Splitting Payments Locally While Routing Interdimensionally

Lisa Eckey
lisa.eckey@tu-darmstadt.de
TU Darmstadt

Sebastian Faust
sebastian.faust@tu-darmstadt.de
TU Darmstadt

Kristina Hostáková
kristina.hostakova@tu-darmstadt.de
TU Darmstadt

Stefanie Roos
s.roos@tudelft.nl
TU Delft

ABSTRACT

Payment Channel Networks (PCNs) enable fast, scalable, and cheap payments by moving transactions off-chain, thereby overcoming debilitating drawbacks of blockchains. However, current algorithms exhibit frequent payment failures when a payment is routed via multiple intermediaries. One of the key challenges for designing PCNs is to drastically reduce this failure rate. In this paper, we design a Bitcoin-compatible protocol that allows intermediaries to split payments on the path. Intermediaries can thus easily adapt the routing to the local conditions, which the sender is unaware of. Our protocol provides both termination and atomicity of payments, and provably guarantees that no participant loses funds even in the presence of malicious parties. An extended version of our protocol further provides unlinkability between two partial split payments belonging to the same transaction, which – as we argue – is important to guarantee the success of split payments. Besides formally modeling and proving the security of our construction, we conducted an in-depth simulation-based evaluation of various routing algorithms and splitting methods. Concretely, we present Interdimensional SpeedyMurmurs, a modification of the SpeedyMurmurs protocol that increases the flexibility of the route choice combined with splitting. Even in the absence of splitting, Interdimensional SpeedyMurmurs increases the success ratio of transactions drastically in comparison to a Lightning-style protocol, by up to 1/3. Splitting further increases the probability of success, e.g., from about 84% to 97% in one scenario.

KEYWORDS

payment channels, payment networks, Bitcoin, routing

1 INTRODUCTION

The most pressing obstacle to mass adoption of cryptocurrencies like Bitcoin [18] and Ethereum [33] is their limited scalability. The limited transaction throughput of public blockchains lead to long delays and high fees [5]. Payment channels [17, 24] are a powerful tool to mitigate these scalability challenges of blockchains. They allow two users to send funds to each other off-chain and only require interaction with the blockchain during creation and closure of the channels. Moreover, the underlying blockchain can be used to resolve disputes, e.g., when parties disagree on the latest balance of the channel. Multiple channels can be connected and form a payment channel network (PCN), where payments can be routed via (several) intermediaries to the receiver [1, 6, 7, 24]. The largest and most widely adopted PCN is the Lightning network with more than 12,000 nodes and over 35,000 open channels[1].

In order to successfully route a payment of v coins through a large PCN, it is crucial to find one or several routes with sufficient capacities on every link between the sender and the receiver. While there exists a multitude of proposed single or multi-path routing protocols for PCNs [9, 13, 14, 27, 30, 31, 34], most of them rely on source routing, where the sender selects the route of the payment.

The main technique for routing payments is to use conditional payments for each involved channel. In Lightning, which uses only one path, the atomicity of these payments follows from the use of Hash-Time-Locked-Contracts (HTLC) [24]. On a high level, in an HTLC, the receiver of a payment sends a hash $h_R := \mathcal{H}(x_R)$ of a random value x_R to the sender. The sender then creates a conditional payment of v coins with the first intermediary I_1 on the path. The conditional payment can be redeemed by presenting a preimage of h_R. Once the conditional payment is setup between the sender and I_1, the intermediary I_1 initiate a conditional payment with the next intermediary on the path and so on, until the receiver is reached. The receiver then reveals the preimage x_R, which settles all conditional payments on the path. The above construction has been extended by the Atomic Multi-Path (AMP) payment protocol [19]. In AMP, the sender of a v coin payment can split the payment over multiple paths, such that an amount of $v_i \leq v$ coins is routed along the i-th path. On a technical level, AMP also relies on HTLCs, but the preimage is split by the sender into preimage shares. Only if all shares arrive at the receiver, the payment is completed.

We argue that leaving it to the sender to determine the paths, or at least the number of paths, is a key reason for the high failure rate of PCNs. While the sender knows the channels in the network and their initial funding, he has no knowledge about the current channel capacities, except for the channels that he is involved in. Therefore, a sender can only guess which routes will be successful. Especially when the transaction value is high, it is likely that channel capacities on the path are insufficient and hence the payment fails.

In this work, we follow a more flexible and adaptable approach similar to SpeedyMurmurs [27], in which intermediaries on the path can freely choose the next hop of the payment based on their local view on channel capacities. In contrast to AMP, where the splitting of the payment is done by the sender, we design a novel protocol that allows splitting payments by intermediaries on the fly. This option enables the intermediaries to route an incoming payment of v coins, even if he does not have a single outgoing channel with sufficient capacity, thus increasing the success probability.

[1] https://1ml.com/statistics

If the routing of all parts of the payment succeeds, the receiver recombines all partial payments and obtains the v coins.

We formally define suitable security properties and prove that our protocol satisfies these properties. Concretely, our protocol guarantees *balance neutrality*, which says that no honest intermediary or receiver can lose coins by executing the protocol and *bounded loss for the sender* ensuring that a sender does not lose more coins than what he wanted to pay. The *atomicity* property of our protocol guarantees that if an honest sender loses v coins, then he obtains a valid payment receipt over the correct transfer of v coins. At the same time, an honest receiver will never issue a correct receipt unless he gets v coins in exchange. Finally, the protocol *terminates* in finitely many rounds and achieves *correctness*, which ensures that the payment will succeed if all parties in the protocol are honest and all channels in the network have sufficient capacity.

Furthermore, we extend our basic protocol to provide *unlinkability* between split payments. This additional security property guarantees that even if intermediaries collude, they cannot link parts of the same split payment. Unlinkability prevents (rational) intermediaries from censoring split payments, which – as we argue – they might do in order to optimize the amount of earned fees from payment forwarding. Our extended construction uses a preimage-resistant hash function that is additively homomorphic and an additively homomorphic encryption scheme. It can be instantiated using exponentiation in a group for which Dlog is hard and Paillier's encryption scheme. We emphasize that both the basic construction and the extended protocol can be integrated into Bitcoin.

Our protocol description follows a modular approach that allows to instantiate the protocol with a variety of routing algorithms. To this end, the sender (and each intermediary) can choose between multiple options for (i) determining a suitable set of candidates to route a payment over, and (ii) for appropriately splitting the payment into multiple subpayments that are routed via some subset of these candidates. The candidate set is selected via the Closer algorithms, for which we design two options. The first is similar to Lightning's routing protocol, with candidates being selected such that the routing takes a shortest path. As a second algorithm, we design *Interdimensional SpeedyMurmurs*, a variant of the tree-based routing protocol SpeedyMurmurs [27], that combines the information from multiple spanning trees. Based on this information, Interdimensional SpeedyMurmurs offers a high number of paths towards the receiver and hence a high flexibility in choosing the candidate set.

Given the candidate set, the splitting algorithm Split selects a subset of candidates over which the splitting is carried out. We compare three splitting variants: (a) no splitting, (b) splitting over the candidates with the shortest paths to be the receiver, and (c) splitting only if the payment would fail otherwise.

To compare all possible routing combinations, we measured the success ratio and communication overhead of the algorithms in a simulation based on data from a real-world Lightning snapshot. Our simulation considers a wide range of scenarios with regard to channel capacities, transactions, network dynamics, and routing algorithms. For all considered scenarios, we find that Interdimensional SpeedyMurmur's flexibility in routing choice drastically increases the success ratio at an inconsequentially increased communication overhead. Strategic splitting variants increase the success ratio, in particular when we apply (b) splitting over the candidates with the shortest paths to be the receiver. More precisely, *Interdimensional SpeedyMurmurs* with splitting reaches a success ratio of over 95% whereas Lightning-style algorithms only succeed in around 55% of cases for larger payments.

In summary, we design a novel modular payment channel routing protocol that supports local splitting, prove its security, and confirm its superior performance with an in-depth evaluation.

2 NOTATION AND SECURITY MODEL

2.1 Preliminaries

Notation. We denote by $\mathbb{N}$, $\mathbb{Z}$ and $\mathbb{R}$ the set of all natural, integer and real numbers, respectively. We denote by $x \leftarrow_{\$} \mathcal{X}$ the uniform sampling of the variable x from the set $\mathcal{X}$. Throughout this paper, n denotes the security parameter and all our algorithms run in polynomial time in n. A function $\mathsf{negl}\colon \mathbb{N} \to \mathbb{R}$ is *negligible in* n if for every $k \in \mathbb{N}$, there exists $n_0 \in \mathbb{N}$ s.t. for every $n \geq n_0$, it holds that $|\nu(n)| \leq 1/n^k$. We implicitly assume that functions are negligible in the security parameter. By writing $x \leftarrow \mathsf{A}(y)$, we mean that a *probabilistic polynomial time* algorithm A (or ppt for short) on input y, outputs x. If A is a *deterministic*, we use the notation $x := \mathsf{A}(y)$. In our protocol descriptions, we use the following arrow notation. Instead of the instruction: "Send a message m to party P", we write "$m \hookrightarrow P$". Similarly, instead of the instruction "Upon receiving a message m from party P", we write " Upon $m \hookleftarrow P$".

Graphs. A directed graph $\mathcal{G}$ is a tuple $(\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is a non-empty finite set of *nodes* and $\mathcal{E} \subseteq \{(U, V) \mid U, V \in \mathcal{V}\}$ is a set of *edges*. If $(U, V) \in \mathcal{E}$, U and V are *neighbors*. A *path* between two nodes V_1, V_{n+1} is a finite sequence of edges $(e_1, \ldots, e_n)$ for which there is a sequence of vertices $(V_1, \ldots, V_{n+1})$ such that $e_i = (V_i, V_{i+1})$, for $i \in [1, n]$ and $V_i \neq V_j$, for $i \neq j$. The number of edges in the path is called *length* of the path. In this paper, we assume that all graphs are connected, i.e., there is a path between all distinct $V, U \in \mathcal{V}$. We define a *hop-distance function of* $\mathcal{G}$ as $d_{\mathcal{G}}\colon \mathcal{V} \times \mathcal{V} \to \mathbb{N}_0$ that on input two nodes $V, U \in \mathcal{V}$ outputs the length of a shortest path between V and U. If $U = V$, then $d_{\mathcal{G}}(U, V) = 0$.

A graph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ is a *subgraph* of $\mathcal{G}$ if $\mathcal{V}' \subseteq \mathcal{V}$ and $\mathcal{E}' \subseteq \mathcal{E}$. A *spanning tree ST* of a connected graph $\mathcal{G}$ is a subgraph $(\mathcal{V}, \mathcal{E}')$ of $\mathcal{G}$ that is a *tree*, i.e., a graph such that there exists exactly one path between every pair of nodes. We consider *rooted trees*, i.e., trees with one designated root node $root \in \mathcal{V}$. In a rooted spanning tree, for each node $V \in \mathcal{V} \setminus \{root\}$, the neighbor with a shorter path to the root is V's *parent*. Neighbors that are not the parent of a node V are V's *children*. We call a rooted spanning tree ST of a graph $\mathcal{G}$ a *Breadth-First Search (BFS) spanning tree* if the path between the root and each node in the tree is a shortest path in $\mathcal{G}$.

Cryptographic primitives. We briefly recall the basic cryptographic primitives that are used in this work. The formal definitions can be found in Appendix A.

A public key encryption scheme Ψ with a message space $\mathbb{M}$ and ciphertext space $\mathbb{C}$ is a triple of ppt algorithms (Gen, Enc, Dec) s.t. for every message $m \in \mathbb{M}$ it holds that $\Pr[\mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(m)) =$

$m \mid (pk, sk) \leftarrow \mathsf{Gen}(1^n)] = 1$. In this work, we use encryption schemes that are *indistinguishabile under chosen paintext attack* (IND-CPA secure for short), guaranteeing, on a high level, that a ppt adversary is not able to distinguish the encryption of two messages of his choice. We say that Ψ is *additively homomorphic* if for every $x, y \in \mathbb{M}$ and public key pk, $\mathsf{Enc}_{pk}(x) +_{\mathbb{C}} \mathsf{Enc}_{pk}(y) \equiv \mathsf{Enc}_{pk}(x +_{\mathbb{M}} y)$, where $\equiv$ denotes equality of probability distributions.

A digital signature scheme Σ is a triple of ppt algorithms (Gen, Sign, Vrfy), where $\Pr[\mathsf{Vrfy}_{pk}(\mathsf{Sign}_{sk}(m)) \mid (pk, sk) \leftarrow \mathsf{Gen}(1^n)] = 1$ holds for every message m. In this work, we use signature schemes that are *existentially unforgeable under chosen message attack* (EUF-CMA secure for short), guaranteeing, on a high level, that a ppt adversary, learning polynomially many signatures of messages of his choice, cannot produce a valid signature for a new message.

A function $\mathcal{H} : \mathbb{P} \to \mathbb{H}$ is called a *preimage-resistant hash function* if it is polynomial-time computable and for every ppt adversary $\mathcal{A}$, given $y = \mathcal{H}(x)$, for a randomly sampled $x \in \mathbb{P}$, the probability that the adversary $\mathcal{A}$ outputs $x' \in \mathbb{P}$ s.t. $\mathcal{H}(x') = y$ is negligible. We say that $\mathcal{H}$ is *additively homomorphic* if for every $x, y \in \mathbb{P}$, it holds that $\mathcal{H}(x +_{\mathbb{P}} y) = \mathcal{H}(x) +_{\mathbb{H}} \mathcal{H}(y)$.

In order to simplify the exposition, we often drop the subscript in $+_{\mathbb{M}}, +_{\mathbb{C}}, +_{\mathbb{P}}, +_{\mathbb{H}}, 0_{\mathbb{M}}, 0_{\mathbb{C}}, 0_{\mathbb{P}}, 0_{\mathbb{H}}$ when the set is clear.

2.2 Security model

Modeling payment channel networks. We model a payment channel network (PCN) as a connected directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ together with a capacity function $C : \mathcal{E} \to \mathbb{R}^+$. The set of vertices $\mathcal{V}$ represents the parties involved in the PCN, the set of edges $\mathcal{E}$ represents payment channels open between parties, and the capacity function assigns coins to parties in a channel. To simplify the notation in our formalization, we represent a payment channel as two uni-directional channels and require that $(P, Q) \in \mathcal{E} \Leftrightarrow (Q, P) \in \mathcal{E}$. Hence, the value $C(P, Q)$ represents the amount of coins that party P has in the channel between P and Q and $C(Q, P)$ represents the amount of coins that Q has in that channel. Let us emphasize that this is equivalent to modeling a PCN as an undirected graph with a capacity function that on input an edge $\{P, Q\}$ and a party $R \in \{P, Q\}$ outputs the amount of coins party R has in the channel. We define $\mathcal{E}_P := \{e \in \mathcal{E} \mid \exists Q \in \mathcal{V} \text{ s.t. } e = (P, Q)\}$ as the set of all channels in which a party $P \in \mathcal{V}$ has locked coins and use $C_P := C\big|_{\mathcal{E}_P}$ to denote the restriction of the capacity function C to the set $\mathcal{E}_P$. We note that $\mathcal{E} = \dot{\bigcup}_{P \in \mathcal{V}} \mathcal{E}_P$, where $\dot{\bigcup}$ denotes the disjoint union of sets.

Recall that our goal is to design a protocol that allows parties to securely route payments through a PCN. In order to design such a protocol, we do not need to fix one concrete PCN implementation, e.g., the Lightning Network. In fact, we aim for our protocol to apply to any PCN in which parties can perform conditional payments and payment routing. To this end, we abstractly specify the functionality and input/output behavior of a PCN and allow parties in our protocol to interact with such PCN in a black-box way. Let us stress that abstracting from payment channel mechanics does not only generalize our work but also significantly simplifies the protocol descriptions and game-based security definitions. Our abstraction is formally described below, possible instantiations are discussed in Appendix B.

We model the functionality of payment channels using an *ideal functionality* $\mathcal{F}(\mathcal{G}, C_0, \Delta)$, parameterized by a connected directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $(P, Q) \in \mathcal{E} \Leftrightarrow (Q, P) \in \mathcal{E}$, and the initial capacity function $C_0 : \mathcal{E} \to \mathbb{R}^+$. The set of vertices $\mathcal{V}$ defines the parties from which the functionality can receive messages. Furthermore, the functionality has a timing parameter Δ representing the upper bound on the blockchain delay. Let us explain the functionality of PCN on a high level first and thereafter provide its formal description. Every party $P \in \mathcal{V}$ can instruct the functionality to perform a payment of v from P to Q by sending a message "pay". If P has a sufficiently funded channel with Q, the functionality subtracts v coins from (P, Q) and adds v coins to (Q, P). We assume that all such payment takes 1 round.[2]

In addition to standard payments, the functionality supports conditional payments. Such a payment can be initiated by a party $P \in \mathcal{V}$ via the message "cPay". In addition to specifying the channel (P, Q) and amount of coins v being conditionally transferred to Q, party P needs to define the *condition* $\varphi : \{0,1\}^* \to \{0,1\}$ and the *time-lock* $T \in \mathbb{N}$ of the payment. Additionally, P has the option of attaching some auxiliary information *info* $\in \{0,1\}^*$. If the channel is sufficiently funded, the functionality subtracts v from the channel (P, Q) and informs Q about the conditional payment. If party Q submits, via the message "cPay–unlock", a witness w s.t. $\varphi(w) = 1$, v coins are added to the channel (Q, P). After the round specified by the time-lock T, party P can request a refund via the message "cPay–refund" in which case the functionality adds v coins back to the channel (P, Q). In order to model the fact that operations triggered by the unlock and refund instructions might require blockchain interaction, the execution of those instruction might be delayed by at most Δ rounds.

The state of the functionality consists of a capacity function $C : \mathcal{E} \to \mathbb{R}^+$ keeping track of balances in the network (initially set to C_0) and a function $\Theta : \{0,1\}^* \to \{0,1\}^*$ keeping track of conditional payments currently being executed in the network. This means that on input a payment identifier $pid \in \{0,1\}^*$, the function returns either $\perp$, signaling that no payment with this identifier is currently open, or information about the conditional payment in the form of the tuple (e, v, φ, T). Here $e \in \mathcal{E}$ is the edge on which the payment takes place, $v \in \mathbb{R}^+$ denotes the amount of coins being transferred, $\varphi : \{0,1\}^* \to \{0,1\}$ is the condition of the payment and T the time-lock. The function Θ is initialized such that it outputs $\perp$ for any input.

Payment channel functionality $\mathcal{F}(\mathcal{G}, C_0, \Delta)$

The initial state is set to $C := C_0$ and $\Theta(pid) := \perp$ for all $pid \in \{0,1\}^*$.

- Upon receiving (pay, e, v) $\hookleftarrow P$, where $e = (P, Q) \in \mathcal{E}$ and $C(e) \geq v$, define $e' := (Q, P)$. In the next round, set $C(e) := C(e) - v$ and $C(e') := C(e') + v$ and send (paid, e', v) $\hookrightarrow Q$.
- Upon receiving (cPay, *pid*, e, v, φ, T, *info*) $\hookleftarrow P$, where $e = (P, Q) \in \mathcal{E}$, $C(e) \geq v$ and $\Theta(pid) = \perp$, wait for one round to set $C(e) :=$

[2] In many payment channel constructions, payments require more than 1 round of off-chain communication between the two channel users. Hence, it would be more accurate to replace the constant 1 by a parameter δ (we stress that the value of δ would always be a constant w.r.t. the blockchain delay Δ). We choose to implicitly set $\delta = 1$ to simplify the exposition.

$C(e) - v$, store $\Theta(pid) := (e, v, \varphi, T)$ and send (cPaid, *pid*, e, v, φ, T, *info*) $\hookrightarrow \mathcal{Q}$.

- Upon receiving (cPay–unlock, *pid*, w) $\hookleftarrow \mathcal{Q}$, wait for at most $\Delta + 1$ rounds. If $((P, Q), v, \varphi, T) := \Theta(pid) \neq \perp$ and $\varphi(w) = 1$, then set $C(e') := C(e') + v$, for $e' = (Q, P)$, set $\Theta(pid) = \perp$ and send (cPay–unlocked, *pid*, w) $\hookrightarrow P$.
- Upon receiving (cPay–refund, *pid*) $\hookleftarrow P$, wait for at most $\Delta + 1$ rounds. If $(e, v, \varphi, T) := \Theta(pid) \neq \perp$ and the current round number is larger than T, then set $C(e) := C(e) + v$, $\Theta(pid) = \perp$ and send (cPay–refunded, *pid*) $\hookrightarrow P$.

Protocol execution. We consider a protocol π whose execution is parameterized by a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ defines the set of parties running the protocol and $\mathcal{E}$ defines the existing payment channels that exist between parties from the set $\mathcal{V}$; an initial capacity function C defines the amount of coins in each payment channel; a party $S \in \mathcal{V}$ being the sender of a payment of $v \in \mathbb{R}^+$ coins to a receiver $R \in \mathcal{V}$. The protocol is executed in presence of a ppt adversary $\mathcal{A}$ that can corrupt an arbitrary number of parties from $\mathcal{V}$ at the beginning of the protocol (i.e., we consider so-called static corruption). The adversary takes full control over the actions of every corrupt party (i.e., the messages it sends to other parties and to the functionality).

