A Composable Security Treatment of the Lightning Network

Aggelos Kiayias
University of Edinburgh and IOHK
Email: akiayias@inf.ed.ac.uk

Orfeas Stefanos Thyfronitis Litos
University of Edinburgh
Email: o.thyfronitis@ed.ac.uk

***Abstract*—The high latency and low throughput of blockchain protocols constitute one of the fundamental barriers for their wider adoption. Overlay protocols, notably the *lightning network*, have been touted as the most viable direction for rectifying this in practice. In this work we present for the first time a full formalisation and security analysis of the lightning network in the (global) universal composition setting that leverages a global ledger functionality, for which realisability by the Bitcoin blockchain protocol has been demonstrated in previous work [Badertscher et al., Crypto'17]. As a result, our treatment delineates exactly how the security guarantees of the protocol depend on the properties of the underlying ledger and the frequent availability of the protocol participants. Moreover, we provide a complete and modular description of the core of the lightning protocol that highlights precisely its dependency to underlying basic cryptographic primitives such as igital signatures, pseudorandom functions, identity-based signatures and a less common two-party primitive, which we term a combined digital signature, that were originally hidden within the lightning protocol's implementation.**

1. Introduction

Improving the latency of blockchain protocols, in the sense of the time it takes for a transaction to be "finalised", as well as their throughput, in the sense of the number of transactions they can handle per unit of time, are perhaps the two most crucial open questions in the area of permissionless distributed ledgers and remain fundamental barriers for their wider adoption in applications that require large scale and reasonably expedient transaction processing, cf. [1]. The Bitcoin blockchain protocol, introduced by Nakamoto [2], provides settlement with probability of error that drops exponentially in the number of blocks k that accumulate over a transaction of interest. This has been informally argued in the original white paper, and further formally demonstrated in [3], from where it can be inferred that the total delay in actual time for a transaction to settle is linear in k in the worst case. These results were subsequently generalised to the setting of partial synchrony [4] and variable difficulty [5]. Interestingly, this latency "deficiency" is intrinsic to the blockchain approach (see below), i.e., latency's dependency on k is not a side-effect of the security analysis but rather a characteristic of the underlying protocol and the threat model it operates in.

Given the above state of affairs, one has to either change the underlying settlement protocol or devise some other mechanism that, in conjunction with the blockchain protocol, achieves high throughput and low latency. A number of works proceeded with the first direction, e.g., hybrid consensus [6], Algorand [7]. A downside of this approach is that the resulting protocols fundamentally change the threat model within which Bitcoin is supposed to operate, e.g., by reducing the threshold of corrupted players, strengthening the underlying cryptographic assumptions or complicating the setup assumption required (e.g., from a public to a private setup).

The alternative approach is to build an *overlay* protocol that utilises the blockchain protocol as a "fall back" layer and does not relax the threat model in any way while it facilitates fast "off-chain" settlement under certain additional assumptions. We note that in light of the impossibility result regarding protocol "responsiveness" from [6] that states that no protocol can provide settlement in time proportional to actual network delay (i.e., fast settlement) and provide a security threshold over 1/3, we know that maintaining Bitcoin's threat model will require some additional assumption for the overlay protocol to offer fast settlement.

The first instance of this approach and by far the most widely known and utilised to date, came with the *lightning network* [8][1] that functions over the Bitcoin blockchain and leverages the concept of a bilateral payment channel. The latency for a transaction becomes linear to actual network delay and another factor that equals the number of bilateral payment channel hops in the path that connects the two end-points of the transaction. Implicated parties are guaranteed that, if they wish so, *eventually* the ledger will record a "gross"

1. The specification available online is a more descriptive reference for the inner workings of the protocol, see https://github.com/lightningnetwork/lightning-rfc/blob/master/02-peer-protocol.md. See also the raiden network that implements Lightning over Ethereum, https://raiden.network.

settlement transaction that reflects the balance resulting from all in-channel payments. Deviations from this guarantee are cryptographically feasible but deincentivised: a malicious party trying to commit to an outdated state will lose funds to a peer that provides evidence of a subsequent state. Moreover, note that no record of a specific payment transaction need ever appear on-chain thus the number of lightning transactions that can be exchanged can reach the maximum capacity the network allows between the parties, without being impeded by any restrictions of the underlying blockchain protocol.

The lightning network has been very influential in the space and spun a number of follow up research and implementations (see below for references). We note that the lightning network is not the only option for building an overlay over a blockchain. See e.g., [9] for an alternative approach focusing on reducing latency, where it is shown that if the assumption is raised to a security threshold of 3/4 plus the honesty of an additional special player, it is possible to obtain optimal latency. Nevertheless, this approach does not offer the throughput benefits that are intrinsic to the lightning network.

Despite the importance of the lightning network for blockchain scalability there is still no work so far providing a thorough formal security analysis. This is a dire state of affairs given the fact that the protocol is actually currently operational[2] and its complexity makes it difficult to extract precise statements regarding the level of security it offers.

Our Results. We present the first, to our knowledge, complete security analysis of the lightning network, which we carry out in the universal composition (UC) setting. We model the payment overlay that the lightning network provides as an ideal functionality and we demonstrate how it can be implemented in a hybrid world which assumes a global ledger functionality. Our treatment is general and does not assume any specific implementation for the underlying ledger functionality. The "paynet" functionality that we introduce abstracts all the salient security features achieved by the lightning network. We subsequently describe the whole lightning protocol in this setting and we prove that it realises our paynet functionality under standard cryptographic assumptions; the security guarantees of the functionality reflect specific environmental conditions regarding the availability of the honest parties to poll the status of the network. In more details our results are as follows.

1) We present the $\mathcal{F}_{\text{PayNet}}$ functionality which abstracts the syntax and security properties provided by the lightning network. We describe our $\mathcal{F}_{\text{PayNet}}$ assuming a global ledger functionality $\mathcal{G}_{\text{Ledger}}$ as defined in [10], and further refined in [11], which we know that is realised by the Bitcoin blockchain. Our approach not only captures lightning, but it is also general as it can be applied to any payment network by finely tuning the following parts of the functionality: the exact channel opening message sequence, the details of the on-chain checks performed by $\mathcal{F}_{\text{PayNet}}$, the negligence time bounds and the penalty in case of a malicious closure being caught. Using $\mathcal{F}_{\text{PayNet}}$, parties can open and close channels, forward payments along channel paths in the network as well as poll its status. Importantly, the functionality keeps track of all the off-chain and on-chain balances and ensures that when a channel closes, the on-chain balances are in line with the off-chain ones. In order to handle realistic adversarial deviations, $\mathcal{F}_{\text{PayNet}}$ allows the adversary to choose one of the following outcomes for each multi-hop payment: (i) let it go through as requested, (ii) charge it to an adversarial party along the path, (iii) charge it to a *negligent* honest party along the path. This last outcome is a crucial security characteristic of the lightning network: honest parties are required to poll the functionality with a frequency that corresponds to their level of involvement in the network and the properties of the underlying ledger. If a party does not poll often enough, $\mathcal{F}_{\text{PayNet}}$ identifies it as negligent and it may lose funds.
2) We identify for the first time the exact polling requirements so that honest parties do not lose funds, as a function of the parameters of $\mathcal{G}_{\text{Ledger}}$. The polling requirements for each party are two-fold: (i) the first type refers to monitoring for closures of channels of which the party is a member, and is specified by the parameter "delay" (chosen by the party), (ii) the second type refers to monitoring for specific events related to receiving and relaying payments. In detail, let *Alice* be an intermediary of a multi-hop payment. When the payment starts, she specifies two blockheights h, h'. Also, let a be the upper bound to the number of blocks that may be finalised in the ledger from the time a certain transaction is emitted to the time it becomes finalised (i.e. it is included in a block in the "stable" part of the ledger). *Alice* should then poll twice while her local view of the chain advances from blockheight h to blockheight $h' - a$. Moreover, the two pollings should be separated by a time window so that the chain can grow by at least a blocks.
3) We provide a complete pseudocode description of the lightning network protocol Π_{LN} and prove that it realises $\mathcal{F}_{\text{PayNet}}$ in the Random Oracle model. We identify a number of underlying cryptographic primitives that have been used in the lightning network in a non-black-box fashion and without reference. Interestingly, while most of these primitives are quite standard (a PRF, a Digital Signature scheme and an Identity Based Signature scheme), there is also one that is less standard and requires a new definition. The *combined digital signature* – as we will call it – is a special case of an asymmetric

2. For current deployment statistics see e.g., https://1ml.com/statistics.

two-party digital signature primitive (e.g., see [12] and references therein) with the following characteristic: One of the two parties, called the shareholder, generates and stores a share of the signing key. The public key of the combined signature can be determined non-interactively from public information produced by both parties. Issuing signatures requires the availability of the share, which is verifiable given the public information provided by the shareholder. We formalise the combined digital signature primitive and show that the construction lying within the lightning specification realises it under standard cryptographic assumptions. In summary, the realisation of $\mathcal{F}_{\text{PayNet}}$ is achieved assuming the security of the underlying primitives, which in turn can be based on EC-DLOG and the Random Oracle model.

4) We prove that a more idealized ledger functionality, i.e. a ledger with instant finality, is unrealisable even assuming a synchronous multicast network. This result supports our decision to use the more realistic ledger functionality of [10], since it establishes that if our analysis was based on such a perfect ledger, it would not be relevant for any real world deployment of a payment network since such software would – necessarily – depend on a non-perfect ledger. This choice also distinguishes our work compared to previous attempts [13], [14], [15], [16] to formalize payment networks, as well as highlights the considerable latency improvement that the protocol offers in comparison to directly using the ledger.

Related Work. A first suggestion for building a unidirectional payment channel appeared in [17]. Bidirectional payment channels were developed in [18] and of course as part of the lightning network [8]. Subsequent work on the topic dealt with either improving payment networks by utilising more expressive blockchains such as Ethereum [16], hardware assumptions, see e.g., [19], or extending its functionality beyond payments, to smart contracts, [15] or finally enhancing their privacy, see e.g., [14], [20], [21]. Additional work looked into developing supporting protocols for the payment networks such as rebalancing [22] or finding routes in a decentralised fashion [23], [24]. With respect to idealising the payment network functionality in the UC setting, a number of previous papers [13], [14], [15], [16] presented ideal functionalities abstracting the concept, but they did not prove that the lightning network realises them. The main advantage of our approach however here is that, for the first time, we present a payment network functionality that interoperates with a global ledger functionality for which we know, in light of the results of [10], that is realisable by the Bitcoin blockchain and hence also reflects the actual parameters that can be enforced by the implementation and the exact participation conditions needed for security. In contrast, previous work [13], [14], [16] utilized "too idealised" ledger functionalities for their analysis which offer instant finality; as we prove in Theorem 3, a representative variant of these functionalities (Fig. 8) is unrealisable even under strong network assumptions (cf. Section 8). It is worth noting here that, were such a ledger realizable, layer-2 payment networks would not be as useful in practice because one of their two main motivations is the high latency of real blockchains. On the other hand, in [15] the ledger is not explicitly specified as a functionality, but only informally described. Several smart contracts are formally defined instead as UC ITMs, which are the entities with which protocols ultimately interact. The execution model of these contracts and their interaction with the blockchain is explained in an intuitive way, but a complete formalization of the ledger is missing. Lastly, the ledger used in [13] cannot be used directly by protocol parties, only accessed via higher-level functionalities. This limitation is imposed because otherwise any party could arbitrarily change the balances of other parties, given the definition of the functionality. This ledger is therefore a useful abstraction for higher-level protocols, but not amenable to direct usage, let alone concrete realisation.

