(r_1)^{-1} + \rho q)$
if $\mathsf{V}_{\mathsf{NIZK}}(\mathrm{stmt}_1, \pi_1) \neq 1$ then abort		
$s \leftarrow \mathsf{Dec}_{\mathsf{HE}}(\mathsf{sk}_{\mathsf{HE}}, c')$		
$(r_x, r_y) := R = r_0 \cdot R_1'$		
if $s \cdot R_1 \neq r_x \cdot \mathsf{pk} + H(m) \cdot G$ then abort	$\xrightarrow{s' := s \cdot r_0^{-1} \bmod q}$	if $s' \cdot r_1 \cdot R_0 \neq r_x \cdot \mathsf{pk} + H(m) \cdot G$ then abort
return $((m, \mathsf{pk}), (s', m, \mathsf{pk}))$		return $((m, \mathsf{pk}), (r_x, s'))$

```
$\underline{\mathsf{Rel}(k, (s^I, s^L, s^R))}$
parse $s^I$ as $(Y', Y, y)$, $k$ as $(r, s)$, $s^L$ as $(w_0, w_1)$, $s^R$ as $(s', m, \mathsf{pk})$
$t := w_1 \cdot (\frac{s'}{s} - y)^{-1};\ t' := w_1 \cdot (-\frac{s'}{s} - y)^{-1}$
if $\mathsf{Vf}_{\mathsf{ECDSA}}(\mathsf{pk}, (w_0, \mathsf{min}(t, -t)), m) = 1$ return $(r, \mathsf{min}(t, -t))$
if $\mathsf{Vf}_{\mathsf{ECDSA}}(\mathsf{pk}, (w_0, \mathsf{min}(t', -t')), m) = 1$ return $(r, \mathsf{min}(t', -t'))$
```

```
$\underline{\mathsf{Vf}(\ell, k)}$
parse $\ell$ as $(m, \mathsf{pk})$
parse $k$ as $(r, s)$
return 1 iff $(r, \cdot) = \frac{H(m)}{s} \cdot G + \frac{r}{s} \cdot \mathsf{pk}$ and $s \leq \frac{q-1}{2}$
```

Fig. 5: Algorithms and protocols for the ECDSA-based construction.

Let $\mathbb{G}$ be an elliptic curve group of order q with base point G and let $H : \{0,1\}^* \to \{0,1\}^{|q|}$ be a hash function. The ECDSA-based construction is shown in Fig. 5. Each pair of users (U_i, U_j) generates a shared ECDSA public key $\mathsf{pk} = (x_i \cdot x_j) \cdot G$ via the $\mathcal{F}_{\mathsf{kgen}}^{\mathsf{ECDSA}}$ functionality. Additionally, U_i receives a Paillier secret key sk and his share x_i, whereas and U_j receives the share x_j and the Paillier encryption c of x_i. The key generation functionality is fully described in Appendix E.

The setup here is very similar to the setup of the generic construction in Fig. 4 except that the one-way function g is now instantiated with discrete logarithm over elliptic curves. Each intermediate user U_i receives a triple (Y_{i-1}, Y_i, y_i) such that $Y_i := Y_{i-1} + y_i \cdot G$, from $\mathcal{F}_{\mathsf{anon}}$. For technical reasons, the initiator of the AMHL also includes a proof of wellformedness for each Y_i.

The locking algorithm is initiated by two users U_i and U_{i+1} who agree on a message m (which encodes a unique id) and on a value $Y_i := y^* \cdot G$ of unknown discrete logarithm. The two parties then run a coin tossing protocol to agree on a randomness $R = (r_0 \cdot r_1) \cdot Y_i$. When compared to the Schnorr instance, the crucial technical challenge here is that the randomnesses are composed multiplicatively due to the structure of the ECDSA signature and therefore, the trick applied in the Schnorr construction no longer works here. R is computed through a Diffie-Hellman-like protocol, where the parties exchange $r_0 \cdot Y_i$ and $r_1 \cdot Y_i$ and locally recompute R. As before, the shared ECDSA signature is computed by "ignoring" the term Y_i, since the parties are unaware of its discrete logarithm. The corresponding tuple $\left(r_x, s' := \frac{r_x \cdot (x_0 \cdot x_{i+1}) + H(m)}{r_0 \cdot r_1}\right)$ is jointly computed using the encryption of x_0 and the homomorphic properties of Paillier encryption. This effectively means that $(r_x, s') = (r_x, s^* \cdot y^*)$, where (r_x, s^*) is a valid ECDSA signature on m. In order to check the validity of s', the parties additionally need to exchange the value $R^* := (r_0 \cdot r_1) \cdot G = (y^*)^{-1} \cdot R$. The computation of R^* (together with the corresponding consistency proof) is piggybacked in the coin tossing. Given R^*, the validity of s' can be easily verified by both parties by recomputing it "in the exponent".

From the perspective of U_{i+1}, releasing his left lock without a key for his right lock implies solving the discrete logarithm of Y_i. On the converse, once the right lock is released, the value $y^* + y_{i+1}$ is revealed (where y_{i+1} is part of the state of U_{i+1}) and a valid signature can be computed as $\left(r_x, \frac{s'}{y^*}\right)$. The security of the construction is established by the following theorem (see Appendix E for a full proof).

Theorem 3. *Let* COM *be a secure commitment scheme and let* NIZK *be a non-interactive zero knowledge proof. If ECDSA signatures are strongly existentially unforgeable and Paillier encryption is* ecCPA *secure, then the construction in Fig. 5 UC-realizes the ideal functionality* $\mathcal{F}$ *in the* $(\mathcal{F}_{\mathsf{kgen}}^{\mathsf{ECDSA}}, \mathcal{F}_{\mathsf{syn}}, \mathcal{F}_{\mathsf{smt}}, \mathcal{F}_{\mathsf{anon}})$*-hybrid model.*

E. Hybrid AMHLs

We observe that, when instantiated over the same elliptic curve $\mathbb{G}$, the setup protocols of the Schnorr and ECDSA constructions are identical. This means that the initiator of the lock does not need to know whether each intermediate lock is computed using the ECDSA or Schnorr method. This opens the doors to hybrid AMHLs: Given a unified setup, the intermediate pair of users can generate locks using an arbitrary locking protocol. The resulting AMHL is a chaining of (potentially) different locks and the release algorithm needs to be adjusted accordingly. For the case of ECDSA-Schnorr the user needs to extract the value y^* from the right Schnorr signature (R^*, s^*) and his state $s^R := s' = s^* - y^* + y_{i+1}$ and $s^I := (Y_i, Y_{i+1}, y_{i+1})$. Given y^*, he can factor it out of its left state $s^L = ((r, s \cdot y^*), m, \mathsf{pk})$ and recover a valid ECDSA signature.

The complementary case (Schnorr-ECDSA) is handled mirroring this algorithm. Similar techniques also apply to the generic construction, when the one-way function is instantiated appropriately (i.e., with discrete logarithm over the same curve). This flexibility enables atomic swaps and cross-currency payments (see Section VII). The security for the hybrid AMHLs follows similar to the standard case.

VI. Performance Analysis

A. Implementation Details

We have developed a prototypical Python implementation to demonstrate the feasibility of our construction and evaluate its performance. We have implemented the cryptographic operations required by AMHLs as described in this work. We have used the Charm library [3] for the cryptographic operations. We have instantiated ECDSA over the elliptic curve *secp256k1* (the one used in Bitcoin) and we have implemented the homomorphic one-way function as $g(x) := x \cdot G$ over the same curve. Zero-knowledge protocols for discrete logarithms have been implemented using Σ protocols [21] and made non-interactive using the Fiat-Shamir heuristic [27]. For a commitment scheme we have used SHA-256 modeled as a random oracle [13].

B. Evaluation

Testbed. We conducted our experiments on a machine with an Intel Core i7, 3.1 GHz and 8 GB RAM. We consider the Setup, Lock, Rel and Vf algorithms. We do not consider KGen as we use off-the-shelf algorithms without modification. Moreover, the key generation is executed only once upon creating a link and thus does not affect the online performance of AMHLs. We refer to [39] for a detailed performance evaluation of the ECDSA key generation. The results of our performance evaluation are shown in Table I.

Computation Time. We measure the computation time required by the users to perform the different algorithms. For the case of two-party protocols (e.g., Setup and Lock) we consider the time for the two users together. We make two main observations: First, the script-based construction based on discrete logarithm is faster than scriptless AMHLs. Second, all the algorithms require computation time of at most 60 milliseconds on a commodity hardware.

		Generic	**Schnorr**	**ECDSA**
Setup	Time (ms)	$0.3 \cdot n$	$1 \cdot n$	$1 \cdot n$
	Comm (bytes)	$96 \cdot n$	$128 \cdot n$	$128 \cdot n$
Lock	Time (ms)	–	2	60
	Comm (bytes)	32	256	416
Rel	Time (ms)	–	0.002	0.02
	Comm (bytes)	0	0	0
Vf	Time (ms)	–	0.6	0.06
	Comm (bytes)	0	0	0
Comp Cost (gas)		$350849 \cdot n$	0	0
Lock size (bytes)		32	$32 + \|m\|$	$32 + \|m\|$
Open size (bytes)		32	64	64

TABLE I: Comparison of the resources required to execute the algorithms for the different AMHLs. We denote by n the length of the path. We denote the negligible computation times by – (e.g., single memory read). We denote the size of an application-dependent message by $|m|$ (e.g., a transaction in a payment-channel network).

Communication Overhead. We measure the communication overhead as the amount of information that users need to exchange during the execution of interactive protocols, in particular, Setup and Lock. As expected, the generic construction based on discrete logarithm requires less communication overhead than scriptless constructions. The scriptless construction based on ECDSA requires a higher communication overhead. This is mainly due to having the signing key distributed multiplicatively and a more complex structure of the final signature when compared to the Schnorr approach.

Computation Cost. We measure the computation cost in terms of the gas required by a smart contract implementing the corresponding algorithm in Ethereum. Naturally, we consider this cost only for the generic approach based on discrete logarithm. We observe that setting up the corresponding contract requires 350849 unit of gas per hop. At the time of writing, each AMHL therefore costs considerably less than 0.01 USD.

Application Overhead. We measure the overhead incurred by the application in terms of the memory required to handle application-dependent data, i.e., information defining the lock and the opening. In tune with the rest of measurements, the generic construction based on discrete logarithms requires the smallest amount of memory, both for lock and opening information. The different scriptless approaches require the same amount of memory from the application.

Scalability. We study the running time and communication overhead required by each of the roles in a multi-hop lock protocol (i.e., sender, receiver and intermediate user). We consider only the generic approach and the ECDSA construction as representative of the scriptless approach. In the absence of significant metrics from current PCNs, we consider a path length of ten hops as suggested for similar payment networks such as the Ripple credit network [41].

Regarding the computation time, the sender requires 3ms with the generic approach and 10ms with the ECDSA scriptless approach. The computation time at intermediate users remain below 1ms for ECDSA and negligible with the generic approach as they only have to check the consistency of the locks with the predecessor and the successor, independently of the length of the path. Similarly, the computation overhead of the receiver remains below 1ms as she only checks if a given key is valid to open the lock according to the Vf algorithm. In summary, a non-private payment over a path of 5 users takes over 600ms as reported in [42]. Extending it with the constructions presented in this work provides formal privacy guarantees at virtually no overhead.

Regarding the communication overhead, the sender must send a message of about 960 bytes for the generic approach while about 1280 bytes are required instead if ECDSA scriptless locks are used. Since Sphinx, the anonymous communication network used in the LN, requires padded messages at each node to ensure anonymity, we foresee that every intermediate user must forward a message of the same size.

Comparing these results with other multi-hop and privacy-preserving PCNs available in the literature, we make the following observations. First, the overhead for the constructions presented in this work are in tune with TeeChain [38], where the overhead per hop is about 0.4 ms in a setting where cryptographic operations required for the multi-hop locks have been replaced by a trusted execution environment. Second, our constructions significantly reduce the communication and computation overhead required by multi-hop HTLC [42]: While a payment using multi-hop HTLC requires approximately 5 seconds and 17MB of communication, our approach requires only few milliseconds and less than 1MB.

In summary, the evaluation results show that even with an unoptimized implementation, our constructions offer significant improvements on computation and communication overhead and are ready to be deployed in practice.

VII. Applications

A. Payment-Channel Networks

AMHLs can be generically combined with a blockchain B to construct a fully-fledged PCN. Loosely speaking, the transformation works as follows: In the first round the sender sets up the locks running the Setup algorithm, then each pair of intermediate users executes the Lock protocol and establishes the following AMHL contract.

AMHL (Alice, Bob, ℓ, x, t):
1) If Bob produces the condition k such that $\mathsf{Vf}(\ell, k) = 1$ before t days, Alice pays Bob x coins.
2) If t days elapse, Alice gets back x coins.

Where ℓ is the output lock and x and t are chosen as specified in Section II. Note that we have to assume that B supports the Vf algorithm and time management in its script language. The rest of the payment is unchanged except that the intermediate users execute the Rel algorithm to extract a valid key k to claim the corresponding payment. In Appendix F, we provide the exact description of the algorithms and we prove the following theorem.

Theorem 4 (Informal)**.** *Let* B *a secure blockchain and let* $\mathbb{L}$ *be a secure AMHL, then we can construct a secure PCN (as defined in [42]).*

Note that even though we defined security of AMHLs in the UC framework, the composition of multiple AMHL instances in one protocol as needed for realizing PCNs does not come for free if those instances have shared state. Formally, such shared state can arise from the use of a shared KGen algorithm. Consequently, we need to show for the KGen algorithms of the presented constructions that they behave independently over multiple invocations and finally make use of the JUC theorem [19] to obtain the composability result.

This shows that AMHLs are the only cryptographic primitive (except for the blockchain) needed to construct PCNs. The only limitation is that the blockchain needs to support the verification of the corresponding contract in their scripting language (see the discussion above). For this reason, the scriptless-construction are preferred for those blockchains where the scripting language does not support the evaluation of a homomorphic one-way function (such as Bitcoin).

Application to the Lightning Network. When applied to the LN, the ECDSA AMHL construction conveys several advantages: First, it eliminates the security issues existing in the current LN due to the use of the HTLC contract. Second, it reduces the transaction size as a single signature is required per transaction. This has the benefit of lowering the communication overhead, the transaction fees, and the blockchain memory requirements for closing a payment channel. In fact, we have received feedback from the LN community indicating the suitability of our ECDSA-based construction. Moreover, results from the implementation and testing done by LN developers are available [28], [29].