The protocol execution begins with a setup phase during which (1) the ideal functionality $\mathcal{F}(\mathcal{G}, C, \Delta)$, representing the PCN functionality, is initialized by the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and the initial capacity function C; (2) every party $P \in \mathcal{V}$ gets as input the graph $\mathcal{G}$ and the capacity of its channels, i.e., the partial function C_P; moreover, the sender S and the receiver R additionally get as input the tuple (S, R, v); (3) the adversary $\mathcal{A}$, learning $\mathcal{G}$, decides which parties from the set $\mathcal{V}$ it corrupts and sets the inputs of all those parties. We denote by $\mathsf{Honest} \subseteq \mathcal{V}$ the set of all parties that were not corrupted by the adversary.

After the setup phase, parties can arbitrarily interact with each other and the ideal functionality $\mathcal{F}(\mathcal{G}, C, \Delta)$. The protocol terminates once all honest parties produce an *output* $m \in \{0,1\}^* \cup \{\top\}$. The special symbol $\top$ signals that a party wants to terminate the protocol without producing any particular output. Looking ahead, this is the case for all parties in our protocol except for the sender S who outputs a receipt when the payment is successful. The set of honest parties, the output of the sender and the final state of the functionality $\mathcal{F}(\mathcal{G}, C, \Delta)$ form the output of the protocol.[3] We denote this output as $\mathsf{EXEC}^{\mathcal{F}}_{\pi,\mathcal{A}}(\mathcal{G}, C, \Delta, S, R, v)$.

Finally, let us briefly comment on fees in PCNs. Fees incentivize intermediaries to forward payments and thus play an important role for the PCN ecosystem. While we do not explicitly add fees into our protocol description, it is easy to extend our construction to integrate them. For instance, in the Lightning network fees for intermediaries are calculated as the difference between the incoming and outgoing payment. Hence, they are easily realized by adjusting the conditional payments accordingly.

We assume that parties are peer-to-peer connected with authenticated communication channels with guaranteed delivery of 1 round. This guarantee indicates that if a party P sends a message to party Q in round t, then Q receives this message in round $t + 1$ and is certain that this message was sent by P. Hence, the adversary cannot drop or introduce any message in the channel between P and Q. However, we assume that he can see the content of messages and reorder messages that were sent in the same round. For simplicity, we assume that any local computation takes zero rounds. The interaction between the parties/adversary and the functionality takes zero rounds as well.

[3] If the sender is malicious and does not produce any output before the protocol terminates, it is automatically set to $\top$.

2.3 Security definitions

We now define the security properties that we require our protocol to satisfy. Before we state the properties formally, let us give a high-level explanation of each of them. Firstly, we want the protocol to *terminate*, meaning that all honest parties produce an output in finitely many rounds. Secondly, we want the protocol to guarantee that no party loses money. The second requirement is formalized by two properties: *balance neutrality* that says that no honest intermediary or honest receiver loses any coins, and *bounded loss for the sender* that says the monetary loss of an honest sender is never more than the v coins he wanted to send. Moreover, we want the protocol to guarantee payment *atomicity*. Briefly, this property guarantees to an honest sender that if he loses any coins, then he holds a *receipt* signed by the receiver that he paid v coins; and it grantees to an honest receiver that if a sender holds a valid receipt for v coins, then the receiver earned at least v coins. Finally, in order to exclude a trivial protocol in which payments always fail, we require the protocol to satisfy *correctness*, meaning that if all parties are honest and capacity of all channels is at least v, then the payment completes successfully.

In order to formalize the properties above, we need to precisely describe what *valid receipt* means. To this end, we define a validation function $\mathsf{Validate}\colon \mathcal{V} \times \mathcal{V} \times \mathbb{R}^+ \times \{0,1\}^* \to \{0,1\}$ that takes as input a sender S, a receiver R, an amount v and a receipt $\mathsf{rec} \in \{0,1\}^*$ and outputs a 0/1 to signal the validity of the receipt. Moreover, for every graph $\mathcal{G}$, we define a family of functions $\{\mathsf{net}_{C,C'}\}_{C,C'}$, where C and C' are two capacity functions of $\mathcal{G}$ and the function $\mathsf{net}_{c,c'}\colon \mathcal{V} \to \mathbb{R}$ is defined as follows: $\mathsf{net}_{C,C'}(P) := \sum_{W \in \mathcal{V}:(P,W)\in\mathcal{E}} C'(P,W) - C(P,W)$. In other words, the value of $\mathsf{net}_{C,C'}(P)$ represents the difference between the amount of coins P owns according to the capacity function C compared to the capacity function C'.

Definition 2.1 (Secure payment protocol). We say that a protocol π run among a set of parties $\mathcal{V}$ is a *secure payment protocol* with respect to a validation function Validate if for every connected directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $(P, Q) \in \mathcal{E} \Leftrightarrow (Q, P) \in \mathcal{E}$, every capacity function $C\colon \mathcal{E} \to \mathbb{R}^+$, every $S, R \in \mathcal{V}$, s.t. $S \neq R$, every $v \in \mathbb{R}^+$, every $\Delta \in \mathbb{N}$ and every ppt adversary $\mathcal{A}$, the protocol terminates in finitely many rounds with $(\mathsf{Honest}, \mathsf{rec}, C') \leftarrow \mathsf{EXEC}^{\mathcal{F}}_{\pi,\mathcal{A}}(\mathcal{G}, C, \Delta, S, R, v)$ that satisfies the following properties with overwhelming probability.

Balance neutrality: $\forall P \in \mathcal{V}\setminus\{S\}\colon P \in \mathsf{Honest} \Rightarrow \mathsf{net}_{C,C'}(P) \geq 0$.

Bounded loss for sender: $S \in \mathsf{Honest} \Rightarrow \mathsf{net}_{C,C'}(S) \geq -v$,

Atomicity: It holds that

(i) $S \in \mathsf{Honest} \wedge \mathsf{net}_{C,C'}(S) < 0 \Rightarrow \mathsf{Validate}(S, R, v, \mathsf{rec}) = 1$,

(ii) $R \in \mathsf{Honest} \wedge \mathsf{Validate}(S, R, v, \mathsf{rec}) = 1 \Rightarrow \mathsf{net}_{C,C'}(R) \geq v$.

Correctness: If Honest $= \mathcal{V}$ and for every $e \in \mathcal{E}$ $C(e) \geq v$, then $\text{net}_{C,C'}(S) = -v$, $\text{net}_{C,C'}(R) = v$ and $\text{net}_{C,C'}(P) = 0$ for all $P \in \mathcal{V} \setminus \{S, R\}$.

We stress that our notion of a secure payment protocol captures routing of one payment between sender S and receiver R only. In other words, our security definition does not consider multiple parallel executions of a payment protocol. We leave the extension of our security model to the concurrent setting as an interesting direction for future research. Note that while we do not consider parallel executions of the protocol, corrupt parties might still perform arbitrary payments during the single protocol execution.

3 PAYMENT PROTOCOL

The idea of our protocol is fairly simple. A receiver first samples a random preimage x_R and sends its hash $h_R := \mathcal{H}(x_R)$ to the sender. The sender uses this hash value to initiate a conditional transfer of v coins to the receiver. In contrast to many other PCN protocols, the sender does not specify the entire path from the sender to the receiver, which the payment has to take. In fact, the sender only chooses the first hop of the payment and attaches routing information (such as the identity of the receiver) to the conditional payment. Moreover, the sender can decide to split the payment of v coins into multiple smaller payments and send each of them via a different first hop. In our simple protocol, we assume that the same hash value is used for all conditional payments.

Once an intermediary receives a conditional payment with attached routing information, the intermediary can freely decide how to split and route the payment based on his local view of the current capacities of his channels. If the intermediary receives multiple conditional payments with the same condition and the same routing information, the partial payments can also be combined into one (and then potentially split again).

A receiver waits until he receives sufficiently many conditional payments locked by the hash value h_R such that their values add up to at least v. Then he uses the preimage x_R to unlock all the payments and receive the promised v coins.

Routing. The main question that we study in this paper is how the sender and the intermediaries decide on the local routing. Namely, to which neighbors they should route the payment and how many coins should they send through each link. We identify several sensible options in Section 4 and evaluate and compare their performance in Section 5. For the purpose of the formal protocol description, we assume an algorithm $\text{Route}_{\mathcal{G}}$ that takes as input the amount of coins v to be routed, the identifier of the party P performing the routing, P's local view on the capacity function C_P, routing information consisting of the identifier of the receiver R, and the set *excl* containing all nodes that were already visited on the payment path between the sender and the party P. The algorithm outputs either $\perp$ (signaling that routing failed), or k edge/value pairs $\{(e_j, v_j)\}_{j\in[k]} \subseteq (\mathcal{E}_P \times \mathbb{R}^+)^k$ satisfying the following three conditions: (i) $C_P(e_j) \geq v_j$ for every $j \in [k]$, (ii) $e_j = (P, Q_j)$ s.t. $Q_j \notin$ *excl* for every $j \in [k]$ and (iii) $\sum_{j\in[k]} v_j = v$. In other words, the algorithm decides how to split the v coins among P's neighbors and excludes all neighbors that are in the set *excl*.

Providing a receipt. In order to turn the above simple protocol into a secure payment protocol satisfying atomicity (as defined in Def. 2.1), we need to discuss when and how the receiver provides a receipt to the sender. Obviously, the receiver does not want to give a receipt before he is sure that v coins are routed to him. On the other hand, the sender does not want to start the conditional transfer of v coins before he has a guarantee that the receiver provides the receipt if the last part of the conditional transfer completes successfully. Hence, we need a method that allows the receiver to provide the receipt *conditionally* such that (a) the sender can verify that the conditional receipt can be turned into a valid receipt if the preimage for h_R is known and (b) the receiver has the guarantee that the sender cannot generate a valid receipt without knowing the preimage of h_R.

Validate(S, R, v, rec)

Parse $(h, \sigma, x) :=$ rec

return $\text{Vrfy}_{pk_R}((S, R, v, h), \sigma) \wedge (\mathcal{H}(x) = h)$

Figure 1: Receipt validation function.

To this end, the receiver signs a statement saying that he received v coins from the sender if a preimage of the hash value h_R is attached. He sends this signature to the sender, together with the hash value h_R, at the beginning of the protocol. The sender can verify the receiver's signature and use the hash value h_R for the conditional payments. If at least one of them is unlocked, the sender can attach the revealed preimage x_R to the receiver's signature and output a valid receipt. The formal definition of the receipt validation function can be found in Figure 1.

Time-locks. An intermediary forwarding a conditional payment must decrease the time-lock in order to be sure that he never loses coins. More precisely, let T be the time-lock of the incoming payment and T' the time lock of the outgoing payment. The difference $|T - T'|$ must be such that if the outgoing payment completes, i.e., the intermediary loses coins but learns the witness x_R, he has enough time to submit this witness to the functionality $\mathcal{F}$ and unlock the incoming payment. The latest point when the intermediary can learn the witness x_R is in round $T' + (\Delta + 1)$ and submission of the witness takes at most $(\Delta + 1)$ rounds. Hence, the time-lock for the outgoing payment is set to $T' := T - 2 \cdot (\Delta + 1)$.

Ideally, the sender sets the time-lock of its conditional payments to $\text{now} + \ell \cdot (1 + 2 \cdot (\Delta + 1))$, where ℓ is the length of the payment path and now is the round in which the sender initiates the conditional payments. The factor $(1 + 2 \cdot (\Delta + 1))$ comes from the fact that it takes 1 round to set up a conditional payment and at most $2 \cdot (\Delta + 1)$ for an intermediary to unlock a conditional payment as discussed above. In contrast to source routing, computation of the ideal time-locks might be impossible for the sender in our protocol since he does not know the paths partial payments take. To this end, we instruct an honest sender to set the time-out with $\ell = |\mathcal{V}|$ since the longest possible path between two nodes in a graph is upper bounded by the number of nodes in the graph (recall that a path never visits the same node twice). This guarantees that payments never fail due to time-outs. Let us stress that once a concrete routing algorithm

is chosen and the graph topology is fixed, tighter upper bounds can be used to increase the efficiency of the protocol. To keep our formal protocol description generic and simple, we do not include those optimizations.

Termination. In order to prove that our protocol satisfies Def. 2.1, we need to define when honest parties terminate and what they output. An honest sender terminates and outputs $\top$ in case the receiver does not provide a valid signature σ on a tuple (S, R, v, h_R) in round $t_0 + 1$, where t_0 is first round of the protocol execution. Furthermore, the sender terminates if all conditional payment expire and get successfully refunded. If at least one of the conditional payments is unlocked, the honest sender learns the preimage x_R of the hash value h_R and hence can output a valid receipt, i.e., the hash value h_R, signature of the receiver σ and the preimage x_R.

Let us now discuss termination for the receiver. Since setting up a conditional payment via the PCN functionality takes at most 1 round, the receiver should receive all partial payments latest in the round $t_0 + |\mathcal{V}| + 1$. Hence, if the receiver does not receive conditional payments whose values add up to v by this round, he terminates, i.e., outputs $\top$, and does not unlock any payment. If v coins are promised by this round, the receiver unlocks all the payments and once he receives all the coins, he terminates, i.e., outputs $\top$.

It remains to define the termination of honest intermediaries. If a payment should be routed via an intermediary, it must happen before round $t_0 + |\mathcal{V}|$. Therefore, we instruct an honest intermediary to stop forwarding payments after this round, wait until all outgoing conditional payments are unlocked or refunded, unlock all forwarded incoming payments and terminate, i.e., output $\top$.

3.1 Extended protocol with unlinkability

Recall that the main purpose of our work is to increase the success ratio of large payments that are routed through a payment channel network by allowing parties to split large payments into multiple smaller payments on the fly. This argumentation quietly assumes that intermediaries do not censor payments that have been split.

However, rational intermediaries might prefer to forward payments that have not been split. Consider the following situation. An intermediary is asked to route two payments of the same value, the same time-lock, the same receiver and both payments offer the same fee for successful payment completion. One of the two payments has previously been split, the other one has not. Then the failure probability of the split payment is higher than for the monolithic one. More precisely, the partial payment routed by the intermediary has the same conditions as the monolithic payment and hence the same probability of reaching the receiver. Yet, for the split payment, the other partial payments also have to succeed, meaning that the probability for the complete payment to be successful is lower. A rational intermediary hence prefers to route the payment that has not been split if they only have the capacity for forwarding one of the two payments.

As intermediaries typically do not encounter the exact situation above, with two payments of the same value arriving at the same time, rational intermediaries might start dropping partial payments by default in order to have free collateral for monolithic payments. In doing so, they drop payments that might have been successful. Such behavior can easily negate the advantages of our approach.

Hence, in order to make our splitting approach effective, we need to make sure that intermediaries cannot distinguish between monolithic payments and payments that have been split. Unfortunately, in the simple protocol that we described earlier in this section, the hash-locks on all partial payment paths are the same, which makes it trivial for colluding intermediaries to see that they are routing parts of the same large payment. Censorship of payments that have been split is hence possible. Appendix C substantiates this claim by simulating the attack and finding that it indeed severely reduces the success ratio.

To overcome this issue, we present an extension to our protocol that remains secure but addresses the linkability issue caused by the identical hash-locks. Our approach is to design a splitting algorithm that produces k partial payments with hash values $(h_1, \ldots, h_k)$ satisfying the following.

(1) The vector $(h_1, \ldots, h_k)$ is computationally indistinguishable from a vector $(h'_1, \ldots, h'_k)$, where $h'_i := \mathcal{H}(x'_i)$ for a randomly chosen preimage x'_i.
(2) In order to learn the preimage x_R for the hash value h_R, the sender only needs to learn a preimage x_i for one of the hash values h_i; hence, the atomicity property is fulfilled.
(3) The receiver is able to compute a witness for all received partial payments; hence, correctness of the protocol is preserved.

In order to achieve all these properties simultaneously, we make use of a hash function that is additively homomorphic. For each partial payment $i \in [k]$, the sender first samples a random x_i, sets the hash-lock to $h_i := h_R + \mathcal{H}(x_i) = \mathcal{H}(x_R + x_i)$ and attaches $c_i \leftarrow \mathsf{Enc}(x_i)$ to the payment. Property (1) follows from the fact that the values x_i are independent and uniformly distributed, hence so are the values $x_R + x_i$. Moreover, the security of the encryption scheme guarantees that attaching c_i to the conditional payment does not affect the unlinkability. Property (2) is satisfied as well since upon learning a value x s.t. $\mathcal{H}(x) = h_i$, for some $i \in [k]$, the sender can compute a preimage of h_R as $x - x_i$ (this follows from the additive homomorphism of $\mathcal{H}$). Finally, correctness holds since the receiver can decrypt c_i, learn x_i and compute a preimage of h_i as $x := x_i + x_R$.

Assume now that an intermediary receives a conditional payment with a hash-lock h and attached ciphertext c, where $h = h_R + \mathcal{H}(x)$ and $c = \mathsf{Enc}_{pk_R}(x)$ for some x. The intermediary can split the payment in k parts by sampling $(x_1, \ldots, x_k)$ and computing $(h_1, \ldots, h_k)$ exactly as the sender did; namely, for every $i \in [k]$ he chooses random x_i and computes $h_i := h + \mathcal{H}(x_i) = \mathcal{H}(x_R + x + x_i)$. It remains to discuss how the intermediary reveals the value x_i to the receiver without breaking the unlinkability. To this end, we make use of an additively homomorphic encryption scheme which allows the intermediary to compute a ciphertext $c_i \leftarrow c + \mathsf{Enc}_{pk_R}(x_i) = \mathsf{Enc}_{pk_R}(x + x_i)$.

We would like to argue that the value $x_R + \mathsf{Dec}_{sk_R}(c_i) = x_R + x + x_i$, computed by the receiver, is a preimage of h_i, hence the correctness holds. The problem with this argumentation is that it assumes $x_R +_{\mathbb{M}} x +_{\mathbb{M}} x_i = x_R +_{\mathbb{P}} x +_{\mathbb{P}} x_i$, where $\mathbb{M}$ is the message space of the encryption scheme and $\mathbb{P}$ is the domain of $\mathcal{H}$. Unfortunately, we do not know how to instantiate the additively homomorphic encryption scheme and the additively homomorphic hash function such that this holds. Hence, we design the preimage recovery slightly

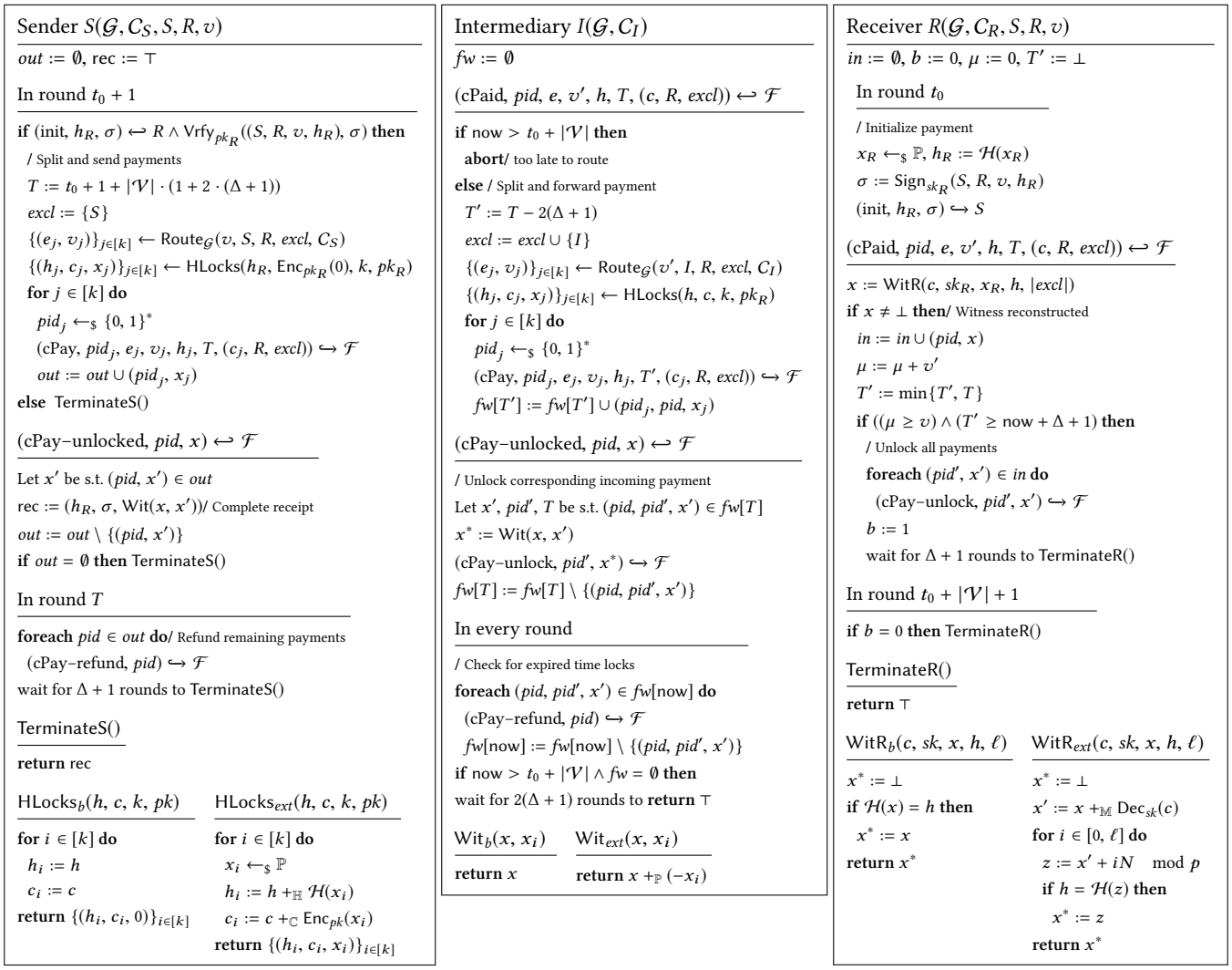


Figure 2: Generic description of the protocol initiated in round t_0. For the extended protocol, we assume $\mathbb{P} = \mathbb{Z}_p$ and $\mathbb{M} = \mathbb{Z}_N$.

differently. In our solution, we assume that $\mathbb{M} = \mathbb{Z}_N$ and $\mathbb{P} = \mathbb{Z}_p$ for $p < N = q \cdot q'$ and q, q', p coprime primes since this is the case for the encryption scheme of Paillier and hash function defined as exponentiation in a group where Dlog is hard (see Appendix B for discussion about instantiations). Under this assumption, we know that $((x_R +_{\mathbb{M}} x +_{\mathbb{M}} x_i) + j \cdot N) \mod p = x_R +_{\mathbb{P}} x +_{\mathbb{P}} x_i$, where j is upper bounded by the length ℓ of the payment path, i.e., the number of times we added values in $\mathbb{Z}_N$. Hence, the receiver can simply try to hash each of the $\ell \leq |\mathcal{V}|$ possible preimages and compare the result to the hash-lock h_i.