Organisation. In Section 2 we present preliminaries for the model we employ and the relevant cryptographic primitives. In Section 3 we present an overview of the lightning network, accompanied by figures of the relevant transactions. Our payment network functionality is given an overview description in Section 4. Our abstraction of the core lightning protocol is provided in Section 5. We give more details about the combined digital signature primitive in Section 6. In Section 7 we provide an overview of the security proof of the main simulation theorem. Finally, in Section 8 we formalise our claim that a ledger functionality with instant finality is unrealisable. We refer the reader to the appendix of the full version [25] for the formal definition of the paynet functionality $\mathcal{F}_{\text{PayNet}}$ and the protocol Π_{LN}, along with the proof that the latter UC-realises the former. Our claim on the unrealisability of a perfect ledger is also proven there.

2. Preliminaries

In this section we give a brief overview of the tools and frameworks used in this work.

Universal Composability framework. In simulation-based security, cryptographic tasks are defined via an ideal "functionality" $\mathcal{F}$, which can be thought of as an uncorruptible entity that gets the inputs of all parties and returns the expected outputs while also interacting with the adversary in a prescribed manner. In this way, the functionality expresses the essence of a cryptographic task and its security features. A protocol Π realises the functionality $\mathcal{F}$ if for any real world adversary we can define a "simulator" $\mathcal{S}$, acting as an ideal world adversary, such that any environment $\mathcal{E}$ cannot distinguish between the real world and the ideal world executions. Albeit a powerful

tool, simulation-based security only works when a single instance of the protocol is run in isolation. However, real-world systems almost always run several programs concurrently, which furthermore may run different instances of the same protocol. To facilitate this, the Universal Composability [26] framework allows us to analyse a single instance of the protocol and then take advantage of a generic composition theorem to infer the security of the protocol more broadly. This is achieved by allowing arbitrary interactions between the environment and the real-world adversary.

As mentioned, lightning network members have to periodically check the blockchain to ensure the security of their funds. However, the execution model of the UC framework allows $\mathcal{E}$ to impose extended periods of inactivity to any party. We opted to avoid the complication of using the clock functionality to force regular activation. Restricting the analysis only to environments that always cater for the needed activations would preclude the composability of our model. We instead allow $\mathcal{E}$ to deny activation to players (therefore becoming negligent) and provide security guarantees conditional on $\mathcal{E}$ permitting the necessary monitoring of the blockchain.

Hybrid functionalities used. Both our main protocol and the corresponding functionality use $\mathcal{G}_{\text{Ledger}}$ [10], [11] as a hybrid. $\mathcal{G}_{\text{Ledger}}$ formalizes an ideal distributed append-only data structure akin to a blockchain. Any participating party can read from $\mathcal{G}_{\text{Ledger}}$, which returns an ordered list of transactions. Furthermore parties can submit new transactions which, if valid, will be added to the ledger and made visible to all parties at the discretion of the adversary, but necessarily within a predefined time window. This property is called liveness. Once a transaction is added to the ledger, it becomes visible to all parties at the discretion of the adversary, but within another predefined time window, and cannot be removed or reordered. This is called persistence. The exact definition can be found in the full version [25]. The current work makes heavy use of these two security properties, as the security of the lightning network relies crucially on the security of the underlying ledger.

Furthermore, $\mathcal{G}_{\text{Ledger}}$ needs the $\mathcal{G}_{\text{CLOCK}}$ functionality, which models the notion of time. Every participating party can request to read the current time (which is initialized to 0) and inform $\mathcal{G}_{\text{CLOCK}}$ that her round is over. $\mathcal{G}_{\text{CLOCK}}$ increments the time by one once all parties have declared the end of their round.

As already mentioned, the protocol and functionality defined in the current work do not make direct use of $\mathcal{G}_{\text{CLOCK}}$. Indeed, the only notion of time both in the lightning protocol and in our work is provided by the height of the blockchain, as reported respectively by the underlying Bitcoin node and $\mathcal{G}_{\text{Ledger}}$. We therefore omit it in the statement of Theorem 2 for simplicity of notation; it should normally appear as hybrid along with $\mathcal{G}_{\text{Ledger}}$. Its exact definition can be found in the full version [25]. We also note that $\mathcal{G}_{\text{Ledger}}$ and $\mathcal{G}_{\text{CLOCK}}$ are global functionalities [27] and therefore can be accessed directly by the environment, whereas $\mathcal{F}_{\text{PayNet}}$ is not.

Transaction structure. $\mathcal{G}_{\text{Ledger}}$ does not define what is a valid transaction, but leaves it as a system parameter. Importantly, no notion of coins is built in $\mathcal{G}_{\text{Ledger}}$. We therefore specify a valid transaction, closely following concepts put forth in Bitcoin [2], but avoiding specifying the entire Bitcoin script.

At a high level, every transaction consists of inputs and outputs. Each output has an associated value in coins and a number of "spending methods". A spending method specifies the exact requirements for spending the output. Each input must be connected to exactly one output and satisfy one of its spending methods.

Transactions in $\mathcal{G}_{\text{Ledger}}$ form a DAG. A new transaction is valid only if each of its inputs correctly spends an output with no other connected input and the sum of the values of its outputs does not exceed the sum of the values of the outputs connected to its inputs. We refer the reader to the full version [25] for a complete overview.

Cryptographic Primitives. In the Lightning Network specification, a custom scheme for deriving keys is used. Its syntax and security aims closely match those of previously studied Identity Based Signature schemes [28], [29], thus we use the latter to abstract away the complexity of the construction and highlight the security requirements it satisfies. We slightly modify previous IBS schemes by adding an algorithm that, on input of the public parameters mpk and a label l, returns the verification key pk_l. Such an IBS scheme provides 5 algorithms:

- $(mpk, msk) \leftarrow \textsc{Setup}(1^k)$: master keypair generation
- $(pk_l, sk_l) \leftarrow \textsc{KeyDer}(mpk, msk, l)$: keypair derivation with label l
- $pk_l \leftarrow \textsc{PubKeyDer}(mpk, l)$: verification key derivation with label l
- $\sigma \leftarrow \textsc{SignIBS}(m, sk_l)$: signature generation with signing key sk_l
- $\{0,1\} \leftarrow \textsc{VerifyIBS}(\sigma, m, pk_l)$: signature verification

We refer the reader to [29] for more details. Other cryptographic primitives used are digital signatures and pseudorandom functions. Finally, a less common two-party cryptographic primitive is employed that we formalise as *combined digital signatures*, see Section 6.

3. Lightning Network overview

Two-party channels. The aim of LN is to enable fast, cheap, off-chain transactions, without compromising security. Specifically no trust between counterparties is needed. This is achieved as follows: Two parties that plan to have recurring monetary exchanges lock up some funds with one special on-chain transaction. We say that they opened a new channel. They can then transact with the locked funds multiple times solely by interacting

privately, without informing the blockchain. If they want to use their funds in the usual, on-chain way again, they have to close the channel and unlock the funds with one more on-chain transaction. Each party can unilaterally close the channel and retrieve the coins they are entitled to – according to the latest channel state – and thus neither party has to trust the other.

In more detail, to open a channel *Alice* and *Bob* first exchange a number of keys and desired timelocks $\text{relDel}_A, \text{relDel}_B$ (explained below) and then build locally some transactions. The "funding transaction" F contains a 2-of-2 multisig output with one "funding" public key $pk_{F,A}, pk_{F,B}$ for each counterparty. This multisig output needs signatures for both designated public keys in order to be spent. F is funded with c_F coins that belong only to one of the two parties, say *Alice*.

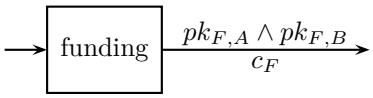

Figure 1. Funding TX (on-chain): Rules over, coins below output.

Each party also builds a slightly different version of the "commitment transaction" C_A, C_B. *Alice* uses her "delayed payment" key $pk_{\text{dcom},A}$ and *Bob*'s "revocation" key $pk_{\text{rev},B}$ (received before), whereas *Bob* uses *Alice*'s "payment" key $pk_{\text{com},A}$ (received before). Both C_A and C_B spend the funding output of F and allow *Alice* to retrieve her funds if she acts honestly, as we will explain shortly. *Alice* sends *Bob* the signature of C_B made with her $sk_{F,A}$ and vice-versa.

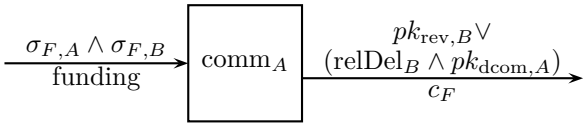

Figure 2. *Alice*'s initial commitment TX (off-chain): Required data over input, spent output below input.

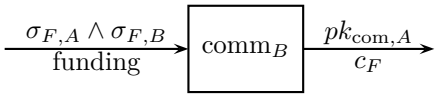

Figure 3. *Bob*'s initial commitment TX (off-chain): All coins belong to *Alice*, so she can immediately spend them if *Bob* closes.

She now broadcasts F; once both parties see that it is confirmed, they generate and exchange new "commitment" keys (used for updating the channel later) and the channel is open.

Every time they want to make a payment to each other, they exchange a series of messages that have two effects. First, a new pair of commitment transactions, along with their signatures by the funding keys, is created, one for each counterparty. Each of these transactions ensures that, if broadcast, each party will be able to spend the appropriate share from the coins contained in the funding output. Second, the two old commitment transactions are revoked. This ensures that no party can close a channel using an old commitment transaction which may be more beneficial to her than the latest one.

Invalidating past commitments requires some care. The reason is that it is impossible to actually make past commitments invalid without spending the funding output on-chain; doing this for every update would however defeat the purpose of LN. The following idea is leveraged instead: If *Alice* broadcasts an old commitment and *Bob* sees it, he can punish *Alice* by taking all the money in the channel. Therefore *Alice* is technically able to broadcast an old commitment, but has no financial benefit in doing so. The same reasoning holds if *Bob* broadcasts an old commitment. On the donwside this imposes the requirement that parties must observe the blockchain periodically — see below the explanation of timelocks and how they facilitate a time window within which parties should react.

The punishing mechanism operates as follows. Suppose *Alice* considers posting her old local commitment which has an output that carries her old share of the funds. This output can be spent in two ways: either with a signature by *Alice*'s "delayed payment" secret key $sk_{\text{dcom},A}$ which is a usual ECDSA key, or with a signature by *Bob*'s "revocation" secret key $sk_{\text{rev},B}$, which is also an ECDSA key, but with an additional characteristic that we will explain soon. Thus, if *Alice* broadcasts an old commitment, *Bob* will be able to obtain her funds by spending her output using his "revocation" key. This privilege of course opens the possibility for *Bob* to abuse it (in particular, when a channel is closed — see below — *Bob* may steal *Alice*'s funds by using such a revocation key) and hence this side effect should also be carefully mitigated. The mitigation has the following form. At the time of creation of a new commitment, both parties will know *Bob*'s "revocation" public key $pk_{\text{rev},B}$, but no party knows its corresponding secret – the key can only be computed by combining one secret value $sk_{\text{com},n,A}$ that *Alice* knows and one secret value $sk_{\text{com},n,B}$ that *Bob* knows. *Alice* therefore can prevent *Bob* from using his revocation key until she sends him $sk_{\text{com},n,A}$. Therefore, *Alice* will send *Bob* $sk_{\text{com},n,A}$ only after the new commitment transaction is built and signed. Thus *Bob* cannot abuse his revocation key on a commitment before this transaction is revoked. We note that the underlying cryptographic mechanism that enables such "revocation keys" is not straightforward and, as part of our contributions, we formalise it as a new two-party cryptographic primitive. We call it "combined signature" and we prove that the construction hidden in the LN implementation realizes it in the random oracle model under the assumption that the underlying digital signature scheme is secure in the full version [25].