The applicability of our proposals are not restricted to the LN or Bitcoin: There exist other PCNs that could similarly take advantage of the scriptless AMHLs presented in this work. For instance, the Raiden Network has been presented as a payment channel network for solving the scalability issue in Ethereum. The adoption of our ECDSA scriptless AMHLs would bring the same benefits to the Raiden Network.

B. Atomic Swaps

Assume two users U_0 and U_1 holding coins in two different cryptocurrencies and want to exchange them. An *atomic swap* protocol ensures that either the coins are swapped or the balances are untouched, i.e., the exchange must be performed atomically. The widely used protocol for atomic swaps described in [16] leverages the HTLC contract to perform the swap. In a nutshell, an atomic swap can be seen as a multi-hop payment over a path of the form (U_0, U_1, U_0). This approach inherits the security concerns of HTLC contract. Scriptless AMHLs also enhance this application domain with formally proven security guarantees.

Additionally, our constructions contribute to the *fungibility* of the coins, a crucial aspect for any available (crypto)currency. Current protocols rely on transactions that are clearly distinguishable from regular payments (i.e., one-to-one payments). In particular, atomic swap transactions contain the HTLC contract, in contrast to regular transactions. Scriptless AMHLs eliminate this issue since even atomic swap transactions only require a single signature from a public key, making them indistinguishable from regular payments. Similar arguments also apply for multi-hop payments in PCNs.

C. Interoperable PCNs

Among the cryptocurrencies existing today, an interesting problem consists in performing a multi-hop payment where each link represents a payment channel defined in a different cryptocurrency. In this manner, a user with a payment channel funded in a given cryptocurrency can use it to pay to another user with a payment channel in a different cryptocurrency. Currently, the InterLedger protocol [54] tackles this problem with a mechanism to perform cross-currency multi-hop payments that relies on the HTLC contract, aiming to ensure the payment atomicity across different hops.

However, apart from the already discussed issues associated with HTLC, the InterLedger protocol mandates that all cryptocurrencies implement HTLC contracts. This obviously hinders the deployment of this approach. Instead, it is possible to use the different AMHL constructions presented in this work on a single path, as described in Section V-E, therefore expanding the domain of cross-currency multi-hop payments.

VIII. Related Work

A recent work [24] shows a protocol to compute an ECDSA signature using multi-party computation. However, it is not as efficient as Lindell's approach [39].

There exists extensive literature proposing constructions for payment channels [22], [23], [37], [50]. These works focus on a single payment channel, and their extension to PCNs remain an open challenge. TumbleBit [33] and Bolt [32] support off-chain payments while achieving payment anonymity guarantees. However, the anonymity guarantees of these approaches are restricted to single-hop payments and their extension to support multi-hop payments remains an open challenge.

State channels [25], [35], [44] and state channel networks [26] cannot work with prominent cryptocurrencies except from Ethereum. TeeChain [38] requires the availability of a trusted execution environment at each user. Instead, our proposal can be seamlessly deployed today in virtually all cryptocurrencies, including Ethereum. In addition, AMHL enables operations between different blockchains, which is clearly not the case for Ethereum-only solutions. If we focus on the specific setting of payment channels in Ethereum, AMHL is more efficient (i.e., it requires less gas and bytes) as payment conditions are encoded in the signature and not in additional scripts. Finally, [25], [26] provide a different privacy notion: two endpoints can communicate privately but the intermediate nodes know that a virtual channel is opened between them. This information is instead concealed with AMHL. Formalizing this privacy leakage and comparing it with our privacy definition is an interesting future work.

The LN has emerged as the most promising approach for PCN in practice. Its current description [6] is being followed by several implementations [5], [8], [10]. However, these implementations suffer from the security and privacy issues with PCNs as described in this work. Instead, we provide several constructions for AMHLs that can be leveraged to have secure and anonymous multi-hop payments.

Malavolta et al. [42] propose a protocol for secure and anonymous multi-hop payments compatible with the current LN. Their approach, however, imposes an overhead of around 5 MB for the nodes in the network, therefore hindering its deployability. Here, we propose several efficient constructions that require only a few bytes of communication.

In the recent literature, we can find proposals for secure and privacy-preserving atomic swaps. Tesseract [14] leverages trusted hardware to perform real time cryptocurrency exchanges. The Merkleized Abstract Syntax Trees (MAST) protocol has been proposed as a privacy solution for atomic swaps [36]. However, MAST relies on scripts that are not available in the major cryptocurrencies today. Moreover, specific contracts for atomic swaps hinder the fungibility of the currency: An observer can easily differentiate between a regular payment and a payment resulting from an atomic swap.

IX. Conclusion

We rigorously study the cryptographic core functionality for security, privacy, and interoperability guarantees in PCNs, presenting a new attack on today's PCNs (the wormhole attack) and proposing a novel cryptographic construction (AMHLs). We instantiate AMHLs in two settings: script-based and scriptless. In the script-based setting, we demonstrate that AMHLs can be realized from any (linear) homomorphic operation. In the scriptless setting, we propose a construction based on ECDSA, thereby catering the vast majority of cryptocurrencies deployed today. Our performance evaluation shows that AMHLs are practical: All operations take less than 100 milliseconds to run and introduce a communication overhead of less than 500 bytes.

We show that AMHLs can be combined in a single path and are of interest in several applications apart from PCNs, such as atomic swaps and interoperable PCNs. In fact, LN developers have implemented and tested AMHL for LN. In the future, we plan to devise cryptographic instantiations of PCNs for the few cryptocurrencies not yet covered, most notably Monero.

Acknowledgements. The authors would like to thank Elizabeth Stark, Conner Fromknecht and Olaluwa Osuntokun (Lightning Network Labs) for insightful discussions on the writeup of this paper. They would also like to thank Foteini Baldimitsi for her helpful comments on Universal Composability.

This work has been partially supported by the National Science Foundation under grant CNS-1719196, the European Research Council (ERC) under the European Unions Horizon 2020 research (grant agreement No 771527-BROWSEC); by Netidee through the project EtherTrust (grant agreement 2158) and PROFET (grant agreement P31621); by the Austrian Research Promotion Agency through the Bridge-1 project PR4DLT (grant agreement 13808694); by COMET K1 SBA, ABC; by Chaincode Labs and the Austrian Science Fund (FWF) through the Meitner program; by the German research foundation (DFG) through the collaborative research center 1223; by the German Federal Ministry of Education and Research (BMBF) through the project PROMISE (16KIS0763); and by the state of Bavaria at the Nuremberg Campus of Technology (NCT). NCT is a research cooperation between the Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU) and the Technische Hochschule Nürnberg Georg Simon Ohm (THN).

References

[1] "5 potential use cases for bitcoin's lightning network," https://tinyurl.com/y6u4tnda.

[2] "Blockchain explorer information," https://blockchain.info/.

[3] "Charm: A framework for rapidly prototyping cryptosystems," https://github.com/JHUISI/charm.

[4] "Ethereum website," https://www.ethereum.org/.

[5] "Lightning network daemon," https://github.com/lightningnetwork/lnd.

[6] "Lightning network specifications," https://github.com/lightningnetwork/lightning-rfc.

[7] "Raiden network," http://raiden.network/.

[8] "A scala implementation of the lightning network," https://github.com/ACINQ/eclair.

[9] "Stress test prepares visanet for the most wonderful time of the year," Blog entry, http://www.visa.com/blogarchives/us/2013/10/10/stress-test-prepares-visanet-for-the-most-wonderful-time-of-the-year/index.html.

[10] "c-lightning – a lightning network implementation in c," Acceses in May 2018, https://github.com/ElementsProject/lightning.

[11] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. D. Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolic, S. W. Cocco, and J. Yellick, "Hyperledger fabric: A distributed operating system for permissioned blockchains," in *EuroSys*, 2018, pp. 30:1–30:15.

[12] M. Backes, A. Kate, P. Manoharan, S. Meiser, and E. Mohammadi, "Anoa: A framework for analyzing anonymous communication protocols," in *CSF*, 2013, pp. 163–178.

[13] M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols," in *CCS*, 1993.

[14] I. Bentov, Y. Ji, F. Zhang, Y. Li, X. Zhao, L. Breidenbach, P. Daian, and A. Juels, "Tesseract: Real-time cryptocurrency exchange using trusted hardware," in *ePrint Archive*, 2017, p. 1153. [Online]. Available: http://eprint.iacr.org/2017/1153

[15] M. Blum, P. Feldman, and S. Micali, "Non-interactive zero-knowledge and its applications," in *Symposium on Theory of Computing*, 1988, pp. 103–112.

[16] S. Bowe and D. Hopwood, "Hashed time-locked contract transactions," Bitcoin Improvement Proposal, https://github.com/bitcoin/bips/blob/master/bip-0199.mediawiki.

[17] J. Camenisch and A. Lysyanskaya, "A formal treatment of onion routing," in *CRYPTO*, 2005.

[18] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *FOCS*, 2001, pp. 136–.

[19] R. Canetti and T. Rabin, "Universal composition with joint state," in *Annual International Cryptology Conference*, 2003, pp. 265–281.

[20] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. Gün Sirer, D. Song, and R. Wattenhofer, "On Scaling Decentralized Blockchains," in *FC*, 2016, pp. 106–125.

[21] I. Damgård, "On σ-protocols," *Lecture Notes, University of Aarhus, Department for Computer Science*, 2002.

[22] C. Decker, R. Russel, and O. Osuntokun, "eltoo: A simple layer2 protocol for bitcoin," https://blockstream.com/eltoo.pdf.

[23] C. Decker and R. Wattenhofer, "A fast and scalable payment network with bitcoin duplex micropayment channels," in *Stabilization, Safety, and Security of Distributed Systems*, 2015.

[24] J. Doerner, Y. Kondi, E. Lee, and a. shelat, "Secure two-party threshold ecdsa from ecdsa assumptions," in *S&P*, 2018.

[25] S. Dziembowski, L. Eckey, S. Faust, and D. Malinowski, "Perun: Virtual payment hubs over cryptocurrencies," in *ePrint*, 2017. [Online]. Available: https://eprint.iacr.org/2017/635

[26] S. Dziembowski, S. Faust, and K. Hostakova, "General state channel networks," in *CCS*, 2018.

[27] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Conference on the Theory and Application of Cryptographic Techniques*, 1986.

[28] C. Fromknecht, "Instantiating scriptless 2p-ecdsa: fungible 2-of-2 multisigs for bitcoin today," Talk at Scaling-Bitcoin 2018, https://tokyo2018.scalingbitcoin.org/transcript/tokyo2018/scriptless-ecdsa.

[29] ——, "tpec: 2p-ecdsa signatures," Github repository, https://github.com/cfromknecht/tpec.

[30] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Secure distributed key generation for discrete-log based cryptosystems," *Journal of Cryptology*, vol. 20, no. 1, pp. 51–83, 2007.

[31] 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, 1988.

[32] M. Green and I. Miers, "Bolt: Anonymous payment channels for decentralized currencies," in *CCS*, 2017.

[33] E. Heilman, L. Alshenibr, F. Baldimtsi, A. Scafuro, and S. Goldberg, "TumbleBit: An untrusted bitcoin-compatible anonymous payment hub," in *NDSS*, 2017.

[34] J. Katz, U. Maurer, B. Tackmann, and V. Zikas, "Universally composable synchronous computation," in *Theory of cryptography*, 2013, pp. 477–498.

[35] R. Khalil and A. Gervais, "Revive: Rebalancing off-blockchain payment networks," in *CCS*, 2017, pp. 439–453.

[36] J. Lau, "Merkelized abstract syntax tree," Bitcoin Improvement Proposal, https://tinyurl.com/yc9jh6lv.

[37] J. Lind, I. Eyal, P. R. Pietzuch, and E. G. Sirer, "Teechan: Payment channels using trusted execution environments," 2016, http://arxiv.org/abs/1612.07766.

[38] J. Lind, O. Naor, I. Eyal, F. Kelbert, P. R. Pietzuch, and E. G. Sirer, "Teechain: Reducing storage costs on the blockchain with offline payment channels," in *Systems and Storage Conference*, 2018, p. 125.

[39] Y. Lindell, "Fast Secure Two-Party ECDSA Signing," in *CRYPTO*, 2017, pp. 613–644.

[40] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," in *CCS*, 2016, pp. 17–30.

[41] G. Malavolta, P. Moreno-Sanchez, A. Kate, and M. Maffei, "SilentWhispers: Enforcing security and privacy in credit networks," in *NDSS*, 2017.

[42] G. Malavolta, P. Moreno-Sanchez, A. Kate, M. Maffei, and S. Ravi, "Concurrency and privacy with payment-channel networks," in *CCS*, 2017.

[43] P. McCorry, M. Möser, S. F. Shahandashti, and F. Hao, "Towards bitcoin payment networks," in *ACISP*, 2016.

[44] A. Miller, I. Bentov, R. Kumaresan, and P. McCorry, "Sprites: Payment channels that go faster than lightning," in *FC*, 2019.

[45] P. Moreno-Sanchez, N. Modi, R. Songhela, A. Kate, and S. Fahmy, "Mind your credit: Assessing the health of the ripple credit network," in *WWW*, 2018, pp. 329–338.

[46] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *International Conference on the Theory and Applications of Cryptographic Techniques*, 1999, pp. 223–238.

[47] A. Pfitzmann and M. Hansen, "A Terminology for Talking about Privacy by Data Minimization: Anonymity, Unlinkability, Undetectability, Unobservability, Pseudonymity, and Identity Management," https://dud.inf.tu-dresden.de/literatur/Anon_Terminology_v0.34.pdf, Aug. 2010, v0.34.

[48] A. Poelstra, "Lightning in scriptless scripts," Mailing list post, https://lists.launchpad.net/mimblewimble/msg00086.html.

[49] ——, "Scriptless scripts," Presentation slides, https://download.wpsoftware.net/bitcoin/wizardry/mw-slides/2017-05-milan-meetup/slides.pdf.

[50] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," Technical Report, https://lightning.network/lightning-network-paper.pdf.

[51] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," *Journal of the ACM*, vol. 56, no. 6, p. 34, 2009.

[52] S. Roos, P. Moreno-Sanchez, A. Kate, and I. Goldberg, "Settling payments fast and private: Efficient decentralized routing for path-based transactions," in *NDSS*, 2018.

[53] C.-P. Schnorr, "Efficient signature generation by smart cards," *Journal of cryptology*, vol. 4, no. 3, pp. 161–174, 1991.

[54] E. S. Stefan Thomas, “A Protocol for Interledger Payments,” Whitepaper, https://interledger.org/interledger.pdf.

