arXiv:2005.07574v1 [cs.CR] 15 May 2020

CryptoMaze: Atomic Off-Chain Payments in Payment Channel Network

[1]Subhra Mazumdar and Sushmita Ruj[2]

[1] Indian Statistical Institute Kolkata, India
subhram_r@isical.ac.in
[2] Data61, CSIRO, Australia
Sushmita.Ruj@data61.csiro.au

Abstract. Payment protocols developed to realize off-chain transactions in *Payment channel network (PCN)* assumes the underlying routing algorithm transfers the payment via a single path. However, a path may not have sufficient capacity to route a transaction. It is inevitable to split the payment across multiple paths. If we run independent instances of the protocol on each path, the execution may fail in some of the paths, leading to partial transfer of funds. A payer has to reattempt the entire process for the residual amount. We propose a *secure* and *privacy-preserving* payment protocol, *CryptoMaze*. Instead of independent paths, the funds are transferred from sender to receiver across several payment channels responsible for routing, in a breadth-first fashion. Payments are resolved faster at reduced setup cost, compared to existing state-of-the-art. Correlation among the partial payments in captured, guaranteeing *atomicity*. Further, two party ECDSA signature can be used for establishing scriptless locks among parties involved in the payment. It reduces space overhead by leveraging on core Bitcoin scripts. We provide a formal model in the *Universal Composability* framework and state the privacy goals achieved by CryptoMaze. We compare the performance of our protocol with the existing single path based payment protocol, *Multi-hop HTLC*, applied iteratively on one path at a time on several instances. It is observed that *CryptoMaze* requires less communication overhead and low execution time, demonstrating efficiency and scalability.

Keywords: Payment Channel Network; Breadth-First Traversal; Privacy; Atomicity.

1 Introduction

Cryptocurrencies, like Bitcoin [39], is gaining prominence as an alternative method of payment. Blockchain, a decentralized public ledger, forms the backbone of such currencies. It not only allows transacting parties to remain pseudonymous but also guarantees reliability and security. The records stored in this distributed ledger are immutable and can be verified by anyone in the network. It is replicated across users who use consensus algorithms like Proof-of-Work [39], [40], [9], Proof-of-Stake [28], [29]) for reaching an agreement. However, consensus algorithms have their own computation-overhead and quite resource-intensive. It slows down the performance and reduces scalability [13], [41]. Hence, scaling blockchain transactions has become a pressing concern, in order to compete with traditional methods of payment like Visa, PayPal [51] etc.

1.1 Background

In this section, we provide the required background on the payment channel network, routing and atomic multi-path payment. The terms source/payer means the sender node. Similarly, sink/payee/destination means the receiver node and transaction means payment transfer.

Payment Channel Several Layer 2 solutions like [14], [15], [32] have been proposed for enhancing scalability of Blockchain. Amongst these, *Payment Channel*, like Lightning Network for Bitcoin [41] and Raiden Network for Ethereum [4], stood out as a widely deployed solution. Any two users, with mutual consent, can open a payment channel by locking their funds. These two parties can perform several off-chain payments between themselves, without recording it on blockchain. This is done by locally agreeing on the new deposit balance, enforced cryptographically by hash-based scripts [41], scriptless locking [35]. A party can close the payment channel, with or without the cooperation of counterparty, broadcasting the latest transaction on blockchain. Broadcasting of older transaction leads to loss of funds of the cheating party. Since opening and closing of payment channel is a costly operation, in terms of time and amount of funds locked, parties that are not connected directly leverage on the set of existing payment channels for transfer of funds. This set of payment channels form the *Payment Channel Network* or PCN [41].

Payment Channel Network A Payment Channel Network (PCN) [34] is defined as a bidirected graph $G := (V, E)$, where V is the set of accounts dealing with cryptocurrency and E is the set of payment channels opened between a pair of accounts. A PCN is defined with respect to a blockchain. Apart from the opening and closing of the payment channel, none of the transaction gets recorded on the blockchain. Upon closing the channel, cryptocurrency gets deposited into each user's wallet according to the most recent balance in the payment channel. Every node $v \in V$ charge a processing fee $fee(v)$, for relaying funds across the network. Each payment channel (v_i, v_j) has an associated capacity $cap(v_i, v_j)$, denoting the amount locked by v_i and $cap(v_j, v_i)$ denoting the amount locked by v_j. $remain(v_i, v_j)$ signifies the residual amount of coins v_i can transfer to v_j. Suppose that a node s, also denoted by v_0, wants to transfer amount α to node r through a path $v_0 \rightarrow v_1 \rightarrow v_2 \ldots \rightarrow v_n \rightarrow r$, with each node v_i charging a processing fee $fee(v_i)$. If $remain(v_i, v_{i+1}) \geq \alpha_i : \alpha_i = \alpha - \Sigma_{k=i}^{n} fee(v_k), i \in [0, n-1]$, then funds can be relayed across the channel (v_i, v_{i+1}). The capacity is updated as follows : $remain(v_i, v_{i+1}) = remain(v_i, v_{i+1}) - \alpha_i$ and $remain(v_{i+1}, v_i) = remain(v_{i+1}, v_i) + \alpha_i$.

Routing Payment across a Single Path The major challenge in designing any protocol for PCN is to ensure the privacy of the payer and payee and hiding the payment value transferred. No party, other than the payer and payee, should get any information about the transaction. Routing algorithm generally focused on finding a single path for routing a transaction and were centralized in nature. Canal [50] uses a centralized server for computing the path, Flare [42] requires intermediate nodes to inform the source node about their residual capacity. However, in order to preserve the transaction privacy, it was not in the best interest to have a single coordinator with all information control the routing algorithm.

Any routing or payment algorithm designed for such a network must be decentralized, where individual nodes take decisions based on the information received from its neighbor. Several payment algorithms like [38], [21], [37], [34], [35] deal with the transfer of payment between payer and payee

across a single path. However, finding a single route for routing high-valued transaction becomes a challenging task, especially after several payments got executed in the network. Channels in a path may not have sufficient balance to relay the funds. It is better to split such high-valued transaction across several paths. This is analogous to the situation of breaking a high-valued transaction into several microtransactions.