More details can be found in the next section where both our protocols are described formally. The instantiation of cryptographic primitives used in the protocols as well and their Bitcoin compatibility is discussed in Appendix B. We state and discuss the security of our schemes in Section 6, where also the formal definition of payment unlinkability can be found.

3.2 Formal protocol description

In Figure 2, we present the formal description of both protocols presented in this section; namely, the basic protocol that we denote Π_b(Route) and the extended protocol with unlinkable payment that we denote Π_{ext}(Route). Recall that protocols are parameterized by a routing algorithm Route whose purpose is to decide how to locally route a payment. Concrete instantiations of Route are discussed in Section 4.

Since both protocols are very similar, we decided to follow a modular approach and present the protocol description Π as a protocol calling multiple sub-routines. To this end, Π is, in addition to Route, parameterized by three algorithms that define the differences between the two protocol: HLocks, WitR, and Wit. The algorithm HLocks, run by the sender and intermediaries during the routing phase, takes as input a hash value h, ciphertext c, an integer

$k \in [n]$ and a public key pk, and outputs k tuples (h_i, c_i, x_i) consisting of hash values, ciphertext, and a preimage. The algorithm WitR, run by the receiver, takes as input a ciphertext c, a secret key sk, a preimage x, a hash value h and integer ℓ, and outputs a preimage x' such that $h = \mathcal{H}(x')$. Finally, the algorithm Wit, run by the intermediaries and the sender during the unlocking phase, takes as input two preimages x and x_i, and outputs another preimage x'. We formally define these algorithms for the basic and extended protocol in Figure 2. Hence, $\Pi_b(\mathsf{Route}) := \Pi(\mathsf{Route}, \mathsf{HLocks}_b, \mathsf{WitR}_b, \mathsf{Wit}_b)$ and $\Pi_{ext}(\mathsf{Route}) := \Pi(\mathsf{Route}, \mathsf{HLocks}_{ext}, \mathsf{WitR}_{ext}, \mathsf{Wit}_{ext})$.

In the formal description presented in Figure 2, t_0 denotes the first round of the protocol execution. We assume that each party maintains a set of unsettled payments. More precisely, the sender stores all outgoing conditional payments in a set *out*, the receiver stores all incoming payments in a set *in*, and every intermediary I maintains a set of all forwarded payment *fw*. Recall that the only functions that parties use for conditional payments are hash preimage verifications, i.e., functions $\mathsf{Hash}_h^{\mathcal{H}}$ that take as input a preimage x and output 1 if $\mathcal{H}(x) = h$ and 0 otherwise. For brevity, we replace the condition $\mathsf{Hash}_h^{\mathcal{H}}$ with the hash value h. Finally, for the sake of simplicity, our formal description excludes the option of partial payment recombination by the intermediaries. We stress that the description could be easily adjusted to capture this feature and further improve the communication complexity.

4 ROUTING ALGORITHMS

In this section, we consider several realizations of the routing algorithm $\mathsf{Route}_{\mathcal{G}}$, as defined in Section 3.

Internally, Route always consists of two algorithms: Closer and Split. In a nutshll, Closer determines a candidate set of potential next hops and Split splits the payment value over these candidates. There exist various realizations for both Closer and Split.

First, Closer takes the node $P \in \mathcal{V}$, the receiver $R \in \mathcal{V}$, and the capacity function C_P as input and outputs tuples consisting of an edge $e = (P, U) \in \mathcal{E}_P$ to a potential next user U, the capacity c of e, and a value indicating an algorithm-dependent closeness measure for U with regard to R. Afterwards, the algorithm removes potential edges to avoid loops. More concretely, if a returned edge is with a node that has previously been on the path, the edge is removed from the set of candidates. In practice, the set *excl* can be realized in an efficient and privacy-preserving manner through the use of a Bloom filter [11].

The second algorithm Split takes the set of candidate channels, their capacities, and closeness measures, and a payment value as input. It then splits the payment value over a subset of these payment channels.

The following subsections introduce two realizations of Closer and three realizations of Split, which can be combined arbitrarily. We describe the algorithms here, the formal protocol descriptions can be found in the Appendix H. In addition to these realizations, random splitting was considered and the results are in Appendix G.

4.1 Determining potential next hops (Closer)

Our first realization of Closer considers nodes that have a lower shortest path length to the receiver than the node calling the algorithm. The second realization considers a set of spanning trees, and every neighbor that is closer to the receiver in terms of at least one spanning tree distance is a potential next hop.

Hop Distance (*HOP*):

The hop distance gives the length of the shortest path between two nodes, i.e., the value of the function $d_{\mathcal{G}}$. As the graph is available, each node can compute the distance locally by applying a shortest path algorithm on the graph. Thus, Closer_{HOP} determines the payment channels to nodes that are closer to the receiver.

Without splitting, the hop distance results in similar paths as Lightning routing [24] when all nodes charge the same fees. Nodes select a shortest and hence cheapest path. However, instead of the sender deciding the path in advance, nodes locally select the next hop. As a consequence, the nodes making local routing decisions can take the balances of neighboring nodes into consideration, which are unknown to the sender. In this manner, they can avoid some routing failures that lead to the need for rerouting in Lightning. Rerouting in Lightning entails high latencies, as the sender has to wait for timelocks to expire. Thus, in terms of the success ratio for the first routing attempt, this algorithm is a version of Lightning that enables local decisions and hence can act as a baseline for splitting.

Interdimensional SpeedyMurmurs (*INTSM*): In this section, a novel realization of Closer is introduced. The novel realization is a modification of the atomic multi-path algorithm SpeedyMurmurs [27] that is more suitable for splitting.

SpeedyMurmurs establishes BFS spanning trees $ST_1, \ldots, ST_{\text{dim}}$ using a standard distributed spanning tree protocol. In practice, there are a number of distributed spanning tree algorithms that can also efficiently repair the spanning tree if the graph topology changes (e.g., [21]). In its original form, SpeedyMurmurs routes each partial payment using a different spanning tree. More precisely, for the i-th partial payment, the hop distance function of the i-th spanning tree, denoted by d_i, is used to determine the next hop. In this manner, SpeedyMurmurs also considers channels that are not part of the i-th spanning tree: If a neighbor is close to the receiver according to d_i, SpeedyMurmurs chooses the corresponding channel regardless of whether it is included in the spanning tree. In contrast to routing using only spanning tree edges, routing based on a spanning tree distance with the inclusion of other edges is resilient to node failures and even attacks that remove nodes strategically from the network [26].

In contrast, our variant does consider all spanning trees concurrently. Note that if a node U is closer to R than P with regard to only one distance d_i, there is a loop-free path from P to R via U. Thus, U makes a good candidate for a next hop. As a consequence, Closer_{INT-SM} determines the set of candidate channels as those leading to nodes that are closer to the receiver according to at least one of the dim distance functions.

More precisely, Closer_{INT-SM} considers all spanning trees for each edge (P, U). Once it finds that if U has a lower distance to R in one spanning tree, it determines the minimal distance of U to R over all spanning trees. Indicating the minimal distance allows Split to prefer short routes. After computing the minimal distance, Closer_{INT-SM} adds the tuple consisting of the channel (P, U), its capacity, and the minimal distance to the candidate set. Afterwards, it proceeds with the next channel. In this manner, Closer_{INT-SM}

selects a large set of neighbors that offer a loop-free but not necessarily shortest path to the receiver. Thus, whereas the hop distance only considers shortest paths, Interdimensional SpeedyMurmurs offers a higher flexibility in choosing paths, thus increases the chance of successfully completing a payment.

4.2 Splitting over potential next hops (Split)

Our realizations of Split use the following three approaches: i) not splitting (baseline), ii) splitting according to the distance to R, and iii) splitting only if necessary.

No Split (Split_{No}): The first considered realization of Split is not to split. For each node in the candidate set, the algorithm checks if the capacity of the corresponding channel is sufficient. From the set of channels with sufficient capacity, it selects a node with minimal distance to R, breaking ties randomly. If none of the channels has sufficient capacity, the payment fails.

Split By Distance (Split_{Dist}): Our second realization of Split iterates over the candidate channels in order of decreasing closeness to the receiver, breaking ties randomly. For each candidate channel, it assigns a partial value that is either the channel capacity or the part of the total payment value that has not been assigned previously, whichever is less. The algorithm terminates when the total payment value has been split or all channels have been considered. In the latter case, the total capacity of all channels is insufficient for the payment value and hence the payment fails.

Split If Necessary (Split_{IfN}): Our third realization only splits if necessary and hence aims to minimize the number of splits in one particular forwarding decision. Note that such a greedy approach does not necessarily minimize the total number of splits as it might prefer longer paths. These longer paths then in turn might lead to more splits. If it is possible to forward without splitting, the algorithm corresponds to Split_{No}. Otherwise, it proceeds analogously to Split_{Dist} but considers the candidate channels in decreasing order of their capacity, breaking ties randomly.

5 PERFORMANCE EVALUATION

This section deals with the overarching question of quantifying the extent to which splitting affects the performance in terms of the success ratio and the overhead. For this purpose, a simulation study evaluates the effect of the Closer and Split functions of the routing algorithm, the topology, the transaction values, and the channels capacities.

5.1 Performance Metrics

The *success ratio* is defined as the fraction of successful payments. Note that we only consider one payment attempt per transaction. As the algorithms are non-deterministic, further attempts are bound to increase the success ratio. However, a second attempt requires waiting for the expiration of the locks, which can be in the order of hours or days. For instance, Bitcoin's Lightning network proposes a timeout of 9 blocks or approximately 1.5h[4]. As the goal for off-chain transaction is completion within seconds, it seems reasonable to consider any transaction that does not succeed in the first attempt failed.

Our second metric is the communication overhead of the routing. The key difference between the routing algorithms are the number of cPay calls, which is the only operation during routing that requires communication between nodes. As a consequence, we quantify the overhead as the number of these calls. In order to account for dependencies between overhead and success of routing, the overhead is considered both as an average over all payments and as an average only for successful payments.

5.2 Simulation Model

We extended the simulation framework for SpeedyMurmurs to include our novel routing algorithms[5].

The simulation framework allows for two options: static and dynamic. In a static setting, the topology is fixed and channel capacities are reset to their initial values after each transaction. In the dynamic setting, the topology can change and balances are not reset. In our evaluation, we did not consider changes to the topology, only to the balances.

When aiming to evaluate the differences between the algorithms, it is sufficient to implement the routing process. Thus, the simulation excluded prior and subsequent communication between sender and receiver. Furthermore, the cryptographic operations are the same for all routing algorithms. Hence, the simulation did not include the cryptographic operations in order to increase the scalability to thousands of nodes.

First, the simulation generated the local information necessary for the routing algorithms, such as the local topology snapshot and spanning trees for Interdimensional SpeedyMurmurs. Afterwards, the simulation proceeded in a round-based manner. Each cPay call was modeled to require one round. In each round, nodes combined any payment shares they received. Intermediaries then split the combined payment value from these shares over neighboring nodes. They did not combine shares received in different rounds.

Concurrency was not simulated. The main effect of concurrency is deadlocks, i.e., multiple payments blocking each others. By adapting the order in which collateral is locked, payments can be serialized and deadlocks are removed [32]. In this manner, a system with concurrency can achieve the same success ratio as one without.

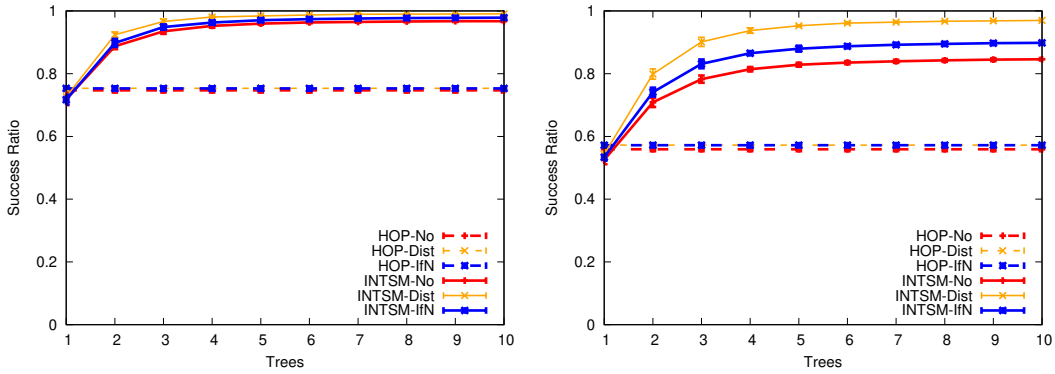
In practice, heterogeneous network latencies entail that the time required for operations such as cPay varies between nodes. As a consequence, the round-based model, in which all cPay calls require one round, is not realistic. However, the main goal of this study is to show that our algorithms improve the success ratio. In the absence of concurrency, heterogeneous latencies can change the order of messages and the way shares are combined but do not affect success or failure of one payment.

5.3 Data Sets and Parameters

The considered factors influencing the performance of the routing algorithms were topology, channel capacities, and transactions. In general, all experiments were averaged over 20 runs. For each run, the simulation framework first generated the data sets and then run all routing algorithms on the exact same data set. For Interdimensional SpeedyMurmurs, the number of trees was between 1 and 10 with randomly selected root nodes.

[4] https://github.com/lightningnetwork/lightning-rfc/blob/master/11-payment-encoding.md

[5] https://github.com/stef-roos/PaymentRouting



(a) $1/\lambda = 50$ **(b)** $1/\lambda = 200$

Figure 3: Routing success for 6 routing algorithms when transaction value follows exponential distribution with $1/\lambda$ being a) 50 and b) 200 (exponential capacities with expected value 200, Lightning topology)

Topology: The topologies were a real-world Lightning snapshot. The Lightning snapshot was from March 1, 2020, snapshot 04_00[6]. It contains 6329 nodes with an average number of channels being 10.31. Appendix D presents results for synthetic scale-free and random graphs.

Capacities and transactions: Our capacities and transactions were synthetic, though motivated by real-world data or strategical choices to highlight the impact of the respective parameters.

Initial channel capacities followed an exponential distribution. In March 2020, 200 was close to the average channel capacity in euro for Lightning and the capacity distribution was highly skewed with most channels having a low capacity[7]. Hence, an exponential distribution seemed a suitable fit with a normal distribution as an alternative that highlights the impact of the distribution. Note that the success ratio depends on the relation between capacities and transaction values rather than the actual values, thus it was sufficient to vary the expected transaction value and keep the expected capacity constant. The capacities of the two directions of a channel were chosen independently. Alternative distributions for channel capacity, as well as transaction values, are evaluated in Appendix D.

For the static setting, each run consisted of 10,000 transactions. For the dynamic setting, each run considered 1 million transactions to evaluate the changes in performance over time. In the absence of real-world transaction data, choosing sender and receiver uniformly at random was the most straight-forward option. The choice of the transaction value proceeded in two steps: First, a preliminary transaction value was chosen according to either an exponential distribution. Exponential distributions indicate many transactions of a small value with few expensive purchases. The expected value of the distribution was $1/\lambda \in \{1, 5, 10, 20, 50, 100, 200, 300\}$. Second, for the static setting, the preliminary transaction value had to be below the maximal flow of the sender and receiver. In this manner, the computed success ratio was the success ratio for the set of transaction that were possible to settle, thus making it possible to quantify how close the routing algorithm was to an optimal solution. If a preliminary transaction value was higher than the maximum flow, the random selection process was repeated until either a sufficiently low transaction value was found or a maximum of 1000 attempts at choosing the transaction value. In the latter case, the transaction value was chosen to be the maximum flow. The details of the dynamic setup are in Appendix F.

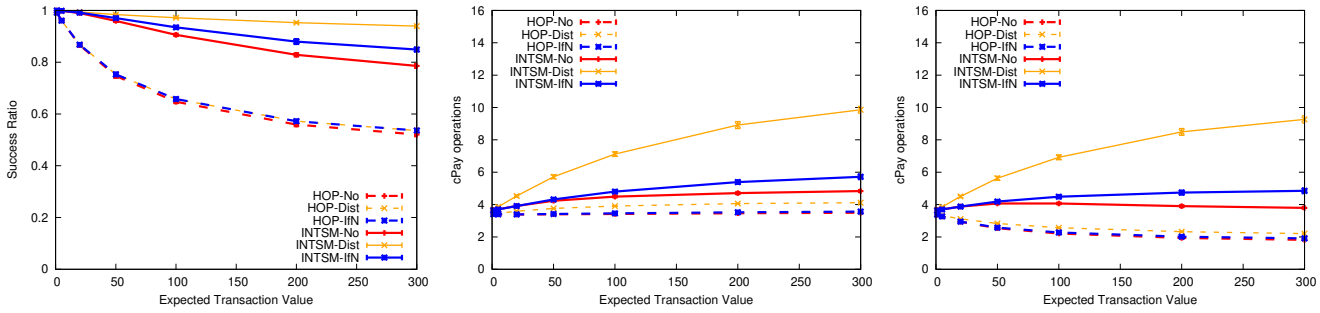
Timeouts: The remainder of the section sets timeouts in accordance with the generic protocol from Section 3, i.e., assuming that the maximal path length is equal to the number of nodes. However, we discuss and evaluate the impact of shorter timeouts in Appendix E. Indeed, the routing is essential equally successful when the applied timeout is about 500 times shorter than the one used in the generic protocol.

5.4 Results

We use the abbreviations from Section 4 throughput the remainder of this section. Generally, a combination of a realization C of Closer and a realization S of Split is written as C-S. For readability, the legends in the figures use *No*, *Dist*, and *IfN* rather than Split_{No}, Split_{Dist}, and Split_{IfN}, respectively.

Success ratio for static scenario: If the typical transaction value was low, all routing algorithms achieved a success ratio of nearly 100%. Differences only became apparent if the typical transaction value was more than 20% of the expected capacity. Then, Interdimensional SpeedyMurmurs with any splitting method and more than one spanning tree outperformed HopDistance as it offered a higher flexibility in the path choice. More concretely, Interdimensional SpeedyMurmurs does not require that a shortest path is taken. Figure 3 portrays examples of this behavior for $1/\lambda = 50$ and $1/\lambda = 200$, i.e., for $1/\lambda$ being 25% and 100% of the expected capacity. The success ratio is displayed in relation to the number of spanning trees. As HopDistance does not utilize the spanning trees, the corresponding results are horizontal lines in the figure. Increasing the number of trees and hence the number of options for paths increased the success ratio of Interdimensional SpeedyMurmurs. However, more than 5 trees had little additional impact.

[6]https://gitlab.tu-berlin.de/rohrer/discharged-pc-data
[7]https://1ml.com/statistics



(a) Success Ratio **(b) Overhead: Successful** **(c) Overhead: All**

Figure 4: a) success ratio and overhead for b) successful transactions and c) all transactions with exponentially distributed transaction values varying $1/\lambda$ (exponential capacities with expected value 200, Lightning topology, dim $= 5$ for Interdimensional SpeedyMurmurs)

The choice of the splitting algorithm only improved the success ratio slightly when HopDistance was used as a realization of Closer. The result stemmed from the low number of shortest paths, especially disjoint shortest paths.

For Interdimensional SpeedyMurmurs, the impact of splitting was considerable. The most successful splitting approach was Split_{Dist}, i.e., splitting only over nodes with the least distance to the receiver. In contrast, Split_{IfN}, which tried to greedily minimize the number of splits, did improve the success ratio but not as much as Split_{Dist}. Split_{IfN} preferred nodes at a higher distance, hence leading to longer paths than Split_{Dist}, which for our data set were more likely to lead to failures, even if they incurred less splits than Split_{Dist}.

The results in terms of best-performing algorithms were consistent for all considered capacity and transaction value distributions as well as topologies, with the concrete results being resented in Appendix D.

Overhead: Figure 4 displays the trade-off between success ratio and number of cPay calls when varying $1/\lambda$. As increasing the success ratio comes with a higher flexibility in choosing longer paths and more splitting, a higher success ratio indicates also a higher overhead for successful transactions. When considering all transactions, the overhead was typically lower and could even decrease with the average transaction value, as clearly indicated for the hop distance in Figure 4c. The decrease stemmed from the early failure of many transactions, i.e., transactions failing in the first hops due to a lack of available funds did require a low number of cPay calls. As a consequence, a lower success ratio in Figure 4a corresponds to a lower overhead in Figure 4c. However, the overhead was generally inconsequential, at less than 15 calls per routing on average, in comparison to disseminating a transaction in the whole network.