[55] P. Wuille, “Schnorr Bitcoin Improvement Proposal,” https://github.com/sipa/bips/blob/bip-schnorr/bip-schnorr.mediawiki.

Appendix

A. Wormhole Attack

In this section, we first describe generically the wormhole attack. We then formally prove that no two-round multi-hop payment in a payment-channel network (without broadcasts) is robust against this attack.

First, we give a model for such two-round multi-hop payments that is capturing existing two-round constructions such as the standard HTLC-based construction as it is presented in [42]. As we are only interested in the underlying locking mechanism of PCNs, we omit information on channel capacities and fees and simply model PCNs as graphs of the form $\mathbb{G} = (\mathbb{V}, \mathbb{E})$.

For arguing about the communication in protocols, we formally state the notions of a path in a PCN as well as of a communication round along a path.

Definition 3 (Path in a PCN). *Let $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ be a PCN. We call a vector $(u_1, \dots, u_n)$ of users a* path in *$\mathbb{G}$ if it holds for all $i \in [1, n-1]$ that $(u_i, u_{i+1}) \in \mathbb{E}$*

In the following, we represent a message from u_1 to u_2 with content m as a tuple $M = \langle u_1, m, u_2 \rangle$. Intuitively, one round of communication in a PCN allows for the consecutive execution of pairwise protocols along a path. This is formally captured by the following definition:

Definition 4 (Communication round in PCNs). *Let $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ be a PCN and let $\pi = (u_1, \dots, u_n)$ be a path in $\mathbb{G}$ with length longer than one. We we call a vector of consecutive messages $(M_1, \dots, M_l)$ a* round of communication along π *if the following conditions hold:*

1) $M_1 = \langle u_1, m, u_2 \rangle$ for some message m

2) for all $i \in [1; l]$ it holds that either

a) $M_i = \langle u_j, m, u_{j+1} \rangle$ or

b) $M_i = \langle u_{j+1}, m, u_j \rangle$ for some $j \in [1; n-1]$ and some message m

3) for all $i \in [1; l-1]$ it holds that if $M_i = \langle u_j, m_i, u_k \rangle$ then either

a) $M_{i+1} = \langle u_k, m, u_j \rangle$ or

b) $M_{i+1} = \langle u_{\max(j,k)}, m, u_{\max(j,k)+1} \rangle$ for some message m

Here condition 1) ensures that a round always starts with a message of the first user in the path to its right neighbor. Condition 2) makes sure that messages can only be exchanged between neighbors in the path. Finally, condition 3) encodes that every message in a path can either be followed by a message in the reversed direction (in case that a protocol between two neighbors is performed) or alternatively a protocol between the next two neighboring nodes is initiated (by the ‚right’ party in the former protocol now sending a message to its right neighbor). Together these definitions ensure that only pairwise protocols can be performed and once that they need to be carried out consecutively along the path.

For describing the essence of the wormhole attack, we use an abstract view on payments in PCNs where we assume payments along a path π to consist of a *commitment phase* and a *releasing phase*. In the commitment phase (which constitutes a communication round along π), pairwise locks (e.g. HTLCs) between the parties along the payment path are created in pairwise locking protocols between the neighboring nodes.

Definition 5 (Commitment phase in a PCN). *Let $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ be a PCN and let $\pi = (u_1, \dots, u_n)$ be a path in $\mathbb{G}$. A protocol encompassing one communication round along π is called a* commitment phase along π *if as a result each user u_i (for $i \in [1; n]$) learns some (shared) pieces of commitment information $\ell_{i,i+1}$, $\ell_{i-1,i}$ (with $\ell_{n,n+1}$ and $\ell_{0,1}$ being empty). We call $\{\ell_{i,i+1}\}_{i \in 1,n-1}$ the output of the commitment phase.*

In the releasing phase, starting from the payment's receiver, keys for ’opening’ the locks are released (as the condition R in the lightning network). The releasing phase thereby consists of a communication round along *rev*(π) (the reversal of π). These keys should satisfy the property that each path node can – given a valid key for an outgoing lock – derive a key for it's incoming lock.

Definition 6 (Releasing phase in a PCN). *Let $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ be a PCN and let $\pi = (u_1, \dots, u_n)$ be a path in $\mathbb{G}$. Further let C be a commitment phase along π and Vf be a boolean function taking two arguments. Additionally, let π' be a subpath of π. A protocol encompassing one communication round along rev(π') is called a* releasing phase for C along *rev*(π') *if as a result, each user u_i in π' learns some information $k_{i-1,i}$ such that $\mathsf{Vf}(\ell_{i-1,i}, k_{i-1,i}) = 1$.*

With the notion of a subpath being defined as follows:

Definition 7 (Subpath). *Let $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ be a PCN and let $\pi = (u_1, \dots, u_n)$ be a path in $\mathbb{G}$. A path $\pi' = (u_1, \dots, u_m)$ is considered a subpath of π if for all u_i in π' there are paths π_i (for $i \in [0; m]$) such that*

$\pi = \pi_0 \cdot (u_1) \cdot \pi_1 \cdot (u_2) \cdot \pi_2 \cdot \cdot \cdot \pi_{m-1} \cdot (u_m) \cdot \pi_m$ *(where $\cdot$ denotes path concatenation).*

Note that we assume communication to be restricted to nodes connected by a direct link in the PCN (as ensured by 4). This prevents that besides the specified messages in the releasing phase, keys can be sent to previous nodes in the path (e.g., via broadcast).

Figure 6 shows the payment from Alice to Edward in the abstract setting. Initially, Edwards gives a trapdoor t to Alice. Using this, Alice starts the commitment phase by creating the lock $\ell_{A,B}$ with Bob. To this end, Alice and Bob might use their secret local states s_A and s_B. In the same fashion all following pairwise locks are created in the commitment phase till reaching Edward. Edward then starts the releasing phase by using the trapdoor he initially sent to Alice for creating the key $k_{D,E}$ for opening lock $\ell_{D,E}$. From this key (and the information learned in the commitment phase), Dave can derive key $k_{C,D}$. In this fashion the whole lock chain can be released.

Note that in the setting of only two rounds of communication (one along the payment path π and one along the reversed payment path *rev*(π)), the initial secret local states of the users involved in a payment are completely independent from π and consequently from the local states of the other nodes in the path. This is as none of the users received any path-specific information upfront.

As a consequence, two nodes u_i and u_j (with $1 < i + 1 < j$) on a payment path can exclude intermediate nodes u_k (with $i < k < j$) from taking part in the releasing phase as follows: After completing the commitment phase in an honest fashion, the releasing phase proceeds honestly till reaching u_j. At this point u_j can derive a key $k_{j-1,j}$ for releasing the lock $\ell_{j-1,j}$ with (honest) user u_{j-1}. Instead of releasing this lock, u_j forwards $k_{j-1,j}$ to u_i which again can use this key for producing a valid key for lock $\ell_{i-1,i}$ with its predecessor u_{i-1}. This is possible as no secret information from the nodes $\{u_k\}_{i<k<j}$ is required for generating a valid key for $l_{i-1,i}$. Otherwise also opening the final lock would already require secret information from some intermediate nodes. As these pieces of secret information from the intermediate nodes are however completely unrelated to the path and consequently from the trapdoor t even the receiver could not earn the necessary knowledge from the trapdoor for opening the last lock. So finally, node u_i can release lock $\ell_{i-1,i}$ and consecutively all remaining locks can be released without contacting nodes $\{u_k\}_{i<k<j}$ at all. Together with the assumption that nodes $\{u_k\}_{i<k<j}$ cannot receive information through other channels than the direct communication with their immediate neighbors in the path and the fact that keys for locks can only be derived from the initial key, there is no way for nodes $\{u_k\}_{i<k<j}$ to open the locks with their successors in the path.

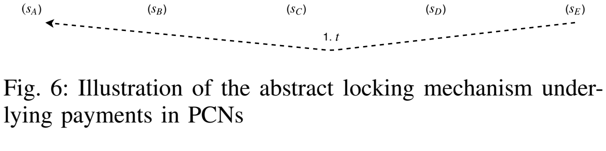

Fig. 6: Illustration of the abstract locking mechanism underlying payments in PCNs

1) Inevitability of wormhole attacks in two-round payment protocols: In PCNS, payments that only encompass two rounds of communication are inevitably vulnerable to wormhole attacks. [4] More specifically, this means that when no path-specific information was communicated to the intermediate nodes of the payment path before performing the payment, nodes located between two corrupted users in the path can always be bypassed in the releasing phase. This situation occurs in cases where the path is not known upfront, but routing is performed dynamically (e.g., [52]).

We formally prove the following theorem:

Theorem 5. *Let $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ be a PCN and let $\pi = (u_1, \ldots, u_n)$ be a path in $\mathbb{G}$ with length longer than two. Further, let u_i, u_j be nodes in π for some $0 < i < j \leq n$ and let $\pi' = (u_1, \ldots, u_i, u_j, \ldots, u_n)$ be the path omitting the nodes between u_i and u_j. Assume that u_1 and u_n share initial common knowledge t while all other nodes do not have any initial shared knowledge. Then each payment along π that is carried out by a protocol P consisting of a commitment phase C along π and a releasing phase along rev(π) for C, can also be carried out by a protocol P' consisting of C and a releasing phase for C along rev(π') given that u_i and u_j collude. Additionally, for all users u_k with $i < k < j$, P' is indistinguishable from the protocol only consisting of C.*

Proof: Let $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ be a PCN and let $\pi = (u_1, \ldots, u_n)$ be a path in $\mathbb{G}$ with length longer than two. Further, let u_i, u_j be nodes in π for some $0 < i < j \leq n$ and let $\pi' = (u_1, \ldots, u_i, u_j, \ldots, u_n)$ be the path omitting the nodes between U_i and U_j. Assume a payment along the path π with u_1 being the sender and u_n being the receiver. Without loss of generality, assume u_i and u_j to be controlled by the attacker and all other nodes on the path to be honest. We show that

[4] Note that we assume here PCNs of the previously described structure hence requiring that payments encompass a commitment and a revealing phase and communication to be restricted to direct neighbors.

the view of honest nodes u_l with $l < i$ or $l > j$ in the scenario of u_i and u_j performing a wormhole attack on a successful payment don't differ from the view in the scenario of a successful payment. More precisely, we show how to construct a one-round successful releasing phase along *rev*(π'), where successful means that each user in π' can derive a valid key for its outgoing locks.

In addition, we show that the view of honest nodes u_m with $i < m < j$ in the scenario of the wormhole attack do not differ from their view in the scenario of an unsuccessful payment. More formally, we show that it is indistinguishable for those nodes whether the one-round releasing phase was omitted or carried out along *rev*(π').

To this end, we first show how an attacker can simulate the behavior of the nodes $u_{i+1}, \dots, u_{j-1}$ without changing the view of the honest nodes u_l with $l < i$ or $l > j$.

In the commitment phase, the locks along the path π have been created correctly. In the locking protocol between u_l and u_i, u_i behaves honestly and consequently u_l's view is the same as in the honest case. In parallel to starting the locking protocol between u_i and u_{i+1}, the attacker locally simulates the locking protocols for the user's $u_{i+1}, \dots, u_j$ and creates simulated locks $\ell_{i+1}, \dots, \ell_j$. This is possible as by sampling random local states for those nodes, the attacker can run the locking protocol locally. Finally, u_j can continue the commitment phase in an honest manner using as own local state the one resulting from the simulated commitments. This cannot be distinguished by node u_{j+1} as it's own local state is unrelated to the local states of the other intermediate nodes.

As we consider the case of a successful payment, the releasing phase will be performed honestly by nodes $u_n, \dots u_{j+1}$. When u_{j+1} releases the lock between u_j and u_{j+1} with key k_j, then the attacker can simulate releasing the locks $\ell_{j-1}, \dots, \ell_i$ locally without publishing the corresponding keys. This is possible as the attacker can use the local states of the intermediate nodes $u_{i+1}, \dots, u_{j-1}$ from the simulated commitment phase for deriving keys $k_{j-1}, \dots, k_{i-1}$. As k_{i-1} is hence also a valid key for the honestly created lock l_{i-1}, the releasing phase can from this point be concluded in an honest manner.

Finally, we can observe that nodes $u_{i+1}, \dots, u_{j-1}$ are not contacted at all in the releasing phase of the payment which is the same as in the case that the payment was unsuccessful, i.e., the releasing phase was not initiated by the receiver at all. ■

B. AMHLs Correctness

In this section, we define the notion of correctness for AMHLs.

Definition 8 (Correctness of AMHLs). *Let $\mathbb{L}$ be a AMHL, $\lambda \in \mathbb{N}^+$ and $n \in \mathsf{poly}(\lambda)$. Let $(U_0, \dots, U_n) \in \mathbb{U}^n$ be a vector of users, $(\mathsf{sk}_0, \dots, \mathsf{sk}_{n-1})$ and $(\mathsf{sk}_1^*, \dots, \mathsf{sk}_n^*)$ two vectors of private keys and $(\mathsf{pk}_0, \dots, \mathsf{pk}_{n-1})$ a vector of shared public keys such that for all $0 \le i < n$, it holds that*

$$\{(\mathsf{sk}_i, \mathsf{pk}_i), (\mathsf{sk}_{i+1}^*, \mathsf{pk}_i)\} \leftarrow \langle \mathsf{KGen}_{U_i}(1^\lambda), \mathsf{KGen}_{U_{i+1}}(1^\lambda)\rangle.$$

Let $(s_0^I, \dots, s_n^I)$ be vector of initial states and k_n be a key such that for all $0 \le i < n$

$$\{s_0^I, \dots, (s_n^I, k_n)\} \leftarrow \left\langle \begin{array}{c} \mathsf{Setup}_{U_0}(1^\lambda, U_1, \dots, U_n) \\ \dots \\ \mathsf{Setup}_{U_n}(1^\lambda) \end{array} \right\rangle$$

Furthermore, let $(\ell_0, \dots, \ell_{n-1})$ be a vector of locks, $(s_1^L, \dots, s_n^L)$ and $(s_0^R, \dots, s_{n-1}^R)$ vectors of states, and $(k_0, \dots, k_{n-1})$ a vector of keys such that for all $0 \le i < n$, it holds that

$$\{(\ell_i, s_i^R), (\ell_i, s_{i+1}^L)\} \leftarrow \left\langle \begin{array}{c} \mathsf{Lock}_{U_i}(s_i^I, \mathsf{sk}_i, \mathsf{pk}_i) \\ \mathsf{Lock}_{U_{i+1}}(s_{i+1}^I, \mathsf{sk}_{i+1}^*, \mathsf{pk}_i) \end{array} \right\rangle$$

and

$$k_i \leftarrow \mathsf{Rel}(k_{i+1}, (s_{i+1}^I, s_{i+1}^L, s_{i+1}^R))$$

where s_n^R is $\perp$. We say that $\mathbb{L}$ is correct if there exists a negligible function negl *such that for all $0 \le i < n$ it holds that*

$$\Pr[\mathsf{Vf}(\ell_i, k_i) = 1] \ge 1 - \mathsf{negl}(\lambda).$$

C. Schnorr-based Scriptless Construction

In the following we cast the idea of Poelstra [48] in our framework.