Splitting Payment across Multiple Paths and Problem of Atomicity

As discussed, it is better to split the high-valued transaction and transmit it over different paths. It eliminates the constraint of finding out a single route from sender to receiver with sufficient channel capacity to support larger payment. Several distributed routing algorithms [43], [42], [33], [45], [50], [53], [26], [36] have been proposed for relaying transaction across multiple paths. But applying existing privacy-preserving payment protocols on each of the path concurrently doesn't guarantee atomicity. Each instance of the protocol runs independent of the other. It is quite possible that an instance of the protocol might fail in a particular path due to resource constraint or malicious behavior of nodes [18], [44]. In the example shown in Fig. 1, payment from sender S to receiver R

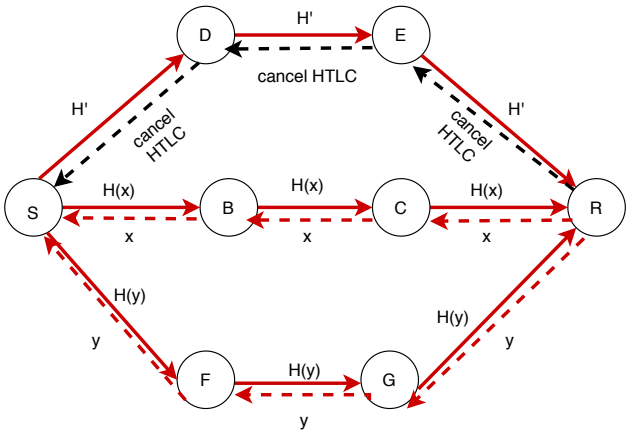


Fig. 1: Failure of payment in path S->D->E->R

is split across three paths, S->D->E->R, S->B->C->R and S->F->G->R. On each path, Hashed Timelock Contract [41] is used concurrently for ensuring secure transfer of funds. For the two paths S->B->C->R and S->F->G->R, the payment hash used is $H(x)$ and $H(y)$ with R releasing the preimage x and y respectively. However for the path S->D->E->R, R does not have the preimage. It immediately reports error and cancels contract with E. E in turn asks D to cancel contract and finally S cancels contract with D, resulting in payment failure. But this results in partial of transfer, violating atomicity. This problem is encountered since each path is considered in isolation and the commitment used across each path are not correlated.

Atomic Multi-Path Payment The goal of the receiver is to receive the full payment. In other words, the payment must be *atomic* - either all the microtransactions succeeds or it fails completely. If funds get transferred partially, sender has to make several attempts for the residual

amount. Existing payment protocols like [2], [8] uses secret sharing [47] for achieving correlation in commitments used across multiple paths. Receiver is able to claim payment if and only if all the paths have confirmed locking of funds for transfer of payments. It reconstructs the secrets from the shares received and resolves the payment. This method guarantees atomicity but either at the cost of high latency or redundancy, involving high computation overhead. Failure of forwarding payment across some path stalls the entire transaction.

This leads to the question of whether is it possible to design an efficient, atomic payment protocol with low setup cost, ensuring secure transfer of funds from payer to payee across several paths (not necessarily edge-disjoint) in the network.

1.2 High Level Overview of CryptoMaze

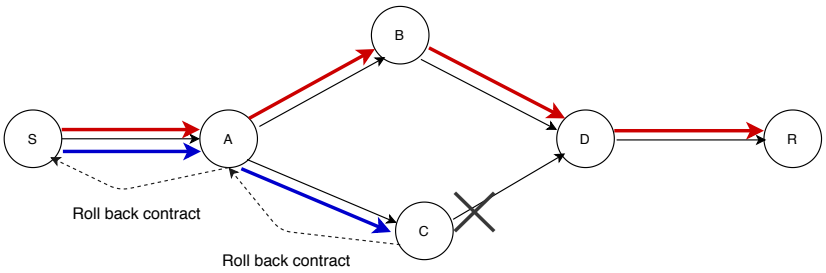


Fig. 2: Protocol fails in node C of p_2

Considering all the above factors, we reached a conclusion that if we want to achieve atomicity as well as low latency, it is better to avoid sending payments via multiple paths. In a flow network, except the source and sink, the incoming flow is equal the outgoing flow. So if a node knows the total cumulative flow and does not receive enough incoming off-chain contracts accounting for it, then it will abort the protocol without proceeding further. In multiple path setting, only receiver has knowledge of the number of partial payments created. Consider the case as shown in Fig.2, with a off-chain contract established on path $p_1 = \langle S, A, B, D, R\rangle$ and path $p_2 = \langle S, A, C, D, R\rangle$, concurrently. If the payment protocol encounters error at node C of path p_2, then the contracts will be canceled in the channels AC and SA corresponding to p_2. However, R will wait for certain time before triggering failure on path p_1. Except R, none of the intermediate nodes knew about the correlation between off-chain contracts established in p_1 and p_2.

We propose a new privacy-preserving payment protocol, *CryptoMaze*, providing an instantiation of the same in Fig. 3. Each node involved in the payment has knowledge about its incoming contracts and outgoing contracts and information about the neighbours which will be sending the request for contract formation. This information helps in faster resolution of payments in the event of failure. Over here, D knows that it will receive incoming contracts from both B and C. Upon not receiving any response from node C, it would have triggered a failure, asking B to cancel all its incoming off-chain contracts. Meanwhile C would have triggered failure as well, canceling contracts on AC. A needs at least one signal from any of its outgoing neighbour for canceling all the incoming contracts. It receives one, either from B or C and cancels contract established with S. The benefit of forming

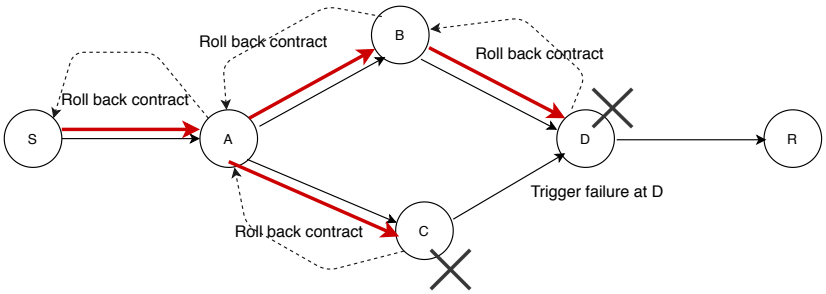


Fig. 3: Handling failure of payment in CryptoMaze

contract on each channel in a breadth first fashion minimizes the number of contract established on shared edges, as paths need not be edge-disjoint.

1.3 Our Contributions

- We have proposed a privacy-preserving payment protocol, *CryptoMaze*, for secure transfer of funds from payer to payee, guaranteeing atomicity, i.e. either the payment succeeds fully or fails entirely. It transfers funds across across several payment channels involved in routing in a breath-first fashion, instead of considering each path individually. This ensures faster resolution of payment.
- Two party ECDSA signature [35] can be easily integrated into our framework for establishing scritpless locking. This reduces space overhead unlike other script-based payment protocols which depends on cryptographic primitives.
- We have defined the privacy notions of *CryptoMaze* based on the Universal Composability framework and provided a detailed security analysis. The security of our proposed scheme depends on the discrete logarithm problem in random oracle.
- We have implemented the proposed protocol on real instances - Ripple Network [34] and Lightning Network [46]. The code is given in [1]. *CryptoMaze* takes around 10s to complete the payment with a communication overhead of less than 1.5 MB compared to Multi-hop HTLC, which takes around 65s to complete the protocol and incurs a communication overhead of 26 MB. In an instance of Lightning Network, it takes around 485ms and communication overhead of 0.16 MB as compared to 10.6s and communication overhead of 24 MB by Multi-hop HTLC.
- The proposed payment protocol is modular and functionally independent and hence works perfectly for any underlying routing algorithm.