Dynamic Setting: When transactions permanently change the balances of channels, the success ratio tends to drop as some channels become depleted [27, 29]. Indeed, the success ratio for Interdimensional SpeedyMurmurs drops initially in the dynamic setting. However, the success ratio seemed to reach a steady state after about 10,000 transactions and remained considerably higher than the success ratio for the hop distance. Splitting still increased the success ratio but less so for splitting algorithm Split_{Dist}, which tends to use the full capacity of channels and hence deplete them. The details of the study are in Appendix F.

In summary, Interdimensional SpeedyMurmurs increases the success ratio consistently and drastically over a distance function that is merely based on the length of shortest paths. Furthermore, splitting increases the success ratio if the transaction value is similar or higher than the typical channel capacities.

6 SECURITY ANALYSIS

Let $\mathcal{R}$ be the set of all routing algorithms discussed in Section 4. More precisely, we define the set $\mathcal{R}$ as

$$\left\{ \text{Route}(\text{Closer}, \text{Split}) \;\middle|\; \begin{array}{c} \text{Closer} \in \{\text{Closer}_{HOP}, \text{Closer}_{INT-SM}\}, \\ \text{Split} \in \{\text{Split}_{No}, \text{Split}_{Dist}, \text{Split}_{IfN}\} \end{array} \right\}.$$

Our goal is to show that for any routing algorithm $\text{Route} \in \mathcal{R}$, both protocols $\Pi_b(\text{Route})$ and $\Pi_{ext}(\text{Route})$ that were discussed in Section 3 satisfy Def. 2.1 with respect to the receipt validation function Validate from Figure 1.

Theorem 6.1. *Assume that Σ is an* EUF–CMA*-secure signature scheme, Ψ is an encryption scheme with message space $\mathbb{M}$, and $\mathcal{H}$ a preimage-resistant hash function with domain $\mathbb{P}$. For any* $\text{Route} \in \mathcal{R}$*, the protocol $\Pi_b(\text{Route})$ is a secure payment protocol with respect to the function $\text{Validate}_{\Sigma,\mathcal{H}}$.*

If, in addition, Ψ and $\mathcal{H}$ are additively homomorphic, and $\mathbb{M} = \mathbb{Z}_N$, $\mathbb{P} = \mathbb{Z}_p$ for p, N coprime and $p < N$, then for any $\text{Route} \in \mathcal{R}$*, the protocol $\Pi_{ext}(\text{Route})$ is a secure payment protocol with respect to the function $\text{Validate}_{\Sigma,\mathcal{H}}$.*

The formal proof of the theorem can be found in Appendix I, here we discuss its main ideas. The protocol termination follows directly from the protocol description and was discussed in detail already in Section 3. Since an honest sender makes conditional payments of total value at most v, the sender's loss is bounded by v. Similarly, an honest receiver never makes any payment and hence cannot lose coins. For the balance neutrality of intermediaries, we observe that an honest intermediary never forwards more coins than what he can potentially receive. Moreover, if an outgoing conditional payment is unlocked, the intermediary has a guarantee of unlocking

the corresponding incoming payment. This follows from correctly set timeouts and hash-locks. In the extended protocol, the latter follows from the homomorphic property of $\mathcal{H}$.

Atomicity for the sender S is guaranteed by the fact that S does not initiate any payment without holding a valid signature of the receiver R on (S, R, v, h_R). Moreover, the sender sets the hash-locks such that from the preimage of any of them, a preimage of h_R can be computed. In the extended protocol, the guarantee of preimage recovery follows from the homomorphic property of $\mathcal{H}$. Atomicity for R follows from EUF–CMA security of Σ and preimage resistance of $\mathcal{H}$ − S cannot forge R's signature and hence must present a preimage of h_R, where h_R has been chosen by R. Moreover, S cannot compute a preimage of h_R without R revealing it, which happens only if R gets v coins.

Finally, we need to argue correctness, i.e., if all parties are honest and all channels have enough coins, the payment succeeds. We prove that no matter which routing algorithm Route $\in \mathcal{R}$ we consider, the routing never fails. Moreover, we show that the initial time-lock set by the sender is sufficient to guarantee that all partial payments reach the receiver. Due to the correctness and additive homomorphism of Ψ and $\mathcal{H}$, R can then compute all witnesses and start unlocking payments. Since the total amount of coins in the system cannot increase (parties cannot create coins), we complete the proof by applying the security properties: balance neutrality and bounded loss for the sender.

6.1 Payment unlinkability

Assume that an honest party (the sender or an intermediary) splits a payment of v coins into k partial payments $(v_1, \ldots, v_k)$ routed over k different neighbors. We want to guarantee that even if all of these neighbors collude, they cannot decide whether the conditional payments are part of the same payment or if k independent payments of values $(v_1, \ldots, v_k)$ have been sent. Since the partial payments are unlinkable right after the split takes place, further forwarding does not influence their linkability. Hence, our definition also captures the case when the colluding parties happens later in the partial payment paths (and not right after the splitting). Formally, we define unlinkability as a property of the algorithm HLocks in the following definition. In Appendix J, we discuss in detail why our definition captures the intuition above.

Definition 6.1 (Unlinkability). Let Ψ be an encryption scheme with plaintext space $\mathbb{M}$ and let $\mathcal{H} : \mathbb{P} \to \mathbb{H}$ be a hash function with $\mathbb{P} \subseteq \mathbb{M}$. Algorithm $\mathsf{HLocks}^{\mathcal{H},\Psi}$ produces *unlinkable conditions* if for every ppt adversary $\mathcal{A}$, every $x \in \mathbb{P}$, $y \in \mathbb{M}$ and $k \in \mathbb{N}$, we have $\Pr[\mathsf{GameLink}_{\mathcal{A},\mathsf{HLocks}^{\mathcal{H},\Psi}}(x, y, k, n) = 1] \leq \frac{1}{2} + \mathsf{negl}(n)$, where the game GameLink is defined as follows:

$\mathsf{GameLink}_{\mathcal{A},\mathsf{HLocks}^{\mathcal{H},\Psi}}(x, y, k, n)$

$b \leftarrow_{\$} \{0, 1\}$

$(pk, sk) \leftarrow \mathsf{Gen}(1^n)$, $h := \mathcal{H}(x)$, $c \leftarrow \mathsf{Enc}_{pk}(y)$

if $b = 1$ **then** $\{(h_j, c_j, x_j)\}_{j\in[k]} \leftarrow \mathsf{HLocks}^{\mathcal{H},\Psi}(h, c, k, pk)$

else foreach $j \in [k]$ **do** $x_j \leftarrow_{\$} \mathbb{P}$, $h_j := \mathcal{H}(x_j)$, $c_j := \mathsf{Enc}_{pk}(x_j)$

$b' \leftarrow \mathcal{A}(x, y, \{(h_j, c_j)\}_{j\in[k]}, pk)$

return $b = b'$

We note that our definition only talks about the unlinkability of the hash values and attached ciphetexts. In particular, we do not aim to hide the payment history (e.g., the sender) or the receiver since this information is crucial for our local routing algorithms. If there are only few concurrent payments, these metadata might link the partial payments even if the hash values and ciphertexts are unlinkable.

In Appendix J, we prove the following theorem stating that our algorithm HLocks_{ext} satisfies the unlinkability definition.

Theorem 6.2. *Let Ψ be an additively homomorphic* IND–CPA*-secure encryption scheme with message space $\mathbb{M} = \mathbb{Z}_N$, and $\mathcal{H}$ an additively homomorphic preimage-resistant hash function with domain $\mathbb{P} = \mathbb{Z}_p$ such that p, N coprime and $p < N$. Then $\mathsf{HLocks}^{\mathcal{H},\Psi}_{ext}$ produces unlinkable conditions.*

7 RELATED WORK

The vast majority of payment channel routing algorithms are either single or multi-path [13, 14, 24, 27, 31, 34], without intermediaries splitting or recombining payments. Atomicity for multi-path payments is possible [19]; however, the algorithm requires the sender to split the payment and is not applicable for splitting on the path.

Spider [30] is a routing algorithm that splits the total payment value into small units and routes every unit individually, an idea previously proposed by Piatkivskyi and Nowostawski [22]. By making the individual payments small, the likelihood of a channel not having sufficient capacity is low and the routing is very flexible in terms of the amounts routed per path. However, a large fraction of the small payments take the same path, creating a large overhead our splitting scheme avoids. Furthermore, while the authors state that atomicity is possible, they leave open how atomicity relates to their local queuing of transactions in term of timeouts.

Boomerang [3] enables redundancy in the sense that the sender can route more than the transaction value using multiple paths. Afterwards, the sender can reclaim any funds that exceed the transaction value. In this manner, partial payments can fail while still transacting the requested value. Boomerang thus presents a complementary approach to splitting on the path and the two protocols can easily be combined. The disadvantage of Boomerang is that routing higher payment values result in increased locked collateral, which splitting does not.

The only work on splitting payments on the path is Ethna [8]. However, the authors do not evaluate the concrete impact of splitting and do not discuss possible routing algorithms. Moreover, the overall goal of the cryptographic protocol is different from ours since Ethna explicitly does not aim for atomicity and allows a partial payment (with a partial receipt) to be a valid outcome of the protocol.

A similar notion of payment unlinkability is informally discussed in the atomic multi-path payment proposal [19]; however, no formal definition or proof is provided. A different unlinkability notion is considered by anonymous mulit-hop locks and its predecessor [15, 16]. This line of work considers only monolithic payments, source-routed by the sender. Their aims is to prevent colluding intermediaries on one payment path to see that they are forwarding the same payment. The objective of this "on-path" unlinkability definition is to hide the sender's and receiver's identities. Although

we could argue that hash-locks on one partial payment path in our extended protocol are "on-path" unlinkable, the routing information attached to the payment would trivially reveal the sender and the receiver. Since our goal is to increase the success ratio of payments by allowing intemediaries to locally decide on the route, hiding routing information (e.g., identity of the receiver) is not desirable.

8 CONCLUSION

This paper presented a protocol for locally splitting payments in a PCN that guarantees termination, atomicity, balance neurality, bounded loss for the sender, correctness, and optionally unlinkability. Our evaluation illustrated the advantages of splitting and compared a variety of splitting methods.

A comparison between the static and the dynamic setting highlights that splitting approaches that utilize the full channel capacity in order to get closer to the receiver have a negative effect on the long-term success ratio. Restricting the capacity of channels that a payment can use when another path is available is a promising avenue for improving the success ratio further.

ACKNOWLEDGMENTS

This work was partly supported by the German Research Foundation (DFG) Emmy Noether Program *FA 1320/1-1*, by the *DFG CRC 1119 CROSSING* (project S7), by the German Federal Ministry of Education and Research (BMBF) *iBlockchain project* (grant nr. 16KIS0902), by the German Federal Ministry of Education and Research and the Hessen State Ministry for Higher Education, Research and the Arts within their joint support of the *National Research Center for Applied Cybersecurity ATHENE*, and by Ripple's University Blockchain Research Initiative.

REFERENCES

[1] 2020. Raiden Network. (2020). https://raiden.network/.
[2] Lukas Aumayr, Oguzhan Ersoy, Andreas Erwig, Sebastian Faust, Kristina Hostakova, Matteo Maffei, Pedro Moreno-Sanchez, and Siavash Riahi. 2020. Generalized Bitcoin-Compatible Channels. Cryptology ePrint Archive, Report 2020/476. (2020). https://eprint.iacr.org/2020/476.
[3] Vivek Bagaria, Joachim Neu, and David Tse. 2020. Boomerang: Redundancy Improves Latency and Throughput in Payment Networks. In *Financial Cryptography and Data Security*.
[4] Albert-László Barabási and Réka Albert. 1999. Emergence of scaling in random networks. *science* 286, 5439 (1999).
[5] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, et al. 2016. On scaling decentralized blockchains. In *International conference on financial cryptography and data security*. Springer, 106–125.
[6] Stefan Dziembowski, Lisa Eckey, Sebastian Faust, and Daniel Malinowski. 2019. Perun: Virtual Payment Hubs over Cryptocurrencies. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*. 106–123.
[7] Stefan Dziembowski, Sebastian Faust, and Kristina Hostáková. 2018. 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*. 949–966.
[8] Stefan Dziembowski and Paweł Kędzior. 2020. Ethna: Channel Network with Dynamic Internal Payment Splitting. https://eprint.iacr.org/2020/166.pdf. (2020).
[9] Christoph Egger, Pedro Moreno-Sanchez, and Matteo Maffei. 2019. Atomic Multi-Channel Updates with Constant Collateral in Bitcoin-Compatible Payment-Channel Networks. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19)*. ACM, 801–815.
[10] Paul Erdős and Alfréd Rényi. 1960. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci* 5, 1 (1960).
[11] Nathan S Evans and Christian Grothoff. 2011. R5n: Randomized recursive routing for restricted-route networks. In *5th International Conference on Network and System Security*.
[12] Arpita Ghosh, Mohammad Mahdian, Daniel M Reeves, David M Pennock, and Ryan Fugger. 2007. Mechanism design on trust networks. In *International Workshop on Web and Internet Economics*.
[13] Philipp Hoenisch and Ingo Weber. 2018. AODV–Based Routing for Payment Channel Networks. In *International Conference on Blockchain*. Springer, 107–124.
[14] Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, and Matteo Maffei. 2017. SilentWhispers: Enforcing Security and Privacy in Decentralized Credit Networks.. In *NDSS*.
[15] Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, Matteo Maffei, and Srivatsan Ravi. 2017. Concurrency and Privacy with Payment-Channel Networks. In *ACM CCS 17*, Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM Press, 455–471.
[16] Giulio Malavolta, Pedro Moreno-Sanchez, Clara Schneidewind, Aniket Kate, and Matteo Maffei. 2019. Anonymous Multi-Hop Locks for Blockchain Scalability and Interoperability.. In *NDSS*.
[17] Andrew Miller, Iddo Bentov, Ranjit Kumaresan, and Patrick McCorry. 2017. Sprites: Payment Channels that Go Faster than Lightning. *CoRR* abs/1702.05812 (2017). http://arxiv.org/abs/1702.05812
[18] Satoshi Nakamoto. 2009. Bitcoin: A Peer-to-Peer Electronic Cash System. (2009). http://bitcoin.org/bitcoin.pdf.
[19] Olaoluwa Osuntokun. 2018. AMP: Atomic Multi-Path Payments over Lightning. https://lists.linuxfoundation.org/pipermail/lightning-dev/2018-February/000993.html. (2018).
[20] Pascal Paillier. 1999. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *EUROCRYPT'99 (LNCS)*, Jacques Stern (Ed.), Vol. 1592. Springer, Heidelberg, 223–238.
[21] Radia Perlman. 1985. An algorithm for distributed computation of a spanningtree in an extended lan. *ACM SIGCOMM Computer Communication Review* 15, 4 (1985), 44–53.
[22] Dmytro Piatkivskyi and Mariusz Nowostawski. 2018. Split payments in payment networks. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Springer, 67–75.
[23] Andrew Poelstra. 2017. Scriptless scripts. (May 2017). https://download.wpsoftware.net/bitcoin/wizardry/mw-slides/2017-05-milan-meetup/slides.pdf.
[24] Joseph Poon and Thaddeus Dryja. 2016. The bitcoin lightning network: Scalable off-chain instant payments. https://www.bitcoinlightning.com/wp-content/uploads/2018/03/lightning-network-paper.pdf. (2016).
[25] Elias Rohrer, Julian Malliaris, and Florian Tschorsch. 2019. Discharged Payment Channels: Quantifying the Lightning Network's Resilience to Topology-Based Attacks. *arXiv preprint arXiv:1904.10253* (2019).
[26] Stefanie Roos, Martin Beck, and Thorsten Strufe. 2016. Anonymous addresses for efficient and resilient routing in f2f overlays. In *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*. IEEE, 1–9.
[27] Stefanie Roos, Pedro Moreno-Sanchez, Aniket Kate, and Ian Goldberg. 2018. Settling payments fast and private: Efficient decentralized routing for path-based transactions. In *NDSS*.
[28] Claus-Peter Schnorr. 1991. Efficient Signature Generation by Smart Cards. *Journal of Cryptology* 4, 3 (1991), 161–174.
[29] Vibhaalakshmi Sivaraman, Shaileshh Bojja Venkatakrishnan, Mohammad Alizadeh, Giulia Fanti, and Pramod Viswanath. 2018. Routing cryptocurrency with the spider network. In *Proceedings of the 17th ACM Workshop on Hot Topics in Networks*. 29–35.
[30] Vibhaalakshmi Sivaraman, Shaileshh Bojja Venkatakrishnan, Kathleen Ruan, Parimarjan Negi, Lei Yang, Radhika Mittal, Giulia Fanti, and Mohammad Alizadeh. 2020. High Throughput Cryptocurrency Routing in Payment Channel Networks. In *17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20)*. 777–796.
[31] Peng Wang, Hong Xu, Xin Jin, and Tao Wang. 2019. Flash: efficient dynamic routing for offchain networks. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*. 370–381.
[32] Shira Werman and Aviv Zohar. 2018. Avoiding deadlocks in payment channel networks. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*.
[33] Gavin Wood. 2014. ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER. (2014). http://gavwood.com/paper.pdf.
[34] Yuhui Zhang, Dejun Yang, and Guoliang Xue. 2019. Cheapay: An optimal algorithm for fee minimization in blockchain-based payment channel networks. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*. IEEE, 1–6.

A CRYPTOGRAPHIC PRIMITIVES

Homomorphic encryption. A public key encryption scheme Ψ with a message space $\mathbb{M}$ and ciphertext space $\mathbb{C}$ is a triple of ppt

algorithms (Gen, Enc, Dec) with the following syntax: The algorithm Gen on input the security parameter n, outputs a key pair (pk, sk). The algorithm Enc on input a message $m \in \mathbb{M}$ and a public key pk, outputs a ciphertext $c \in \mathbb{C}$. Finally, Dec is a deterministic algorithm that on input a secret key sk and a ciphertext $c \in \mathbb{C}$ outputs a message $m \in \mathbb{M}$. For every message $m \in \mathbb{M}$ it holds that

$$\Pr[\mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(m)) = m \mid (pk, sk) \leftarrow \mathsf{Gen}(1^n)] = 1,$$

where the probability is taken over the randomness of Gen and Enc. In this work we use encryption schemes that satisfy the notion of *indistinguishability under chosen paintext attack* (IND-CPA secure for short) defined below.

Definition A.1 (IND-CPA security). An encryption scheme $\Psi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is IND–CPA secure if for every ppt adversary $\mathcal{A}$ there exists a negligible function negl such that:

$$\Pr[\mathsf{PubK}^{\mathsf{cpa}}_{\mathcal{A},\Psi}(n) = 1] \leq \mathsf{negl}(n),$$

where the experiment $\mathsf{PubK}^{\mathsf{cpa}}_{\mathcal{A},\Psi}$ is defined as follows:

$\mathsf{PubK}^{\mathsf{cpa}}_{\mathcal{A},\Psi}(n)$

$(sk, pk) \leftarrow \mathsf{Gen}(1^n)$

$(m_0, m_1) \leftarrow \mathcal{A}(pk)$

$b \leftarrow_{\$} \{0, 1\}$, $c_b \leftarrow \mathsf{Enc}_{pk}(m_b)$

$b^* \leftarrow \mathcal{A}(c_b)$

return $|m_0| = |m_1| \wedge b = b^*$

Definition A.2 (Additively homomorphic encryption). Let $\Psi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be an encryption scheme, where the message space $(\mathbb{M}, +_{\mathbb{M}}, 0_{\mathbb{M}})$ forms an Abelian group and the ciphertext space $\mathbb{C}$ is closed under the binary operation $+_{\mathbb{C}}$. We say that Ψ is *additively homomoprhic* if for every $x, y \in \mathbb{M}$ and every public key pk it holds that

$$\mathsf{Enc}_{pk}(x) +_{\mathbb{C}} \mathsf{Enc}_{pk}(y) \equiv \mathsf{Enc}_{pk}(x +_{\mathbb{M}} y),$$

where $\equiv$ denotes equality of distributions.

Signature scheme. A digital signature scheme Σ is a triple of ppt algorithms (Gen, Sign, Vrfy) with the following syntax: The algorithm Gen on input the security parameter n, outputs a key pair (pk, sk). The algorithm Sign on input a message m and a signing key sk, outputs a signature σ. Finally, Vrfy is a deterministic algorithm that on input a verification key pk, a message m and a signature σ outputs a bit b. For every message m it holds that

$$\Pr[\mathsf{Vrfy}_{pk}(\mathsf{Sign}_{sk}(m)) \mid (pk, sk) \leftarrow \mathsf{Gen}(1^n)] = 1,$$

where the probability is taken over the randomness of Gen and Sign. In this work, we use signature schemes that are *existentially unforgeable under chosen message attack* (or EUF-CMA secure for short) defined as follows.