Schnorr Signatures. Let $\mathbb{G}$ be an elliptic curve group of order q with base point G and let $H : \{0,1\}^* \to \{0,1\}^{|q|}$ be a collision resistant hash function (modeled as a random oracle). The key generation algorithm $\mathsf{KGen}_{\mathsf{schnorr}}(1^\lambda)$ of a Schnorr signature [53] samples some $x \leftarrow_\$ \mathbb{Z}_q$ and sets the corresponding public key as $Q := x \cdot G$. To sign a message m, the signing algorithm $\mathsf{Sig}_{\mathsf{schnorr}}(\mathsf{sk}, m)$ samples some $k \leftarrow_\$ \mathbb{Z}_q$, computes $e := H(Q \| k \cdot G \| m)$, sets $s := k - xe$, and returns $\sigma := (R, s)$, where $R := k \cdot G$. The verification $\mathsf{Vf}_{\mathsf{schnorr}}(\mathsf{pk}, \sigma, m)$ returns 1 if and only if $s \cdot G = R + H(Q \| R \| m) \cdot Q$. Schnorr signatures are known to be strongly unforgeable against the discrete logarithm assumption [31]. We assume the existence of a 2-party protocol $\Pi_{\mathsf{KGen}}^{\mathsf{schnorr}}$ where the two players, on input x_0 and x_1, set a shared public key $Q := (x_0 + x_1) \cdot G$. Such a protocol can be implemented using standard techniques

```
$\underline{\mathsf{Lock}_{U_i}(s_i^I, \mathsf{sk}_i, \mathsf{pk})}$                                  $\underline{\mathsf{Lock}_{U_{i+1}}(s_{i+1}^I, \mathsf{sk}_{i+1}, \mathsf{pk})}$
parse $s_i^I$ as $(Y_0', Y_0, y_0)$                                   parse $s_{i+1}^I$ as $(Y_1', Y_1, y_1)$
$r_0 \leftarrow\$ \mathbb{Z}_q$                                              $r_1 \leftarrow\$ \mathbb{Z}_q$
$R_0 := r_0 \cdot G$                                                  $R_1 := r_1 \cdot G$
$\pi_0 \leftarrow \mathsf{P_{NIZK}}(r_0, \{\exists r_0 \text{ s.t. } R_0 = r_0 \cdot G\})$           $\pi_1 \leftarrow \mathsf{P_{NIZK}}(r_1, \{\exists r_1 \text{ s.t. } R_1 = r_1 \cdot G\})$
                                      $\xleftarrow{\mathsf{com}}$ $(\mathsf{decom}, \mathsf{com}) \leftarrow \mathsf{Commit}(1^\lambda, (R_1, \pi_1))$
                                      $\xrightarrow{(R_0, \pi_0)}$ $b_1 \leftarrow \mathsf{V_{NIZK}}(\{\exists r_0 \text{ s.t. } R_0 = r_0 \cdot G\}, \pi_0)$
                                                                      if $b_1 = 0$ then abort
                                                                      $e := H(\mathsf{pk} \| R_0 + R_1 + Y_1' \| m)$
if $\mathsf{V_{com}}(\mathsf{com}, \mathsf{decom}, (R_1, \pi_1)) \neq 1$ then abort $\xleftarrow{(\mathsf{decom}, R_1, \pi_1, s)}$ $s := r_1 + e \cdot \mathsf{sk}_{i+1} \bmod q$
$b_0 \leftarrow \mathsf{V_{NIZK}}(\{\exists r_1 \text{ s.t. } R_1 = r_1 \cdot G\}, \pi_1)$
if $b_0 = 0$ then abort
$e := H(\mathsf{pk} \| R_0 + R_1 + Y_0 \| m)$
if $s \cdot G \neq R_1 + e \cdot (\mathsf{pk} - \mathsf{sk}_i \cdot G)$ then abort
$s' := s + r_0 + e \cdot \mathsf{sk}_i \bmod q$                           $\xrightarrow{s'}$ if $s' \cdot G \neq R_0 + R_1 + e \cdot \mathsf{pk}$ then abort
return $((m, \mathsf{pk}), s')$                                          return $((m, \mathsf{pk}), (R_0 + R_1 + Y_1', s'))$
```

```
$\underline{\mathsf{Rel}(k, (s^I, s^L, s^R))}$
parse $s^I$ as $(Y', Y, y)$
parse $k$ as $(R, s)$
parse $s^L$ as $(W_0, w_1)$
$w := w_1 + s - (s^R + y)$
$\bmod\ q$
return $(W_0, w)$
```

```
$\underline{\mathsf{Vf}(\ell, k)}$
parse $\ell$ as $(m, \mathsf{pk})$
parse $k$ as $(R, s)$
$e := H(\mathsf{pk} \| R \| m)$
return $s \cdot G = R + e \cdot \mathsf{pk}$
```

Fig. 7: Algorithms and protocols for the Schnorr-based construction. The Setup protocol is as defined in Fig. 5.

and we denote the corresponding ideal functionality by $\mathcal{F}_{\mathsf{kgen}}^{\mathsf{schnorr}}$.

Description. Let $\mathbb{G}$ be an elliptic curve group of order q with base point G and let $H : \{0,1\}^* \to \{0,1\}^{|q|}$ be a hash function. The Schnorr-based construction is formally described in Fig. 7. The key generation algorithm consists of a call to the $\mathcal{F}_{\mathsf{kgen}}^{\mathsf{schnorr}}$ functionality. At the end of a successful run, U_i receives (x_i, pk) whereas U_j obtains (x_j, pk), where $\mathsf{pk} := (x_i + x_j) \cdot G$. The setup of a AMHL is identical to the ECDSA-based construction and can be found in Fig. 5.

Prior to the locking phase, two users U_i and U_{i+1} (implicitly) agree on the value Y_i and on a message m to be signed. Each message is assumed to be unique for each session (e.g., contains a transaction identifier). The locking algorithm consists of an "incomplete" distributed signing of m. First, the two parties agree on a randomly chosen element $R_0 + R_1$ using a standard coin tossing protocol, then they set the randomness of the signature to be $R := R_0 + R_1 + Y_i$. Note that at this point the parties cannot complete the signature since they do not know the discrete logarithm of Y_i. Instead, they jointly compute the value $s := r_0 + r_1 + e \cdot (x_0 + x_1)$ as if Y_i was not part of the randomness, where e is the hash of the transcript so far. The resulting (R, s) is *not* a valid signature on m, since the additive term y^* (where $y^* \cdot G = Y_i$) is missing from the computation of s. However, rearranging the terms, we have that $(R, s + y^*)$ is a valid signature on m. This implies that, once the discrete logarithm of Y_i is revealed, a valid signature m can be computed by U_{i+1}. Leveraging this observation, U_{i+1} can enforce an *atomic* opening: The subsequent locking (between U_{i+1} and U_{i+2}) is conditioned on some $Y_{i+1} = Y_i + y_{i+1} \cdot G$. This way, the opening of the right lock reveals the value $y^* + y_{i+1}$ and U_{i+1} can immediately extract y^* and open its left lock with a valid signature on m. The security of the construction is shown by the following theorem. We refer the reader to Appendix E for a full proof.

Theorem 6. *Let* COM *be a secure commitment scheme, and let* NIZK *be a non-interactive zero knowledge proof. If Schnorr signatures are strongly existentially unforgeable, then the construction in Fig. 7 UC-realizes the ideal functionality* $\mathcal{F}$ *in the* $(\mathcal{F}_{\mathsf{kgen}}^{\mathsf{schnorr}}, \mathcal{F}_{\mathsf{syn}}, \mathcal{F}_{\mathsf{smt}}, \mathcal{F}_{\mathsf{anon}})$*-hybrid model.*

D. Comparison of Privacy Notions and Guarantees

In this section we discuss our notion of relationship anonymity as the privacy notion of interest for PCNs and compare it with other possible privacy notions described in the literature related to PCN.

Our privacy model faithfully captures the reality of currently deployed PCN. In particular, Malavolta et al. [42] showed that it captures the well established notion of relationship anonymity. In a nutshell, relationship anonymity [47] requires that, given two simultaneous successful payment operations between sender$_{\{0,1\}}$ and receiver$_{\{0,1\}}$ that share the same path with at least one honest intermediate user, corrupted intermediate users cannot determine the correct pair (sender$_b$, receiver$_b$) for a given payment with probability better than $1/2$ (i.e., guessing). Note that this holds only for payments of the same value, since such an information is trivially leaked to intermediate users, i.e., each user can monitor how adjacent links evolve and infer the amount that was transferred.

An alternative privacy notion is described in BOLT [32]. There, authors propose *payment anonymity*. Intuitively, payment anonymity requires that the merchant, even in collaboration with a set of malicious customers, learns nothing about a customer's spending pattern beyond the information available outside the payment protocol.

While this privacy notion additionally hides the value that is transacted, it is restricted to single-hop payments and does not consider the crucial aspect of conditional payment required when more than one intermediate user takes part in the payment. As discussed in Section III, many well-established networks use paths with multiple intermediaries and it is reasonable to expect long paths also in the LN. To obtain the best of both worlds, one could envision a protocol where private one-hop payments are performed "at the edges" (i.e., between sender and first hop as well as between last hop and the receiver) while the rest of intermediate users carry out a multi-hop payment à la LN.

However, this approach raises several questions. First, it is unclear whether the hypothetical resulting privacy guarantees are stronger or weaker than those presented in this work. It is possible that the naïve combination of the two systems would completely break down the guarantees of both schemes. Techniques presented in both works might be required to develop a new system. Second, BOLT requires a blockchain supporting a rich scripting language and it is therefore not compatible with prominent cryptocurrencies (such as Bitcoin). Thus, making this system Bitcoin-compatible would require fundamentally new techniques.

In summary, although it seems to be an interesting research direction, further work is required to study this approach and its privacy properties.

E. Security Analysis

Throughout the analysis we denote by $\mathsf{poly}(\lambda)$ any function that is bounded by a polynomial in λ. We denote any function that is negligible in the security parameter by $\mathsf{negl}(\lambda)$. We say that an algorithm is PPT if it is modelled as a probabilistic Turing machine whose running time is bounded by some function $\mathsf{poly}(\lambda)$.

In the following we recall the (non standard) ideal functionalities that our protocols build on and we elaborate on the security analysis of our constructions. We stress that the copies of these functionality that are invoked as subroutines are fresh independent instances and therefore the composition theorem [18] directly applies to our settings.

Key Generation Functionalities. Our ideal functionality for key generation of Schnorr signatures $\mathcal{F}^{\mathsf{schnorr}}_{\mathsf{keygen}}$ provides the users with the interface described below. This essentially models a distributed key generation for discrete logarithm-based schemes, which is a very well-studied problem (see, e.g., [30]).

KeyGen($\mathbb{G}$, G, q)

```
Upon invocation by both $U_0$ and $U_1$ on input ($\mathbb{G}$, $G$, $q$):
sample $x \leftarrow_{\$} \mathbb{Z}_q$ and compute $Q = x \cdot G$
set $\mathsf{sk}_{U_0,U_1} = x$
sample $x_0$ and $x_1$ randomly
sample a hash function $H : \{0,1\}^* \to \{0,1\}^{|q|}$
send $(x_0, Q, H)$ to $U_0$ and $(x_1, Q, H)$ to $U_1$
ignore future calls by $(U_0, U_1)$
```

Our ideal functionality for key generation of ECDSA $\mathcal{F}^{\mathsf{ECDSA}}_{\mathsf{keygen}}$ is taken almost in verbatim from [39] and it is given in the following.

KeyGen($\mathbb{G}$, G, q)

```
Upon invocation by both $U_0$ and $U_1$ on input ($\mathbb{G}$, $G$, $q$):
sample $x \leftarrow_{\$} \mathbb{Z}_q$ and compute $Q = x \cdot G$
sample $x_0$ and $x_1$ randomly
sample a hash function $H : \{0,1\}^* \to \{0,1\}^{|q|}$
sample a key pair $(\mathsf{sk}_{U_0,U_1}, \mathsf{pk}_{U_0,U_1}) \leftarrow \mathsf{KGen}_{\mathsf{HE}}(1^\lambda)$
compute $c \leftarrow \mathsf{Enc}_{\mathsf{HE}}(\mathsf{pk}, \tilde{r})$ for a random $\tilde{r}$
send $(x_0, Q, H, \mathsf{sk})$ to $U_0$ and $(x_1, Q, H, c)$ to $U_1$
ignore future calls by $(U_0, U_1)$
```

Generic Construction. Here we elaborate on the proof of Theorem 2.

Proof:

We define the following sequence of hybrids, where we gradually modify the initial experiment.

$\mathcal{H}_0$: Is identical to the protocol as described in Section V-B.

$\mathcal{H}_1$: Consider the following ensemble of variables in the interaction with $\mathcal{A}$: A honest user U_i, a key pair $(\mathsf{sk}_i, \mathsf{pk})$, a state s^I, a tuple $(\ell_i, \ell_{i+1}, s^L, s^R)$ such that

$$\{\cdot, (\ell_i, s^L)\} \leftarrow \langle \cdot, \mathsf{Lock}_{U_i}(s^I, \mathsf{sk}_i, \mathsf{pk})\rangle$$

and

$$\{(\ell_{i+1}, s^R), \cdot\} \leftarrow \langle \mathsf{Lock}_{U_i}(s^I, \mathsf{sk}_i, \mathsf{pk}), \cdot\rangle.$$

If, for any set of these variables, the adversary returns some k such that $\mathsf{Vf}(\ell_{i+1}, k) = 1$ and $\mathsf{Vf}(\ell_i, \mathsf{Rel}(k, (s^I, s^L, s^R))) \neq 1$, then the experiment aborts.