1.4 Organization

Section 2 defines the problem statement, privacy goals and a formal definition of the security under Universal Composability Model is given in Section 2.2. The basic operations in ideal world is stated in Section 2.3. The details of our proposed protocol is stated in Section 3, with formal description of *CryptoMaze* in Section 3.4 and its privacy analysis in Section 3.5. Performance Analysis of *CryptoMaze* protocol is provided in Section 4. Section 5 discusses the state-of-the-art in PCN and Section 6 concludes the paper.

2 Problem Statement & Motivation

We formalize the notion of a PCN which allows faster atomic transfer of payment from sender to receiver. An ideal world functionality for the PCN has been provided, discussing the privacy goals.

Definition 1. *A PCN is defined as a bidirected graph $G := (V, E)$, where V is the set of accounts dealing with cryptocurrency and E is the set of payment channels opened between a pair of accounts. Each payment channel is defined by tuple $(id_{(U_i,U_j)}, \beta_i^{start}, \beta_j^{start}, \beta_i^{current}, \beta_j^{current}, t)$, where $id_{(U_i,U_j)}$ is the channel identifier, $\beta_{U_i}^{start}$ denotes the initial deposit amount of U_i in the channel, $\beta_{U_j}^{start}$ denotes the initial deposit amount of U_j in the channel, $\beta_{U_i}^{current}$ denotes the current balance of U_i in the channel, $\beta_{U_j}^{current}$ denotes the current balance of U_j in the channel, where $\beta_{U_i}^{start} + \beta_{U_j}^{start} = \beta_{U_i}^{current} + \beta_{U_j}^{current}$ and t is the channel timeout period. We consider a blockchain $\mathbb{B}$ which will records the node's bitcoin address, denoted by U_i, and its on-chain balance, addressed by $\mathbb{B}[U_i]$. The current timestamp of blockchain as time($\mathbb{B}$). Basic operations of PCN consists three operations (`openPaymentChannel`, `closePaymentChannel`, `payChannel`) -*

- `openPaymentChannel`$(U_i, U_j, \beta_i, \beta_j, t) \rightarrow \{0, 1\}$ *: For a given pair of accounts $U_i, U_j \in V$, with initial balances β_i and β_j, $B[U_i] \geq \beta_i, B[U_j \geq \beta_j$, and a channel timeout period as t, U_i and U_j mutually cooperate to open a channel denoted by $(id_{i,j}, \beta_i, \beta_j, \beta_i, \beta_j, t) \in E$, where $id_{i,j}$ is the channel identifier, provided both U_i and U_j has authorized to do so. If it succeeds, the blockchain is updated as follows: $B[U_i] = B[U_i] - \beta_i$ and $B[U_j] = B[U_j] - \beta_j$ and it returns 1. Upon failure, it returns 0.*
- `closePaymentChannel`$(id_{i,j}) \rightarrow \{0, 1\}$ *: Given a channel identifier $id_{i,j}$ for channel (U_i, U_j), retrieve $(id_{i,j}, \beta_i^{start}, \beta_j^{start}, \beta_i^{current}, \beta_j^{current}, t) \in E$. If timeout period t has expired, i.e. $t <$ time($\mathbb{B}$) then update $\mathbb{B}$ as follows : $B[U_i] = B[U_i] + \beta_i^{start}$ and $B[U_j] = B[U_j] + \beta_j^{start}$, remove the entry from E and return 0. Else, update blockchain as follows: $B[U_i] = B[U_i] + \beta_i^{current}$ and $B[U_j] = B[U_j] + \beta_j^{current}$, remove the entry from E and return 1.*
- `payChannel`$(\{id_{i,j} : (U_i, U_j) \in \mathbb{PC}\}, val) \rightarrow \{0, 1\}$ *: Given a set of payment channels $\mathbb{PC}$ responsible for relaying of funds val from payer U_0 to payee U_n. $\mathbb{PC}$ is denoted by set of channel identifiers $id_{i,j}, U_i, U_j \in V$. Retrieve $(id_{i,j}, \beta_i^{start}, \beta_j^{start}, \beta_i^{current}, \beta_j^{current}, t_{i,j}) \in E$ for each channel. U_i wants to transfer $val_{i,j}$ to U_j, provided U_j has authorized the same. If $\beta_i^{current} \geq val_{i,j}$, then update the channel as $(id_{i,j}, \beta_i^{start}, \beta_j^{start}, \beta_i^{current} - val_{i,j}, \beta_j^{current} + val_{i,j}, t_{i,j})$ and return 1. Else none of the balances of the payment channels in $\mathbb{PC}$ is modified and* `payChannel` *returns 0.*

2.1 Privacy Goals of the Protocol

- **Value Privacy** - It guarantees that neither the participants involved in routing the payment nor any corrupted user outside the payment path will have any knowledge about the transaction amount being send from sender to receiver.
- **Relationship Anonymity** - Given two simultaneous successful pay operations (U_0', U_n', val) and (U_0'', U_n'', val) via same set of intermediaries $U_i \in V$ where $U_0' \neq U_0''$ and $U_n' \neq U_n''$, with each intermediate channel forwarding the same amount of flow for both the cases. If at least one intermediate party is honest and rest all are corrupted, then none of the corrupted intermediate parties can distinguish between payment $(U_0,', U_n', val)$ and (U_0'', U_n'', val) with probability more than $1/2$.

– **Consistency** - The protocol is consistent if none of the nodes can claim funds from the predecessor contract without obtaining the solution from the successor time-locked contracts. Non-adjacent parties, upon collusion, cannot unlock their contracts by bypassing honest intermediaries.
– **Atomicity** - A payment is said to be atomic if the receiver can claim the payment upon receiving all the partial payment flow. The payment channels involved in routing aggregate their individual secrets and provide it to the receiver. Upon receiving this value, the receiver can withdraw funds from the network. If any of the party misbehaves and does not lock fund then the transaction fails.

2.2 Ideal World Functionality

For modeling security and privacy definition of payment across several payment channels under concurrent execution of an instance of *CryptoMaze*, we take the help of Universal Composability framework, first proposed by Canetti et al. [12]. Our modeling of ideal functionality is similar to [34] in terms of notation and assumption, opening and closing of channel. However the difference lies in the procedure of payment. We do not consider linear path based payment. Instead we check the condition on each channel whether the incoming off-chain contracts are consistent to form the outgoing off-chain contracts.