Definition A.3 (EUF-CMA security). A signature scheme Σ is EUF–CMA secure if for every ppt adversary $\mathcal{A}$ there exists a negligible function negl such that: $\Pr[\mathsf{SigForge}_{\mathcal{A},\Sigma}(n) = 1] \leq \mathsf{negl}(n)$, where the experiment $\mathsf{SigForge}_{\mathcal{A},\Sigma}$ is defined as follows:

$\mathsf{SigForge}_{\mathcal{A},\Sigma}(n)$

$(sk, pk) \leftarrow \mathsf{Gen}(1^n)$

$(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}_{sk}(\cdot)}(pk)$

Let Q denote the set of queried messages of $\mathcal{A}$ to the signing oracle

return $\mathsf{Vrfy}_{pk}(m^*, \sigma^*) \wedge m \notin Q$

Homomorphic hash function. A hash function is a pair of ppt algorithms $(\mathsf{Gen}, \mathcal{H})$ with the following syntax: the algorithm Gen on input a security parameter n outputs a key s. The algorithm $\mathcal{H}$ on input the key s and a value $x \in \mathbb{P}_s$, outputs a value $y \in \mathbb{H}_s$. In this work, we consider preimage-resistant hash functions formally defined as follows.

Definition A.4 (Preimage-resistant hash function). A hash function $(\mathsf{Gen}, \mathcal{H})$ is *preimage resistant* if for every ppt adversary $\mathcal{A}$, there exists a negligible function negl such that

$$\left[\Pr[\mathcal{H}_s(x') = y] \;\middle|\; \begin{array}{c} s \leftarrow \mathsf{Gen}, x \leftarrow_{\$} \mathbb{P}, \\ y := \mathcal{H}_s(x), x' \leftarrow \mathcal{A}(s, y) \end{array}\right] = \mathsf{negl}.$$

Definition A.5 (Homomorphic function). Consider two Abelian groups $(\mathbb{P}, +_{\mathbb{P}}, 0_{\mathbb{P}})$ and $(\mathbb{H}, +_{\mathbb{H}}, 0_{\mathbb{H}})$. We say that $\mathcal{H} \colon \mathbb{P} \to \mathbb{H}$ is a *homomorphism* if for every $x, y \in \mathbb{P}$ it holds that $\mathcal{H}(x +_{\mathbb{P}} y) = \mathcal{H}(x) +_{\mathbb{H}} \mathcal{H}(y)$.

B INSTANTIATION OF BUILDING BLOCKS

Let us now discuss how to instantiate the building blocks used by our protocols. Our basic protocol requires the underlying PCN to support conditional payments locked by a hash preimage verification, i.e., $\mathsf{Hash}^{\mathcal{H}}_h$, for $\mathcal{H}$ being a preimage-resistant hash function. Such a PCN can be built over most common blockchains, including Bitcoin and Ethereum, typically using SHA256. A concrete PCN that is currently deployed on top of the Bitcoin blockchain is the Lightning Network [24]. An example of a PCN over Ethereum is the Raiden network [1]. Moreover, our protocol relies on a EUF–CMA-secure digital signature that can be instantiated by, e.g., the signature scheme of Schnorr [28] or ECDSA.

The requirements of our extended protocol are higher as we need a preimage-resistant hash function that is homomorphic. Such a hash function can be instantiated as follows. Let G be a group of prime order p and let g be a generator of G. Then the function $\mathcal{H} \colon \mathbb{Z}_p \to G$ defined as $\mathcal{H}(x) := g^x$ is additively homomorphic and under the assumption that dlog is hard in G, $\mathcal{H}$ is preimage resistant.

While replacing SHA256 with a homomorphic hash function is not a problem for PCNs built over blockchains whose scripting languages are Turing complete, the situation is more complicated for legacy cryptocurrencies such as Bitcoin where the evaluation of such hash functions is not supported. In order to overcome this difficulty, we can make use of *Adaptor signatures* – primitive proposed by Poelstra [23], instantiated based on both Schnorr and ECDSA signatures by [16] and recently formalized by [2]. On a high level, adaptor signatures allow one party to *pre-sign* a message with respect to some hash value h. Such pre-signature can be adapted into a valid signature by any party if and only if this party knows a preimage of a hash value h. And importantly, a signer observing the adapted valid signature can extract the preimage of h. In the

context of payment channels, the role of a signer would be taken by the payer (the party that wants to conditionally pay), the message would be the unlocking transaction assigning coins to the payee (the party conditionally receiving coins), and the hash value would be the condition of the payment. We refer to [2, 16] for more details about adaptor signatures.

Finally, our construction assumes an additively homomorphic encryption scheme with message space $\mathbb{Z}_N$ for $p < N$ and p, N coprime. The encryption scheme of Paillier [20] satisfies these properties.

C ATTACK EFFECTIVENESS

Using the simulation model from Section 5, this section evaluates the attack of nodes dropping payments that have been split. In order to detect such split payments, the attacker has to observe the same hash multiple times. Note that it seems unlikely that an attacker will drop the payment if they already committed to a different partial payment with the same hash. Hence, the attacker only drops if they have not yet committed, meaning that either i) the adversary observes two partial payment in the same round or ii) the adversary delays the payment until they are relatively sure that they will not observe a second payment with the same hash. In our model, a randomly selected fraction p of nodes applies the attack. Attackers might be individual non-colluding rational nodes or a group of colluding nodes.

Our simulation used the Lightning topology with exponentially distributed capacities and transaction values. The evaluated routing algorithm was Interdimensional SpeedyMurmurs, as introduced in section 4, with the three well-performing splitting algorithms: Split_{No}, Split_{Dist}, and Split_{IfN}. The fraction p varied between 0 and 1.0 in steps of 0.1. When using a delay, the delay was set to 12, the length of the longest shortest path in the Lightning topology. For more details on the data sets and parameters, please check Section 5.

Figure 5 displays the results regarding the attack effectiveness. If splitting was only applied if necessary, the success ratio dropped towards the scenario when there was no splitting as all split payments were dropped. Split_{Dist}, which splits to minimize the number of hops, splits more frequently and hence suffered more drastically from the attack. Indeed, the success ratio dropped below the success ratio of not splitting if 0.2 or more of the network apply the attack. The drop was more pronounced for colluding attackers and if delays were applied. Both colluding and delays allowed the attacker to detect more splits. Yet, even without collusion and delays, the success ratio was considerably reduced. Given that Split_{Dist} is the most effective splitting mechanism, the severity of the attack is evident.

In summary, the discussed attack can indeed negate the positive impact of splitting. Hence, introducing unlinkability is important from a security perspective.

D IMPACT OF DISTRIBUTIONS AND TOPOLOGY

In this section, we evaluate the impact of the capacity distribution, the transaction value distribution, and the topology.

For the channel capacity, we consider a normal distribution with average 200 and a standard deviation of 10 in comparison to an exponential distribution with the same average. Similarly, normally transaction values were evaluated. The expected value of the distribution was $\mu = 100$ with the standard deviation of $0.1 \cdot \mu$. As described in Section 5, we only considered transactions that could succeed.

For the scale-free and random graphs, Barabasi-Albert [4] and Erdos-Renyi [10] graphs, respectively, with the same number of nodes and a similar average degree, were generated. More concretely, each node added to the Barabasi-Albert graph established 5 channels to already-existing nodes. For the random graph, the probability of an edge between nodes was chosen such that the expected degree was 10.31. Nodes outside of the largest connected component were removed.

Figure 6a displays the success ratio for different distributions. Normal distributions in the capacity reduced the number of failures as channel capacities were more uniform. Hence, it was less likely to contain situations when a node had only channels of low capacity. Indeed, the advantages of Interdimensional SpeedyMurmurs and splitting were more pronounced for scale-free and random networks as they offered a higher diversity of paths than Lightning's hub-and-spoke topology [25], as can seen in Figure 6b.

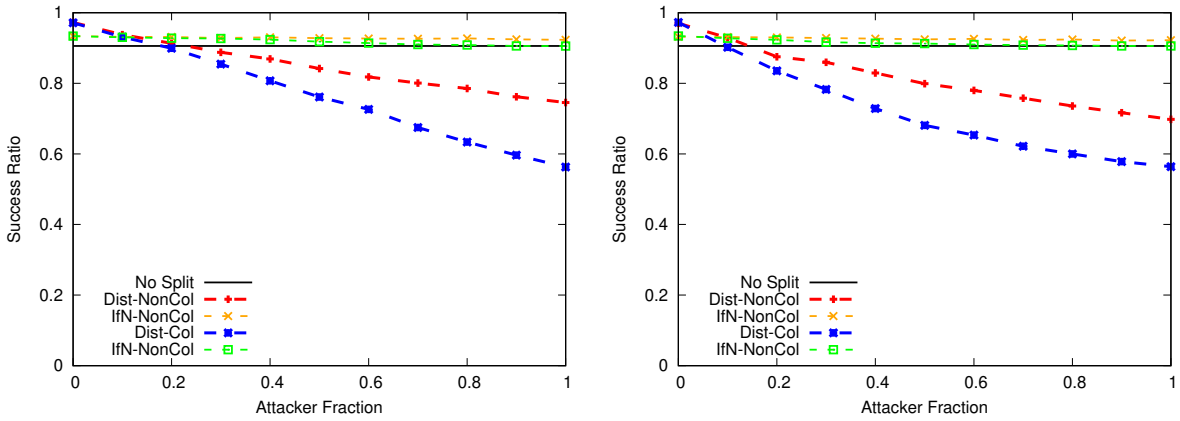
E IMPACT OF SHORTER TIMEOUTS

The generic protocol in Figure 2 assumes that the maximal length of a (partial) routing path is equal to the number of nodes in the network. However, in practice, such high timeouts are not desired as they lock collateral for unnecessary long periods.

For the hop distance, an initial timeout corresponding to the initial distance is always sufficient by construction of the distance as the length of the shortest path. For Interdimensional SpeedyMurmurs, termination can be guaranteed for $T = |\mathcal{V}|$ because loops are not possible. However, in practice, such long timeouts are undesirable, especially if they do not increase the success ratio sufficiently. Thus, our evaluation also considered the following four options for timeouts:

(1) the minimum of the dim distances between sender S and receiver R, i.e., $\min_{i \in [\dim]}\{d_i(S, R)\}$ (denoted by MIN)
(2) the maximum of the dim distances between sender S and receiver R, i.e., $\max_{i \in [\dim]}\{d_i(S, R)\}$ (denoted by MAX)
(3) the diameter of the network, i.e., the longest shortest path in the network (denoted by $DIAMETER$)
(4) twice the diameter of the network, i.e., the longest shortest path in the network (denoted by $2DIAMETER$)

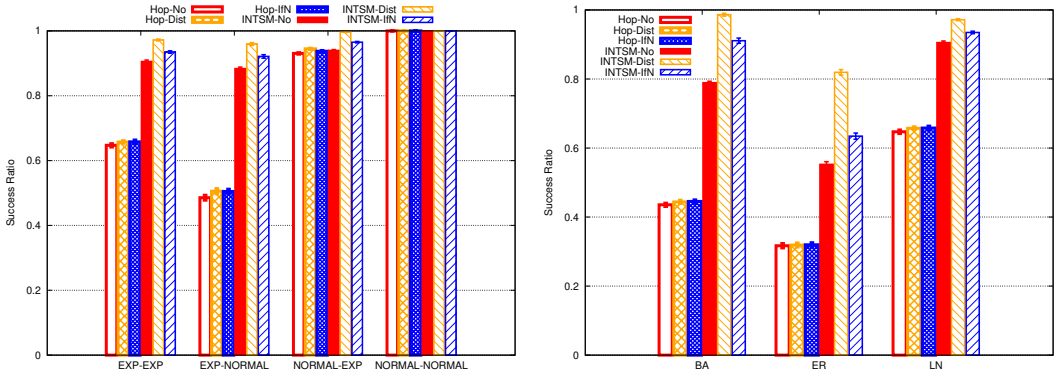
The minimum distance indicates that there is at least one path of this length between S and R. However, the routing algorithm might take longer paths, in which case the payment will fail even if the routing could succeed with a higher timeout. The maximum distance d_{max} indicates that there is a path in every tree whose length is at most d_{max}. However, Interdimensional SpeedyMurmurs utilizes all spanning trees in parallel and hence the function giving the distance after j hops might not be monotonously decreasing. As a consequence, the routing algorithm might still take longer paths. Similarly, there are no guarantees that Interdimensional SpeedyMurmurs does not result in paths longer than the diameter or even twice the diameter. However, it seems unlikely for paths to



(a) No Delay

(b) Delay (12 rounds)

Figure 5: Impact of the attack on success ratio (exponential capacities with expected value 200, exponential transaction values with $1/\lambda = 100$, Lightning topology, $\dim = 5$ for Interdimensional SpeedyMurmurs) for colluding (Col) and non-colluding (NonCol) attackers



(a) Capacity/Transaction Distribution

(b) Topology

Figure 6: Routing success for 6 routing algorithms for a) exponential and normal capacity (first part of x-axis label) and transaction value (second part of x-axis label) distributions on Lightning and b) three topologies: Barabasi-Albert (BA), Erdos-Renyi (ER), Lightning (LN) (expected capacity 200, transaction values for $1/\lambda = 100$ or $\mu = 100$, $\dim = 5$ for Interdimensional SpeedyMurmurs)

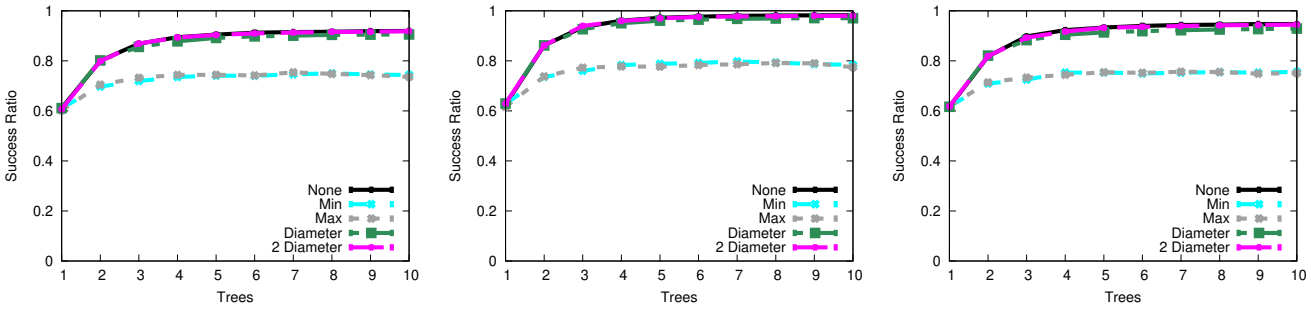
exceed such a length. Our evaluation substantiates this intuition with concrete numbers.

Figure 7 depicts the impact of different maximal timeouts on the success ratio for three different splitting algorithms. The results were consistent for all three algorithms: Using *MIN* and *MAX* did reduce the success ratio considerable, so it is not advisable to use them as timeouts. However, using the diameter of 12 or a multiple of the diameter produced nearly the same success ratio as a timeout of $|\mathcal{V}| = 6000$. For the Lightning snapshot, we can hence use a timeout that is approximately $6000/12 = 500$ times shorter than the theoretical bounds while essentially maintaining the same success ratio.

F DYNAMIC SCENARIO

Setup: Note that our simulation setup differed with regard to the generation of transactions. As channel capacities changed based on the routing, different routing algorithms led to different capacities over time, so that it was not easily possible to easily determine whether a transaction is possible via computing the maximal flow. Indeed, as indicated by previous work, a maximum flow algorithm can have a lower success ratio in the dynamic setting than local alternatives [27]. Furthermore, the problem of determining if a sequence of transactions that changes balances is possible to fulfill is NP-complete [12]. As a consequence, the second step was not executed for the dynamic setting.

Results: The relation between the evaluated splitting mechanism differed between the static and the dynamic setting. Figure 8



(a) No Split (b) Split by Dist (c) Split If Necessary

Figure 7: Success ratio of Interdimensional SpeedyMurmurs for different initial timelock values (exponential capacities with expected value 200, exponential transaction values with $1/\lambda = 100$, Lightning topology): a) no splitting, b) splitting over nodes closest to receiver, c) splitting only if necessary

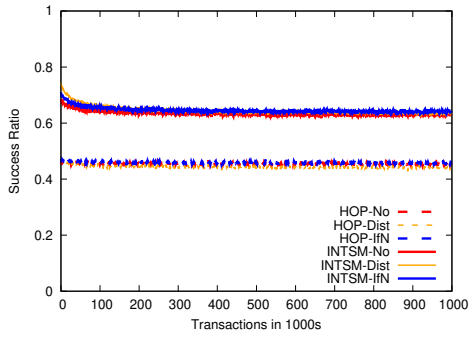


Figure 8: Dynamic setting: Success ratio for Interdimesional SpeedyMurmurs with dim $= 5$ and HopDistance over time, averaged over 1000 transactions each (exponential capacities with expected value 200, exponential transaction values with $1/\lambda = 100$, Lightning topology)

displays the average success ratio over time, with time being represented by the transaction number. In particular, Split_{Dist}, which selects the nodes closest to the receiver even if they have hardly any available funds, frequently utilized all available funds of a channel, resulting in a depleted channel. Thus, over time, the success ratio of Split_{Dist} dropped below that of the other splitting algorithms despite initially having the highest success ratio. In contrast, Split_{lfN}, which only splits if necessary, offered the highest success ratio.

G RANDOM SPLITTING

We also evaluated the impact of random splitting in contrast to more sophisticated splitting mechanisms. Due to space constraints, we excluded the description and results from the main body.

G.1 Algorithm Description

The key component of the algorithm is the function $randomSplit(v, k)$ that splits a value v at random such that the random variables X_i of the i-th partial value are identically distributed in $[0, v]$. Furthermore, we have $\sum_{i=1}^{k} X_i = v$, i.e., the total transaction value is split.

In the first step, the splitting algorithm applies $randomSplit$ with v being the total transaction value and k being the number of potential channels. However, such a random assignment of partial values can potential lead to partial values exceeding the capacities of some channels. As a consequence, the partial value assigned to each channel is reduced to the channel's capacity if it is too high. The respective channels is marked as being fully collateralized and the remaining value is recorded for being reassigned to another channel.

In each subsequent step, $randomSplit$ re-distributes the remaining value $\tilde{v}$ that has not been assigned to a channel over the number of channels that still have capacity left. The process continues until either the total transaction value has been assigned or none of the channels has capacity left. In the latter case, the payment fails.

G.2 Evaluation

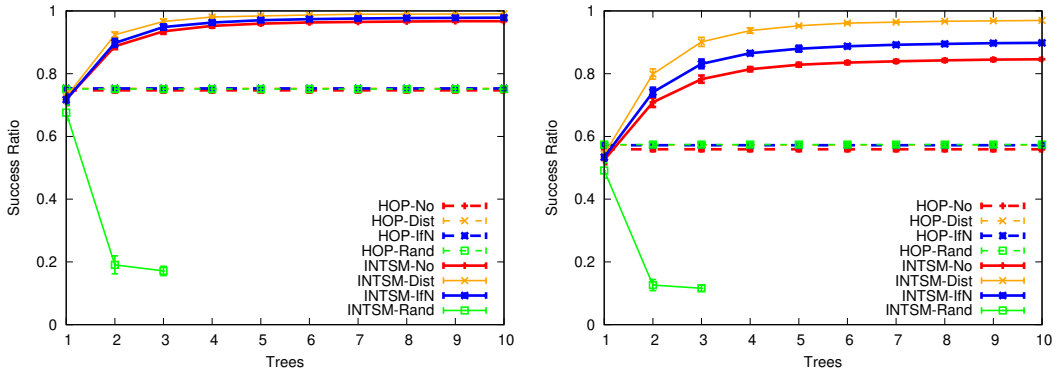
Our implementation of random splitting Split_{rand} included a threshold for a value to be split, i.e., values below the threshold were not split to reduce the computation time of the algorithm. The threshold for splitting was set to 1. As random splitting introduced a high computation overhead, so that we only simulated it for up to three trees.

Figure 9 is a version of Figure 3 that includes the results for random splitting with up to three trees. Random splitting decreased the success ratio drastically because if only one of the many shares did not reach the receiver, the payment failed. Hence, splitting too much and in a non-strategic manner did not improve the performance.

H ROUTING ALGORITHM: PSEUDOCODE

Figure 10 describes the generic routing algorithm.

Figures 11 and 12 display the two choices for Closer: Hop Distance and Interdimensional SpeedyMurmurs. Moreover, Figures 13–16 show the four splitting methods: No Split, Split by Distance, Split If Necessary, and Random Splitting.