$\mathcal{H}_2$: Consider the following ensemble of variables in the interaction with $\mathcal{A}$: A pair of honest users (U_0, U_i) a set of (possibly corrupted) users $(U_1, \dots, U_n)$, a key pair $(\mathsf{sk}_i, \mathsf{pk})$, a set of initial states

$$(s_0^I \dots, s_n^I) \leftarrow \left\langle \begin{array}{c} \mathsf{Setup}_{U_0}(1^\lambda, U_1, \dots, U_n), \\ \dots, \\ \mathsf{Setup}_{U_n}(1^\lambda) \end{array} \right\rangle,$$

and a pair of locks (ℓ_{i-1}, ℓ_i) such that

$$\{\cdot, (\ell_{i-1}, \cdot)\} \leftarrow \langle \cdot, \mathsf{Lock}_{U_i}(s_i^I, \mathsf{sk}_i, \mathsf{pk})\rangle$$

and

$$\{(\ell_i, \cdot), \cdot\} \leftarrow \langle \mathsf{Lock}_{U_i}(s_i^I, \mathsf{sk}_i, \mathsf{pk}), \cdot\rangle.$$

If, for any set of these variables, the adversary returns some k_{i-1} such that $\mathsf{Vf}(\ell_{i-1}, k_{i-1}) = 1$ before the user U_i outputs a key k_i such that $\mathsf{Vf}(\ell_i, k_i) = 1$, then the experiment aborts.

$\mathcal{H}_3$: Let $S = (U_0, \dots, U_m)$ be an ordered set of (possibly corrupted) users. We say that that an ordered subset $A = (U_1, \dots, U_j)$ is *adversarial* if U_i is honest and $(U_{i+1}, \dots, U_j)$ are corrupted. Note that every set of users can be expressed as a concatenation of adversarial subsets, that is $S = (A_1 || \dots || A_{m'})$, for some $m' \leq m$. Whenever a honest user is requested to set up a lock for a certain set $S = (A_1 || \dots || A_{m'})$, it initializes an independent lock for each subset (A_i, A_{i+1}^0), where A_{i+1}^0 is the first element of the $(i+1)$-th set, if present. Whenever some A_{i+1}^0 is requested to release the key for the corresponding lock (recall that all A_{i+1}^0 are honest nodes) it releases the key for the fresh lock (A_i, A_{i+1}^0) instead.

$\mathcal{S}$: The interaction of the simulator is identical to $\mathcal{H}_3$ except that the actions of $\mathcal{S}$ are dictated by the interaction with $\mathcal{F}$. The simulator reads the communication of $\mathcal{A}$ with the honest users via $\mathcal{F}_{\mathsf{anon}}$ and is queried by $\mathcal{F}$ on the following set of inputs.

1) $(\cdot, \cdot, \cdot, \cdot, \mathsf{Init})$: The simulator reconstructs the adversarial set (defined above) from the ids and sets up a fresh lock chain.
2) $(\cdot, \mathsf{Lock})$: The simulator initiates the locking procedure with the adversary and replies with $\perp$ if the execution is not successful.
3) $(\cdot, \mathsf{Rel})$ The simulator releases the key of the corresponding lock and publishes it.

If $\mathcal{A}$ interacts with a honest user (e.g., by releasing a lock) the simulator queries the corresponding interface of $\mathcal{F}$.

Note that the simulator is efficient and interacts as the adversary with the ideal world. Furthermore, the simulation is always consistent with the ideal world, i.e., if the adversary's action is not supported by the interfaces of $\mathcal{F}$ the simulation aborts. What is left to be shown is that the neighboring hybrids are indistinguishable to the eyes of the environment $\mathcal{E}$.

Lemma 1. *For all PPT distinguishers $\mathcal{E}$ it holds that*

$$\mathsf{EXEC}_{\mathcal{H}_0, \mathcal{A}, \mathcal{E}} \equiv \mathsf{EXEC}_{\mathcal{H}_1, \mathcal{A}, \mathcal{E}}.$$

Proof: Follows from the homomorphic property of the function g: Recall that a key-lock pair (k, ℓ) is valid if and only if $g(k) = \ell$. Let (k_i, ℓ_i) be the output of $\mathcal{A}$, by construction we have that $\ell_i = \ell_{i-1} + g(y_i)$, for some (ℓ_{i-1}, y_i), which is part of the state of the honest node. Since the release algorithm computes $k_i - y_i$ we have that

$$\begin{aligned} g(k_i - y_i) &= g(k_i) - g(y_i) \\ &= \ell_i - g(y_i) \\ &= \ell_{i-1} + g(y_i) - g(y_i) \\ &= \ell_{i-1} \end{aligned}$$

with probability 1, by the homomorphic property of g. ■

Lemma 2. *For all PPT distinguishers $\mathcal{E}$ it holds that*

$$\mathsf{EXEC}_{\mathcal{H}_1, \mathcal{A}, \mathcal{E}} \approx \mathsf{EXEC}_{\mathcal{H}_2, \mathcal{A}, \mathcal{E}}.$$

Proof: Let $q \in \mathsf{poly}(\lambda)$ be a bound on the number of interactions. Recall that $\mathcal{H}_1$ and $\mathcal{H}_2$ differ only for the case where the adversary outputs a key for a honestly generated lock before the trapdoor is released. Assuming towards contradiction that the probability that this event happens is non-negligible, we can construct the following reduction against the one-wayness of g: On input some $Y^* \in \mathcal{R}$, the reduction guesses a session $j \in [1, q]$ and some index $i \in [1, n]$. The setup algorithm of the j-th session is modified as follows: Y_i is set to be Y^*. Then, for all $\iota \in [i-1, 0]$, the setup samples some

$y_\iota \in \mathcal{D}$ and returns $(Y_\iota = Y_{\iota+1} - g(y_\iota), Y_{\iota+1}, y_\iota)$. The setup samples a random $y_i \in \mathcal{D}$ and sets $Y_{i+1} = g(y_i)$. Then, for $\iota \in [i+1, n-1]$, the setup samples $y_\iota \in \mathcal{D}$ returns $(Y_\iota, Y_\iota + g(y_\iota), y_\iota)$. The nodes $(U_1, \dots, U_{n-1})$ are given the corresponding output (except for U_i) and U_n is given $(Y_{n-1}, \sum_{j=i}^{n-1} y_j)$. If the node U_i is requested to release the lock, the reduction aborts. At some point of the execution the adversary $\mathcal{A}$ outputs some $y*$, and the reduction returns $y^* + y_{i-1}$.

The reduction is clearly efficient and, whenever j and i are guessed correctly, the reduction does not abort. Since the group defined by g is abelian, the distribution induced by the modified setup algorithm is identical to the original (except for the initial state of U_1). Also note that, whenever j and i are guessed correctly, the user U_i is honest and therefore the adversary does not not see the corresponding internal state. It follows that the reduction is identical to $\mathcal{H}_1$, to the eyes of the adversary. Finally, whenever the adversary outputs some valid k_{i-1} for ℓ_{i-1}, then it holds that $g(k_{i-1}) = \ell_{i-1}$. Substituting we have that

$$\begin{aligned} g(k_{i-1}) &= \ell_{i-1} \\ g(y^*) &= Y_{i-1} \\ g(y^*) &= Y^* - g(y_{i-1}) \\ g(y^*) + g(y_{i-1}) &= Y^* \\ g(y^* + y_{i-1}) &= Y^*. \end{aligned}$$

It follows that the reduction is successful with probability at least $\frac{1}{q \cdot n \cdot \mathsf{poly}(\lambda)}$. This proves our statement. ■

Lemma 3. *For all PPT distinguishers $\mathcal{E}$ it holds that*

$$\mathsf{EXEC}_{\mathcal{H}_2,\mathcal{A},\mathcal{E}} \equiv \mathsf{EXEC}_{\mathcal{H}_3,\mathcal{A},\mathcal{E}}.$$

Proof: Recall that adversarial sets are always interleaved by a honest node. Therefore in $\mathcal{H}_2$ for each adversarial set starting at index i there exists a y such that $Y_i = Y_{i-1} + g(y)$ and $\mathcal{A}$ is not given y. Since y is randomly sampled from $\mathcal{D}$ we have that $Y_{i-1} + g(y) \equiv Y'$, for some Y' sampled uniformly from $\mathcal{R}$, which corresponds to the view of $\mathcal{A}$ in $\mathcal{H}_3$. ■

Lemma 4. *For all PPT distinguishers $\mathcal{E}$ it holds that*

$$\mathsf{EXEC}_{\mathcal{H}_3,\mathcal{A},\mathcal{E}} \equiv \mathsf{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{E}}.$$

Proof: The changes between the two experiments are only conceptual and the equivalence of the views follows. ■

This concludes our analysis. ■

Schnorr-based Construction. Here we prove Theorem 6.

Proof:

We define the following sequence of hybrids, where we gradually modify the initial experiment.

$\mathcal{H}_0$: Is identical to the protocol as described in Appendix C.

$\mathcal{H}_1$: All the calls to the commitment scheme are replaced with interactions with the ideal functionality $\mathcal{F}_{\mathsf{com}}$, defined in the following.

Commit(*sid*, m)

Upon invocation by U_i (for $i \in \{0,1\}$):
record (sid, i, m) and send $(\mathsf{com}, \mathit{sid})$ to U_{1-i}
if some $(\mathit{sid}, \cdot, \cdot)$ is already stored ignore the message

Decommit(*sid*)

Upon invocation by U_i (for $i \in \{0,1\}$):
if (sid, i, m) is recorded then send $(\mathsf{decom}, \mathit{sid}, m)$ to U_{1-i}

Instead of calling the Commit algorithm on some message m, the parties sent a message of the form **Commit(sid, m)** to the ideal functionality, and the decommitment algorithm is replaced with a call to **Decommit(sid)**. The verifying party simply records messages from $\mathcal{F}_{\mathsf{com}}$.

$\mathcal{H}_2$: All the calls to the NIZK scheme are replaced with interactions with the ideal functionality $\mathcal{F}_{\mathsf{NIZK}}$:

Prove(*sid*, x, w)

Upon invocation by U_i (for $i \in \{0,1\}$):
if $R(x, w) = 1$ then send $(\mathsf{proof}, \mathit{sid}, x)$ to U_{1-i}

Instead of running the proving algorithm in input (x, w), the proving party queries the functionality on **Prove(sid, x, w)**. The verifier records the messages from $\mathcal{F}_{\mathsf{NIZK}}$.

$\mathcal{H}_3, \mathcal{H}_4, \mathcal{H}_5, \mathcal{S}$: The subsequent hybrids are defined as $\mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3, \mathcal{S}$, respectively, in Theorem 2.

As argued before, the simulator is efficient and the interaction is consistent with the inputs of the ideal functionality. In the following we prove the indistinguishability of the neighboring experiments.

Lemma 5. *For all PPT distinguishers $\mathcal{E}$ it holds that*

$$\mathsf{EXEC}_{\mathcal{H}_0,\mathcal{A},\mathcal{E}} \approx \mathsf{EXEC}_{\mathcal{H}_1,\mathcal{A},\mathcal{E}}.$$

Proof: Follows directly from the security of the commitments scheme COM. ■

Lemma 6. *For all PPT distinguishers $\mathcal{E}$ it holds that*

$$\mathsf{EXEC}_{\mathcal{H}_1,\mathcal{A},\mathcal{E}} \approx \mathsf{EXEC}_{\mathcal{H}_2,\mathcal{A},\mathcal{E}}.$$

Proof: Follows directly from the security of the non-interactive zero-knowledge scheme NIZK. ■

Lemma 7. *For all PPT distinguishers $\mathcal{E}$ it holds that*

$$\mathsf{EXEC}_{\mathcal{H}_2,\mathcal{A},\mathcal{E}} \approx \mathsf{EXEC}_{\mathcal{H}_3,\mathcal{A},\mathcal{E}}.$$

Proof: In order to show this claim, we introduce an intermediate experiment.

$\mathcal{H}_2^*$: The locking algorithms are substituted with the following ideal functionality. Such an interface is called by both parties on input m and $y = \sum_{j=0}^{i} y_j$, where i is the position of the lock in the chains and the y_j are defined as in the original protocol. Note that the key sk_{U_0,U_1} refers to the previously established key in the call to the $\mathcal{F}_{\mathsf{kgen}}^{\mathsf{schnorr}}$.

Sign(m, y)

```
Upon invocation by both $U_0$ and $U_1$ on input $(m, y)$:
compute $(R, s) = \mathsf{Sig}_{\mathsf{schnorr}}(\mathsf{sk}_{U_0,U_1}, m)$
return $(R, s - y)$
```

We defer the indistinguishability proof to lemma 8. Let cheat by the event that triggers an abort of the experiment in $\mathcal{H}_3$, that is, the adversary returns some k such that $\mathsf{Vf}(\ell_{i+1}, k) = 1$ and that $\mathsf{Vf}(\ell_i, \mathsf{Rel}(k, (s^I, s^L, s^R))) \neq 1$. Assume towards contradiction that $\Pr[\mathsf{cheat} \mid \mathcal{H}_2^*] \geq \frac{1}{\mathsf{poly}(\lambda)}$, then we can construct the following reduction against the strong-existential unforgeability of Schnorr signatures: The reduction receives as input a public key pk and samples an index $j \in [1, q]$, where $q \in \mathsf{poly}(\lambda)$ is a bound on the total amount of interactions. Let Q be the key generated in the j-th interaction, the reduction sets $Q = \mathsf{pk}$. All the calls to the signing algorithm are redirected to the signing oracle. If the event cheat happens, the reduction returns corresponding $(k^*, \ell^*) = (\sigma^*, (m^*, \mathsf{pk}^*))$, otherwise it aborts.

The reduction is clearly efficient. Assume for the moment that j is the index of the interaction where cheat happens, and let $i + 1$ be the index that identifies the lock ℓ^* in the corresponding chain. Note that in case the guess of the reduction is correct we have that $\mathsf{pk}^* = \mathsf{pk}$. Since cheat happens we have that $\mathsf{Vf}_{\mathsf{schnorr}}(\mathsf{pk}^*, m^*, \sigma^*) = 1$ and the release fails, i.e., $\mathsf{Vf}(\ell_i, \mathsf{Rel}(k, (s_i^I, s_i^L, s_i^R)) \neq 1$ (where ℓ_i is the lock in the previous position as ℓ^* in the same chain). Recall that the release algorithm parses s_i^L as $(W_{i,0}, w_{i,1})$ and σ^* as (R^*, s^*) and returns $(W_{i,0}, w_{i,1} + s^* - (s_i^R + y_i))$. Substituting with the corresponding values

$$\begin{aligned}&\left(W_{i,0}, w_{i,1} + s^* - (s_i^R + y_i)\right)\\ &= \left(R_i, s_i - \sum_{j=0}^{i-1} y_j + s^* - s_j - \sum_{j=0}^{i} y_j + y_i\right)\\ &= \left(R_i, s_i + s^* - s_j\right),\end{aligned}$$

where s_j is the answer of the oracle on the j-th session on input m_j. This implies that $s^* \neq s_j$, otherwise (R_i, s_i) would be a valid signature since it is an output of the signing oracle. Since each message uniquely identifies a session (the same message is never queried twice to the interface **Sign**$(m$,$y)$) this implies that $(\sigma^*, (m^*, \mathsf{pk}^*))$ is a valid forgery. By assumption this happens with probability at least $\frac{1}{q \cdot \mathsf{poly}(\lambda)}$, which is a contradiction and proves that $\Pr[\mathsf{cheat} \mid \mathcal{H}_2^*] \leq \mathsf{negl}(\lambda)$. Since the experiments $\mathcal{H}_2$ and $\mathcal{H}_3$ differ only when cheat happens (and $\mathcal{H}_3$ aborts), we are only left with showing the indistinguishability of $\mathcal{H}_2$ and $\mathcal{H}_2^*$.