Attacker Model Using the model suggested in [34], the nodes of the network are modeled as interactive Turing machines, denoted by $\mathbb{U} = \{U_i\}, i \in V$, U_0 denotes the initiator of protocol and U_n denotes the receiver, which communicates with an ideal functionality $\mathcal{F}$ via secure and authenticated channels. We model the attacker $\mathcal{A}$ as a PPT machine that is allowed to corrupt a subset of nodes in the network. Upon corruption, it gets access to its internal state and controls any transmission of information to and from the corrupted node. As of now, only static corruption is allowed, i.e. adversary must specify the nodes it wants to corrupt before the start of the protocol.

Communication Model For encoding anonymous communication between two parties in the ideal world, we define it in the following way - Using anonymous message transmission functionality $\mathcal{F}_{anon}$, U_i sends packet $(sid, \text{instruction}, U_i, U_j, m)$, containing the secret message m to U_j. $(sid, \text{instruction}, U_j, |m|)$ is leaked to Sim [12], [11], without revealing the content of the message and the identity of the sender.

An attacker can delay the delivery of messages arbitrarily. The network model is assumed to be synchronous [12], [16], where any message sent out at i^{th} round, gets delivered to the intended recipient at $(i+1)^{th}$ round. Computation in this model is assumed to be instantaneous. However, since we deal with the asynchronous network in the real world, a maximum time bound for message transmission is set. If no message is delivered by the pre-decided expiration time, then the message is considered as $\perp$.

Assumptions We define an ideal functionality $\mathcal{F}$ for the PCN. Dummy parties in the set $\mathbb{U}$ communicate with each other via $\mathcal{F}$. If a user u in the network wishes to communicate anonymously

with user v, it will use $\mathcal{F}_{anon}$. Consider an underlying blockchain B which acts like a trusted append-only ledger recording opening and closing of payment channels. An ideal functionality $\mathcal{F}_B$ maintains B locally. B is updated as per the transaction between parties. Any user can send a *read* instruction to $\mathcal{F}_B$, where the whole transcript of B is sent as a reply. The number of entries of B is denoted by $|B|$. An arbitrary condition can be specified in the contract in order to execute a transactions in B. $\mathcal{F}_B$ is entrusted to enforce that a contract is fulfilled before the corresponding transaction is executed. Time is modeled as the number of entries of the blockchain B. By adding dummy entries to B, time can be elapsed artificially. Users figure out the current time by counting the entries of B. $\mathcal{F}$ uses $\mathcal{F}_{anon}$ and $\mathcal{F}_B$ as subroutines.

Notations

Any payment channel existing in B is denoted by $(id_{i,j}, v_{i,j}, t_{i,j}, f_i)$, where $id_{i,j}$ is the channel identifier of the payment channel existing between dummy parties U_i and U_j, $v_{i,j}$ is the capacity of the channel, $t_{i,j}$ is the expiration time of the channel and f_i is the fee charged by the node U_i. $\mathcal{F}$ maintains two lists internally - one for keeping track of the list of closed channels, denoted by $\mathcal{C}$ and one for keeping track of the list of off-chain payments, denoted by $\mathcal{L}$ [34]. Upon executing an off-chain payment in the channel $id_{i,j}$, $(id_{i,j}, v'_{i,j}, t'_{i,j}, h_{i,j})$ is entered into $\mathcal{L}$ where $v'_{i,j}$ is the payment forwarded to node U_j by U_i and $t'_{i,j}$ is the expiration time of the payment, $h_{i,j}$ is the event identifier. When a channel (U_i, U_j) is closed on-chain, the channel identifier $id_{i,j}$ is entered into list $\mathcal{C}$. For routing payment from U_0 to U_n, payment channels involved in doing so is put in set $\mathbb{PC}$, added serially upon breadth first traversal of the network, starting from U_n. The flow in each channel $id_{i,j}$ present in $\mathbb{PC}$ is denoted by $val_{i,j}$.

2.3 Basic Operations of Payment Channel Network in Ideal World

$\mathcal{F}$ initialized pair of local empty lists $(\mathcal{L}, \mathcal{C})$. Users in set $\mathbb{U}$ can query $\mathcal{F}$ for opening and closing of channel, provided they are valid operations in sync with the state in $\mathcal{L}$. We describe the basic operations in PCN in the ideal world - *open channel, close channel* and *payChannel*.

- OPEN CHANNEL : Considering a user U_i wants to open a channel with U_j. U_i invokes $\mathcal{F}$ by sending the message $(open, id_{i,j}, U_j, v_{i,j}, t_{i,j}, f)$, where $v_{i,j}$ is the channel capacity, $t_{i,j}$ is the expiration time of the channel and f is the associated fee charged on using the channel. If there is no other entry in B and no other inconsistencies are found, $\mathcal{F}$ sends $(id_{i,j}, v_{i,j}, t_{i,j}, f)$ to U_j. Upon authorization by both parties, $\mathcal{F}$ adds $(id_{(U_i,U_j)}, v_{i,j}, t_{i,j}, f)$ to B and $(id_{i,j}, v_{i,j}, t_{i,j}, h_{i,j})$ to $\mathcal{L}$ where $h_{i,j}$ is the event identifier. The event identifier $h_{i,j}$ is returned to U_i and U_j.
- CLOSE CHANNEL : For a channel between U_i and U_j, if either of the party wants to close the channel, it invokes $\mathcal{F}$ with the message $(close, id_{i,j}, h_{i,j})$. $\mathcal{F}$ checks for an entry in B of the form $(id_{i,j}, v_{i,j}, t_{i,j}, f)$ and checks the list $\mathcal{L}$ for an entry $(id_{i,j}, v'_{i,j}, t'_{i,j}, h_{i,j})$, given that $h_{i,j}$ is a valid event identifier. If $id_{i,j} \in \mathcal{C}$ or $t'_{i,j} > |B|$, $t_{i,j} \leq t'_{i,j}$, then $\mathcal{F}$ aborts. Else $(id_{(U_i,U_j)}, v'_{i,j}, t'_{i,j}, h'_{i,j})$ is added to B and $id_{i,j}$ gets added to $\mathcal{C}$. Both U_i and U_j is notified with the message $(id_{i,j}, h_{i,j})$.
- PAY: Given the tuple $(pay, \{(id_{i,j}, val_{i,j}, t_{i,j}) : (U_i, U_j) \in \mathbb{PC}\})$ as input from U_0, $\mathcal{F}$ executes the following protocol:
  - For a given node U_i, $\forall U_k \in V, (U_i, U_k) \in \mathbb{PC}$, $\mathcal{F}$ samples a random $h_{i,k}$ and checks B for an entry $(id_{i,k}, v_{i,k}, t_{i,k}, f)$, $\forall U_j \in V, (U_j, U_i) \in \mathbb{PC}$, $\mathcal{F}$ samples a random $h_{j,i}$ and checks B for an entry $(id_{j,i}, v_{j,i}, t_{j,i}, f)$. If all these entries exists, then it forms $\overrightarrow{I_{in,U_i}} = \{(h_{j,i}, id_{j,i}, val_{j,i}) :$