(a) $1/\lambda = 50$ **(b)** $1/\lambda = 200$

Figure 9: Routing success for 8 routing algorithms when transaction value follows exponential distribution with $1/\lambda$ being a) 50 and b) 200 (exponential capacities with expected value 200, Lightning topology)

Route$_{\mathcal{G}}(v, P, R, excl, C_P)$

/ Get candidate set

$\{((P, U_j), c_j, d_j)\}_{j\in[k']} \leftarrow$ Closer(P, R, C_P)

/ Remove visited nodes

$\mathbf{M} \leftarrow \emptyset$

for $j \in [k']$ **do**

  if $((U_j \notin excl)$ **then**

    $\mathbf{M} := \mathbf{M} \cup \{((P, U_j), c_j, d_j)\}$

/ Split

$\{(\tilde{e}_j, \tilde{v}_j)\}_{j\in[k]} \leftarrow$ Split$(\mathbf{M}, v)$

return $\{(\tilde{e}_j, \tilde{v}_j)\}_{j\in[k]}$

Figure 10: Generic routing algorithm

Closer$_{HOP}(P, R, C_P)$

$Cand := \emptyset$/ candidate set

$dI \leftarrow d_{\mathcal{G}}(P, R)$

foreach $(P, U) \in \mathcal{E}$ **do**

  $d \leftarrow d_{\mathcal{G}}(U, R)$

  if $d < dI$ **then**/ Closer to R

    $Cand := Cand \cup ((P, U), C_P((P, U)), d)$

return $Cand$

Figure 11: Hop Distance for determining candidate channels for routing

Closer$_{INTSM}(P, R, C_P)$

$Cand := \emptyset$/ candidate set

foreach $(P, U) \in \mathcal{E}$ **do**

  for $i \in$ [dim] **do**/ Iterate over trees

    $d \leftarrow d_i(U, R)$

    $dI \leftarrow d_i(P, R)$

    if $d < dI$ **then**/ lower distance

      $dmin := d$

      foreach $j \in$ [dim] **do**

        $dj \leftarrow d_j(U, R)$

        if $dj < dmin$ **then**

          $dmin := dj$

      $Cand := Cand \cup ((P, U), C_P((P, U)), dmin)$

      break/ Stop loop as lower distance found

return $Cand$

Figure 12: Interdimensional SpeedyMurmurs for determining candidate channels for routing

$\text{Split}_{No}(\{(e_j, c_j, d_j)\}_{j\in[k]}, v)$

$B := \emptyset$/ Edges with nodes closest to R
$d := \infty$
for $j \in [k]$ **do**
  if $v \leq c_j$ **then**/ enough capacity
    if $d_j < d$ **then**/ new lowest distance
      $B := \emptyset$
      $d := d_j$
    if $d = d_j$ **then**/ lowest distance
      $B := B \cup \{e_j\}$
if $B = \emptyset$ **then**
  return $\perp$/ No channel has sufficient capacity without splitting
else
  $e \leftarrow_{\$} B$/ Choose a random channel if multiple channels with lowest distance
  return $\{(e, v)\}$

Figure 13: Routing without Splitting

$\text{Split}_{Dist}(\{(e_j, c_j, d_j)\}_{j\in[k]}, v)$

$sortByIncDistance(\{(e_j, c_j, d_j)\}_{j\in[k]})$/ Sort by increasing d_j
$sum := 0$/ capacity already assigned
$j := 0$/ index
$Res := \emptyset$/ edges and partial payment values
while $(sum < v) \wedge (j < k)$ **do**
  $v_j \leftarrow \min\{v - sum, c_j\}$
  $Res := Res \cup \{e_j, v_j\}$
  $sum := sum + v_j$
  $j := j + 1$
if $sum < v$ **then**
  return $\perp$/ value could not be split as total capacity insufficient
else return Res

Figure 14: Routing with splitting over channels at least distance to receiver

$\text{Split}_{IfN}(\{(e_j, c_j, d_j)\}_{j\in[k]}, v)$

$Res \leftarrow \text{Split}_{No}(\{(e_j, c_j, d_j)\}_{j\in[k]}, v)$
if $Res \neg \perp$ **then**
  return Res
$sortByDecCapacity(\{(e_j, c_j, d_j)\}_{j\in[k]})$/ Sort by decreasing c_j
$sum := 0$/ capacity already assigned
$j := 0$/ index
$Res := \emptyset$/ edges and partial payment values
while $(sum < v) \wedge (j < k)$ **do**
  $v_j \leftarrow \min\{v - sum, c_j\}$
  $Res := Res \cup \{e_j, v_j\}$
  $sum := sum + v_j$
  $j := j + 1$
if $sum < v$ **then**
  return $\perp$/ value could not be split as total capacity insufficient
else
  return Res

Figure 15: Routing with splitting only when necessary and then with a minimal number of splits

$\text{Split}_{Rand}(\{(e_j, c_j, d_j)\}_{j\in[k]}, v)$

$sum := 0$
$done := \emptyset$/ channels with capacity used
for $j \in [k]$ **do**
  $v_j := 0$/ init partial payment as 0
while $(|done| < k) \wedge (sum < v)$ **do**
  $count := k - |done|$
  $\{\tilde{v}_i\}_{i\in[count]} \leftarrow randomSplit(v - sum, count)$
  $sum := 0$
  $i := 0$
  for $j \in [k]$ **do**/ increase partial values
    if $i \notin done$ **do**
      $v_j \leftarrow \min\{v_j + \tilde{v}_i, c_j\}$
      $sum := sum + v_j$
      if $v_j = c_j$ **then**
        $done := done \cup \{j\}$/ channel has no capacity left
      $i := i + 1$
if $sum < v$ **then**
  return $\perp$/ value could not be split as total capacity insufficient
else
  return $\{(e_j, v_j)\}_{j\in[k]}$

Figure 16: Routing with random splits

I SECURITY ANALYSIS

Recall from Section 6 that our goal is to prove the security of our payment protocol for any routing algorithm from the set $\mathcal{R}$ which was defined as

$$\left\{ \mathsf{Route}(\mathsf{Closer}, \mathsf{Split}) \;\middle|\; \begin{array}{c} \mathsf{Closer} \in \{\mathsf{Closer}_{HOP}, \mathsf{Closer}_{INT-SM}\}, \\ \mathsf{Split} \in \{\mathsf{Split}_{No}, \mathsf{Split}_{Dist}, \mathsf{Split}_{IfN}\} \end{array} \right\}.$$

It is convenient for our proofs to also define two subsets of $\mathcal{R}$: the subset $\mathcal{R}_{HOP} \subset \mathcal{R}$ containing all routing algorithms with Closer = Closer_{HOP} and the $\mathcal{R}_{INT-SM} \subset \mathcal{R}$ containing all routing algorithms with Closer = Closer_{INT-SM}.

Note that the routing algorithm that uses random splitting (as described in Appendix G) is excluded from the set $\mathcal{R}$. The reason is that protocol using this routing algorithm does not satisfy the correctness property. In a nutshell, due to the loop-detection mechanism, it might happen that routing fails even is all parties behave honestly and all channels have enough coins.

Before we present the proof of Theorem 6.1, let us restate the theorem here for completeness.

Theorem 6.1. *Assume that Σ is an* EUF–CMA*-secure signature scheme, Ψ is an encryption scheme with message space $\mathbb{M}$, and $\mathcal{H}$ a preimage-resistant hash function with domain $\mathbb{P}$. For any* Route $\in \mathcal{R}$*, the protocol* Π_b(Route) *is a secure payment protocol with respect to the function* $\mathsf{Validate}_{\Sigma,\mathcal{H}}$.

If, in addition, Ψ and $\mathcal{H}$ are additively homomorphic, and $\mathbb{M} = \mathbb{Z}_N$, $\mathbb{P} = \mathbb{Z}_p$ for p, N coprime and $p < N$, then for any Route $\in \mathcal{R}$*, the protocol* Π_{ext}(Route) *is a secure payment protocol with respect to the function* $\mathsf{Validate}_{\Sigma,\mathcal{H}}$.

Since large parts of the proof are the same for both protocols, we present the proof for $\Pi \in \{\Pi_b, \Pi_{ext}\}$ and distinguish between the two protocols only when the proof steps differ. The structure of our proof is the following. Firstly, we prove that the protocol always terminates in Appendix I.1. Thereafter, in Appendix I.2, we prove that the monetary loss of an honest sender is bounded and no other honest party can ever lose coins, i.e., the security properties *bounded loss for the sender* and *balance neutrality*. Next, we prove that our protocol satisfies *atomicity*. This is done in Appendix I.3. Before we complete the proof by showing the *correctness* of our protocol in Appendix I.5, we prove two technical statements about our routing algorithms in Appendix I.4.

I.1 Termination

Fix a connected directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ for which $(P, Q) \in \mathcal{E} \Leftrightarrow (Q, P) \in \mathcal{E}$, a capacity function $\mathcal{C}$, parties $S, R \in \mathcal{V}$ s.t. $S \neq R$, a value $v \in \mathbb{R}^+$, a blockchain delay $\Delta \in \mathbb{N}$ and a ppt adversary $\mathcal{A}$. Let t_0 denote the first round of the protocol execution. As a first step, we show that the protocol Π terminates. In other words, it cannot happen that an honest party waits for infinitely many rounds to produce an output.

Claim 1. *The protocol Π terminates in finitely many rounds.*

Proof. The protocol description instructs an honest sender to terminate latest in round $T+\Delta+1$, were $T = t_0+1+|\mathcal{V}|\cdot(1+2\cdot(\Delta+1))$. An honest receiver terminates in round $t_0 + |\mathcal{V}| + 1$ if $b = 0$. If $b \neq 0$, then the receiver is in the process of unlocking all incoming payments and terminates within $\Delta + 1$ rounds. Thus, latest in round $t_0 + |\mathcal{V}| + \Delta + 2$, the receiver terminates.

It remains to discuss the termination for honest intermediaries (i.e., any other party $P \in \mathcal{V} \setminus \{S, R\}$). The protocol description instruct an intermediary I to stop forwarding payments after round $t_0 + |\mathcal{V}|$. Let T be the maximal time-lock on all outgoing payments in round $t_0 + |\mathcal{V}|$ (set to 0 if there is no such payment). Then latest in round $t_0 + |\mathcal{V}| + T$, it holds that $fw = \emptyset$ since for each conditional payment, it holds that either it was unlocked until this round or the intermediary has requested a refund. In both cases, the intermediary removes the conditional payment from the set fw. This implies that latest in round $t_0 + |\mathcal{V}| + T + 2(\Delta + 1)$ the intermediary terminates.

Since the protocol terminates once all honest parties terminate, this concludes the proof. □

Since the protocol terminates, after finitely many rounds the protocol execution returns as output

$$(\mathsf{Honest}, \mathsf{rec}, \mathcal{C}') \leftarrow \mathsf{EXEC}^{\mathcal{F}}_{\Pi,\mathcal{A}}(\mathcal{G}, \mathcal{C}, \Delta, S, R, v).$$

Let us now prove that this output satisfies the remaining security properties.

I.2 Balance neutrality

Our next step is to show that the monetary loss of an honest sender is bounded by the amount of coins that he wants to send, and none of the other honest parties can ever lose coins. To this end, we make the following simple but important observation about the payment channel ideal functionality $\mathcal{F} := \mathcal{F}(\mathcal{G}, \mathcal{C}, \Delta)$ that parties call in the protocol execution.

Observation 1. *The ideal functionality $\mathcal{F}$ never reduces coins of any party unless it receives the instruction* cPay *or* pay *from this party.*

Claim 2 (Bounded loss for the sender)**.** *It holds that*

$$S \in \mathsf{Honest} \Rightarrow \mathsf{net}_{C,C'}(S) \geq -v.$$

Proof. An honest sender S never sends any pay-message to $\mathcal{F}$. By our protocol description, the sender either does not send any cPay-messages (this happens if the receiver does not send a valid signature in the first protocol round, or if execution of Route outputs $\perp$), or sends k cPay-messages of values $(v_1, \ldots, v_k)$. Since the values $(v_1, \ldots, v_k)$ are returned by the routing algorithm Route, we know that $\sum_i v_i = v$ (recall the routing property (iii) on page 5). Hence, the total value of all the cPay-messages sent by S to $\mathcal{F}$ is at most v which by Observation 1 implies that $\mathsf{net}_{C,C'}(S) \geq -v$. □

As a next step, we show that balance neutrality for the intermediaries and the receiver. The balance neutrality for the receiver follows directly from Observation 1 since honest receiver never makes any payments. The balance neutrality for intermediaries is slightly more evolved. Firstly, we observe that no honest intermediary ever conditionally pays more coins than what he can potentionally receive. As a second step, we show that if an outgoing conditional payment is unlocked, the intermediary has a guarantee that the corresponding incoming payment is unlocked as well. This follows from the fact that (i) the difference between time-lock is sufficiently large and (ii) the intermediary can compute a witness

for the incoming payment from the revealed witness of the outgoing payment. The part (ii) is trivial form the basic protocol and for the extended protocol follows from the homomorphic property of $\mathcal{H}$. To this end, we state the following auxiliary lemma about additively homomorphic functions.

Lemma I.1. *Let $\mathcal{H}: \mathbb{P} \to \mathbb{H}$ be an additively homomorphic function. Let $h \in \mathbb{H}$, $x_i \in \mathbb{P}$ and let us define $h_i := h + \mathcal{H}(x_i)$. Then for every $x' \in \mathbb{P}$ s.t. $\mathcal{H}(x') = h_i$, it holds that $\mathcal{H}(x' - x_i) = h$.*

Proof. By definition of h_i, we know that $h = h_i - \mathcal{H}(x_i)$, hence

$$h = h_i - \mathcal{H}(x_i) = \mathcal{H}(x') - \mathcal{H}(x_i) \overset{\text{hom.}}{=} \mathcal{H}(x' - x_i).$$

□

Claim 3 (Balance neutrality)**.** *It holds that*

$$P \in \mathsf{Honest} \setminus \{S\} \Rightarrow \mathsf{net}_{C,C'}(P) \geq 0.$$

Proof. Let us first consider the case $P = R$. Since an honest receiver R never sends any pay-message or cPay-message to $\mathcal{F}$, by Observation 1, $\mathsf{net}_{C,C'}(R) \geq 0$.

Let us fix an arbitrary intermediary $I \in \mathsf{Honest} \setminus \{S, R\}$. An honest I never sends any pay-message to $\mathcal{F}$ and never sends a cPay-message to $\mathcal{F}$ without receiving a cPaid-messages from $\mathcal{F}$ before. In other words, every outgoing conditional payment is triggered by an incoming conditional payment.

Assume now that I receives a conditional payment of value v, condition h and time-lock T. Then I executes the routing algorithm Route on input value v and obtains k values $(v_1, \ldots, v_k)$ such that $\sum_j v_j = v$. Thereafter, the intermediary executes the algorithm HLocks on input h and k and obtains k hash values $(h_1, \ldots h_k)$. For every $i \in [k]$, the intermediary sends a cPay-message of value v_i, condition h_i and time-lock $T' := T - 2 \cdot (\Delta + 1)$. This implies that an honest I never conditionally pays more coins than what he can conditionally receive. As a next step, we prove that if one of the outgoing payments is completed, i.e., I pays v_i coins, then I has the guarantee of unlocking the corresponding incoming payments, i.e. receive v coins.

First we show that if I learns a preimage of at least one of the hash values $h_1, \ldots, h_k$, then he can compute a preimage for h.
$\underline{\Pi = \Pi_b}$: Since for every $i \in [k]$ we have $h_i = h$, the statement trivially holds.
$\underline{\Pi = \Pi_{ext}}$: For every $i \in [k]$, the value h_i is computed as $h + \mathcal{H}(x_i)$. Upon learning x' such that $h_i = \mathcal{H}(x')$, I computes $x' - x_i$ which by Lemma I.1 is a preimage of h.
The latest point at which the intermediary can learn x' from an outgoing payment is in round $T' + (\Delta + 1)$. Submission of a witness for the corresponding incoming payments takes at most $(\Delta + 1)$ rounds. Hence, I has the guarantee that the unlocking request is executed by $\mathcal{F}$ latest in round $T'' := T' + 2 \cdot (\Delta + 1)$. Since honest intermediary sets T' during the routing phase as $T' := T - 2(\Delta + 1)$, we have $T'' = T$. Since a conditional payment can be refunded by the payer earliest in round $T + 1$, the intermediary has a guarantee of successful unlocking.

The intermediary makes no outgoing conditional payments after round $t_0 + |\mathcal{V}|$. Hence, the size of the set of all outgoing payments fw can only decrease from this point on. An outgoing conditional payment is removed from fw when (i) the payment was unlocked or (ii) I requests a refund. If an outgoing payment is unlocked, then within the next $(\Delta+1)$ rounds the corresponding incoming payment is unlocked as we showed above. By our protocol description, I requests a refund for a conditional payment in the round when its time-lock expires. $\mathcal{F}$ processes the refund request within $(\Delta + 1)$ rounds and accepts it unless the conditional payment was already unlocked in which case we fall into case (i). This means that I has to unlock the corresponding incoming payments, which takes at most $(\Delta + 1)$ rounds. Hence, it takes at most $2(\Delta + 1)$ rounds before I settles a payment that was removed from fw.

Recall that by the protocol description, an honest intermediary does not terminate before round $t_0 + |\mathcal{V}|$. After this round, I waits for the first round when the set of outgoing payments fw is empty. Once this happens, I waits $2(\Delta + 1)$ rounds and only then produces an output and terminates. Hence, in the round when I terminates, all forwarded payments were settled.

□

I.3 Atomicity

We now show that if an honest sender loses coins, then he holds a valid receipt, i.e., a triple (h_R, σ, x_R), where $h_R = \mathcal{H}(x_R)$ and $\mathsf{Vrfy}_{pk_R}(S, R, v, h_R, \sigma) = 1$. Our argumentation in the proof is essentially the following. Since we assume that the sender lost coins, he had to make at least one conditional payment which was unlocked. In the basic protocol, all time-locks are set to h_R, hence one unlocked payment implies knowledge of the desired preimage x_R. In the extended protocol, condition of an outgoing payment is set to $h_i = h_R + \mathcal{H}(x_i)$. By the homomorphic property of $\mathcal{H}$ (Lemma I.1), we know that if $h_i = \mathcal{H}(x')$, then $x' - x_i$ is a preimage of h_R. Let us now state and proof the atomicity for the sender formally.

Claim 4 (Atomicity for the sender)**.** *It holds that*

$$S \in \mathsf{Honest} \wedge \mathsf{net}_{C,C'}(S) < 0 \Rightarrow \mathsf{Validate}_{\Sigma,\mathcal{H}}(S, R, v, \mathsf{rec}) = 1.$$

Proof. The assumption that the sender is honest and lost some coins implies that in the initialization phase, the sender received a statement from the receiver of the form (h_R, σ) for some hash value h_R and σ being a valid signature of R on (S, R, v, h_R). This implication follows from the fact that if the sender does not receive such a valid statement, he immediately terminates the protocol without ever making any payment. By Observation 1 we know that in such a case, the sender never loses any coins contradiction the assumption of this lemma.

After receiving a valid statement from the receiver, the sender makes (multiple) conditional payments via the functionality $\mathcal{F}$, all of which have the same time-lock T. The sender does not send any further cPay-message or pay-message to $\mathcal{F}$ and does not produce an output until all conditional payments are either unlocked or refunded. This follows from the fact that in round T, i.e., in a round when all time-locks expire, the sender instructs the ideal functionality to refund all conditional payments that were not unlocked. This process takes at most $\Delta + 1$ rounds and hence in round $T + \Delta + 1$ when the sender terminates, all conditional payments are indeed settled. Since we assume that the sender lost coin, at least one of the conditional payments must have been unlocked (and not refunded).

$\underline{\Pi = \Pi_b}$: In the base protocol, all conditional payments had the same hash-lock h_R. The fact that at least one of the conditional

payments was unlocked implies that the sender must have learned a value x'_R such that $\mathcal{H}(x'_R) = h_R$. In this case, the sender outputs a receipt $\text{rec} = (h_R, \sigma, x'_R)$. Since σ is a valid signature of the receiver R and $\mathcal{H}(x'_R) = h_R$, it holds that $\text{Validate}_{\Sigma,\mathcal{H}}(S, R, v, \text{rec}) = 1$.

$\Pi = \Pi_{ext}$: In the extended protocol, each conditional payment has a different hash-lock produced by the algorithm HLocks_{ext}. Recall that each hash-lock h_i is computed as $h_R + \mathcal{H}(x_i)$ for a randomly chosen x_i. The fact that at least one of the conditional payments was unlocked implies that the sender must have learned a preimage of at least one h_i. In other words, the sender learns x' s.t. $\mathcal{H}(x') = h_i$. According to our protocol description, the sender executes $\text{Wit}(x', x_i)$, returning $x' - x_i$, which by Lemma I.1 is a preimage of h_R. In this case, the sender outputs a receipt $\text{rec} = (h_R, \sigma, x' - x_i)$. Since σ is a valid signature of the receiver R and $\mathcal{H}(x' - x_i) = h_R$, it holds that $\text{Validate}_{\Sigma,\mathcal{H}}(S, R, v, \text{rec}) = 1$. □

Next, we argue about the atomicity for the receiver, saying that if the sender holds a valid receipt, then the receiver earned at least v. The intuition about this proof is the following. Unforgeability of the signature scheme guarantees that the only valid signature of receiver that the sender could output is the signature on (S, R, v, h_R) produced by the receiver. Hence, the valid receipt must contain a preimage of h_R. By preimage resistance of $\mathcal{H}$, the sender is not able to output such value without R revealing x_R. We finalize the proof by showing that the receiver does not reveal x_R unless he has a guarantee of unlocking v coins and does not terminate before all the payments are unlocked.

Claim 5 (Atomicity for the receiver)**.** *It holds that*

$$R \in \text{Honest} \wedge \text{Validate}_{\Sigma,\mathcal{H}}(S, R, v, \text{rec}) = 1 \Rightarrow \text{net}_{C,C'}(R) \geq v$$

with overwhelming probability.

Proof. Assume that $R \in \text{Honest}$ and $\text{Validate}_{\Sigma,\mathcal{H}}(S, R, v, \text{rec}) = 1$. Let us parse rec as (h', σ', x'). Since rec is a valid receipt, by the definition of the validation function $\text{Validate}_{\Sigma,\mathcal{H}}$ we know that σ' is a valid signature of the receiver on the tuple (S, R, v, h') and $h' = \mathcal{H}(x')$. The EUF–CMA-security of the signature scheme Σ guarantees that (with overwhelming probability) the signature σ' had to be produce by the receiver. By our protocol, the receiver computes only one signature. Namely, in the first round of the protocol, where he randomly samples x_R, computes $h_R := \mathcal{H}(x_R)$ and produces a signature $\sigma \leftarrow \text{Sign}_{sk_R}(S, R, v, h_R)$. Hence, with overwhelming probability $h' = h_R$.