The last element needed to make channel updates secure is the already mentioned "relative timelock". If

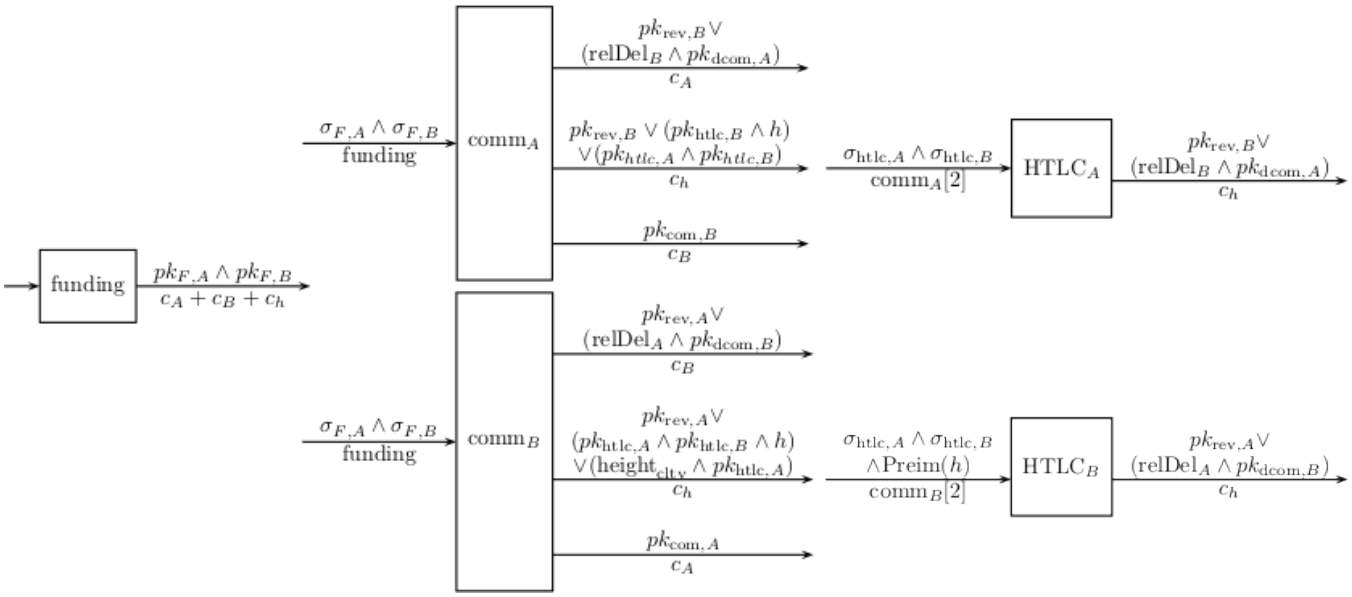

Figure 4. All transactions of an open channel with an HTLC in flight. *Alice* owns c_A coins, *Bob* c_B coins, and c_h coins will be transferred to *Bob* if he discloses the preimage of h until the ledger has $\text{height}_{\text{cltv}}$ blocks, otherwise they will return to *Alice*. "funding" is in the ledger, *Alice* keeps locally "comm_A" and "HTLC_A", and *Bob* keeps "comm_B" and "HTLC_B".

Alice broadcasts a commitment transaction, she is not allowed to immediately spend her funds with her "delayed payment" key. Instead, she has to wait for the transaction to reach a pre-agreed block depth (the relative timelock, negotiated during the opening of the channel and hardcoded in the output script of the commitment transaction) in order to give some time to *Bob* to see the transaction and, if it does not correspond to the latest version of the channel, punish her with his "revocation" key. This avoids a scenario in which *Alice* broadcasts an old commitment transaction and immediately spends her output, which would prevent *Bob* from ever proving that this commitment was old.

Lastly, if *Alice* wants to unilaterally close a channel, all she has to do is broadcast her latest local commitment (the only one not revoked) and any outstanding HTLC transactions (explained below) and wait for the timelock to expire in order to spend her funds. The LN specification further allows for cooperative channel closure, achieved by negotiating and broadcasting the "closing transaction" which is not encumbered with a timelock, providing immediate availability of funds.

As mentioned timelocks provide specific time windows within which both parties have to check the blockchain in order to punish a misbehaving counterparty who broadcasts an old commitment transaction. This means that parties have to be regularly online to safeguard against theft. Furthermore, LN makes it possible to trustlessly outsource this to so-called watchtowers, but this mechanism is not analyzed in the current work.

Multi-hop payments. Having funds locked down for exclusive use with a particular counterparty would be a serious limitation. LN goes beyond that by allowing multi-hop payments. In a situation where *Alice* has a channel with *Bob* and he has another channel with *Charlie*, it is possible for *Alice* to pay *Charlie* off-chain by leveraging *Bob*'s help. Remarkably, this can be achieved without any one party trusting any of the other two. One can think of *Alice* giving some "marked" money to *Bob*, who in turn either delivers it to *Charlie* or returns it to *Alice* – it is impossible for *Bob* to keep the money. It is also impossible for *Alice* and *Charlie* to make *Bob* pay for this transaction out of his pocket.

We will now give a brief overview of how this counterintuitive dynamic is made possible. *Alice* initiates the payment by asking *Charlie* to create a new hash for a payment of x coins. *Charlie* chooses a random secret, hashes it and sends the hash to *Alice*. *Alice* promises *Bob* to pay him x in their channel if he shows her the preimage of this particular hash within a specific time frame. *Bob* makes the same promise to *Charlie*: if *Charlie* tells *Bob* the preimage of the same hash within a specific time frame (shorter than the one *Bob* has agreed with *Alice*), *Bob* will pay him x in their common channel. *Charlie* then sends him the preimage (which is the secret he generated initially) and *Bob* agrees to update the channel to a new version where x is moved from him to *Charlie*. Similarly, *Bob* sends the preimage to *Alice* and once again *Alice* updates their channel to give *Bob* x coins. Therefore x coins were transmitted from *Alice* to *Charlie* and *Bob* did not gain or lose anything, he just increased his balance in the channel with *Alice* and reduced his balance by an equal amount in the channel with *Charlie*.

This type of promise where a preimage is exchanged for money is called Hashed Time Locked Contract (HTLC). It is enforceable on-chain in case the payer does not cooperatively update upon disclosure of the preimage, thus no trust is needed. It is realised as an additional output of the commitment transactions, which contains the specified hash and transfers its funds either to the party that should provide the preimage or to the other party after a timeout. A corresponding "HTLC transaction" that can spend this output is built by each party. In the previous example with *Alice*, *Bob* and *Charlie*, two HTLC transactions were signed and fulfilled in total for the payment to go through. Two updates happened in each channel: one to sign the HTLC and one to fulfill it. The time frames were chosen so that every intermediary has had the time to learn the preimage and give it to the previous party on the path. Figure 4 shows all transactions implicated in a channel that has an HTLC in flight.

In LN zero-hop payments are also carried out using HTLCs.

LN gives the possibility for intermediaries to charge a fee for their service, but such fees are not incorporated in the current analysis for the benefit of avoiding the added complexity and making it easier for the functionality to keep track of the correct balances. We note in passing that the "wormhole" attack described in [30] is captured by our model, as an adversary that controls two non-neighbouring nodes on a payment path can skip the intermediate nodes. Nevertheless, such an attack is inconsequential in our analysis given the lack of fees. Furthermore, LN leverages the Sphinx onion packet scheme [31] to increase the privacy of payments, but we do not formaly analyze the privacy of LN in this work – we just use it in our protocol description to syntactically match the message format used by LN.

4. Overview of $\mathcal{F}_{\text{PayNet}}$

One of our contributions is the specification of $\mathcal{F}_{\text{PayNet}}$, a functionality that describes the functional and security guarantees given by an ideal payment network. Its definition can be found in the full version [25]. The central aim of $\mathcal{F}_{\text{PayNet}}$ is opening payment channels, keeping track of their state, updating them according to requested payments and closing them, as requested by honest players, all in a secure manner. In particular, the three main messages it can receive from *Alice* are (openChannel), (pay), (closeChannel) and (forceCloseChannel).

When $\mathcal{F}_{\text{PayNet}}$ receives (openChannel, *Alice*, *Bob*, x, *tid*) from *Alice*, it informs simulator $\mathcal{S}$ of the intention of environment $\mathcal{E}$ to create a channel between *Alice* and *Bob* where *Alice* owns x coins. When it receives (pay, *Bob*, $x, \overrightarrow{\texttt{path}}$, `receipt`) from *Alice*, it informs $\mathcal{S}$ that $\mathcal{E}$ asked to perform a multi-hop payment of x coins from *Alice* to *Bob* along the $\overrightarrow{\texttt{path}}$. In the same vein, when $\mathcal{F}_{\text{PayNet}}$ receives (closeChannel, `receipt`, *pchid*) or (forceCloseChannel, `receipt`, *pchid*) from *Alice* (for a cooperative or unilateral close respectively), it leaks to $\mathcal{S}$ the fact that $\mathcal{E}$ wants to close the relevant channel.

In order to provide security guarantees, there are various moments when $\mathcal{F}_{\text{PayNet}}$ verifies whether certain expected events have actually taken place. A number of messages prompt $\mathcal{F}_{\text{PayNet}}$ to read from $\mathcal{G}_{\text{Ledger}}$ and perform these checks. In the actual implementations of LN these checks are done periodically by a polling daemon. Such checks are done by $\mathcal{F}_{\text{PayNet}}$ in the following cases:

- On receiving (poll) by *Alice*, $\mathcal{F}_{\text{PayNet}}$ asks $\mathcal{G}_{\text{Ledger}}$ for *Alice*'s latest state Σ_{Alice} and verifies that no bad events have happened. In particular, $\mathcal{F}_{\text{PayNet}}$ halts if any of *Alice*'s channels has been closed maliciously with a transaction at height h and, even though *Alice* has polled within $[h, h + \texttt{delay}(Alice) - 1]$, she did not manage to punish the counterparty. If $\mathcal{F}_{\text{PayNet}}$ does not halt, it leaks to $\mathcal{S}$ the polling details (including the identity of the poller and the state of the ledger in their view).
- $\mathcal{F}_{\text{PayNet}}$ expects $\mathcal{S}$ to send a (resolvePays, `charged`) message that gives details on the outcome of one or more multi-hop payments that include the identity of the party that is charged. Moreover, for each resolved payment, the message includes two expiry values, expressed in absolute block height: `OutgoingCltvExpiry`, which is the highest block in which the charged party could claim money from the previous hop (closer to the payer) and `IncomingCltvExpiry`, which is the lowest block in which the charged party could claim money from the next hop (closer to the payee). $\mathcal{F}_{\text{PayNet}}$ checks that for each payment the charged party was one of the following: (a) the one that initiated the payment, (b) a malicious party or (c) an honest party that is negligent. The latter case happens when the honest party either:
  1) did not poll in time to catch a malicious closure (similarly to the check performed when a poll message is handled, as described above) or
  2) did not poll twice while the block height in the view of the player was in $[\texttt{OutgoingCltvExpiry}, \texttt{IncomingCltvExpiry} - (2 + r)\,\texttt{windowSize}]$ with a distance of at least $(2 + r)\,\texttt{windowSize}$ between the two polls or
  3) did not enforce the retrieval of the funds lost as a result of this payment when the chain in her view had height $\texttt{IncomingCltvExpiry} - (2 + r)\,\texttt{windowSize}$ with a fulfillOnChain message, as discussed below.