$U_j \in V, (U_j, U_i) \in \mathbb{PC}\}$, $\overrightarrow{I_{out,U_i}} = \{(h_{i,k}, id_{i,k}, val_{i,k}, t_{i,k}) : U_k \in V, (U_i, U_k) \in \mathbb{PC}\}$ $\mathcal{F}$ conveys $(\overrightarrow{I_{in,U_i}}, \overrightarrow{I_{out,U_i}}, t_{prev,i})$ to U_i via anonymous communication channel. As special case, it sends $(\overrightarrow{I_{in,U_n}}, \perp, t_{prev,n})$ to U_n. $\mathcal{F}$ checks whether for a given payment channel (U_i, U_j) having entries of the form $(id_{i,j}, v'_{i,j}, *, *) \in \mathcal{L}$, the following conditions hold true: $v'_{i,j} \geq val_{i,j}$ and $t_{prev,i} \geq t_{i,j} - \Delta$. If it holds true, then $\mathcal{F}$ adds $m_{i,j} = (id_{i,j}, v'_{i,j} - val_{i,j}, t_{i,j}, \perp)$ to $\mathcal{L}$, $v'_{i,j}$ being the last updated capacity of the payment channel (U_i, U_j). If any of the conditions fails, $\mathcal{F}$ removes all such entries from $\mathcal{L}$ entered in this session and aborts.

- For all $U_i \in \mathbb{PC}$, $\mathcal{F}$ queries U_i with $(\{h_{j,i} : U_j \in V, (U_j, U_i) \in \mathbb{PC}\}, \{h_{i,k} : U_k \in V, (U_i, U_k) \in \mathbb{PC}\})$, through an anonymous channel. If a node U_k returns $\perp$ to $\mathcal{F}$ then update $m_{k,v}, \forall v \in V, (k, v) \in \mathbb{PC}$ to $(\_, \_, \_, h_{k,v})$ in $\mathcal{L}$. $\forall w \in V, (w, k) \in \mathbb{PC}$, remove $m_{w,k}$ from $\mathcal{L}$.

Discussion The ideal functionality $\mathcal{F}$ captures the privacy properties of PCN:

- Value Privacy: Since the payment value is split across multiple channels, we claim that intermediate nodes as well as any adversary (without any access to $\mathcal{F}$) lying outside the payment path cannot figure out the total amount of the transaction.
- Relationship Anonymity: If there exist at least one honest intermediate node, then it receives unique event identifier from $\mathcal{F}$ for each payment over any of its outgoing payment channel. Since all the event identifiers are independently generated, any corrupted node neither draw correlation nor figure out which of the payment (U'_0, U'_n, val) or (U''_0, U''_n, val) got forwarded firs with probability greater then $1/2$.
- Consistency: As per the second step of payment, if a node returns failure while processing payment, none of the incoming payment channels can process their payment.
- Atomicity: $\mathcal{F}$ maintains all the contract list for each user $U_i \in \mathbb{PC}$ and keeps track of their status in $\mathcal{L}$. In the first step of payment, if at any node there is not enough capacity or discrepancy in expiration time of event, then a failure is returned to U_0. In case all conditions gets satisfied, only then does the second step gets executed, with U_n being able to claim payment.

Definition 2. ***UC Definition of Security****. An environment $\mathcal{Z}$ present in both the ideal and real world invokes the steps of execution of an instance by providing the input and receiving the output. If $\mathcal{Z}$ cannot distinguish between Π in the real world and the ideal world functionality, then it is said to be UC-secure. Formally stated,*

Theorem 1. *Given that λ is the security parameter, a protocol denoted by Π, UC-realizes an ideal functionality $\mathcal{F}$ if for all computationally bounded adversary $\mathcal{A}$ attacking Π there exist a probabilistic polynomial-time simulator Sim such that for all probabilistic polynomial time environment $\mathcal{Z}$ such that $IDEAL_{\mathcal{F},Sim,\mathcal{Z}}$ and $REAL_{\Pi,\mathcal{A},\mathcal{Z}}$ are computationally indistinguishable.*

3 Our Proposed Construction

3.1 Network Model and its Assumptions

The topology of the network is known by any node in the network since any opening or closing of a channel is recorded on the blockchain. The payer chooses a set of paths to the receiver according to her own criteria. The current value on each payment channel is not published but instead kept locally by the users sharing a payment channel. Every user is aware of the payment fees charged by each other user in the PCN. Pairs of users sharing a payment channel communicate through secure and authenticated channels.

3.2 Cryptographic Building Blocks

Consider an elliptic curve group with generator $\mathcal{G}$, with $|\mathcal{G}| = q$ and λ be the security parameter.

Discrete Logarithm Problem Given the elliptic curve $\mathbb{G}$ over a finite field $\mathbb{F}_q$, where $q = p^n$ and p is prime, the elliptic curve discrete logarithm problem (ECDLP) is the following computational problem: Given points $P, Q \in \mathbb{G}(\mathbb{F}_q)$, find an integer a such that $Q = aP$, if a exists. This computational problem is called the *Elliptic Curve Discrete Logarithm Problem* which forms the the fundamental building block for elliptic curve cryptography [19].

Two Party ECDSA Signature An efficient two party ECDSA protocol stated by Lindell [31]. Given a collision resistant hash function $H : \{0,1\}^* \rightarrow \{0,1\}^{|q|}$. A private and public key pair is generated by sampling a random value x and corresponding public key $Q = xG$. The signature algorithm over a message m proceeds as follows - Sample a random value k, construct $R = kG$ and $e = H(m)$. Take r_x, which is the x co-ordinate of R. Compute $r = r_x \mod q$ and $s = \frac{e+rx}{k} \mod q$. The signature is the tuple (r, s). Note that $(r, -s)$ also forms a valid signature.

Given (m,r,s) and public key Q, the verification algorithm proceeds as follows - Compute $e = H(m)$ and calculate $S' = \frac{eG+r.Q}{s}$. Let x co-ordinate of S' be s_x. If $r \stackrel{?}{=} s_x \mod q$ then return 1 else return 0.