The preimage resistance of the hash function $\mathcal{H}$ guarantees that the probability of sender outputting x' without the receiver revealing his secret preimage x_R is negligible. Hence, it remains to show that if the receiver R reveals the secret preimage x_R, then $\text{net}_{C,C'}(R) \geq v$.

By our protocol description, the receiver reveals x_R only once the following conditions are satisfied: (i) the receiver received conditional payments of total value v, (ii) the receiver knows the witness for all conditional payments, and (iii) in the round when the last conditional payment is received, let us denote it t, the time-lock of all conditional payments is at least $t + \Delta + 1$. Since unlocking of a conditional payment takes at most $\Delta + 1$ rounds, the aforementioned properties guarantee that if the receiver starts unlocking the payments in round t, all conditional payments will be successfully unlocked latest in round $t + \Delta + 1$. Since the receiver waits for $\Delta + 1$ before producing an output and terminating, we have $\text{net}_{C,C'}(R) \geq v$ as we wanted to prove. □

I.4 Correctness of routing algorithms

In this section we prove two technical lemmas about our routing algorithms from the set $\mathcal{R}$. Recall that this set contains all routing algorithms defined in Section 4.

We first consider the routing algorithms using Closer_{HOP} to determine candidate nodes for routing, i.e., algorithms from the set $\mathcal{R}_{HOP}$. On a high level, the lemma says that if a party P executes such a routing algorithm given a set *excl* that contains only nodes that are strictly further away from the receiver than P, then the routing algorithm never outputs $\perp$. This statement holds under the assumption that all channels in the graph are sufficiently funded. Moreover, we prove that all nodes that the routing algorithm outputs are strictly closer to the receiver than P. Hence, if nodes output by the routing algorithm add P to the set *excl*, this set again satisfies the assumptions of our lemma.

For convenience, we define a set $\mathcal{S}^{P,R}_{HOP}$ for every $P, R \in \mathcal{V}$ as

$$\mathcal{S}^{P,R}_{HOP} := \{Q \in \mathcal{V} \mid d_{\mathcal{G}}(Q, R) > d_{\mathcal{G}}(P, R)\}.$$

Lemma I.2 (Hop-distance routing). *Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a connected directed graph, $P, R \in \mathcal{V}$ be two nodes such that $P \neq R$ and C be a capacity function of $\mathcal{G}$ such that $C(e) \geq v$ for every $e \in \mathcal{E}_P$. For every set $excl \subseteq \mathcal{S}^{P,R}_{HOP}$, and every routing algorithm $\text{Route} \in \mathcal{R}_{HOP}$, the output of $\text{Route}_{\mathcal{G}}(v, P, R, excl, C_P)$ is never equal to $\perp$. Moreover for every returned edge (P, I) it holds that $excl \cup \{P\} \subseteq \mathcal{S}^{I,R}_{HOP}$.*

Proof. Let *Cand* be the set returned by Closer_{HOP}. By the description of Closer_{HOP}, we have

$$Cand := \left\{((P, I), v, d_{\mathcal{G}}(I, R)) \mid (P, I) \in \mathcal{E} \wedge d_{\mathcal{G}}(P, R) > d_{\mathcal{G}}(I, R)\right\}$$

In other words, the set *Cand* contains all neighbors of P who are closer to the receiver. As a first step, we prove that $Cand \neq \emptyset$.

Since $\mathcal{G}$ is connected, there exists at least one path between P and R in $\mathcal{G}$. Let us consider one shortest such path and let $(P, I_1, \ldots, I_\ell, R)$ be the nodes associated with this path. Then $(P, I_1) \in \mathcal{E}$ and $d_{\mathcal{G}}(P, R) = d_{\mathcal{G}}(I_1, R) + 1$ hence $((P, I_1), v, d_{\mathcal{G}}(I_1, R)) \in Cand$.

As a next step, the routing algorithm applies the loop-detection filter. Formally, a set $\mathbf{M}$ is defined as

$$\mathbf{M} := \{((P, I), v, d_{\mathcal{G}}(I, R)) \in Cand \mid I \notin excl\}.$$

Since $d_{\mathcal{G}}(P, R) > d_{\mathcal{G}}(I_1, R)$, we know that $I_1 \notin excl$ and hence $((P, I_1), v, d_{\mathcal{G}}(P, I_1)) \in \mathbf{M}$. This in particular means that $\mathbf{M} \neq \emptyset$.

The routing algorithm now executes Split on input $\mathbf{M}$ and v. By the specification of the algorithms $\text{Split} \in \{\text{Split}_{No}, \text{Split}_{IfN}, \text{Split}_{Dist}\}$, it is easy to see that since the capacity of all edges is at least v, no matter which of the splitting algorithms was used, the output is exactly one pair $((P, I), v)$. This completes the proof of the first part of the lemma.

It remains to argue that $d_{\mathcal{G}}(I, R) < d_{\mathcal{G}}(P, R)$. This follows from the fact that $((P, I), v)$ is such that $((P, I), v, d_{\mathcal{G}}(P, I)) \in \mathbf{M} \subseteq Cand$. □

Let us now prove an analogous lemma for routing algorithms that use Closer_{INT-SM} to determine candidate nodes for routing, i.e., algorithms from the set $\mathcal{R}_{INT-SM}$. Recall that in contrast to Closer_{HOP}, the algorithm Closer_{INT-SM} does not use hop-distance $d_{\mathcal{G}}$ for measuring the distance of nodes from the receiver. Instead, it considers the distance of nodes in several spanning trees $ST_1, \ldots, ST_{\dim}$ of $\mathcal{G}$. The restriction we put on the input set *excl* is hence slightly different.

The lemma requires that the set *excl* contains only nodes whose minimal distance from the receiver (minimum is taken over all spanning trees) is strictly larger that the minimal distance of the party P from the receiver (again, minimum is taken over all spanning trees). Under this condition and assuming that all channels are sufficiently funded, the routing algorithm executed by P never outputs $\perp$. Moreover, we prove that all nodes that the routing algorithm outputs have a minimal distance from the receiver strictly lower than P. For convenience, we define a set $\mathcal{S}^{P,R}_{INT-SM}$ for every $P, R \in \mathcal{V}$ as

$$\mathcal{S}^{P,R}_{INT-SM} := \left\{ Q \in \mathcal{V} \;\middle|\; \min_{i\in[\dim]} d_i(Q,R) > \min_{i\in[\dim]} d_i(P,R) \right\},$$

where $d_i := d_{ST_i}$ is the distance function in the spanning tree ST_i.

Lemma I.3 (Routing using spanning trees). *Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a connected directed graph, $P, R \in \mathcal{V}$ be two nodes such that $P \neq R$ and C be a capacity function of $\mathcal{G}$ such that $C(e) \geq v$ for every $e \in \mathcal{E}_P$. For every set of spanning trees $ST := \{ST_1, \ldots, ST_{\dim}\}$ of $\mathcal{G}$, every set $excl \subseteq \mathcal{S}^{P,R}_{INT-SM}$, and every routing algorithm $\text{Route} \in \mathcal{R}_{INT-SM}$, the output of $\text{Route}_{\mathcal{G}}(v, P, R, excl, C_P)$ is never equal to $\perp$. Moreover for every returned edge (P, I) it holds that $excl \cup \{P\} \subseteq \mathcal{S}^{I,R}_{INT-SM}$.*

Proof. Let *Cand* be the set returned by Closer_{INT-SM}. By the description of Closer_{INT-SM}, we have

$$Cand := \left\{ ((P,I), v, d^I) \;\middle|\; \begin{array}{c} (P,I) \in \mathcal{E} \wedge d^I := \min_{i\in[\dim]} d_i(I,R) \\ \wedge \\ \exists i \in [\dim]\ d_i(P,R) > d_i(I,R) \end{array} \right\}$$

In other words, the set *Cand* contains all neighbors of P who are closer to the receiver in at least one spanning tree. As a first step, we prove that $Cand \neq \emptyset$.

Since $\mathcal{G}$ is connected, there exists at least one path between P and R in $\mathcal{G}$, which implies that there exists exactly one path between P and R in every spanning tree. Hence, for every spanning tree ST_i there exists $I \in \mathcal{V}$ such that $d_i(P,R) = d_i(I,R) + 1$ implying that

$$\forall i \in [\dim]\ \exists ((P,I), v, d^I) \in Cand \text{ s.t. } d_i(P,R) > d_i(I,R). \quad (1)$$

From eq. (1) we have that $Cand \neq \emptyset$.

As a next step, the routing algorithm applies the loop-detection filter. Formally, a set $\mathbf{M}$ is defined as

$$\mathbf{M} := \{((P,I), v, d^I) \in Cand \mid I \notin excl\}.$$

We show that $\mathbf{M} \neq \emptyset$. Let j denote the index of the spanning tree in which P is closest to R; formally, $j := \arg\min_{i\in\dim} d_i(P,R)$. By eq. (1), we know that there exists $((P,J), v, d^J) \in Cand$ s.t. $d_j(P,R) > d_j(J,R)$ and hence

$$\min_{i\in[\dim]} d_i(P,R) > \min_{i\in[\dim]} (J,R). \quad (2)$$

By the definition of the set *excl*, this implies that $J \notin excl$ and hence $((P,J), v, d^J) \in \mathbf{M}$.

The routing algorithm now executes the splitting algorithm Split on input $\mathbf{M}$ and v. From the specification of the algorithm $\text{Split} \in \{\text{Split}_{No}, \text{Split}_{IfN}, \text{Split}_{Dist}\}$, it is easy to see that since all channels have capacity at least v, exactly one pair $((P,I), v)$ is output no matter which of the splitting algorithms we consider. This in particular implies that the routing algorithm never outputs $\perp$.

It remains to prove that $\min_{i\in[\dim]} d_i(I,R) < \min_{i\in[\dim]} d_i(P,R)$. For each $\text{Split} \in \{\text{Split}_{No}, \text{Split}_{IfN}, \text{Split}_{Dist}\}$, it holds that the output $((P,I), v)$ satisfies the following two conditions: (i) $((P,I), v, d^I) \in \mathbf{M}$ and (ii) d^I is such that for every $((P,I'), v, d^{I'}) \in \mathbf{M}$ it holds that $d^I \leq d^{I'}$. We already proved that $((P,J), v, d^J) \in \mathbf{M}$. Hence

$$\min_{i\in[\dim]} d_i(I,R) = d^I \leq d^J = \min_{i\in[\dim]} d_i(J,R) \overset{(2)}{<} \min_{i\in[\dim]} d_i(P,R)$$

which concludes the proof. □

A simple corollary of the above two lemmas is that if any of the routing algorithms is executed on the set $excl = \emptyset$, which is exactly what the sender does it our protocol, then the routing algorithm never fails.

Corollary I.4 (Routing for the sender). *Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a connected directed graph, $S, R \in \mathcal{V}$ be two nodes such that $S \neq R$ and C be a capacity function of $\mathcal{G}$ such that $C(e) \geq v$ for every $e \in \mathcal{E}_S$. For every routing algorithm $\text{Route} \in \mathcal{R}$, the output of $\text{Route}_{\mathcal{G}}(v, S, R, \emptyset, C_S)$ is never equal to $\perp$.*

Proof. If $\text{Route} \in \mathcal{R}_{HOP}$, then the corollary follows directly from Lemma I.2 since for every S, R it holds that $\emptyset \subseteq \mathcal{S}^{S,R}_{HOP}$. Analogously, if $\text{Route} \in \mathcal{R}_{INT-SM}$, then the corollary follows directly from Lemma I.3 since for every S, R it holds that $\emptyset \subseteq \mathcal{S}^{S,R}_{INT-SM}$. □

Another simple corollary of the two lemmas is that if the nodes output by the routing algorithm $\text{Route}(v, P, R, excl, C_P)$ add the node P to the set *excl* and execute the routing algorithm themselves, then the routing algorithm never fails. In other words, the lemma can be use inductively. Let us stress that this is exactly what happens at intermediary nodes in our protocol.

Corollary I.5 (Routing for intermediaries). *Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a connected directed graph, $P, R \in \mathcal{V}$ be two nodes such that $P \neq R$ and C be a capacity function of $\mathcal{G}$ such that $C(e) \geq v$ for every $e \in \mathcal{E}_P$. Let $(\text{Route}, \mathcal{S}) \in \mathcal{R}_{HOP} \times \{\mathcal{S}^{P,R}_{HOP}\}$ or $(\text{Route}, \mathcal{S}) \in \mathcal{R}_{INT-SM} \times \{\mathcal{S}^{P,R}_{INT-SM}\}$ and $excl \subseteq \mathcal{S}$. For every pair $((P,I), v')$ returned by $\text{Route}(v, P, R, excl, C_P)$ and ever capacity function C' of $\mathcal{G}$ such that $C(e) \geq v'$ for $e \in \mathcal{E}_I$, it holds that $\text{Route}(v', I, R, excl \cup \{P\}, C'_I) \neq \perp$.*

Proof. If $(\text{Route}, \mathcal{S}) \in \mathcal{R}_{HOP} \times \{\mathcal{S}^{P,R}_{HOP}\}$, then the corollary follows directly from Lemma I.2 since for every edge (P,I) returned by $\text{Route}(v, P, R, excl, C_P)$, it holds that $excl \cup \{P\} \subseteq \mathcal{S}^{I,R}_{HOP}$. Analogously, if $(\text{Route}, \mathcal{S}) \in \mathcal{R}_{INT-SM} \times \{\mathcal{S}^{P,R}_{INT-SM}\}$, then the corollary follows directly from Lemma I.3 since for every edge (P,I) returned by $\text{Route}(v, P, R, excl, C_P)$, it holds that $excl \cup \{P\} \subseteq \mathcal{S}^{I,R}_{INT-SM}$. □

I.5 Correctness

Finally, we need to prove that our protocol satisfies correctness meaning that if all parties are honest and all channels in the network have enough coins, then the payment succeeds. The main steps of our proof are the following. Since both sender and receiver are honest, the sender initiates conditional payments of total value v. Using the statements from the previous section, we argue that no matter which routing algorithm Route $\in \mathcal{R}$ we consider, the routing never fails. Moreover, we show that the initial time-out set by the sender is sufficient to guarantee that all partial payments reach the receiver in time and hence the receiver starts unlocking payments. Since the total amount of coins in the system cannot increase (parties cannot create coins), we complete the proof by applying the balance neutrality and bounded loss for the sender. We now state and proof the correctness of our protocol formally.

Claim 6 (Correctness). *If $\mathcal{V}$ = Honest and for every $e \in \mathcal{E}$ it holds that $C(e) \geq v$, then it holds that $\mathsf{net}_{C,C'}(S) = -v$, $\mathsf{net}_{C,C'}(R) = v$ and $\mathsf{net}_{C,C'}(P) = 0$ for every $P \in \mathcal{V} \setminus \{S, R\}$.*

Proof. Since the receiver R is honest, the sender S receives a valid signature σ on the statement (S, R, v, h_R) from the receiver R in the round t_0+1. This means that the sender execute the algorithm Route on input $excl = \emptyset$. By Corollary I.4, we know that the routing algorithm returns $\{(I_i, v_i)\}_{i\in[k]}$ and hence the sender initiates k conditional payments – each of them with set $excl := \{S\}$.

Assume for now that at least one of the conditional payments is completed and hence $\mathsf{net}_{C,C'}(S) < 0$. By Claim 4, this implies that the sender outputs a valid receipt. Since the receiver is honest and the sender outputs a valid receipt, by Claim 5 we know that $\mathsf{net}_{C,C'}(R) \geq v$. Moreover, by Claim 3, we know that none of the intermediaries can lose coins, i.e., $\mathsf{net}_{C,C'}(I) \geq 0$ for every $I \in \mathcal{V} \setminus \{S, R\}$. Since the PCN functionality $\mathcal{F}$ does not allow any party to create coins, it must hold that $\sum_P \mathsf{net}_{C,C'}(P) \leq 0$. Hence, we have

$$\mathsf{net}_{C,C'}(S) \leq -\left(\mathsf{net}_{C,C'}(R) + \sum_{I\in\mathcal{V}\setminus\{S,R\}} \mathsf{net}_{C,C'}(P)\right) \leq -v.$$

The bounded loss for the sender, Claim 2, guarantees that $\mathsf{net}_{C,C'}(S) \geq -v$, hence it must holds that $\mathsf{net}_{C,C'}(S) = -v$. This in turn implies that $\mathsf{net}_{C,C'}(R) = v$ and $\mathsf{net}_{C,C'}(I) = 0$ for any other party $I \in \mathcal{V} \setminus \{S, R\}$.

What remains to prove is that at least one of the conditional payments made by the sender is unlocked before the protocol terminates. Since the sender terminates only once all outgoing payments are settled, it suffices to prove that at least once conditional payment is unlocked and not refunded.

Since the sender is honest, the conditional payments are made in the round $t_0 + 1$ and the time-lock of all of them are set to $T = t_0 + 1 + n(1 + 2(\Delta + 1))$, where $n = |\mathcal{V}|$. It takes one round before the neighbors of S are informed about the conditional payment. Each intermediary, upon being informed about the conditional payment, executes the routing algorithm Route which by Corollary I.5 never fails. Hence, the intermediary decreases the time-lock by $2(\Delta + 1)$ and forwards the payment immediately to parties outputs by the algorithm Route. Since the length of the longest path between the sender and receiver is upper bounded by n, the receiver is informed about all conditional payments latest in round $t_1 := t_0 + 1 + n$. Since the number of intermediaries on each path is upper bounded by $n-1$, the minimal time-lock $T_{\min}$ of incoming conditional payments of the receiver can be lower bounded as

$$T_{\min} \geq T - (n-1)\cdot 2(\Delta+1) = t_0 + 1 + n + 2(\Delta+1))$$

Since $T_{\min} - t_1 > (\Delta + 1)$, the receiver starts unlocking all payments if he knows a witness for all of them.

$\Pi = \Pi_b$: In the base protocol, all conditional payments have the same hash-lock h_R. Since h_R was computed by the receive in the first round as $\mathcal{H}(x_R) = h_R$, R trivially knows the witness x_R.

$\Pi = \Pi_{ext}$: In the extended protocol, a conditional payment has a hash-lock h_i and an attached ciphertext c_i. By our protocol description, the receiver executes $\mathsf{WitR}_{ext}(c_i, sk, x_R, h_i, excl)$ in order to find the preimage of h_i. The algorithm WitR_{ext} instructs the receiver to compute $x' := x_R +_{\mathbb{M}} \mathsf{Dec}_{sk_R}(c_i)$ and loop over all $j \leq |excl|$ to check if $h_i = \mathcal{H}(x' +_{\mathbb{Z}} j\cdot N \mod p)$. We need to argue that if all parties behaved honestly, the receiver will find such j and hence the preimage of h_i. Since all parties in the system are honest, by the additive homomorphism of $\mathcal{H}$ we know that

$$h_i = \mathcal{H}(x_R) +_{\mathbb{H}} \mathcal{H}(x_i^{(1)}) +_{\mathbb{H}} \ldots +_{\mathbb{H}} \mathcal{H}(x_i^{(\ell)}) \tag{3}$$

$$= \mathcal{H}\left(x_R +_{\mathbb{P}} x_i^{(1)} +_{\mathbb{P}} \ldots +_{\mathbb{P}} x_i^{(\ell)}\right), \tag{4}$$

for $\ell = |excl|$. Moreover, by the additive homomorphism of the encryption scheme we know that

$$c_i = \mathsf{Enc}_{pk_R}(0) +_{\mathbb{C}} \mathsf{Enc}_{pk_R}(x_i^{(1)}) +_{\mathbb{C}} \cdots + \mathsf{Enc}_{pk_R}(x_i^{(\ell)}) \tag{5}$$

$$= \mathsf{Enc}_{pk_R}\left(x_i^{(1)} +_{\mathbb{M}} \ldots +_{\mathbb{M}} x_i^{(\ell)}\right). \tag{6}$$

Since $\mathbb{M} = \mathbb{Z}_N$ we have

$$x_R +_{\mathbb{Z}} x_i^{(1)} +_{\mathbb{Z}} \cdots +_{\mathbb{Z}} x_i^{(\ell)} \leq (\ell+1)\cdot N. \tag{7}$$

Recall that $x' := x_R +_{\mathbb{M}} \mathsf{Dec}_{sk_R}(c_i)$. From (6) and (7), we know that there exists $j \leq \ell$ such that

$$x_R +_{\mathbb{Z}} x_i^{(1)} +_{\mathbb{Z}} \cdots +_{\mathbb{Z}} x_i^{(\ell)} = x' +_{\mathbb{Z}} j\cdot N.$$

Since $\mathbb{P} = \mathbb{Z}_p$, this implies that

$$x' +_{\mathbb{Z}} j\cdot N \mod p = x_R +_{\mathbb{P}} x_i^{(1)} +_{\mathbb{P}} \ldots +_{\mathbb{P}} x_i^{(\ell)}.$$

and hence $x' +_{\mathbb{Z}} j\cdot N \mod p$ is a preimage of h_i which we wanted to prove.

We showed that all partial payments arrive to the receiver. Since the receiver can compute all witnesses and he has enough time to unlock all of them, he initiate the unlocking phase. This implies that the intermediaries that conditionally paid to R lose coins. By balance neutrality, Claim 3, we know that each intermediary has the guarantee of unlocking the corresponding incoming payment. Hence, the conditional payments of the sender are unlocked which completes the proof.