  Note that $(2 + r)\,\texttt{windowSize}$ is the maximum number of blocks an honest party needs to wait from the moment a valid transaction is submitted until it is added to the ledger state. $\mathcal{F}_{\text{PayNet}}$ also ensures that the two expiries (`OutgoingCltvExpiry` and `IncomingCltvExpiry`) have a distance of at

least `relayDelay`($Alice$) + (2 + r) `windowSize`, otherwise it halts. In case the charged party was honest and non-negligent, $\mathcal{F}_{\text{PayNet}}$ halts. It also halts if a particular payment resulted in a channel update for which $\mathcal{S}$ did not inform $\mathcal{F}_{\text{PayNet}}$.
- $\mathcal{F}_{\text{PayNet}}$ executes the function `checkClosed`(Σ_{Alice}) every time it receives Σ_{Alice} from $\mathcal{G}_{\text{Ledger}}$. In this case, it checks that every channel that $\mathcal{E}$ has asked to be closed or $\mathcal{S}$ designated as closed indeed has a closing transaction that corresponds to its latest state in Σ_{Alice}. Enough time is given for that transaction to settle in Σ_{Alice}, but if that time passes and the channel is still open or it is closed to a wrong version and no opportunity for punishment was given, $\mathcal{F}_{\text{PayNet}}$ halts.

A number of messages that support the protocol progress are also handled:

- Every player has to send (REGISTER, delay, relayDelay) before participating in the network. This informs $\mathcal{F}_{\text{PayNet}}$ how often the player has to POLL. "delay" corresponds to the maximum time between POLLs so that malicious closures will be caught. "relayDelay" is useful when the player is an intermediary of a multi-hop payment. It roughly represents the size of the time window the player has to learn a preimage from the next and reveal it to the previous node. Subsequently $\mathcal{F}_{\text{PayNet}}$ asks $\mathcal{S}$ to create and send a public key that will hold the player's funds. This public key is subsequently sent back to the player.
- To complete her registration, *Alice* has to send the (TOPPEDUP) message. It lets $\mathcal{F}_{\text{PayNet}}$ know that the desired amount of initial funds have been transferred to *Alice*'s public key. $\mathcal{F}_{\text{PayNet}}$ reads *Alice*'s state on $\mathcal{G}_{\text{Ledger}}$ to retrieve this number and subsequently allows *Alice* to participate in the payment network after it updates her `onChainBalance`.
- When $\mathcal{F}_{\text{PayNet}}$ receives (CHECKFORNEW, *Alice*, *Bob*, *tid*) from *Alice*, it asks $\mathcal{G}_{\text{Ledger}}$ for *Alice*'s latest state Σ_{Alice} and looks for a funding transaction F in it. If one is found, $\mathcal{S}$ is asked to complete the opening procedure.
- (PUSHFULFILL, *pchid*), (PUSHADD, *pchid*) and (COMMIT, *pchid*) all nudge $\mathcal{S}$ to carry on with the protocol that updates the state of a specific channel. $\mathcal{F}_{\text{PayNet}}$ simply forwards these messages to $\mathcal{S}$.
- (FULFILLONCHAIN) prompts $\mathcal{S}$ to close channels in which the counterparty is not willing to pay, even though they have promised to do so upon disclosure of a particular preimage. This message is simply forwarded to $\mathcal{S}$, but $\mathcal{F}_{\text{PayNet}}$ takes a note that such a message was received and the current blockheight in the view of the calling party.

Last but not least, $\mathcal{E}$ sends (GETNEWS) to obtain the latest changes regarding newly opened or closed channels, along with updates to the state of existing ones. Here we make an interesting observation: The most complex part of LN is arguably the negotiations that happen when a multi-hop payment takes place, due to the many channel updates needed; indeed, two complete channel updates for each hop are needed for a successful payment to go through. The fact that a proposal for an update can happen asynchronously with the commitment to this update, along with the fact that a single commitment may commit to many indiviual update proposals only adds to the complexity. It is therefore only natural to want $\mathcal{F}_{\text{PayNet}}$ to be unaware of these details. In order to disentangle the abstraction of $\mathcal{F}_{\text{PayNet}}$ from such minutiae, we allow the adversary full control of the updates that are reported back to $\mathcal{E}$ via $\mathcal{F}_{\text{PayNet}}$. Nevertheless, $\mathcal{F}_{\text{PayNet}}$ enforces that any reporting deviations induced by the adversary will be caught when a channel closes. This is quite intuitive: Consider a user of the payment network that does not understand its inner workings but can read $\mathcal{G}_{\text{Ledger}}$ and count her funds there. $\mathcal{F}_{\text{PayNet}}$ provides no guarantees regarding any specific interim reporting but the user is assured that in case she chooses to close the relevant channel, her state in $\mathcal{G}_{\text{Ledger}}$ will be consistent with all the payments that went through.

5. Lightning Protocol Π_{LN} Overview

In order to prove that software adhering to the LN specification fulfills the security guarantees given by $\mathcal{F}_{\text{PayNet}}$, a concrete protocol that implements LN in the UC model is needed. To that end we define the formal protocol Π_{LN}, an overview of which is given here.

For the rest of this section, we will assume that *Alice*, *Bob* and *Charlie* are interactive Turing machine instances (ITIs) [26] that honestly execute Π_{LN}. Similarly to the ideal world, the main functions of Π_{LN} are triggered when it receives (OPENCHANNEL), (PAY), (CLOSECHANNEL) and (FORCECLOSECHANNEL) from $\mathcal{E}$. These three messages along with (GETNEWS) informally correspond to actions that a "human user" would instruct the system to perform. (REGISTER) and (TOPPEDUP) are sent by $\mathcal{E}$ for player intialization. The rest of the messages sent from $\mathcal{E}$ prompt Π_{LN} to perform actions that a software implementation would spontaneously perform periodically. All messages sent between *Alice*, *Bob* and *Charlie* correspond to messages specified by LN. For clarity of exposition, we avoid mentioning the exact name and contents of every message. We refer the reader to the formal definition of Π_{LN} for further details [25].

Registration. Before *Alice* can use the network, $\mathcal{E}$ first has to send her a (REGISTER, delay, relayDelay) message. She generates her persistent key and sends it back to $\mathcal{E}$. The latter may choose to add some funds to this key and then send (TOPPEDUP) to *Alice*, who checks her state in $\mathcal{G}_{\text{Ledger}}$ and records her on-chain balance.

Channel opening. When she receives (OPENCHANNEL, *Alice*, *Bob*, x, *tid*) from $\mathcal{E}$, *Alice* initiates the message sequence needed to open a channel with *Bob*, funded by

her with x coins. After following the steps described in Section 3, the funding transaction has been submitted to $\mathcal{G}_{\text{Ledger}}$. However the channel is not open yet.

At a later point $\mathcal{E}$ may send (CHECKFORNEW, *Alice*, *Bob*, *tid*) to *Alice*. She then checks if her state in $\mathcal{G}_{\text{Ledger}}$ contains the funding transaction with the temporary ID *tid* and in that case she exchanges new commitment keys with *Bob*, as per Section 3. The channel is now open. Both parties keep a note to give $\mathcal{E}$ a receipt of the new channel the next time they receive (GETNEWS).

Channel closing. When sent by $\mathcal{E}$, the messages CLOSECHANNEL and FORCECLOSECHANNEL prompt *Alice* to close the channel cooperatively or unilaterally respectively, as explained in Section 3. In both cases she takes a note to notify $\mathcal{E}$ that the channel is closed when she receives (GETNEWS).

Performing payments. We will now follow the exact steps needed for a multi-hop payment, filling in many details that we omitted from Section 3. When she receives (PAY, *Charlie*, x, $\overrightarrow{\texttt{path}}$) from $\mathcal{E}$, *Alice* attempts to pay *Charlie* x coins, using the provided $\overrightarrow{\texttt{path}}$. Let us assume that the path is *Alice* – *Bob* – *Charlie*. *Alice* asks *Charlie* for an "invoice" with the HTLC hash, to which *Charlie* reacts by choosing a random preimage and sending back to *Alice* its hash. *Alice* then prepares a Sphinx [31] onion packet with the relevant information for each party on the $\overrightarrow{\texttt{path}}$ and sends it to *Bob*. *Bob* peels his layer of the onion and, after performing sanity checks and extracting the hash, he takes a note of this pending HTLC. He does not yet forward the onion to *Charlie*, because *Alice* is not yet committed to paying *Bob*. The latter happens if *Alice* subsequently receives (COMMIT, $pchid_{AB}$) from $\mathcal{E}$, where $pchid_{AB}$ is the ID of the *Alice* – *Bob* channel. She then sends *Bob* all the signatures needed to make the new commitment transaction spendable, who replies with the secret commitment key of the old commitment transaction (thus revoking it), along with the public commitment key of the future commitment transaction (to allow *Alice* to prepare the next update, when that happens). LN demands that before *Bob* forwards the onion to *Charlie*, he must commit as well to the HTLC to *Alice*. This happens if he receives the relevant COMMIT message from $\mathcal{E}$. Now that both parties have the HTLC in their commitment transaction and all past commitment transactions are revoked, they consider this HTLC "irrevocably committed".

Bob may then receive (PUSHADD, $pchid_{AB}$) from $\mathcal{E}$. *Bob* sends the onion to *Charlie*, who in turn peels it, recognizes that the payment is for him and that indeed he knows the preimage (since he generated it himself) and waits for the HTLC between him and *Bob* to be irrevocably committed. After both *Bob* and *Charlie* receive (COMMIT), *Charlie* awaits for a (PUSHFULFILL, $pchid_{BC}$) message from $\mathcal{E}$. If it arrives, *Charlie* sends the preimage to *Bob*, who sends it back to *Alice*. Once more every party has to receive a (COMMIT) message for each of the channels it participates in in order to remove the HTLC and update the definitive balance of each player to the appropriate value after the payment is complete. After this last update, each party keeps a note to inform $\mathcal{E}$ about the new balance when it receives (GETNEWS). *Alice* and *Charlie* also keep a note to inform $\mathcal{E}$ that the payment it had asked for succeeded.

Observe that while locked up in an HTLC, funds do not belong to either player; they are rather in a temporary, transitive state. If one party learns the preimage, the funds become theirs, whereas if it does not learn the preimage after some time, the other party is entitled to the funds. Also observe that within the UC framework the necessary messages COMMIT, PUSHFULFILL and PUSHADD may never arrive, but in a correct software implementation the corresponding actions happen automatically, without waiting for a prompt by the user.