3.3 Subroutines used in CryptoMaze

We define the subroutines KeyGen, Setup, TimeLockContractCreate and TimeLockContractRelease, which will be used in the payment phase of our protocol.

KeyGen Phase Each node $v \in V$ independently samples a pair of public key and private key $(pk_v, sk_v) : pk_v = sk_v\mathcal{G}$, where $sk_v \stackrel{\$}{\leftarrow} \{0,1\}^\lambda$ and pk_v is a point on the elliptic curve. The public key is a long term key and it is used repeatedly across different instance of the protocol, until and unless the secret key gets compromised.

Setup Phase Given a flow across a network for relaying funds from payer to payee, we map it into set of payment channels, denoted by $\mathbb{PC}$, ordered as per breadth-first traversal. Starting from receiver node, the channels are ordered as per the algorithm stated in Procedure 1.

Consider the network, as shown in Fig.4. Starting from R, it has one incoming payment channel ER with positive flow. This channel is inserted into the set $\mathbb{PC}$. R is inserted into the queue Q. This continues till the last node in Q is the sender S. The set constructed is $\mathbb{PC} = \{ER, DE, CE, BD, BC, SA\}$.

Preprocessing Phase Consider a function $\mathcal{H} : \{0,1\}^* \rightarrow \{0,1\}^\lambda$ as random oracle. U_0 samples $n+1$ independent strings $x_i : x_i \in \mathbb{Z}_q, 0 \leq i \leq n$. Since receiver node is the one with no outgoing flow, funds of payment channel denoted by $id_{u,r}, \forall u \in V, (u,r) \in \mathbb{PC}$, with receiver node as one of the counterparty is locked for the least time. Let this be t_0. For timelocked contracts established with any other pair of nodes (v,u), check the value $t_u = t_{v,u} = max\{t_{u,w} : \forall w \in V, (u,w) \in \mathbb{PC}\} + \Delta$ for some positive value of Δ. Assign $t_{v,u}$ as the lock time for the contract on payment channel (v,u).

Procedure 1: Mapping set of paths $\mathcal{P}$ into set of payment channels

Input: $\mathcal{P}$
Output: $\mathbb{PC}$
Initialize set $\mathbb{PC} = \phi$ and a queue $Q \leftarrow \phi$.
Insert the receiver node r into Q.
while *Q is not empty* **do**
    $v \leftarrow Q.pop()$. Mark v as *visited*.
    Find out the incoming neighbours of v which has a positive flow in order to route the payment.
    Insert these payment channel, with v as the counterparty, into the set $\mathbb{PC}$ and delete it from the set $\mathcal{P}$.
    Insert all *unvisited* incoming neighbours of v into Q.
end

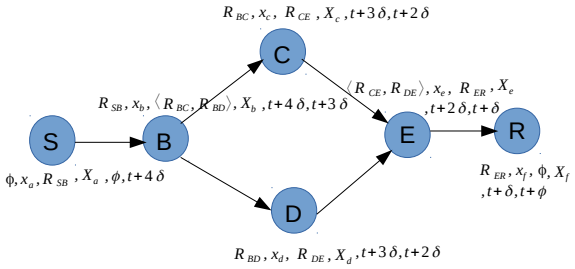


Fig. 4: CryptoMaze : Setup

For each channel $(u, w) \in E$, denoted by $id_{u,w}$ with flow value $val_{u,w}$, a commitment for locking funds is constructed in the following way -

$$R_{u,w} = x_w\mathcal{G} + e_{u,w}.pk_w + \Sigma_{(w,i)\in\mathbb{PC}:i\in V} R_{w,i} \tag{1}$$

where pk_w is the public key of w. $e_{u,w}$ is constructed as

$$e_{u,w} = \mathcal{H}(x_w\mathcal{G} + \Sigma_{(w,i)\in\mathbb{PC}:i\in V} R_{w,i}||id_{u,w}) \tag{2}$$

If $w = U_n$ then $R_{w,i} = \phi$ and $x_w = \tilde{x_w}$, where $\tilde{x}_w$ is defined in Eqn. 4. For each node $U_i, 0 \leq i \leq n$, we construct X_i

$$X_i = (\Sigma_{(U_j,U_i)\in\mathbb{PC}:U_j\in V}\tilde{x_j} + x_i)\mathcal{G} \tag{3}$$

where $\tilde{x}_j$ is defined in Eqn. 4. For all the outgoing neighbours j of u, $R_{u,j}$ will be constructed as shown in Eq. 2. For all the incoming neighbours d of u, $R_{d,u}$ will be constructed as shown in Eq. 1. The packets constructed for vertex $u \in V \setminus \{s, r\}$ is $m_u = (\{(R_{d,u}, val_{d,u}) : d \in V, (d, u) \in \mathbb{PC}\}, t_u, x_u, \{(R_{u,j}, t_{u,j}, val_{u,j}) : j \in V, (u, j) \in \mathbb{PC}\}, X_u)$. Receiver vertex r receives the following information - $m_r = (\{(R_{d,r}, val_{d,r}) : d \in V, (d, r) \in \mathbb{PC}\}, t_r, x_r, \phi, X_r)$. Sender nodes U_0 uses an anonymous secure communication channel to transfer the packets to each of the members $U_i \in \mathbb{PC}, 1 \leq i \leq n$.

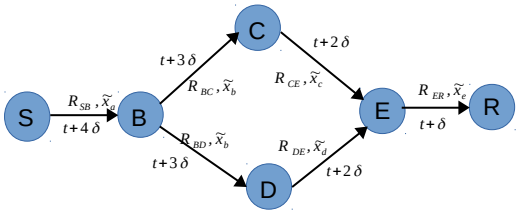


Fig. 5: CryptoMaze : TimeLockContractCreate Phase

TimeLockContractCreate Phase Any node u, except $u = U_0$, waits for incoming contract to be formed for a minimal amount of threshold time t_θ (t_θ of the order of milliseconds). Any contract formed between u and any of its incoming neighbour, say d, is of the form $contract(d, u, R_{d,u}, val_{d,u}, t_{d,u})$. It also obtains a value $\tilde{x}_d$ where

$$\begin{gathered} \tilde{x}_d = x_d + \Sigma_{(j,d)\in\mathbb{PC}}\tilde{x}_j, d \in V \setminus \{s\} \\ \tilde{x}_d = x_s, d = s \end{gathered} \tag{4}$$

Given the information $m_u = (\{(R_{d,u}, val_{d,u}) : d \in V, (d,u) \in \mathbb{PC}\}, t_u, x_u, \{(R_{u,j}, t_{u,j}, val_{u,j}) : j \in V, (u,j) \in \mathbb{PC}\}, X_u)$ for vertex u, it constructs the value R_u where

$$R_u = \Sigma_{(u,j)\in\mathbb{PC}, j\in V} R_{u,j} \tag{5}$$

and checks the following -

$$t_u \stackrel{?}{=} max\{t_{u,j} : (u,j) \in \mathbb{PC}, j \in V\} + \Delta \tag{6}$$