Lemma 8. *For all PPT distinguishers $\mathcal{E}$ it holds that*

$$\mathsf{EXEC}_{\mathcal{H}_2,\mathcal{A},\mathcal{E}} \approx \mathsf{EXEC}_{\mathcal{H}_2^*,\mathcal{A},\mathcal{E}}.$$

Proof: The proof consists of the description of the simulator for the interactive lock algorithm. We describe two simulators depending on whether the honest adversary is playing the role of the "left" or "right" party. For each proof, both the simulators implicitly check that the given witness is valid and abort if this is not the case.

1) Left corrupted: Prior to the interaction the simulator is sent $(Y, y, (\mathsf{prove}, \{\exists y^* \text{ s.t } y^* \cdot G = Y\}, y^*))$, which is the state corresponding to the execution of the lock. After agreeing on a message m, the simulator sends $(\mathsf{com}, \mathit{sid})$ to $\mathcal{A}$, for a random sid. The simulator also queries the interface **Sign** on input m, y^* and receives a signature $\sigma = (R, s)$. At some point of the execution $\mathcal{A}$ sends $(R_0, (\mathsf{prove}, \{\exists r_0 \text{ s.t } r_0 \cdot G = R_0\}, r_0))$. The simulator replies with

$$\left(\begin{array}{l}\mathsf{decom}, \mathit{sid}, \left(\begin{array}{l} R^* = R - (R_0 + Y), \\ \mathsf{proof}, \mathit{sid}, \\ \{\exists r^* \text{ s.t } r^* \cdot G = R^*\}\end{array}\right), \\ R^*, (s - r_0 - e \cdot x_0)\end{array}\right)$$

where $e = H(\mathsf{pk} \| R^* \| m)$ and x_0 is the value returned by the key generation to $\mathcal{A}$. The rest of the execution is unchanged.

2) Right corrupted: Prior to the interaction the simulator is sent $(Y, y, (\mathsf{prove}, \{\exists y^* \text{ s.t } y^* \cdot G = Y\}, y^*))$, which is the state corresponding to the execution of the lock. After agreeing on a message m, the simulator is given

$$\left(\mathsf{com}, \mathit{sid}, \left(\begin{array}{ll} R_1, & \begin{array}{l}\mathsf{prove}, \mathit{sid}, \\ \{\exists r_1 \text{ s.t } r_1 \cdot G = R_1\}, r_1\end{array}\end{array}\right)\right)$$

by $\mathcal{A}$. The simulator then queries the interface **Sign** on input m, y^* and receives a signature $\sigma = (R, s)$. The simulator sends $(R^* = R - (R_1 + Y), (\mathsf{proof}, \mathit{sid}, \{\exists r^* \text{ s.t } r^* \cdot G = R^*\}))$ to $\mathcal{A}$ and receives $((\mathsf{decom}, \mathit{sid}), s^*)$ in response. The simulator checks whether $s^* = r_1 + e \cdot x_1$, where $e = H(\mathsf{pk} \| R^* \| m)$, and returns s if this is the case.

Both simulators are obviously efficient and the distributions induced by the simulated views are identical to the ones of the original protocol. ■

This concludes the proof of lemma 7. ■

Lemma 9. *For all PPT distinguishers $\mathcal{E}$ it holds that*

$$\mathsf{EXEC}_{\mathcal{H}_3,\mathcal{A},\mathcal{E}} \approx \mathsf{EXEC}_{\mathcal{H}_4,\mathcal{A},\mathcal{E}}.$$

Proof: Let $q \in \mathsf{poly}(\lambda)$ be a bound on the number of interactions. Let cheat denote the events that triggers an abort in $\mathcal{H}_4$ but not in $\mathcal{H}_3$. In the following we are going to show that $\Pr[\mathsf{cheat} \mid \mathcal{H}_3] \leq \mathsf{negl}(\lambda)$, thus proving the indistinguishability of $\mathcal{H}_3$ and $\mathcal{H}_4$. Assume that the converse is true, then we can construct the following reduction against the discrete logarithm problem (which is implied by the sEUF of Schnorr): On input some $Y^* \in \mathbb{G}$, the reduction guesses a session $j \in [1, q]$ and some index $i \in [1, n]$. The setup algorithm of the j-th session is modified as follows: Y_i is set to be Y^*. Then, for all $\iota \in [i-1, 0]$, the setup samples some $y_\iota \in \mathbb{Z}_q$ and returns $(Y_\iota = Y_{\iota+1} - y_\iota \cdot G, Y_{\iota+1}, y_\iota)$. The setup samples a random $y_i \in \mathbb{Z}_q$ and sets $Y_{i+1} = y_i \cdot G$. Then, for $\iota \in [i+1, n-1]$, the setup samples $y_\iota \in \mathbb{Z}_q$ returns $(Y_\iota, Y_\iota + y_\iota \cdot G, y_\iota)$. The nodes $(U_1, \ldots, U_{n-1})$ are given the corresponding output (except for U_i) and U_n is given $(Y_{n-1}, \sum_{j=i}^{n-1} y_j)$. If the node U_i is requested to release the lock, the reduction aborts. At some point of the execution the adversary $\mathcal{A}$ outputs some $k^* = (R^*, s^*)$. The reduction parses s^R as the updated state of U_i and returns $s^* + y_{i-1} - s^R$.

The reduction is clearly efficient and, whenever j and i are guessed correctly, the reduction does not abort. Since the group $\mathbb{G}$ is abelian and the U_i is honest, the distribution induced by the modified setup algorithm is identical to the original to the eyes of the adversary. Recall that cheat happens only in the case where k^* is a valid opening for ℓ_i and the release algorithm is successful on input k^* (if the last condition is not satisfied both $\mathcal{H}_3$ and $\mathcal{H}_4$ abort). Substituting, we have that s^R is of the form $r_0 + r_1 + e \cdot (x0 + x_1) - y = s' - y$, for some $y \in \mathbb{Z}_q$. Since the release is successful, then it must be the case that $(R' = (r_0 + r_1) \cdot G + Y_{i-1}, s')$ is a valid Schnorr signature on the message m_{i-1} (agreed by the two parties in the locking algorithm for ℓ_{i-1}), which implies that $y \cdot G = Y_{i-1}$. As argued in the proof of lemma 7, if $s^* \neq s'$, then we have an attacker against the strong unforgeability of the signature scheme. It follows that $s^* = s'$ with all but negligible probability. Substituting we have

$$\begin{aligned}
(s^* + y_{i-1} - s^R) \cdot G &= (s^* + y_{i-1} - s' + y) \cdot G \\
&= (y_{i-1} + y) \cdot G \\
&= y_{i-1} \cdot G + y \cdot G \\
&= y_{i-1} \cdot G + Y_{i-1} \\
&= y_{i-1} \cdot G + (Y^* - y_{i-1} \cdot G) \\
&= Y^*
\end{aligned}$$

as expected. Since, by assumption, this happens with probability at least $\frac{1}{q \cdot n \cdot \mathsf{poly}(\lambda)}$ we have a successful attacker against the discrete logarithm problem. This proves our statement.

■

Lemma 10. *For all PPT distinguishers $\mathcal{E}$ it holds that*

$$\mathsf{EXEC}_{\mathcal{H}_4,\mathcal{A},\mathcal{E}} \equiv \mathsf{EXEC}_{\mathcal{H}_5,\mathcal{A},\mathcal{E}}.$$

Proof: Recall that adversarial sets are always interleaved by a honest node. Therefore in $\mathcal{H}_4$ for each adversarial set starting at index i there exists a y such that $Y_i = Y_{i-1} + y \cdot G$ and $\mathcal{A}$ is not given y. Since y is randomly sampled from $\mathbb{Z}_q$ we have that $Y + i - 1 + y \cdot G \equiv Y'$, for some Y' sampled uniformly from $\mathbb{G}$, which corresponds to the view of $\mathcal{A}$ in $\mathcal{H}_5$. ■

Lemma 11. *For all PPT distinguishers $\mathcal{E}$ it holds that*

$$\mathsf{EXEC}_{\mathcal{H}_5,\mathcal{A},\mathcal{E}} \equiv \mathsf{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{E}}.$$

Proof: The change is only syntactical and the indistinguishability follows. ■

This concludes our analysis. ■

ECDSA-based Construction. In the following we prove Theorem 3.

Proof: The sequence of hybrids that we define is identical to the one described in the proof of Theorem 6. In the following we prove the indistinguishability of neighboring experiments only for the cases where the argument needs to be modified. If the argument is identical, the proof is omitted.

Lemma 12. *For all PPT distinguishers $\mathcal{E}$ it holds that*

$$\mathsf{EXEC}_{\mathcal{H}_2,\mathcal{A},\mathcal{E}} \approx \mathsf{EXEC}_{\mathcal{H}_3,\mathcal{A},\mathcal{E}}.$$

Proof: In order to show this claim, we introduce an intermediate experiment.

$\mathcal{H}_2^*$: The locking algorithms is substituted with the interaction with the following ideal functionality. Recall that the key sk_{U_0,U_1} here refers to the key established during the call of the same pair of users to the key generation functionality. Also note that the locking algorithm is called by both parties on input m and

$y = \sum_{j=0}^{i} y_j$, where i is the position of the lock in the chains and the y_j are defined as in the original protocol.

Sign(m, y)

```
Upon invocation by both $U_0$ and $U_1$ on input $(m, y)$:
compute $(r, s) = \mathsf{Sig}_{\mathsf{ECDSA}}(\mathsf{sk}_{U_0,U_1}, m)$
return $(r, \mathsf{min}(s \cdot y, -s \cdot y))$
```

The indistinguishability proof of $\mathcal{H}_2$ and $\mathcal{H}_2^*$ is formally shown in lemma 13. Let cheat by the event that triggers an abort of the experiment in $\mathcal{H}_3$, that is, the adversary returns some k such that $\mathsf{Vf}(\ell_{i+1}, k) = 1$ and $\mathsf{Vf}(\ell_i, \mathsf{Rel}(k, (s^I, s^L, s^R))) \neq 1$. Assume towards contradiction that $\Pr[\mathsf{cheat} \mid \mathcal{H}_2^*] \geq \frac{1}{\mathsf{poly}(\lambda)}$, then we can construct the following reduction against the strong-existential unforgeability of ECDSA signatures: The reduction receives as input a public key pk and samples an index $j \in [1, q]$, where $q \in \mathsf{poly}(\lambda)$ is a bound on the total amount of interactions. Let Q be the key generated in the j-th interaction, the reduction sets $Q = \mathsf{pk}$. All the calls to the signing algorithm are redirected to the signing oracle. If the event cheat happens, the reduction returns corresponding $(k^*, \ell^*) = (\sigma^*, (m^*, \mathsf{pk}^*))$, otherwise it aborts.

The reduction runs in polynomial time. Assume for the moment that j is the index of the interaction where cheat happens, and let $i+1$ be the index that identifies the lock ℓ^* in the corresponding chain. Note that in case the guess of the reduction is correct we have that $\mathsf{pk}^* = \mathsf{pk}$. Since cheat happens we have that $\mathsf{Vf}_{\mathsf{ECDSA}}(\mathsf{pk}^*, m^*, \sigma^*) = 1$ and the release fails, i.e., $\mathsf{Vf}(\ell_i, \mathsf{Rel}(k^*, k^*, (s_i^I, s_i^L, s_i^R))) \neq 1$ (where ℓ_i is the lock in the previous position as ℓ^* in the same chain). Recall that the release algorithm parses s_i^L as $(w_{i,0}, w_{i,1})$, σ^* as (r^*, s^*), and s_i^R as (s', m, pk) and computes $t = w_1 \cdot (\frac{s'}{s^*} - y)^{-1}$ and $t' = w_1 \cdot (-\frac{s'}{s^*} - y)^{-1}$. Then it returns either $(w_{i,0}, \mathsf{min}(t, -t))$ or $(w_{i,0}, \mathsf{min}(t', -t'))$ depending on which verifies as a valid signature on m under pk. Substituting with the corresponding values (for the case t is the lower term)

$$
\begin{aligned}
(w_{i,0}, t) &= \left(r_i, w_{i,1} \cdot \left(\frac{s'}{s^*} - y \right)^{-1} \right) \\
&= \left(r_i, s_i \cdot \sum_{j=0}^{i-1} y_j \cdot \left(\frac{s_j \cdot \sum_{j=0}^{i} y_j}{s^*} - y_i \right)^{-1} \right)
\end{aligned}
$$

where s_j is the answer of the oracle on the j-th session on input the corresponding message m_j. If we set $s^* = s_j$ then we have

$$
\begin{aligned}
(w_{i,0}, t) &= \left(r_i, s_i \cdot \sum_{j=0}^{i-1} y_j \cdot \left(\sum_{j=0}^{i} y_j - y_i \right)^{-1} \right) \\
&= (r_i, s_i)
\end{aligned}
$$

which is a valid signature on m_i (since it is the output of the signing oracle) and the release would be successful. So this cannot happen and we can assume that $s^* \neq s_j$. A similar argument (substituting t with t') can be used to show that it must be the case that $s^* \neq -s_j$. Since each message uniquely identifies a session (the same message is never queried twice to the interface **Sign(**m**,**y**)**) this implies that $(\sigma^*, (m^*, \mathsf{pk}^*))$ is a valid forgery. By assumption this happens with probability at least $\frac{1}{q \cdot \mathsf{poly}(\lambda)}$, which is a contradiction and proves that $\Pr[\mathsf{cheat} \mid \mathcal{H}_2^*] \leq \mathsf{negl}(\lambda)$. Since the experiments $\mathcal{H}_2$ and $\mathcal{H}_3$ differ only when cheat happens (and $\mathcal{H}_3$ aborts), we are only left with showing the indistinguishability of $\mathcal{H}_2$ and $\mathcal{H}_2^*$.