Polling. Lastly, $\mathcal{E}$ may send (POLL) to *Alice*. She then reads her state in $\mathcal{G}_{\text{Ledger}}$ and checks for closed channels. If she finds maliciously closed channels (closed using old commitments), she punishes the counterparty and takes all the funds in the channel. If she finds in an honestly closed channel a preimage of an HTLC that she has previously signed and for which she is an intermediary, she records it and prepares to send it when she receives (PUSHFULFILL). Finally, if she finds an honestly closed channel with an HTLC output for which she knows the preimage, she spends it immediately. For every closed channel she finds, she keeps a note to report it to $\mathcal{E}$ the next time she receives (GETNEWS).

Remark 1 (Differences between LN and Π_{LN}). In LN, a custom construction for generating a new secret during each channel update is used. It reduces the space needed to maintain a channel from $O(n)$ to $O(\log n)$ in the number of updates. As far as we know, its security has not been formally analyzed. In the current paper we use instead a PRF [32].

As mentioned earlier, LN uses a custom construction that takes advantage of elliptic curve homomorphic properties in order to derive any number of keypairs by combining a single "basepoint" with different "labels". We instead use Identity Based Signatures [28], [29] (IBS) to abstract the properties provided by the construction. We also prove in the full version [25] that it actually implements an IBS.

Additionally, we have chosen to simplify the protocol in a number of ways in order to keep the analysis tractable. In particular LN defines several additional messages that signal various types of errors in transmission. It also specifies exactly how message retransmission should happen upon reconnection, specifically for the case of connection failure while updating a channel. This allows for a more robust system by excluding many cases of accidental channel closures. What is more, an LN user can change their "delay" and "relayDelay" parameters even after registration, which is not the case in Π_{LN}.

Lastly, in order to incentivize users to act as intermediaries or check for channel closures on behalf of others, LN provides for fees for these two roles. Furthermore, in order to reduce transaction size and ensure that bitcoin

nodes relay the transactions, it specifies exact rules for pruning outputs of too low value (known as "dust outputs"). In the current analysis we do not consider these features.

6. The Combined Signature primitive

As previously mentioned, we define a new primitive for combining keys and generating signatures, which is leveraged in the revocation and punishment mechanism of channel updates. Furthermore, we prove that the construction designed by the creators of LN realizes this primitive. We provide here the concrete syntax and correctness definitions, along with the intuition behind it, the exact security definitions, a concrete construction and proof of its security.

Previous work on the subject of multi-party signatures [12], [33], [34], [35], [36], [37] focuses on use-cases where some parties desire to generate a signature without revealing their private information; the latter is created using an interactive protocol. The resulting signatures can be verified by a single verification key, which is also included in the output of the key generation protocol. As we will see however, the primitive defined here has different aims and limitations and, to our knowledge, has not been formalized yet.

A combined signature is a two-party primitive, say between *Alice* and *Bob*, with *Bob* being the signer and *Alice* the holder of a share of the secret key. This share is essential for issuing signatures, which in turn are verifiable with the "combined" verification key. The verification key is generated using public information drawn from *Alice* and *Bob* and is feasible without any party knowing the corresponding signing key. *Bob* will be able to construct the signing key only if *Alice* shares her secret information with him.

More specifically, the seven algorithms used by a Combined Signatures scheme are:

- $(mpk, msk) \leftarrow \textsc{MasterKeyGen}\left(1^k\right)$
- $(pk, sk) \leftarrow \textsc{KeyShareGen}\left(1^k\right)$
- $cpk_l \leftarrow \textsc{CombinePubKey}\left(mpk, pk\right)$
- $(cpk_l, csk_l) \leftarrow \textsc{CombineKey}\left(mpk, msk, pk, sk\right)$
- $\{0,1\} \leftarrow \textsc{TestKey}\left(pk, sk\right)$
- $\sigma \leftarrow \textsc{SignCS}\left(m, csk\right)$
- $\{0,1\} \leftarrow \textsc{VerifyCS}\left(\sigma, m, cpk\right)$

We demand that these three properties hold for a scheme to have correctness:

- $\forall k \in \mathbb{N}$,
  $\Pr[(pk, sk) \leftarrow \textsc{KeyShareGen}\left(1^k\right),$
  $\textsc{TestKey}(pk, sk) = 1] = 1$
  I.e. $\textsc{KeyShareGen}()$ must always generate a valid keypair.
- $\forall k \in \mathbb{N}$,
  $\Pr[(mpk, msk) \leftarrow \textsc{MasterKeyGen}\left(1^k\right),$
  $(pk, sk) \leftarrow \textsc{KeyShareGen}\left(1^k\right),$
  $(cpk_1, csk_1) \leftarrow \textsc{CombineKey}\left(mpk, msk, pk, sk\right),$
  $cpk_2 \leftarrow \textsc{CombinePubKey}\left(mpk, pk\right),$
  $cpk_1 = cpk_2] = 1$
  I.e. for suitable input, $\textsc{CombinePubKey}()$ and $\textsc{CombineKey}()$ produce the same public key.
- $\forall k \in \mathbb{N}, m \in \mathcal{M}$,
  $\Pr[(mpk, msk) \leftarrow \textsc{MasterKeyGen}\left(1^k\right),$
  $(pk, sk) \leftarrow \textsc{KeyShareGen}\left(1^k\right),$
  $(cpk, csk) \leftarrow \textsc{CombineKey}\left(mpk, msk, pk, sk\right),$
  $\textsc{VerifyCS}(\textsc{SignCS}(m, csk), m, cpk) = 1] = 1$
  I.e. for suitable input, honestly generated signatures always verify correctly.

Beyond correctness, combined signatures have two security properties expressed as follows. Share-EUF security expresses security from the point of view of *Alice*, and establishes that *Bob* cannot issue a valid combined signature if he does not possess *Alice*'s corresponding secret share. Formally:

Game share-EUF$^{\mathcal{A}}\left(1^k\right)$

1: $(\mathtt{aux}, mpk, n) \leftarrow \mathcal{A}\left(\textsc{init}\right)$
2: **for** $i \leftarrow 1$ to n **do**
3: $\quad (pk_i, sk_i) \leftarrow \textsc{KeyShareGen}\left(1^k\right)$
4: **end for**
5: $(cpk^*, pk^*, m^*, \sigma^*) \leftarrow \mathcal{A}\left(\textsc{keys}, \mathtt{aux}, pk_1, \ldots, pk_n\right)$
6: **if** $pk^* \in \{pk_1, \ldots, pk_n\} \wedge$
$cpk^* = \textsc{CombinePubKey}\left(mpk, pk^*\right) \wedge$
$\textsc{VerifyCS}\left(\sigma^*, m^*, cpk^*\right) = 1$ **then**
7: $\quad$ **return** 1
8: **else**
9: $\quad$ **return** 0
10: **end if**

Definition 1. *A Combined Signatures scheme is* share-EUF-secure *if*

$$\forall \mathcal{A} \in \mathsf{PPT}, \Pr\left[\textsf{share-EUF}^{\mathcal{A}}\left(1^k\right) = 1\right] = \mathrm{negl}\left(k\right)$$

or equivalently

$$\text{E-share}(k) = \mathrm{negl}\left(k\right) \quad ,$$

where $\text{E-share}(k) = \sup_{\mathcal{A} \in \mathsf{PPT}} \{\Pr[\textsf{share-EUF}^{\mathcal{A}}\left(1^k\right) = 1]\}$

On the other hand, master-EUF-CMA security is modeled very similarly to standard EUF-CMA security, with the difference that *Bob* (the signer) combines malicious shares into his public key and issues signatures with respect to such combined keys. The security property ensures that these signatures provide no advantage to the adversary in terms of producing a forged message for a combined key of its choice. Formally:

Game master-EUF-CMA$^{\mathcal{A}}\left(1^k\right)$

1: $(mpk, msk) \leftarrow \textsc{MasterKeyGen}\left(1^k\right)$
2: $i \leftarrow 0$
3: $(\mathtt{aux}_i, \text{response}) \leftarrow \mathcal{A}\left(\textsc{init}, mpk\right)$
4: **while** response can be parsed as (pk, sk, m) **do**
5: $\quad i \leftarrow i + 1$

store pk, sk, m as pk_i, sk_i, m_i
$(cpk_i, csk_i) \leftarrow \text{CombineKey}(mpk, msk, pk_i, sk_i)$
$\sigma_i \leftarrow \text{SignCS}(m_i, csk_i)$
$(\texttt{aux}_i, \text{response}) \leftarrow \mathcal{A}(\text{SIGNATURE}, \texttt{aux}_{i-1}, \sigma_i)$
end while
parse response as $(cpk^*, pk^*, m^*, \sigma^*)$
if $m^* \notin \{m_1, \ldots, m_i\} \wedge$
$cpk^* = \text{CombinePubKey}(mpk, pk^*) \wedge$
$\text{VerifyCS}(\sigma^*, m^*, cpk^*) = 1$ **then**
return 1
else
return 0
end if

Definition 2. *A Combined Signatures scheme is* master-EUF-CMA-secure *if*

$$\forall \mathcal{A} \in \text{PPT},$$
$$\Pr\left[\text{master-EUF-CMA}^{\mathcal{A}}\left(1^k\right) = 1\right] = \text{negl}(k)$$
or equivalently
$$\text{E-master}(k) = \text{negl}(k) \quad ,$$
where
$$\text{E-master}(k) = \sup_{\mathcal{A} \in \text{PPT}} \{\Pr[\text{master-EUF} - \text{CMA}^{\mathcal{A}}\left(1^k\right) = 1]\}$$

Definition 3. *A Combined Signatures scheme is* combine-EUF-secure *if it is both share-EUF-secure and master-EUF-CMA-secure.*

In conclusion, a collection of algoritms is said to be a secure Combined Signatures scheme if it conforms to the syntax of the seven aforementioned algorithms, it satisfies the three correctness properties and provides existential unforgeability against key-only attacks with respect to key shares and existential unforgeability against chosen message attacks with respect to master keys.

We here define the particular construction for Combined Signatures used in LN and prove its security.

Parameters: hash function $\mathcal{H}$, group generator G

MasterKeyGen(1^k, rand):
return $(G \cdot \text{rand}, \text{rand})$

KeyShareGen(1^k, rand):
return $(G \cdot \text{rand}, \text{rand})$

CombinePubKey(mpk, pk):
return $mpk \cdot \mathcal{H}(mpk \,\|\, pk) + pk \cdot \mathcal{H}(pk \,\|\, mpk)$

CombineKey(mpk, msk, pk, sk):
return $(\text{CombinePubKey}(mpk, pk), msk \cdot \mathcal{H}(mpk \,\|\, pk) + sk \cdot \mathcal{H}(pk \,\|\, mpk))$

TestKey(pk, sk):
if $pk = G \cdot sk$ **then**
return 1
else
return 0
end if

SignCS(m, csk):
return SignDS(m, csk)

VerifyCS(σ, m, cpk):
return VerifyDS(σ, m, cpk)

One can check by inspection that the syntax above matches the one required by the Combined Signatures scheme definition. Furthermore, assuming that SignDS() and VerifyDS() are provided by a correct Digital Signature construction, it is straightforward to confirm that the construction here satisfies the Combined Signatures correctness properties.

We now move on to proving that the construction is also secure.