For all outgoing neighbours j of u, it constructs $\tilde{x}_u$ as defined in Eq. 4. For all incoming neighbours d of u,

$$\begin{gathered} e_{d,u} = \mathcal{H}(x_u\mathcal{G} + R_u || id_{d,u}) \\ R_{d,u} \stackrel{?}{=} (x_u + e_{d,u}.sk_u)\mathcal{G} + R_u \\ X_u \stackrel{?}{=} \tilde{x}_u\mathcal{G} \end{gathered} \tag{7}$$

If all the equations hold true, then u sends value $\tilde{x}_u$ to j as well as forms the contract $contract(u, j, R_{u,j}, val_{u,j}, t_{u,j})$. For forming an off-chain contract in a channel (u, v), we leverage on ECDSA based scriptless locking, as defined in [35]. It is compatible with Bitcoin and incurs less space overhead required for other cryptographic operations like hash based scripts [41].

TimeLockContractRelease Phase The release phase starts from the receiver node $U_n = r$ where it checks whether it can construct the discrete log of X_r

$$\begin{gathered} \tilde{x}_r = x_r + \Sigma_{(u,r)\in\mathbb{PC}:u\in V}\tilde{x}_u \\ X_r \stackrel{?}{=} \tilde{x}_r\mathcal{G} \end{gathered} \tag{8}$$

This shows that the key required by r to claim the money is indirectly dependent on the participation of all the channels present in the route $\mathbb{PC}$. If any of the node deviates from the protocol then the

payment will fail. It then proceeds to release the condition for all the incoming contracts in order to claim the money.

We provide a generic procedure of the release phase followed by a node $u \in V \setminus \{s\}$ for its incoming contract $R_{w,u}, (w,u) \in \mathbb{PC}$ formed with node w. It constructs

$$s_{w,u} = \begin{cases} \tilde{x_u}, u = r \\ x_u + \Sigma_{(u,v)\in\mathbb{PC}:v\in V} r_{u,v}, u \in V \setminus \{s,r\} \\ e_{w,u} = \mathcal{H}(s_{w,u}\mathcal{G}||id_{w,u}) \\ r_{w,u} = s_{w,u} + e_{w,u}sk_u \end{cases} \tag{9}$$

The value $r_{w,u}$ is revealed to node w. Since we use ECDSA based scriptless locking, node u will provide the required information for completing the ECDSA signature, which can be verified by w.

3.4 Formal Description of the Protocol

The operation `openPaymentChannel` and `closePaymentChannel` has already been defined as follows:

`openPaymentChannel`$(U_i, U_j, \beta_i, \beta_j, t)$: This operation results in opening of channel between user U_i and U_j. Given the initial bitcoin addresses of U_i and U_j, each party, upon authorization, deposits β_i and β_j in the channel denoted by channel identifier $id_{i,j}$. The channel timeout period is t. After the operation of bitcoin deposit gets successfully added to the blockchain, it is reported as success by returning 1. Upon failure, 0 is returned,

`closePaymentChannel`$(id_{i,j})$: This operations allows two users U_i and U_j mutually agree to close a channel between them denoted by channel identifier $id_{i,j}$. Get the balance of each U_i and U_j in the channel denoted by $remain(U_i, U_j)$ and $remain(U_j, U_i)$. Update the bitcoin balance as per this information and reflect the same in the Bitcoin blockchain. Return 1 if the operation is successful else return 0.

We first define the interfaces of ideal functionality $\mathcal{F}_{ECDSA-Lock}$ in Fig. 6, which has access to a Random Oracle. The interfaces are **KeyGen, Lock** and **Verify**. KeyGen generates a common public key for a payment channel $id_{i,j}$ between parties U_i and U_j. The Lock Phase is same as generating ECDSA signature but with a difference that instead of $R = kG$, we use the value Y sampled by sender U_0 for establishing locks in each channel. The construction is same as defined in [35]. The Verify phase ensures that the correct key is released for completing the signature and a valid party gets to claim the money, as stated in the timelock contract.

`payChannel`$(\{id_{i,j} : (U_i, U_j) \in \mathbb{PC}\}, val)$: Given a network flow for routing the transaction (s, r, val) from $s = U_0$ to $r = U_n$ using the payment channels in $\mathbb{PC}$, obtained after execution of **Setup Phase**. The details of `payChannel` for sender, receiver and intermediate nodes, considering our protocol has access to ideal functionalities $\mathcal{F}_B, \mathcal{F}_{anon}$ and $\mathcal{F}_{ECDSA-Lock}$.

In Procedure 2, sender first calculates the cost of routing the transaction through the channels in $\mathbb{PC}$, including the fee charged by each intermediate hop, denoted by value val_0. It checks whether it has enough funds remaining in its channel to route the payment, given that val_{U_0,U_j} is the flow from U_0 to each of its outgoing neighbour $U_j : val_0 = \Sigma_{(U_0,U_j)\in\mathbb{PC}} val_{U_0,U_j}, \forall U_j \in V$. If not, it will abort the payment. Else it constructs the contract information for each intermediate node, as shown in **Preprocessing Phase**. The information is propagated to each user via anonymous channel. Next, it establishes contract with each of its outgoing neighbour, assigning a legitimate timeout period within which the counterparty has to resolve the payment.

KeyGen

Upon receiving (key-gen-ecdsa, $sid, ssid, U_j$) from U_i and (key-gen-ecdsa, $sid, ssid, U_i$) from U_j:

- Sample a secret key $sk \leftarrow \mathbb{Z}_q$
- Compute a public key $pk = sk.\mathcal{G}$
- Output the message (key-gen-ecdsa, $sid, ssid, pk$) to U_i and U_j
- Store (key-gen-ecdsa, $sid, ssid, sk$)

Lock

Upon receiving $(lock, sid, ssid, m, Y, pk)$ from both U_i and U_j:

- If $(lock, sid, ssid)$ is already stored, abort.
- Check if (key-gen-ecdsa, $sid, ssid, sk$) for the given pk : $pk = sk\mathcal{G}$ has been stored.
- Sample $k \leftarrow \mathbb{Z}_q$ and compute $(r_x, r_y) = R = kY$
- Query the Random Oracle at point $(sid, ssid, m)$, which returns $\mathcal{H}(m)$.
- Compute $s = k^{-1}(\mathcal{H}(m) + r_x sk)$
- Send a output $(lock, sid, ssid, (s, r_x))$ to U_i and U_j
- Store $(lock, sid, ssid)$