Lemma 13. *For all PPT distinguishers $\mathcal{E}$ it holds that*

$$\mathsf{EXEC}_{\mathcal{H}_2,\mathcal{A},\mathcal{E}} \approx \mathsf{EXEC}_{\mathcal{H}_2^*,\mathcal{A},\mathcal{E}}.$$

Proof: The proof consists of the description of the simulator for the interactive lock algorithm. In the following we describe the two simulators for the locking protocol depending on whether the honest adversary is playing the role of the "left" or "right" party. For each zero-knowledge proof, both the simulators implicitly check that the given witness is valid and abort if this is not the case.

1) Left corrupted: Prior to the interaction the simulator is sent $(Y, y, (\mathsf{prove}, \{\exists y^* \text{ s.t } y^* \cdot G = Y\}, y^*))$, which is the state corresponding to the execution of the lock. After agreeing on a message m, the simulator sends $(\mathsf{com}, \mathit{sid})$ to $\mathcal{A}$, for a random *sid*. The simulator also queries the interface **Sign** on input m, y^* and receives a signature $\sigma = (r, s)$. The simulator sets $R = \frac{H(m)}{s} \cdot G + \frac{r}{s} \cdot \mathsf{pk}$. At some point of the execution $\mathcal{A}$ sends $(R_0, R_0', (\mathsf{prove}, \{\exists r_0 \text{ s.t } r_0 \cdot G = R_0 \text{ and } r_0 \cdot Y = R_0'\}, r_0))$. Then the simulator samples a $\rho \leftarrow \mathbb{Z}_{q^2}$ and computes $c' \leftarrow \mathsf{Enc}_{\mathsf{HE}}(\mathsf{pk}, s \cdot r_0 + \rho q)$. Then it provides the attacker with

$$
\mathsf{decom}, \mathit{sid}, \left(\begin{array}{l} R^* = (r_0)^{-1} \cdot R, \\ R_1 = y^{-1} \cdot R^*, \\ \mathsf{proof}, \mathit{sid}, \\ \left\{ \begin{array}{r} \exists r^* \text{ s.t } r^* \cdot G = R_1 \\ \text{and } r^* \cdot Y = R^* \end{array} \right\} \end{array} \right), \\
R_1, R^*, c'.
$$

The rest of the execution is unchanged.

The executions are identical except for the way c' is computed. In order to show the statistical proximity we invoke a the following helping lemma.

Lemma 14. *[39] For all $(r, s, p) \in \mathbb{Z}_q$ and for a random $\rho \in \mathbb{Z}_{q^2}$, the distributions $\mathsf{Enc}_{\mathsf{HE}}(\mathsf{pk}, r \cdot s \text{ mod } q + pq + \rho q)$ and $\mathsf{Enc}_{\mathsf{HE}}(\mathsf{pk}, r \cdot s \text{ mod } q + \rho q)$ are statistically close.*

In the real world c' is computed as $\mathsf{Enc}_{\mathsf{HE}}(\mathsf{pk}, r \cdot s \bmod q + pq + \rho q)$, for some p which is bounded by q since the only operation performed without modular reduction are one multiplication and one addition, which cannot increase the result by more than q^2. Since the distribution $\mathsf{Enc}_{\mathsf{HE}}(\mathsf{pk}, r \cdot s \bmod q + \rho q)$ is identical to the simulation, the indistinguishability follows.

2) Right corrupted: Prior to the interaction the simulator is sent $(Y, y, (\mathsf{prove}, \{\exists y^* \text{ s.t } y^* \cdot G = Y\}, y^*))$, which is the state corresponding to the execution of the lock. After agreeing on a message m, the simulator is given

$$\mathsf{com}, sid, \left(R_1, R_1', \begin{array}{l} \mathsf{prove}, sid, \\ \left\{ \begin{array}{c} \exists r_1 \text{ s.t } r_1 \cdot G = R_1 \text{ and} \\ r_1 \cdot Y = R_1' \end{array} \right\}, \\ r_1 \end{array} \right)$$

by $\mathcal{A}$. The simulator then queries the interface **Sign** on input m, y^* and receives a signature $\sigma = (r, s)$. Then it sets $R = \frac{H(m)}{s} \cdot G + \frac{r}{s} \cdot \mathsf{pk}$ and $R^* = R - (R_1 + Y)$ and sends $(R_0 = y^{-1} \cdot R^*, R^*, (\mathsf{proof}, sid, \{\exists r^* \text{ s.t } r^* \cdot G = R_0 \text{ and } r^* \cdot Y = R^*\}))$ to $\mathcal{A}$. The attacker sends $((\mathsf{decom}, sid), c')$ in response. The simulator checks

$$\mathsf{Dec}_{\mathsf{HE}}(\mathsf{sk}, c') = \tilde{r} \cdot r \cdot (r_1)^{-1} + H(m) \cdot r_1^{-1} \bmod q,$$

where $\tilde{r}$ was sampled in the key generation algorithm. If the check holds true, the simulator sends s to $\mathcal{A}$.

The distribution induced by the simulator is identical to the real experiment except for the way c is computed. Towards showing indistinguishability, consider the following modified simulator, that is given the oracle $\mathcal{O}(c', a, b)$ as defined in the following security experiment of the Paillier encryption scheme.

```
$\mathsf{Exp} - \mathsf{ecCPA}^{\mathcal{A}}_{\mathsf{HE}}(\lambda)$ :
$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KGen}_{\mathsf{HE}}(1^\lambda)$
$(w_0, w_1) \leftarrow\!\!\$\, \mathbb{Z}_q$
$Q = w_0 \cdot G$
$b \leftarrow\!\!\$\, \{0, 1\}$
$c \leftarrow \mathsf{Enc}_{\mathsf{HE}}(\mathsf{pk}, w_b)$
$b' \leftarrow \mathcal{A}(\mathsf{pk}, c, Q)^{\mathcal{O}(\cdot,\cdot,\cdot)}$
where $\mathcal{O}(c', a, b)$ returns 1 iff $\mathsf{Dec}_{\mathsf{HE}}(\mathsf{sk}, c') = a + b \cdot w_b$
return 1 iff $b = b'$
```

Instead of performing the last check, the simulator queries the oracle on input $(c', a = H(m) \cdot r_1^{-1}, b = r \cdot (r_1)^{-1})$. It is clear that the modified simulator accepts if and only if the simulator described above accepts. Assume towards contradiction that the modified simulator can be efficiently distinguished from the real world experiment. Then we can reduce to the security of Paillier as follows: On input (pk, c, Q), the reduction simulates the inputs of $\mathcal{A}$ as described in the modified simulator using the input pk, Q, and c as the corresponding variables. It is easy to see that the reduction is efficient. Note that if $b = 0$ then we have that $c = \mathsf{Enc}_{\mathsf{HE}}(\mathsf{pk}, w_0)$ and $Q = w_0 \cdot G$, which is identical to the real world execution. On the other hand if $b = 1$ then it holds that $c = \mathsf{Enc}_{\mathsf{HE}}(\mathsf{pk}, w_1)$ and $Q = w_0 \cdot G$, where w_1 is uniformly distributed in $\mathbb{Z}_q$, which is identical to the (modified) simulated experiment. This implies that the modified simulation is computationally indistinguishable from the real world experiment. Since the modified simulation and the simulation (as described above) are identical to the eyes of the adversary, the validity of the lemma follows. ■

This concludes the proof of lemma 12. ■

Lemma 15. *For all PPT distinguishers $\mathcal{E}$ it holds that*

$$\mathsf{EXEC}_{\mathcal{H}_3, \mathcal{A}, \mathcal{E}} \approx \mathsf{EXEC}_{\mathcal{H}_4, \mathcal{A}, \mathcal{E}}.$$

Proof: Let $q \in \mathsf{poly}(\lambda)$ be a bound on the number of interactions. Let cheat denote the event that triggers an abort in $\mathcal{H}_4$ but not in $\mathcal{H}_3$. In the following we are going to show that $\Pr[\mathsf{cheat} \mid \mathcal{H}_3] \leq \mathsf{negl}(\lambda)$, thus proving the indistinguishability of $\mathcal{H}_3$ and $\mathcal{H}_4$. Assume that the converse is true, then we can construct the following reduction against the discrete logarithm problem (which is implied by the sEUF of ECDSA): On input some $Y^* \in \mathbb{G}$, the reduction guesses a session $j \in [1, q]$ and some index $i \in [1, n]$. The setup algorithm of the j-th session is modified as follows: Y_i is set to be Y^*. Then, for all $\iota \in [i-1, 0]$, the setup samples some $y_\iota \in \mathbb{Z}_q$ and returns $(Y_\iota = Y_{\iota+1} - (y_\iota) \cdot G, Y_{\iota+1}, y_\iota)$. The setup samples a random $y_i \in \mathbb{Z}_q$ and sets $Y_{i+1} = y_i \cdot G$. Then, for $\iota \in [i+1, n-1]$, the setup samples $y_\iota \in \mathbb{Z}_q$ and returns $(Y_\iota, Y_\iota + y_\iota \cdot G, y_\iota)$. The nodes $(U_1, \ldots, U_{n-1})$ are given the corresponding output (except for U_i) and U_n is given

$(Y_{n-1}, \sum_{j=i}^{n-1} y_j)$. If the node U_i is requested to release the lock, the reduction aborts. At some point of the execution the adversary $\mathcal{A}$ outputs some $k^* = (r^*, s^*)$. The reduction parses $s^R = (s', m, \mathsf{pk})$ as the updated state of U_i then checks the following:

1) $\left(\frac{s}{s^*} + y_{i-1}\right) \cdot G = Y^*$
2) $-\left(\frac{s'}{s^*} + y_{i-1}\right) \cdot G = Y^*$

and returns the LHS term of the equation that satisfies the relation.

The reduction is clearly efficient and, whenever j and i are guessed correctly, the reduction does not abort. Since the $\mathbb{G}$ is abelian and the U_i is honest, the distribution induced by the modified setup algorithm is identical to the original to the eyes of the adversary. Recall that cheat happens only in the case where k^* is a valid opening for ℓ_i and the release algorithm is successful on input k^* (if the last condition is not satisfied both $\mathcal{H}_3$ and $\mathcal{H}_4$ abort). Substituting, we have that s' is of the form $\frac{x_0 \cdot x_1 \cdot r_x + H(m)}{r_0 \cdot r_1} = \tilde{s} \cdot y$, where $R' = r_0 \cdot r_1 \cdot Y_{i-1} = (r_x, r_y)$, for some $y \in \mathbb{Z}_q$. Since the release is successful, then it must be the case that $(r_x, \tilde{s})$ is a valid ECDSA signature on the message m_{i-1} (agreed by the two parties in the locking algorithm for ℓ_{i-1}). This implies that $y \cdot G = Y_{i-1}$. As argued in the proof of lemma 12, if $s^* \neq \tilde{s}$ and $s^* \neq \tilde{s}$, then we have an attacker against the strong unforgeability of the signature scheme. It follows that $s^* = \tilde{s}$ or $s^* = -\tilde{s}$ with all but negligible probability. Substituting we have

$$\begin{aligned}\left(\frac{s'}{s^*} + y_{i-1}\right) \cdot G &= \left(\frac{\tilde{s} \cdot y}{s^*} + y_{i-1}\right) \cdot G\\ &= \frac{\tilde{s} \cdot y}{s^*} \cdot G + y_{i-1} \cdot G\\ &= y \cdot G + y_{i-1} \cdot G\\ &= Y_{i-1} + y_{i-1} \cdot G\\ &= (Y^* - y_{i-1} \cdot G) + y_{i-1} \cdot G\\ &= Y^*\end{aligned}$$

which implies that condition (1) holds if $s^* = \tilde{s}$. For the other case

$$\begin{aligned}-\left(\frac{s'}{s^*} + y_{i-1}\right) \cdot G &= -\left(\frac{\tilde{s} \cdot y}{s^*} + y_{i-1}\right) \cdot G\\ &= -\frac{\tilde{s} \cdot y}{s^*} \cdot G + y_{i-1} \cdot G\\ &= y \cdot G + y_{i-1} \cdot G\\ &= Y_{i-1} + y_{i-1} \cdot G\\ &= (Y^* - y_{i-1} \cdot G) + y_{i-1} \cdot G\\ &= Y^*\end{aligned}$$

which means that condition (2) is satisfied if $s^* = -\tilde{s}$. Since, by assumption, this happens with probability at least $\frac{1}{q \cdot n \cdot \mathsf{poly}(\lambda)}$ we have a successful attacker against the discrete logarithm problem. This proves our statement. ■

This concludes our proof. ■

F. PCNs from Multi-Hop Locks

In this section we show that AMHLs are sufficient to construct a full-fledged PCN that satisfy the standard security definition from Malavolta et al. [42].

Ideal Functionalities. We assume an ideal realization of AMHLs in the form of an ideal functionality $\mathcal{F}_{\mathbb{L}}$ as described in Fig. 3. That is, all parties have oracle access to $\mathcal{F}_{\mathbb{L}}$ through the specified interfaces.

Furthermore (same as it was done in [42]), we assume the existence of a blockchain B that we model as a trusted append-only bulletin board: The corresponding ideal functionality $\mathcal{F}_{\mathsf{B}}$ maintains B locally and updates it according to the transactions between users. At any point in the execution, anyone can send a distinguished message read to $\mathcal{F}_{\mathsf{B}}$, who sends the whole transcript of B to U. We denote the number of entries of B by $|\mathsf{B}|$. We assume that users can specify arbitrary *contracts*, i.e., transactions in B may be associated with arbitrary conditions which require to be me in order to make the transaction effective. $\mathcal{F}_{\mathsf{B}}$ is entrusted to enforce that a contract is fulfilled before the corresponding transaction is executed.

We model time as the number of entries of the blockchain B, i.e., time t is whenever $|\mathsf{B}| = t$. Note that we can artificially elapse time by adding dummy entries to B and that the current time is available to all parties by simply reading B and counting the number of entries. As discussed before, we assume synchronous communication between users, which is modelled by the functionality $\mathcal{F}_{\mathsf{syn}}$, and secure message transmission channels between users (modelled by $\mathcal{F}_{\mathsf{smt}}$).