Lemma 1. *The construction defined above is share-EUF-secure in the Random Oracle model under the assumption that the underlying signature scheme is strongly EUF-CMA-secure and the range of the Random Oracle coincides with that of the underlying signature scheme signing keys.*

Proof: Let $k \in \mathbb{N}, \mathcal{B}$ PPT algorithm such that

$$\Pr\left[\text{share-EUF}^{\mathcal{B}}\left(1^k\right) = 1\right] = a = \text{non} - \text{negl}(k) \quad .$$

We construct a PPT distinguisher $\mathcal{A}$ (Fig. 5) such that

$$\Pr\left[\text{EUF-CMA}^{\mathcal{A}}\left(1^k\right) = 1\right] = \text{non} - \text{negl}(k)$$

that breaks the assumption, thus proving Lemma 1.

Let Y be the range of the random oracle. The modified random oracle used in Fig. 5 is indistinguishable from the standard random oracle by PPT algorithms since the statistical distance of the standard random oracle from the modified one is at most $\frac{1}{2|Y|} = \text{negl}(k)$ as they differ in at most one element.

Let E denote the event in which $\mathcal{B}$ does not invoke CombinePubKey to produce cpk^*. In that case the values $\mathcal{H}(pk^* \,\|\, mpk)$ and $\mathcal{H}(mpk \,\|\, pk^*)$ are decided after $\mathcal{B}$ terminates (Fig. 5, line 24) and thus

$$\begin{gathered}\Pr[cpk^* = \text{CombinePubKey}(mpk, pk^*) \,|E] = \\ = \frac{1}{|Y|} = \text{negl}(k) \Rightarrow \\ \Pr[cpk^* = \text{CombinePubKey}(mpk, pk^*) \wedge E] = \\ = \text{negl}(k) \quad .\end{gathered} \tag{1}$$

It is

$$\begin{gathered}(\mathcal{B} \text{ wins}) \rightarrow (cpk^* = \text{CombinePubKey}(mpk, pk^*)) \Rightarrow \\ \Pr[\mathcal{B} \text{ wins}] \leq \\ \leq \Pr[cpk^* = \text{CombinePubKey}(mpk, pk^*)] \Rightarrow \\ \Pr[\mathcal{B} \text{ wins} \wedge E] \leq \\ \leq \Pr[cpk^* = \text{CombinePubKey}(mpk, pk^*) \wedge E] \overset{(1)}{\Rightarrow} \\ \Pr[\mathcal{B} \text{ wins} \wedge E] = \text{negl}(k) \quad .\end{gathered}$$

Algorithm $\mathcal{A}(vk)$

1: $j \xleftarrow{\$} U[1, T(\mathcal{B})]$ // $T(M)$ is the maximum running time of M
2: Random Oracle: for every first-seen query q from $\mathcal{B}$ set $\mathcal{H}(q)$ to a random value
3: **return** $\mathcal{H}(q)$ to $\mathcal{B}$
4: $(\mathsf{aux}, mpk, n) \leftarrow \mathcal{A}(\text{INIT})$
5: **for** $i \leftarrow 1$ to n **do**
6: $(pk_i, sk_i) \leftarrow \text{KEYSHAREGEN}(1^k)$
7: **end for**
8: Random Oracle: Let q be the jth first-seen query from $\mathcal{B}$:
9: **if** $q = (mpk \,\|\, x)$ **then**
10: **if** $\mathcal{H}(x \,\|\, mpk)$ unset **then**
11: set $\mathcal{H}(x \,\|\, mpk)$ to a random value
12: **end if**
13: set $\mathcal{H}(mpk \,\|\, x)$ to $(vk - x \cdot \mathcal{H}(x \,\|\, mpk)) \cdot mpk^{-1}$
14: **else if** $q = (x \,\|\, mpk)$ **then**
15: **if** $\mathcal{H}(mpk \,\|\, x)$ unset **then**
16: set $\mathcal{H}(mpk \,\|\, x)$ to a random value
17: **end if**
18: set $\mathcal{H}(x \,\|\, mpk)$ to $(vk - mpk \cdot \mathcal{H}(mpk \,\|\, x)) \cdot x^{-1}$
19: **else**
20: set $\mathcal{H}(q)$ to a random value
21: **end if**
22: **return** $\mathcal{H}(q)$ to $\mathcal{B}$
23: $(cpk^*, pk^*, m^*, \sigma^*) \leftarrow \mathcal{B}(\text{KEYS}, \mathsf{aux}, pk_1, \ldots, pk_n)$
24: **if** $vk = cpk^* \wedge \mathcal{B}$ wins the share-EUF game **then** // $\mathcal{A}$ won the EUF-CMA game
25: **return** (m^*, σ^*)
26: **else**
27: **return** FAIL
28: **end if**

Figure 5.

But we know that $\Pr[\mathcal{B} \text{ wins}] = \Pr[\mathcal{B} \text{ wins} \wedge E] + \Pr[\mathcal{B} \text{ wins} \wedge \neg E]$ and $\Pr[\mathcal{B} \text{ wins}] = a$ by the assumption, thus

$$\Pr[\mathcal{B} \text{ wins} \wedge \neg E] > a - \text{negl}(k) \quad . \tag{2}$$

We now focus at the event $\neg E$. Let F the event in which the call of $\mathcal{B}$ to CombinePubKey to produce cpk^* results in the jth invocation of the Random Oracle. Since j is chosen uniformly at random and using Proposition 1 of [25], $\Pr[F|\neg E] = \frac{1}{T(\mathcal{B})}$. Observe that $\Pr[F|E] = 0 \Rightarrow \Pr[F] = \Pr[F|\neg E] = \frac{1}{T(\mathcal{B})}$.

In the case where the event $(F \wedge \mathcal{B} \text{ wins} \wedge \neg E)$ holds, it is

$$cpk^* = \text{CombinePubKey}(mpk, pk^*) = mpk \cdot \mathcal{H}(mpk \,\|\, pk^*) + pk^* \cdot \mathcal{H}(pk^* \,\|\, mpk)$$

Since F holds, the jth invocation of the Random Oracle queried either the value $\mathcal{H}(mpk \,\|\, pk^*)$ or $\mathcal{H}(pk^* \,\|\, mpk)$. In either case (Fig. 5, lines 9-18), it is $cpk^* = vk$. This means that $\text{VerifyCS}(\sigma^*, m^*, vk) = 1$. We conclude that in the event $(F \wedge \mathcal{B} \text{ wins} \wedge \neg E)$, $\mathcal{A}$ wins the EUF-CMA game. A final observation is that the probability that the events $(\mathcal{B} \text{ wins} \wedge \neg E)$ and F are almost independent, thus

$$\Pr[F \wedge \mathcal{B} \text{ wins} \wedge \neg E] = \\ = \Pr[F] \Pr[\mathcal{B} \text{ wins} \wedge \neg E] \pm \text{negl}(k) \overset{(2)}{=} \\ \frac{a - \text{negl}(k)}{T(\mathcal{B})} \pm \text{negl}(k) = \text{non} - \text{negl}(k)$$

□

Lemma 2. *The construction above is master-EUF-CMA-secure in the Random Oracle model under the assumption that the underlying signature scheme is strongly EUF-CMA-secure and the range of the Random Oracle coincides with that of the underlying signature scheme signing keys.*

Proof: Let $k \in \mathbb{N}, \mathcal{B}$ PPT algorithm such that

$$\Pr\left[\text{master-EUF-CMA}^{\mathcal{B}}(1^k) = 1\right] = a = \text{non} - \text{negl}(k) \quad .$$

We construct a PPT distinguisher $\mathcal{A}$ (Fig. 6) such that

$$\Pr\left[\text{EUF-CMA}^{\mathcal{A}}(1^k) = 1\right] = \text{non} - \text{negl}(k)$$

that breaks the assumption, thus proving Lemma 2.

The modified random oracle used in Fig. 6 is indistinguishable from the standard random oracle for the same reasons as in the proof of Lemma 1.

Let E denote the event in which CombinePubKey is not invoked to produce cpk^*. In that case the values $\mathcal{H}(pk^* \,\|\, mpk)$ and $\mathcal{H}(mpk \,\|\, pk^*)$ are decided after $\mathcal{B}$ terminates (Fig. 6, line 30) and thus

$$\Pr[cpk^* = \text{CombinePubKey}(mpk, pk^*) \,|E] = \\ = \text{negl}(k) \Rightarrow \\ \Pr[cpk^* = \text{CombinePubKey}(mpk, pk^*) \wedge E] = \\ = \text{negl}(k) \quad . \tag{3}$$

We can reason like in the proof of Lemma 1 to deduce that

$$\Pr[\mathcal{B} \text{ wins} \wedge \neg E] > a - \text{negl}(k) \quad . \tag{4}$$

We now focus at the event $\neg E$. Let F the event in which the call of to CombinePubKey that produces cpk^* results in the jth invocation of the Random Oracle. Since j is chosen uniformly at random and using Proposition 1 of [25], $\Pr[F|\neg E] = \frac{1}{T(\mathcal{B})+T(\mathcal{A})}$. Observe that $\Pr[F|E] = 0 \Rightarrow \Pr[F] = \Pr[F|\neg E] = \frac{1}{T(\mathcal{B})+T(\mathcal{A})}$.

Once more we can reason in the same fashion as in the proof of Lemma 1 to deduce that

$$\Pr[F \wedge \mathcal{B} \text{ wins} \wedge \neg E] = \\ = \Pr[F] \Pr[\mathcal{B} \text{ wins} \wedge \neg E] \pm \text{negl}(k) \overset{(4)}{=} \\ \frac{a - \text{negl}(k)}{T(\mathcal{B}) + T(\mathcal{A})} \pm \text{negl}(k) = \text{non} - \text{negl}(k)$$

□

Algorithm $\mathcal{A}(vk)$

$j \xleftarrow{\$} U[1, T(\mathcal{B}) + T(\mathcal{A})]$ // $T(M)$ is the maximum running time of M
  Random Oracle: for every first-seen query q from $\mathcal{B}$ set $\mathcal{H}(q)$ to a random value
  return $\mathcal{H}(q)$ to $\mathcal{B}$
$(mpk, msk) \leftarrow$ MasterKeyGen $\left(1^k\right)$
  Random Oracle: Let q be the jth first-seen query from $\mathcal{B}$ or $\mathcal{A}$:
  if $q = (mpk \,\|\, x)$ **then**
    if $\mathcal{H}(x \,\|\, mpk)$ unset **then**
      set $\mathcal{H}(x \,\|\, mpk)$ to a random value
    end if
    set $\mathcal{H}(mpk \,\|\, x)$ to $(vk - x \cdot \mathcal{H}(x \,\|\, mpk)) \cdot mpk^{-1}$
  else if $q = (x \,\|\, mpk)$ **then**
    if $\mathcal{H}(mpk \,\|\, x)$ unset **then**
      set $\mathcal{H}(mpk \,\|\, x)$ to a random value
    end if
    set $\mathcal{H}(x \,\|\, mpk)$ to $(vk - mpk \cdot \mathcal{H}(mpk \,\|\, x)) \cdot x^{-1}$
  else
    set $\mathcal{H}(q)$ to a random value
  end if
  return $\mathcal{H}(q)$ to $\mathcal{B}$ or $\mathcal{A}$
$i \leftarrow 0$
$(\mathtt{aux}_i, \text{response}) \leftarrow \mathcal{B}(\text{INIT}, mpk)$
while response can be parsed as (pk, sk, m) **do**
  $i \leftarrow i + 1$
  store pk, sk, m as pk_i, sk_i, m_i
  $(cpk_i, csk_i) \leftarrow$ CombineKey (mpk, msk, pk_i, sk_i)
  $\sigma_i \leftarrow$ SignCS (m_i, csk_i)
  $(\mathtt{aux}_i, \text{response}) \leftarrow \mathcal{B}(\text{SIGNATURE}, \mathtt{aux}_{i-1}, \sigma_i)$
end while
parse response as $(cpk^*, pk^*, m^*, \sigma^*)$
if $vk = cpk^* \wedge \mathcal{B}$ wins the master-EUF-CMA game **then** // $\mathcal{A}$ won the EUF-CMA game
  return (m^*, σ^*)
else
  return FAIL
end if

Figure 6.