□

J UNLINKABILITY OF PARTIAL PAYMENTS

On a high level, our unlinkability definition guarantees the following. Assume that an honest party (sender or an intermediary) splits a payment of v coins into k partial payments $(v_1, \ldots, v_k)$ which he routes over k different neighbors $P_1, \ldots, P_k$. Even if all of these

neighbors collude, they cannot decide whether the conditional payments are parts of the same payment or if k independent payments of values $(v_1, \ldots, v_k)$ were sent. Since the partial payments are unlinkable right after the split takes place, further forwarding of the payments does not influence their linkability. Hence, our definition also captures the case when the colluding parties appear later in the partial payment paths (and not right after the splitting). Let us recall the formal definition as stated in Section 6.

Definition 6.1 (Unlinkability). Let Ψ be an encryption scheme with plaintext space $\mathbb{M}$ and let $\mathcal{H} : \mathbb{P} \to \mathbb{H}$ be a hash function with $\mathbb{P} \subseteq \mathbb{M}$. Algorithm $\mathsf{HLocks}^{\mathcal{H},\Psi}$ produces *unlinkable conditions* if for every ppt adversary $\mathcal{A}$, every $x \in \mathbb{P}$, $y \in \mathbb{M}$ and $k \in \mathbb{N}$, we have $\Pr[\mathsf{GameLink}_{\mathcal{A},\mathsf{HLocks}^{\mathcal{H},\Psi}}(x, y, k, n) = 1] \leq \frac{1}{2} + \mathsf{negl}(n)$, where the game GameLink is defined as follows:

$\mathsf{GameLink}_{\mathcal{A},\mathsf{HLocks}^{\mathcal{H},\Psi}}(x, y, k, n)$

$b \leftarrow_\$ \{0, 1\}$

$(pk, sk) \leftarrow \mathsf{Gen}(1^n)$, $h := \mathcal{H}(x)$, $c \leftarrow \mathsf{Enc}_{pk}(y)$

if $b = 1$ **then** $\{(h_j, c_j, x_j)\}_{j\in[k]} \leftarrow \mathsf{HLocks}^{\mathcal{H},\Psi}(h, c, k, pk)$

else foreach $j \in [k]$ **do** $x_j \leftarrow_\$ \mathbb{P}$, $h_j := \mathcal{H}(x_j)$, $c_j := \mathsf{Enc}_{pk}(x_j)$

$b' \leftarrow \mathcal{A}(x, y, \{(h_j, c_j)\}_{j\in[k]}, pk)$

return $b = b'$

Let us briefly discuss why the formal definition captures the unlinkability property discussed above on high level. In case $b = 0$, the game GameLink generates $\{(h_j, c_j)\}_{j\in[k]}$ exactly as a sender would do when sending k independent payments. Namely, for each of the k payments, the sender would sample a random preimage x_i and use the hash value $h_i := \mathcal{H}(x_i)$ as the hash-lock. Moreover, the sender would attach the ciphertext $c_i := \mathsf{Enc}_{pk}(x_i)$ to the conditional payment, where pk is the public key of the receiver, to ensure that the receiver can unlock the conditional payment. In case $b = 1$, the game GameLink generates $\{(h_j, c_j)\}_{j\in[k]}$ exactly as a sender, who sends a single payment splitted into k partial payments using the procedure HLocks.

We emphasize that by quantifying in our definition over all x and y we model that malicious neighbours of a party splitting a payment (sender or intermediary) may learn these values in our protocol. For the value y this is, e.g., the case when the sender (i.e., the first party splitting), executes HLocks on $c = \mathsf{Enc}_{pk_R}(0)$. For x this happens in our protocol when the neighbors of the party splitting the payment might collude with the party generating h and hence know x. For instance, this can occur during a potential collusion between the receiver and intermediaries.

Earlier in this work, we presented two different HLocks algorithms; namely the algorithm HLocks_b and the algorithm HLocks_{ext} (see Figure 2). The simple algorithm HLocks_b outputs k copies of the pair (h, c). Such algorithm clearly does not satisfy the unlinkability definition. We now prove Theorem 6.2 stating that HLocks_{ext} does satisfy the unlikability definition. Before we present the proof, let us restate the theorem here.

Theorem 6.2. *Let Ψ be an additively homomorphic* IND–CPA*-secure encryption scheme with message space $\mathbb{M} = \mathbb{Z}_N$, and $\mathcal{H}$ an additively homomorphic preimage-resistant hash function with domain $\mathbb{P} = \mathbb{Z}_p$ such that p, N coprime and $p < N$. Then $\mathsf{HLocks}^{\mathcal{H},\Psi}_{ext}$ produces unlinkable conditions.*

Proof. Let us fix arbitrary $x \in \mathbb{P}$, $y \in \mathbb{M}$ and $k \in \mathbb{N}$. Let us define the three distributions $\mathcal{D}(pk)$, $\mathcal{D}_x(pk)$, $\mathcal{D}_{x,y}(pk)$ as follows:

$\mathcal{D}(pk)$	$\mathcal{D}_x(pk)$	$\mathcal{D}_{x,y}(pk)$
$a \leftarrow_\$ \mathbb{P}$	$a \leftarrow_\$ \mathbb{P}$	$a \leftarrow_\$ \mathbb{P}$
$c := \mathsf{Enc}_{pk}(a)$	$c := \mathsf{Enc}_{pk}(a)$	$c := \mathsf{Enc}_{pk}(a + y)$
return (a, c)	**return** $(a + x, c)$	**return** $(a + x, c)$

Base on these distributions, we define three games $G_0(x, y, k, n)$, $G_1(x, y, k, n)$ and $G_2(x, y, k, n)$. All three games first generate a key pair (pk, sk) and then sample k pairs $\{(h_j, c_j)\}_{j\in[k]}$. How these pairs are generated differs for each game. In the game G_0, every (h_j, c_j) is sampled from $\mathcal{D}(pk)$, in the game G_1, every (h_j, c_j) is sampled from $\mathcal{D}_x(pk)$ and in G_2, every (h_j, c_j) is sampled from $\mathcal{D}_{x,y}(pk)$. Finally, all of the games output the tuple $(x, y, \{(h_j, c_j)\}_{j\in[k]}, pk)$ (see Figure 17 for formal description).

$G_0(x, y, k, n)$	$G_1(x, y, k, n)$	$G_2(x, y, k, n)$
$(pk, sk) \leftarrow \mathsf{Gen}(1^n)$	$(pk, sk) \leftarrow \mathsf{Gen}(1^n)$	$(pk, sk) \leftarrow \mathsf{Gen}(1^n)$
foreach $j \in [k]$ **do**	**foreach** $j \in [k]$ **do**	**foreach** $j \in [k]$ **do**
$(h_j, c_j) \leftarrow \mathcal{D}(pk)$	$(h_j, c_j) \leftarrow \mathcal{D}_x(pk)$	$(h_j, c_j) \leftarrow \mathcal{D}_{x,y}(pk)$
endfor	**endfor**	**endfor**
$X := \{(h_j, c_j)\}_{j\in[k]}$	$X := \{(h_j, c_j)\}_{j\in[k]}$	$X := \{(h_j, c_j)\}_{j\in[k]}$
return (x, y, X, pk)	**return** (x, y, X, pk)	**return** (x, y, X, pk)

Figure 17: Game hops

Let us note that the game G_0 almost corresponds to the case $b = 0$ in our unlinkability game expect for the fact that h_i is not a random element from $\mathbb{P}$ but the hash value $\mathcal{H}(h_i)$. Similarly, the game G_2 is almost as the case $b = 1$ in our unlinkability game expect for the fact that in the unlinkability game h_i is the hash value $\mathcal{H}(x + x_i)$. However, since for every polynomial-time commutable function f it holds that $G_0 \sim_c G_2 \Rightarrow f(G_0) \sim_c f(G_2)$, it suffices to prove that $G_0 \sim_c G_2$. Here $\sim_c$ denotes computational indistinguishability.

We prove the indistinguishability of G_0 and G_2 in two steps. We first prove that G_0 and G_1 are computationally indistinguishable in Claim 7 and then we show that G_1 and G_2 are computationally indistinguishable in Claim 8.

Claim 7. *Games G_0 and G_1 are computationally indistinguishable.*

Proof. The proof is by a simple hybrid argument. For every $i \in [0, k]$, we define a game $G_{0,i}$ as follows. The game generates the pair (h_j, c_j) as in game G_0 for every $j \leq k - i$, and for every $j > k - i$, the game generates the pair (h_j, c_j) as in game G_1. Formally,

$G_{0,i}(x, y, k, n)$

$(pk, sk) \leftarrow \mathsf{Gen}(1^n)$
foreach $j \in [k]$ **do**
  if $j \leq k - i$ **then**
    $(h_j, c_j) \leftarrow \mathcal{D}(pk)$
  else
    $(h_j, c_j) \leftarrow \mathcal{D}_x(pk)$
  endif
endfor
$X := \{(h_j, c_j)\}_{j\in[k]}$
return (x, y, X, pk)

Note that $G_0 = G_{0,0}$ and $G_1 = G_{0,k}$. As a first step we prove that for every $i \in [0, k-1]$, the games $G_{0,i}$ and $G_{0,i+1}$ are computationally indistinguishable, i.e.,

$$G_0 = G_{0,0} \sim_c G_{0,1} \sim_c \cdots \sim_c G_{0,k} = G_1.$$

Let us fix an arbitrary $i \in [k-1]$ and assume that there exists a ppt adversary $\mathcal{A}_i$ that distinguishes two consecutive games $G_{0,i}$ and $G_{0,i+1}$ with non-negligible probability. We show that then there exists an adversary $\mathcal{A}_{cpa}$ that wins the $\mathsf{PubK}^{\mathsf{cpa}}$ game with the same non-negligible probability which is a contradiction to our assumption that Ψ is a IND-CPA secure encryption scheme.

The games $G_{0,i}$ and $G_{0,i+1}$ differ only in the way how the pair (h_{k-i}, c_{k-i}) is generated. In the game $G_{0,i}$, this pair is sampled from the distribution $\mathcal{D}(pk)$ while in the game $G_{0,i+1}$ it is sampled from $\mathcal{D}_x(pk)$. We use this fact to design an adversary $\mathcal{A}_{cpa}$ as follows (see also Figure 18). The adversary $\mathcal{A}_{cpa}$ samples m_1 uniformly at random and defines $m_0 := m_1 + x$. Upon receiving a challenge ciphertext c_b, the adversary generates the pairs (h_j, c_j), for $j \neq k-i$, as in the games $G_{0,i}$ and $G_{0,i+1}$ and sets $(h_{k-i}, c_{k-i}) := (m_0, c_b)$. The adversary $\mathcal{A}_{cpa}$ now sends $(x, y, \{(h_j, c_j)\}_{j\in[k]}, pk)$ to the adversary $\mathcal{A}_i$ and outputs whatever $\mathcal{A}_i$ outputs.

We now show that if $b = 0$, $\mathcal{A}_{cpa}$ perfectly simulates the game $G_{0,i}$ to $\mathcal{A}_i$ and if $b = 1$, $\mathcal{A}_{cpa}$ perfectly simulates the game $G_{0,i+1}$ to $\mathcal{A}_i$. Consider first that $b = 0$. This means that c_b is an encryption of m_0 and hence

$$(h_{k-i}, c_{k-i}) = (m_0, \mathsf{Enc}(m_0)),$$

implying that (h_{k-i}, c_{k-i}) is distributed according to $\mathcal{D}(pk)$. Hence the tuple $(x, y, \{(h_j, c_j)\}_{j\in[k]}, pk)$ is equally distributed as the output produced by $G_{0,i}$. Now if $b = 1$, then c_b is an encryption of m_1 and hence

$$(h_{k-i}, c_{k-i}) = (m_0, \mathsf{Enc}_{pk}(m_1)) = (m_1 + x, \mathsf{Enc}_{pk}(m_1)),$$

implying that (h_{k-i}, c_{k-i}) is distributed according to $\mathcal{D}_x(pk)$. Hence, the tuple $(x, y, \{(h_j, c_j)\}_{j\in[k]}, pk)$ is identically distributed with $G_{0,i+1}$. This implies that the success probability of $\mathcal{A}_{cpa}$ is the same as the success probability of $\mathcal{A}_i$ which we assume to be non-negligible. This is a contradiction to the IND-CPA security of the encryption scheme Ψ.

We are now prepared to complete the proof that the games $G_{0,0}$ and $G_{0,k}$ are computationally indistinguishable. By using the standard hybrid argument, we know that for any ppt adversary $\mathcal{A}$, the probability of distinguishing $G_{0,0}$ and $G_{0,k}$ is bounded by $\sum_{i\in[k-1]} \nu_i$ where $\nu_i(n)$ is the probability that $\mathcal{A}$ distinguishes $G_{0,i}$

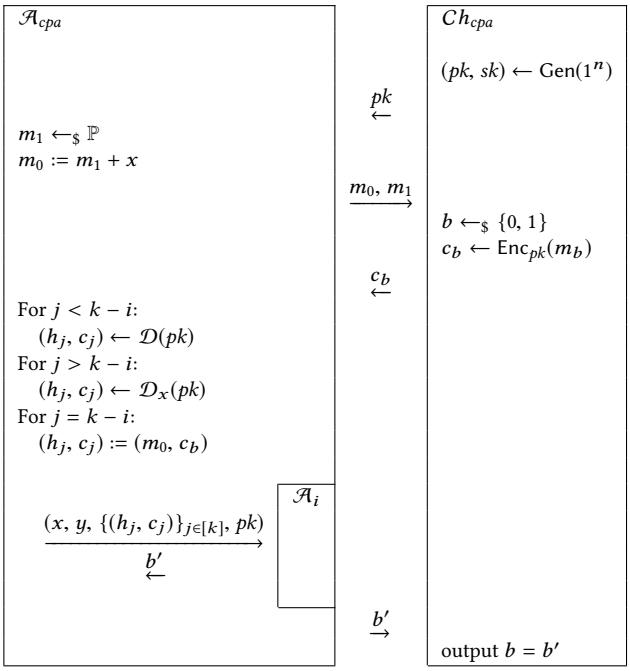


Figure 18: Construction of $\mathcal{A}_{cpa}$ using an adversary $\mathcal{A}_i$ that distinguishes the games $G_{0,i}$ and $G_{0,i+1}$.

and $G_{0,i+1}$. Since we already proved that $G_{0,i}$ and $G_{0,i+1}$ are computationally indistinguishable, we know that ν_i is negligible in the security parameter. Since k is a polynomial in the security parameter and sum of polynomially many negligible functions is a negligible function, we completed the proof. □

Claim 8. *Games G_1 and G_2 are computationally indistinguishable.*

Proof. For every $i \in [0, k]$, we define a game $G_{1,i}$ as follows. The game generates the pair (h_j, c_j) as in game G_1 for every $j \leq k-i$, and for every $j > k - i$, the game generates the pair (h_j, c_j) as in game G_2. Formally,

$G_{1,i}(x, y, k, n)$

$(pk, sk) \leftarrow \mathsf{Gen}(1^n)$
foreach $j \in [k]$ **do**
  if $j \leq k - i$ **then**
    $(h_j, c_j) \leftarrow \mathcal{D}_x(pk)$
  else
    $(h_j, c_j) \leftarrow \mathcal{D}_{x,y}(pk)$
$X := \{(h_j, c_j)\}_{j\in[k]}$
return x, y, X, pk

Note that $G_1 = G_{1,0}$ and $G_2 = G_{1,k}$. As a first step we prove that for every $i \in [0, k-1]$, the games $G_{1,i}$ and $G_{1,i+1}$ are computationally indistinguishable, i.e.,

$$G_1 = G_{1,0} \sim_c G_{1,1} \sim_c \cdots \sim_c G_{1,k} = G_2.$$

Let us fix an arbitrary $i \in [k-1]$ and assume that there exists a ppt adversary $\mathcal{A}_i$ that distinguishes the two games $G_{1,i}$ and $G_{1,i+1}$ with non-negligible probability. We show that then there exists an adversary $\mathcal{A}_{cpa}$ that wins the $\mathsf{PubK}^{\mathsf{cpa}}$ game with the same non-negligible probability which is a contradiction with our assumption that Ψ is a IND-CPA secure encryption scheme.

The games $G_{1,i}$ and $G_{1,i+1}$ differ only in the way how the pair (h_{k-i}, c_{k-i}) is generated. In the game $G_{1,i}$, this pair is sampled from the distribution $\mathcal{D}_x(pk)$ while in the game $G_{1,i+1}$ it is sampled from $\mathcal{D}_{x,y}(pk)$. We use this fact to design an adversary $\mathcal{A}_{cpa}$ as follows (see also Figure 19). The adversary $\mathcal{A}_{cpa}$ samples m_0 uniformly at random message and defines $m_1 := m_0 + y$. Upon receiving a challenge ciphertext c_b, the adversary generates the pairs (h_j, c_j), for $j \neq k-i$, as in the games $G_{1,i}$ and $G_{1,i+1}$ and sets $(h_{k-i}, c_{k-i}) := (m_0+x, c_b)$. The adversary $\mathcal{A}_{cpa}$ now sends $(x, y, \{(h_j, c_j)\}_{j\in[k]}, pk)$ to the adversary $\mathcal{A}_i$. The adversary $\mathcal{A}_{cpa}$ outputs whatever $\mathcal{A}_i$ outputs.

We now show that if $b = 0$, $\mathcal{A}_{cpa}$ perfectly simulates the game $G_{1,i}$ to $\mathcal{A}_i$ and if $b = 1$, $\mathcal{A}_{cpa}$ perfectly simulates the game $G_{1,i+1}$ to $\mathcal{A}_i$. Consider first that $b = 0$. This means that c_b is an encryption of m_0 and hence

$$(h_{k-i}, c_{k-i}) = (m_0 + x, \mathsf{Enc}(m_0)),$$

implying that (h_{k-i}, c_{k-i}) is distributed according to $\mathcal{D}_x(pk)$. Hence the tuple $(x, y, \{(h_j, c_j)\}_{j\in[k]}, pk)$ is equally distributed as the output produced by $G_{1,i}$. Now if $b = 1$, then c_b is an encryption of m_1 and hence

$$(h_{k-i}, c_{k-i}) = (m_0 + x, \mathsf{Enc}_{pk}(m_1)) = (m_0 + x, \mathsf{Enc}_{pk}(m_0 + y)),$$

implying that (h_{k-i}, c_{k-i}) is distributed according to $\mathcal{D}_{x,y}(pk)$. Hence the tuple $(x, y, \{(h_j, c_j)\}_{j\in[k]}, pk)$ is equally distributed as the output produced by $G_{1,i+1}$. This implies that the success probability of $\mathcal{A}_{cpa}$ is the same as the success probability of $\mathcal{A}_i$ which we assume to be non-negligible. This is a contradiction to the IND-CPA security of the encryption scheme Ψ.

We are now prepared to complete the proof that the games $G_{1,0}$ and $G_{1,k}$ are computationally indistinguishable. By using the standard hybrid argument, we know that for any ppt adversary $\mathcal{A}$, the probability of distinguishing $G_{1,0}$ and $G_{1,k}$ is bounded by $\sum_{i\in[k-1]} \nu_i$ where $\nu_i(n)$ is the probability that $\mathcal{A}$ distinguishes $G_{1,i}$ and $G_{1,i+1}$. Since we already proved that $G_{1,i}$ and $G_{1,i+1}$ are computationally indistinguishable, we know that ν_i is negligible in the security parameter. Since k is a polynomial in the security parameter and the sum of polynomially many negligible functions is a negligible function, we completed the proof. □

To conclude, since $G_0 \sim_c G_1 \sim_c G_2$, we have $G_0 \sim_c G_2$ by the hybrid argument. □

Remark 1. Let us stress that $\mathcal{H}$ does not need to be a hash function. In fact, any additively homomorphic function would work for the purposes of the unlikability equally well.

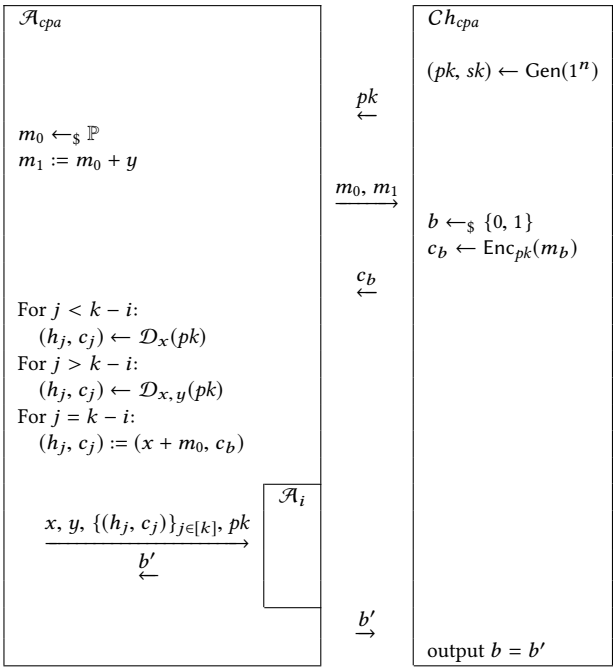


Figure 19: Construction of $\mathcal{A}_{cpa}$ using an adversary $\mathcal{A}_i$ that distinguishes the games $G_{1,i}$ and $G_{1,i+1}$.