Multi-session Extension. A subtlety in the application of the composition theorem is that each call of each ideal functionality assumes to spawn an independent instance. However, the $\mathcal{F}_{\mathbb{L}}$ functionality (described in Fig. 3) formally requires a joint state between sessions: The KGen protocols that are used for establishing pairwise links (or channels, respectively) are shared between multiple locking instances which might potentially result in shared keys between the different instances of PrivMuLs that realize payment channels. Consequently, a study of the concrete realization of those KGen protocols is required when arguing about the composition of several locking instances. Composition with joint states is discussed in [19], where the authors state a stronger version of the composition theorem (the so called JUC theorem) which accounts for joint state and randomness across protocol sessions.

In order to satisfy the conditions for the JUC theorem to apply, we must argue that our protocol realizes a stronger ideal functionality $\tilde{\mathcal{F}}_{\mathbb{L}}$ that makes only independent calls to the underlying interfaces (refer to [19] for a detailed description). More precisely, this means that we need to argue for each of the previously presented concrete realizations of $\mathcal{F}_{\mathbb{L}}$ that a parallel composition of those protocols – with all instances of the protocol sharing the same KGen protocols and running independently otherwise – realizes the functionality $\tilde{\mathcal{F}}_{\mathbb{L}}$. This is shown in the following lemmas.

Lemma 16. *Let g be a homomorphic one-way function, and let $\widehat{\mathbb{L}_{generic}}^{\mathsf{KGen}}$ be the multi-session extension of the protocol described in Fig. 4 using a shared* KGen *algorithm. Then $\widehat{\mathbb{L}_{generic}}^{\mathsf{KGen}}$ UC-realizes the ideal functionality $\tilde{\mathcal{F}}_{\mathbb{L}}$ in the $(\mathcal{F}_{\mathsf{syn}}, \mathcal{F}_{\mathsf{smt}}, \mathcal{F}_{\mathsf{anon}})$-hybrid model.*

Proof: The proof trivially follows from Theorem 2 and the Composition Theorem [18] since the KGen protocol never needs to be invoked for realizing $\mathcal{F}_{\mathbb{L}}$ and hence the different copies of $\mathbb{L}_{generic}$ in $\widehat{\mathbb{L}_{generic}}^{\mathsf{KGen}}$ are fully independent. So the joint state is in fact empty. ■

Lemma 17. *Let* COM *be a secure commitment scheme, let* NIZK *be a non-interactive zero knowledge proof, and let $\widehat{\mathbb{L}_{schnorr}}^{\mathsf{KGen}}$ be the multi-session extension of the protocol described in Fig. 7 using a shared* KGen *algorithm realizing $\mathcal{F}_{\mathsf{kgen}}^{\mathsf{schnorr}}$. If Schnorr signatures are strongly existentially unforgeable, then $\widehat{\mathbb{L}_{schnorr}}^{\mathsf{KGen}}$ UC-realizes the ideal functionality $\tilde{\mathcal{F}}_{\mathbb{L}}$ in the $(\mathcal{F}_{\mathsf{kgen}}^{\mathsf{schnorr}}, \mathcal{F}_{\mathsf{syn}}, \mathcal{F}_{\mathsf{smt}}, \mathcal{F}_{\mathsf{anon}})$-hybrid model.*

Proof: It is easy to see that the $\mathcal{F}_{\mathsf{kgen}}^{\mathsf{schnorr}}$ functionality itself is stateless and therefore consecutive invocations of $\mathcal{F}_{\mathsf{kgen}}^{\mathsf{schnorr}}$ are indistinguishable from the invocation of fresh instances of the functionality. Hence, for multiple protocols, it is identical to query the same $\mathcal{F}_{\mathsf{kgen}}^{\mathsf{schnorr}}$ instance or to work on independent copies (and the same property carries over to protocols realizing this functionality). As a consequence $\widehat{\mathbb{L}_{schnorr}}^{\mathsf{KGen}}$ is indistinguishable from the multi-session extension of $\mathbb{L}_{schnorr}$ using independent KGen copies that realize $\mathcal{F}_{\mathsf{kgen}}^{\mathsf{schnorr}}$. So the claim trivially follows from Theorem 6 and the Composition Theorem [18]. ■

Lemma 18. *Let* COM *be a secure commitment scheme and let* NIZK *be a non-interactive zero knowledge proof, and let $\widehat{\mathbb{L}_{ecdsa}}^{\mathsf{KGen}}$ be the multi-session extension of the protocol described in Fig. 5 using a shared* KGen *algorithm realizing $ideal_{\mathsf{kgen}}^{\mathsf{ECDSA}}$. If ECDSA signatures are strongly existentially unforgeable and Paillier encryption is* ecCPA *secure, and* KGen *then the construction in Fig. 5 UC-realizes the ideal functionality $\tilde{\mathcal{F}}_{\mathbb{L}}$ in the $(\mathcal{F}_{\mathsf{kgen}}^{\mathsf{ECDSA}}, \mathcal{F}_{\mathsf{syn}}, \mathcal{F}_{\mathsf{smt}}, \mathcal{F}_{\mathsf{anon}})$-hybrid model.*

Proof: As $\mathcal{F}_{\mathsf{kgen}}^{\mathsf{ECDSA}}$ satisfies the same independence property as $\mathcal{F}_{\mathsf{kgen}}^{\mathsf{schnorr}}$, the same argument as for lemma 17 applies. ■

System Assumptions. We assume that every user in the PCN is aware of the complete network topology, that is, the set of all users and the existence of a payment channel between every pair of users. We further assume that the sender of a payment chooses a payment path to the receiver according to her own criteria.

The current value on each payment channel is not published but instead kept locally by the users sharing a payment channel. The two users U_0 and U_1 are assumed to maintain locally the capacity of their channel, denoted by $\mathsf{cap}(U_0, U_1)$. We further assume that every user is aware of the payment fees charged by each other user in the PCN. For ease of exposition we define the predicate $\mathsf{fee}(U_i)$ to return the fee charged by the user U_i. We assume that pairs of users sharing a payment channel communicate through secure and authenticated channels (such as TLS), which is easy to implement given that every user is uniquely identified by a public key.

Our System. In the following we describe the three operations (open channel, close channel, and payment) that constitute the core of our system. For the sake of simplicity we restrict each pair of user to at most one channel, however our construction can be easily extended to support multiple channels per pair.

OPEN CHANNEL. The open channel protocol generates a new payment channel between users U_1 and U_2. The user U_1 invokes $\mathcal{F}_{\mathbb{L}}$ on input (U_2, L), depending on the direction of the channel, which returns the users identifiers (U_1, U_2) if the operation was successful. Then the users create an initial blockchain deposit that includes the following information: Their addresses, the initial capacity of the channel, the channel timeout, and the fee charged to use the channel agreed beforehand between both users. After the deposit has been successfully added to the blockchain, the operation returns 1. If any of the previous steps is not carried out as defined, the operation returns 0.

CLOSE CHANNEL. The close channel protocol is run by two users U_1 and U_2 sharing an open payment channel to close it at the state defined by v and accordingly update their bitcoin balances in the Bitcoin blockchain. From this point on, U_1 and U_2 ignore all the requests from $\mathcal{F}_{\mathbb{L}}$ relative to their link.

PAYMENT. A payment operation transfers a value v from a sender (U_0) to a receiver (U_{n+1}) through a path of open payment channels between them $(U_0, \ldots, U_{n+1})$. The sender (prot. 1) first computes the cost of sending v coins to the receiver as $v_1 := v + \sum_{i=1}^{n} \mathsf{fee}(U_i)$, and the corresponding cost at each of the intermediate hops in the payment path. Then it setups up a AMHL by calling the ideal functionality $\mathcal{F}_{\mathbb{L}}$ on the set of identifiers of the intermediate users. Finally, it sends each user the corresponding value to be transferred and a timeout information t_i.

Each intermediate user (prot. 3) checks whether the capacity of the channel is high enough to support the transfer of the coins and whether the timeouts give by the sender are consistent, i.e., $t_{i+1} = t_i - \Delta$ for some fixed Δ. Starting from (U_0, U_1), each pair of users query the ideal functionality $\mathcal{F}_{\mathbb{L}}$ on the **Lock** interface using the *lid* received in the previous phase. If the ideal functionality signals to proceed, then the two users establish a contract specified in the following.

contract(Alice, Bob, *lid*, x, t)

1) If GetStatus(*lid*) = Rel before t days, then Alice pays Bob x coins.
2) If t elapse, then Alice gets back x coins.

The contract is authenticated by both users and can be uploaded to B by either of them at any time. If every user in the path locks the corresponding *lid*, eventually the receiver (prot. 2) is reached. U_{n+1} checks whether the transacted value is what it expects, and whether the latest timeout t_{n+1} is well-formed. If both conditions hold, the receiver releases the lock lid_n by querying the ideal functionality. This triggers a cascade of release calls in the path from the sender to the receiver, thereby enabling the left user in the link to pull the payment (using the previously established contract). If for some reason one of the intermediate links is not released, then all of the previous contracts are voided after the corresponding timeout.

Algorithm 1: Payment routine for the sender

```
Input : $(U_0, \dots, U_{n+1}, v)$
$v_1 := v + \sum_{i=1}^{n} \mathsf{fee}(U_i)$
if $v_1 \leq \mathsf{cap}(U_0, U_1)$ then
    query $\mathcal{F}_{\mathbb{L}}$ on Setup$(U_0, \dots, U_{n+1})$
    $\mathcal{F}_{\mathbb{L}}$ returns $(\bot, lid_0, \bot, U_1, \mathsf{Init})$
    $\mathsf{cap}(U_0, U_1) := \mathsf{cap}(U_0, U_1) - v_1$
    $t_0 := t_{\text{now}} + \Delta \cdot n$
    forall $i \in \{1, \dots, n\}$
        $v_i := v_1 - \sum_{j=1}^{i-1} \mathsf{fee}(U_j)$
        $t_i := t_{i-1} - \Delta$
        send $((U_{i-1}, U_{i+1}, v_{i+1}, t_i, t_{i+1}), \mathsf{fwd})$ to
    $U_i$
    end for
    send $(U_n, v_{n+1}, t_{n+1})$ to $U_{n+1}$
    query $\mathcal{F}_{\mathbb{L}}$ on Lock$(lid_0)$
    if $\mathcal{F}_{\mathbb{L}}$ returns $(lid_0, \mathsf{Lock})$
        $\mathsf{contract}(U_0, U_1, lid_0, v_1, t_1)$
    else
        abort
    end if else
    abort
end if
```

Algorithm 2: Payment routine for the receiver

```
Input : $(U_n, v_{n+1}, t_{n+1}, v)$
$\mathcal{F}_{\mathbb{L}}$ returns $(lid_n, \bot, U_n, \bot, \mathsf{Init})$
if $(t_{n+1} > t_{\text{now}} + \Delta) \wedge (v_{n+1} =$
  $v) \wedge$ (GetStatus($lid_n$) $= \mathsf{Lock}$) then
    query $\mathcal{F}_{\mathbb{L}}$ on Release$(lid_n)$
    send ok to $U_n$
else
    send $\bot$ to $U_n$
end if
```

Analysis. In the following we argue that the system as described above ideally realizes the functionality $\mathcal{F}_{PCN}$ as defined in [42], assuming oracle access to $\mathcal{F}_{\mathbb{L}}$, $\mathcal{F}_{\mathsf{B}}$, and $\mathcal{F}_{\mathsf{syn}}$.

Theorem 7. *The system described above UC-realizes $\mathcal{F}_{PCN}$ (as defined in [42]) in the $(\mathcal{F}_{\mathbb{L}}, \mathcal{F}_{\mathsf{B}}, \mathcal{F}_{\mathsf{syn}}, \mathcal{F}_{\mathsf{smt}})$-hybrid model.*

Proof: The proof consists of the observation that the ideal functionality $\mathcal{F}_{\mathbb{L}}$ enforces balance security and satisfies relationship anonymity (as defined in [42]). A subtlety is that now all users have access to a **GetStatus** interface and they might be able to query the functionality on a certain *lid* and learn its status even when they are not involved in the generation of such a lock. However one can easily show that this happen only with negligible probability since it require guessing *lid*, which is a string sampled uniformly at random. It is also easy to see that $\mathcal{F}_{\mathbb{L}}$ does not allow one to perform wormhole attacks, by construction. What is left to be shown is that the rest of the information exchanged by the machines does not break any of these properties. Note that the only information that is sent outside $\mathcal{F}_{\mathbb{L}}$ consists of user identifiers, timeouts, and values to lock. The first are already known by the intermediate users, whereas the rest of the items are chosen exactly as described in $\mathcal{F}_{PCN}$. Note that it is sufficient here to argue about the individual copies of $\mathcal{F}_{\mathbb{L}}$ in isolation by the JUC theorem [19]. As we showed, the multi-session extended ideal functionality $\tilde{\mathcal{F}}_{\mathbb{L}}$ is realized by our instantiations and therefore the JUC theorem allows us to complete the analysis assuming independent copies of $\mathcal{F}_{\mathbb{L}}$ running in parallel. ■

Algorithm 3: Payment routine for the i-th intermediate user

```
Input : $(m, decision)$
if $decision = \mathsf{fwd}$ then
    parse $m$ as $(U_{i-1}, U_{i+1}, v_{i+1}, t_i, t_{i+1})$
    $\mathcal{F}_{\mathbb{L}}$ returns $(lid_{i-1}, lid_i, U_{i-1}, U_{i+1}, \mathsf{Init})$
    if $(v_{i+1} \leq \mathsf{cap}(U_i, U_{i+1})) \wedge (t_{i+1} = t_i - \Delta) \wedge (\textbf{GetStatus(}lid_{i-1}\textbf{)} = \mathsf{Lock})$ then
        $\mathsf{cap}(U_i, U_{i+1}) := \mathsf{cap}(U_i, U_{i+1}) - v_{i+1}$
        query $\mathcal{F}_{\mathbb{L}}$ on Lock($lid_i$)
        if $\mathcal{F}_{\mathbb{L}}$ returns $(lid_i, \mathsf{Lock})$
            $\mathsf{contract}(U_i, U_{i+1}, lid_i, v_{i+1}, t_{i+1})$
        else
            send $\perp$ to $U_{i-1}$
        end if
    else
        send $\perp$ to $U_{i-1}$
else if $decision = \perp$ then
    $\mathsf{cap}(U_i, U_{i+1}) := \mathsf{cap}(U_i, U_{i+1}) + v_{i+1}$
    send $\perp$ to $U_{i-1}$
else if $(decision = \mathsf{ok}) \wedge \textbf{GetStatus(}lid_i\textbf{)} = \mathsf{Rel}$ then
    query $\mathcal{F}_{\mathbb{L}}$ on Release($lid_{i-1}$)
    send $\mathsf{ok}$ to $U_{i-1}$
else
    send $\perp$ to $U_{i-1}$
end if
```