The two results can then be combined to obtain the desired security property:

Theorem 1. *The construction above is combine-EUF-secure in the Random Oracle model under the assumption that the underlying signature scheme is strongly EUF-CMA-secure.*

Proof: The construction is combine-EUF-secure as a consequence of Lemma 1, Lemma 2 and the definition of combine-EUF-security. □

7. Security proof overview

Theorem 2 (Lightning Payment Network Security)**.** *The protocol* Π_{LN} *realises* $\mathcal{F}_{\text{PayNet}}$ *given a global functionality* $\mathcal{G}_{\text{Ledger}}$ *and assuming the security of the underlying digital signature, identity-based signature, combined digital signature and PRF. Specifically,*

$$\forall k \in \mathbb{N},\ PPT\ \mathcal{E},$$
$$|\Pr[EXEC^{\mathcal{G}_{\text{Ledger}}}_{\Pi_{\text{LN}},\mathcal{A}_{\text{d}},\mathcal{E}} = 1] - \Pr[EXEC^{\mathcal{F}_{\text{PayNet}},\mathcal{G}_{\text{Ledger}}}_{\mathcal{S},\mathcal{E}} = 1]| \leq$$
$$2nm\text{E-ds}(k) + 6np\text{E-ids}(k) + 2nmp\text{E-share}(k) +$$
$$+2nm\text{E-master}(k) + 2\text{E-prf}(k)\ ,$$

where n is the maximum number of registered users, m is the maximum number of channels that a user is involved in, p is the maximum number of times that a channel is updated and the "E-" terms correspond to the insecurity bounds of the primitives.

Proof Sketch: The proof is done in 5 steps of successive game replacement. In the first lemma, we define a simulator $\mathcal{S}_{\text{LN}}$ that internally simulates a full execution of Π_{LN} for each player, and a "dummy" functionality that acts as a simple relay between $\mathcal{E}$ and $\mathcal{S}_{\text{LN}}$. We argue that this version of the ideal world trivially produces the exact same messages for $\mathcal{E}$ as the real world.

In each subsequent step, we incrementally move responsibilities from the simulator to the functionality, while ensuring the change is transparent to both $\mathcal{E}$ and $\mathcal{A}$. Each step defines a different functionality that handles some additional messages from $\mathcal{E}$ exactly like $\mathcal{F}_{\text{PayNet}}$, until the last step where we use $\mathcal{F}_{\text{PayNet}}$ itself. Correspondingly, the simulator of each step is adapted so that the new ideal execution is computationally indistinguishable from the previous one. For each step we exhaustively trace the differences from the previous step in order to prove that, given the same messages from $\mathcal{E}$ and $\mathcal{A}$, the resulting responses remain unchanged.

The second step, lets $\mathcal{F}$ handle registration messages, along with the corruption messages from $\mathcal{S}$. In the third step, the functionality additionally handles messages related to channel opening. It behaves like $\mathcal{F}_{\text{PayNet}}$, but does not execute `checkClosed`(). The fourth step, has the functionality handle all messages sent during channel updates. Lastly, the entire $\mathcal{F}_{\text{PayNet}}$ is used as functionality, by incorporating the message for closing a channel, executing `checkClosed`() normally and handing the message that returns to $\mathcal{E}$ the receipts for newly opened, updated and closed channels. The last two steps introduce a probability of failure in case the various types of signatures used in Π_{LN} are forged. We analyze these cases separately and argue that, if such forgeries do not happen, the emulation is perfect. Therefore we can calculate the concrete security bounds shown in the theorem. □

As a concrete example of the proof approach, the second step entails the following parts: First $\mathcal{F}_{\text{PayNet,Reg}}$ is defined, which is a functionality that behaves exactly like $\mathcal{F}_{\text{PayNet}}$ when receiving the messages REGISTER, REGISTERDONE, TOPPEDUP and CORRUPTED, but simply forwards all other messages along with the sender to $\mathcal{S}$. Then $\mathcal{S}_{\text{LN}-\text{Reg}}$ is defined, which simulates all protocol

instances, but in response to REGISTER messages from $\mathcal{F}_{\text{PayNet,Reg}}$, it provides the public key of the key it just generated (as $\mathcal{F}_{\text{PayNet,Reg}}$ expects). It also keeps track of corruptions and informs $\mathcal{F}_{\text{PayNet,Reg}}$ thereof. Lastly, we argue that the functionality and simulator that were used in the first step can be replaced by their newly defined counterparts without introducing any discernible difference to the transcript that any $\mathcal{E}$ sees. This is achieved by exhaustive enumeration of all possible messages and comparison of the behaviour of the ideal and the real world for each, to conclude that the change is transparent to $\mathcal{E}$. The formal proof can be found in the full version [25].

8. Instant finality ledgers are unrealisable

As already mentioned, previous attempts at formalising payment channels in UC [13], [14], [15], [16] assume a variant of a ledger functionality with instant finality. We here define a representative variant of this approach $\mathcal{F}_{\text{PerfectL}}$ (defined below) where all submitted transactions are instantly added to the ledger and immediately available to be read by all players. Subsequently we argue that, albeit an attractive abstraction, such a functionality is unrealisable, even under strong network assumptions, i.e. a multicast synchronous network $\mathcal{F}^1_{\text{N-MC}}$. Such a network ensures that messages sent by honest parties will be instantly delivered to all other parties; no delays can be introduced by the adversary. We refer the reader to the full version [25] for the formal definition of $\mathcal{F}^1_{\text{N-MC}}$. The adversary however may choose to send its own messages only to specific parties. This allows the adversary to spread conflicting information or withhold data from some parties. This adversarial ability precludes the possibility of such a ledger to be realised. The complete proof can be found in the full version [25].

Theorem 3 (Perfect Ledger is Unrealisable)**.** *For any ITM* Π_{PerfectL} *there exist ITMs* $\mathcal{E}_{\text{PL}}$*,* $\mathcal{A}_{\text{PL}}$ *such that for any ITM* $\mathcal{S}$

$$EXEC^{\mathcal{F}^1_{\text{N-MC}},\bar{\mathcal{G}}_{CLOCK}}_{\Pi_{\text{PerfectL}},\mathcal{A}_{\text{PL}},\mathcal{E}_{\text{PL}}} \not\approx EXEC^{\mathcal{F}_{\text{PerfectL}},\bar{\mathcal{G}}_{CLOCK}}_{\mathcal{S},\mathcal{E}_{\text{PL}}}$$

Proof Sketch: We take advantage of $\mathcal{A}_{\text{PL}}$'s ability to selectively send messages to specific players. In particular, $\mathcal{E}_{\text{PL}}$ starts an execution with two players and generates a random message m. In half of the executions (randomly selected), the adversary simulates a "broken" Π_{PerfectL} execution where the effects of submitting m are only shared with one of the two players, say *Alice* by $\mathcal{A}_{\text{PL}}$ (in the real world). The environment then sends (READ) to the other player, say *Bob*. If *Bob* returns a ledger containing m, then $\mathcal{E}_{\text{PL}}$ concludes that it is the ideal world, otherwise it sends (READ) to *Alice*. If she returns a ledger with m, then $\mathcal{E}_{\text{PL}}$ concludes it is in the real world, otherwise it concludes it is in the ideal world.

The above is not sufficient since a protocol may choose to return an empty ledger; to counter this, in the other half of the executions, $\mathcal{E}_{\text{PL}}$ sends (SUBMIT, m) to *Alice* and then (READ) to *Bob*. If, and only if, *Bob* knows m, then $\mathcal{E}_{\text{PL}}$ concludes this is the ideal world. This forces the Π_{PerfectL} protocol to achieve instant finality and will establish that a distinguishing advantage exists no matter how Π_{PerfectL} is implemented. □

Functionality $\mathcal{F}_{\text{PerfectL}}$

1: State: List of txs $\mathcal{L}$

2: Upon receiving (SUBMIT, m) from *Alice* or $\mathcal{A}$, append m to $\mathcal{L}$ and send (SUBMIT, *Alice* or $\mathcal{A}$, m) to $\mathcal{A}$

3: Upon receiving (READ) from *Alice*, send (READ, $\mathcal{L}$) to *Alice*

9. Future Work and Conclusion

In order to remain tractable, the current analysis omits some parts of the lightning specification. In particular, the specification defines how intermediaries of multi-hop payments can charge a fee for their service. Furthermore, the per-update secret generation is not done with a PRF according to the specification: an optimisation that reduces the storage overhead for the counterparty is used instead. The security of this optimisation however has not been yet formally inspected. Additionally, the specification provisions for a number of failure messages that help in keeping counterparties informed of issues with requested payments and in alleviating the problem of unneeded precautionary channel closures. Transactions that are added on-chain offer a fee to the blockchain miners (unrelated to the fee of the off-chain multi-hop payments). When closing a channel cooperatively, this fee is contributed by both counterparties, therefore the closing sequence of the specification includes an iterative negotiation of said fee where the two parties repeatedly propose a value based on their settings until they converge to a compromise or fail to agree. Lastly, most Bitcoin nodes do not relay transactions that include outputs with tiny amounts of coins, a.k.a "dust" outputs, to avoid bloating the blockchain. The lightning specification provides extensive instructions as to how to prune such outputs.

All aforementioned parts of the protocol were not analysed so that the security of the core parts of the lightning protocol could be discussed without distractions. In order for the analysis to cover the entirety of the current version of the lightning specification however, the aforementioned features should be incorporated and their security should be proven. This expansion of the analysis is left as future work.

In a different direction, big parts of our main security proof consist of an exhaustive enumeration of the possible messages that $\mathcal{E}$ and $\mathcal{A}$ can send to the protocol, the simulator or the functionality and tracking how such messages would change the flow of the execution of the ideal and the real world. It is then argued that in all

cases the messages that would be sent to $\mathcal{E}$ and $\mathcal{A}$ are indistinguishable. These parts of the proof are good candidates for rewriting in the environment of an automated proof assistant [38] to instill additional certainty that all possible execution paths are indeed checked and do not contain subtle sources of distinguishability. Combining our results with the recent mechanization of UC via Easycrypt [39] would be a natural and interesting direction for future work.

The lightning specification is not static, but it is continuously undergoing a number of improvements. The most noteworthy upcoming change is the introduction of Pointlocked TimeLocked Contracts (PTLCs). This mechanism replaces HTLCs and promises to combat the "wormhole" attack [30], while increasing privacy. Our work can be modified to cover the case of PTLCs with relative ease. It also provides a suitable framework for future work that aims to shed light on the exact privacy benefit that PTLCs offer as opposed to HTLCs.

The present analysis constitutes the first comprehensive treatment in the Universal Composability framework of a deployed layer-2 protocol on top of a functional ledger. It can be extended and adapted to analyze other similar protocols that achieve different security goals or use another ledger as base layer.

Conclusion. The present work constitutes a fortunate result, since it conclusively proves that software that adhere to the lightning specification cannot lose funds accidentally to a malicious protocol player. Indeed, such a result reinforces trust to the lightning network and acts as a guarantee to the almost 900 bitcoins currently in circulation in the layer-2 protocol.

By leveraging the guarantees provided by the Universal Composability framework, we further assert that the lightning protocol is freely composable with other protocols. As such it can run side by side with arbitrary protocols, or be used as a subroutine to higher-level protocols without needing to prove its security anew.

By separating particular subroutines of the protocol as distinct cryptographic primitives and analysing them individually, we have contributed to its cryptorgaphic agility. Lastly, by keeping it as protocol-agnostic as possible, our payment network functionality can be adapted to express the functional and security requirements of other layer-2 protocols with relative ease.

References

[1] Croman K., Decker C., Eyal I., Gencer A. E., Juels A., Kosba A., Miller A., Saxena P., Shi E., Sirer E. G., et al.: On scaling decentralized blockchains. In International Conference on Financial Cryptography and Data Security: pp. 106–125: Springer (2016)

[2] Nakamoto S.: Bitcoin: A Peer-to-Peer Electronic Cash System (2008)

[3] Garay J., Kiayias A., Leonardos N.: The Bitcoin Backbone Protocol: Analysis and Applications. Cryptology ePrint Archive, Report 2014/765: https://eprint.iacr.org/2014/765 (2014)

[4] Pass R., Seeman L., Shelat A.: Analysis of the Blockchain Protocol in Asynchronous Networks. IACR Cryptology ePrint Archive: vol. 2016, p. 454: URL http://eprint.iacr.org/2016/454 (2016)

[5] Garay J. A., Kiayias A., Leonardos N.: The Bitcoin Backbone Protocol with Chains of Variable Difficulty. In J. Katz, H. Shacham (editors), Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I: vol. 10401 of *Lecture Notes in Computer Science*: pp. 291–323: Springer: ISBN 978-3-319-63687-0: doi:10.1007/978-3-319-63688-7_10: URL https://doi.org/10.1007/978-3-319-63688-7_10 (2017)

[6] Pass R., Shi E.: Hybrid Consensus: Efficient Consensus in the Permissionless Model. In A.W. Richa (editor), 31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria: vol. 91 of *LIPIcs*: pp. 39:1–39:16: Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik: ISBN 978-3-95977-053-8: doi:10.4230/LIPIcs.DISC.2017.39: URL https://doi.org/10.4230/LIPIcs.DISC.2017.39 (2017)

[7] Micali S.: ALGORAND: The Efficient and Democratic Ledger. CoRR: vol. abs/1607.01341: URL http://arxiv.org/abs/1607.01341 (2016)

[8] Poon J., Dryja T.: The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments. https://lightning.network/lightning-network-paper.pdf (2016)

[9] Pass R., Shi E.: Thunderella: Blockchains with Optimistic Instant Confirmation. In J.B. Nielsen, V. Rijmen (editors), Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II: vol. 10821 of *Lecture Notes in Computer Science*: pp. 3–33: Springer: ISBN 978-3-319-78374-1: doi:10.1007/978-3-319-78375-8_1: URL https://doi.org/10.1007/978-3-319-78375-8_1 (2018)

[10] Badertscher C., Maurer U., Tschudi D., Zikas V.: Bitcoin as a transaction ledger: A composable treatment. In Annual International Cryptology Conference: pp. 324–356: Springer (2017)

[11] Badertscher C., Gaži P., Kiayias A., Russell A., Zikas V.: Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security: pp. 913–930: ACM (2018)

[12] Nicolosi A., Krohn M. N., Dodis Y., Mazières D.: Proactive Two-Party Signatures for User Authentication. In Proceedings of the Network and Distributed System Security Symposium, NDSS 2003, San Diego, California, USA: The Internet Society: ISBN 1-891562-16-9: URL http://www.isoc.org/isoc/conferences/ndss/03/proceedings/papers/15.pdf (2003)

[13] Dziembowski S., Faust S., Hostáková K.: General State Channel Networks. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018: pp. 949–966: doi:10.1145/3243734.3243856: URL https://doi.org/10.1145/3243734.3243856 (2018)

[14] Malavolta G., Moreno-Sanchez P., Kate A., Maffei M., Ravi S.: Concurrency and Privacy with Payment-Channel Networks. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security: CCS '17: pp. 455–471: ACM, New York, NY, USA: ISBN 978-1-4503-4946-8: doi:10.1145/3133956.3134096: URL http://doi.acm.org/10.1145/3133956.3134096 (2017)

[15] Miller A., Bentov I., Kumaresan R., Cordi C., McCorry P.: Sprites and State Channels: Payment Networks that Go Faster than Lightning. ArXiv preprint arXiv:1702.05812 (2017)

[16] Dziembowski S., Eckey L., Faust S., Malinowski D.: Perun: Virtual Payment Hubs over Cryptocurrencies. In 2019 2019 IEEE Symposium on Security and Privacy (SP): pp. 344–361: IEEE Computer Society, Los Alamitos, CA, USA: ISSN 2375–1207: doi:10.1109/SP.2019.00020: URL https://doi.ieeecomputersociety.org/10.1109/SP.2019.00020 (2019)

[17] Spilman J.: Anti dos for tx replacement. https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2013-April/002433.html (2013)

[18] Decker C., Wattenhofer R.: A Fast and Scalable Payment Network with Bitcoin Duplex Micropayment Channels. In A. Pelc, A.A. Schwarzmann (editors), Stabilization, Safety, and Security of Distributed Systems - 17th International Symposium, SSS 2015, Edmonton, AB, Canada, August 18-21, 2015, Proceedings: vol. 9212 of *Lecture Notes in Computer Science*: pp. 3–18: Springer: ISBN 978-3-319-21740-6: doi:10.1007/978-3-319-21741-3_1: URL https://doi.org/10.1007/978-3-319-21741-3_1 (2015)

[19] Lind J., Naor O., Eyal I., Kelbert F., Pietzuch P. R., Sirer E. G.: Teechain: Reducing Storage Costs on the Blockchain With Offline Payment Channels. In Proceedings of the 11th ACM International Systems and Storage Conference, SYSTOR 2018, HAIFA, Israel, June 04-07, 2018: p. 125: ACM: doi:10.1145/3211890.3211904: URL https://doi.org/10.1145/3211890.3211904 (2018)

[20] Green M., Miers I.: Bolt: Anonymous Payment Channels for Decentralized Currencies. In Thuraisingham et al. [40]: pp. 473–489: doi:10.1145/3133956.3134093: URL https://doi.org/10.1145/3133956.3134093 (2017)

[21] Heilman E., Alshenibr L., Baldimtsi F., Scafuro A., Goldberg S.: TumbleBit: An Untrusted Bitcoin-Compatible Anonymous Payment Hub. In 24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017: The Internet Society: URL https://www.ndss-symposium.org/ndss2017/ (2017)

[22] Khalil R., Gervais A.: Revive: Rebalancing Off-Blockchain Payment Networks. In Thuraisingham et al. [40]: pp. 439–453: doi:10.1145/3133956.3134033: URL https://doi.org/10.1145/3133956.3134033 (2017)

[23] Prihodko P., Zhigulin S., Sahno M., Ostrovskiy A.: Flare: An Approach to Routing in Lightning Network: White Paper. https://bitfury.com/content/downloads/whitepaper_flare_an_approach_to_routing_in_lightning_network_7_7_2016.pdf (2016)

[24] Sivaraman V., Venkatakrishnan S. B., Alizadeh M., Fanti G. C., Viswanath P.: Routing Cryptocurrency with the Spider Network. CoRR: vol. abs/1809.05088: URL http://arxiv.org/abs/1809.05088 (2018)

[25] Kiayias A., Litos O. S. T.: A Composable Security Treatment of the Lightning Network. https://eprint.iacr.org/2019/778 (2019)

[26] Canetti R.: Universally Composable Security: A New Paradigm for Cryptographic Protocols. In 42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA: pp. 136–145: doi:10.1109/SFCS.2001.959888: URL https://eprint.iacr.org/2000/067.pdf (2001)

[27] Canetti R., Dodis Y., Pass R., Walfish S.: Universally Composable Security with Global Setup. In Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007, Proceedings: pp. 61–85: doi:10.1007/978-3-540-70936-7_4: URL https://doi.org/10.1007/978-3-540-70936-7_4 (2007)

[28] Shamir A.: Identity-Based Cryptosystems and Signature Schemes. In Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings: pp. 47–53: doi:10.1007/3-540-39568-7_5: URL https://doi.org/10.1007/3-540-39568-7_5 (1984)

[29] Paterson K. G., Schuldt J. C. N.: Efficient Identity-Based Signatures Secure in the Standard Model. In Information Security and Privacy, 11th Australasian Conference, ACISP 2006, Melbourne, Australia, July 3-5, 2006, Proceedings: pp. 207–222: doi:10.1007/11780656_18: URL https://doi.org/10.1007/11780656_18 (2006)

[30] Malavolta G., Moreno-Sanchez P., Schneidewind C., Kate A., Maffei M.: Anonymous Multi-Hop Locks for Blockchain Scalability and Interoperability. In 26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019 (2019)

[31] Danezis G., Goldberg I.: Sphinx: A compact and provably secure mix format. In Security and Privacy, 2009 30th IEEE Symposium on: pp. 269–282: IEEE (2009)

[32] Katz J., Lindell Y.: Introduction to Modern Cryptography, Second Edition. CRC Press: ISBN 9781466570269 (2014)

[33] Bellare M., Sandhu R. S.: The Security of Practical Two-Party RSA Signature Schemes. IACR Cryptology ePrint Archive: vol. 2001, p. 60: URL http://eprint.iacr.org/2001/060 (2001)

[34] Boyd C.: Digital Multisignatures. Cryptography and Coding: pp. 241–246: URL https://ci.nii.ac.jp/naid/10013157942/en/ (1986)

[35] Ganesan R.: Yaksha: augmenting Kerberos with public key cryptography. In 1995 Symposium on Network and Distributed System Security, (S)NDSS '95, San Diego, California, USA, February 16-17, 1995: pp. 132–143: doi:10.1109/NDSS.1995.390639: URL https://doi.org/10.1109/NDSS.1995.390639 (1995)

[36] MacKenzie P. D., Reiter M. K.: Two-Party Generation of DSA Signatures. In Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings: pp. 137–154: doi:10.1007/3-540-44647-8_8: URL https://doi.org/10.1007/3-540-44647-8_8 (2001)

[37] Ganesan R., Yacobi Y.: A secure joint signature and key exchange system. Bellcore TM: vol. 24531 (1994)

[38] McCarthy J.: Computer Programs for Checking Mathematical Proofs. Proceedings of the Fifth Symposium in Pure Mathematics of the American Mathematical Society: pp. 219–227: american Mathematical Society (1961)

[39] Canetti R., Stoughton A., Varia M.: EasyUC: Using EasyCrypt to Mechanize Proofs of Universally Composable Security. In 32nd IEEE Computer Security Foundations Symposium, CSF 2019, Hoboken, NJ, USA, June 25-28, 2019: pp. 167–183: IEEE: ISBN 978-1-7281-1407-1: doi:10.1109/CSF.2019.00019: URL https://doi.org/10.1109/CSF.2019.00019 (2019)

[40] Thuraisingham B. M., Evans D., Malkin T., Xu D. (editors): Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017: ACM: ISBN 978-1-4503-4946-8: doi:10.1145/3133956: URL https://doi.org/10.1145/3133956 (2017)