Verify

Upon receiving $(verify, sid, ssid, m, r', k, pk)$ from both U_i and U_j:

- If $(lock, sid, ssid)$ is not stored then abort.
- Parse k and retrieve (r, s)
- Query the Random Oracle at point $(sid, ssid, m)$, which returns $\mathcal{H}(m)$.
- Compute $s' = \frac{s}{r'}$ and $(s_x, s_y) = S' = \frac{H(m)G + r.pk}{s'}$
- Check $s_x \stackrel{?}{=} r$, if true return $(verified, sid, ssid)$ to U_j

Fig. 6: Interface of ideal world functionality $\mathcal{F}_{ECDSA-Lock}$

The intermediate parties $U_j, j \in V \setminus \{U_0, U_n\}$ get the terms of the contract from the intermediate node along with a decision, as shown in Procedure 3. If the decision is forward, then it will check the consistency of incoming contracts with the terms stated for outgoing contract. Upon validation, it calls the subroutine **TimeLockContractCreate** and accesses $\mathcal{F}_{ECDSA-Lock}$ to establish a partial ECDSA signature on the contract. Only a counterparty with valid information can complete the signature and claim payment upon verification. If the decision is OK, then it calls **TimeLockContractRelease** module, constructs the key to be propagated to the incoming contracts for completing the signature. The party receiving the key checks whether the signature is complete by querying $\mathcal{F}_{ECDSA-Lock}$. Upon verification, the success message is propagated to the predecessor. If any of the phase fails or no decision is sent out, then abort is triggered. Abort restores the channel balance to its previous valid state and requests all parties to cancel the contract.

The receiver gets the secret share from all the nodes and constructs $\tilde{x}_{U_n}$, as shown in Procedure 4. Even if one payment channel fails in establishing contract, receiver will not be able to claim payment. This guarantees the property of *atomicity*. Receiver triggers the release phase and sends the information along with the decision OK to all its incoming neighbours.

Procedure 2: Payment Protocol for sender

Input: $\mathbb{PC}, U_0, val$
$val_0 = val + \Sigma_{U_i \in \mathbb{PC} \setminus \{U_0, U_n\}} fee(U_i)$
if $val_0 \leq \Sigma_{(U_0,U_j) \in \mathbb{PC}}\ remain(U_0, U_j), \forall U_j \in V$ **then**
  for $U_j \in V : (U_0, U_j) \in \mathbb{PC}$ **do**
    $remain(U_0, U_j) = remain(U_0, U_j) - val_{U_0,U_j}$
  end
  $t_0 = t_{now} + \Delta.l$, l is the maximum level traversed during bfs on the set of paths $\mathcal{P}$.
  for $U_i \in [1, n-1]$ **do**
    Call **Preprocessing Phase** on node U_i
    Get $m_{U_i} = (\{(R_{U_k,U_i}, val_{U_k,U_i}) : U_k \in V, (U_k, U_i) \in \mathbb{PC}\}, t_{U_i}, x_{U_i}, \{(R_{U_i,U_j}, t_{U_i,U_j}, val_{U_i,U_j}) : U_j \in V, (U_i, U_j) \in \mathbb{PC}\}, X_{U_i})$
    Send $(m_{U_i}, forward)$ to U_i via $\mathcal{F}_{anon}$.
  end
  Get $m_{U_n} = (\{(R_{U_d,U_n}, val_{U_d,U_n}) : U_d \in V, (U_d, U_n) \in \mathbb{PC}\}, t_{U_n}, x_{U_n}, \phi, X_{U_n})$ to U_n
  Send $(m_{U_n}, forward)$ to U_n via $\mathcal{F}_{anon}$.
  for $U_j \in V : (U_0, U_j) \in \mathbb{PC}$ **do**
    Generate a random message $m \leftarrow \{0,1\}^*$
    U_j sends the message (key-gen-ecdsa, $sid, ssid, U_0$) to $F_{ECDSA-Lock}$, receives a public key (key-gen, $sid, ssid, pk$)
    query $\mathcal{F}_{ECDSA-Lock}$ on $Lock(sid, ssid, m, R_{U_0,U_j}, pk)$
    if $\mathcal{F}_{ECDSA-Lock}$ *returns* $(sid, ssid, (r, s))$ **then**
      contract$(U_0, U_j, (r, s), val_{U_0,U_j}, t_{U_0,U_j})$
    end
    else
      $remain(U_0, U_j) = remain(U_0, U_j) + val_{U_0,U_j}$
    end
  end
end

3.5 Privacy Analysis

Theorem 2. *Given the elliptic curve group of order q generated by the base point $\mathcal{G}$, the protocol* CryptoMaze UC*-realizes the ideal functionality $\mathcal{F}$ in the $(\mathcal{F}_B, \mathcal{F}_{anon}, \mathcal{F}_{ECDSA-Lock}$)-hybrid Random Oracle model.*

In order to prove Theorem 2, the ideal world simulator Sim, a $\mathcal{PPT}$ algorithm, needs to ensure the output of execution of the an instance of the protocol *CryptoMaze* in $(\mathcal{F}_B, \mathcal{F}_{anon}, \mathcal{F}_{ECDSA-Lock})$-hybrid world is indistinguishable as that in the ideal world, even in presence of corrupt parties. We consider here the following cases for basic PCN operations - openPaymentChannel (if any one party is malicious), closePaymentChannel(if any one of the parties is malicious) and payChannel, to be simulated by a PCN. The environment $\mathcal{Z}$ can use the information leaked by adversary $\mathcal{A}$ or actively influence the execution. It supplies the input to the parties, gets the output and can even corrupt any parties to learn their internal values, control the execution by keeping a tab on the input and output send from that party.

openPaymentChannel($id_{i,j}, \beta, t, f$) Given a payment channel between user U_i and U_j, channel identifier $id_{i,j}$, with U_i initiating the request for opening a channel, with balance β, timeout value t and fee f is the fee charged on using the channel.

- U_i **is corrupted**: On corruption of user U_i by adversary $\mathcal{A}$, a channel open request $(id_{i,j}, \beta, t, f)$ is send to Sim. Both the parties engage in two party agreement over a local channel identifier $id_{i,j}$. Upon success, Sim sends $(open, id_{i,j}, \beta, t, f)$ to $\mathcal{F}$, which will return event identifier h.
- U_j **is corrupted**: Sim receives $(id_{i,j}, \beta, t, f)$ from $\mathcal{F}$. It now executes a two party agreement with $\mathcal{A}$ for opening a channel. If successful, Sim sends an accepting message to $\mathcal{F}$, which will return an event identifier h.

closePaymentChannel($id_{i,j}, h$) Given an existing channel $id_{(U_i,U_j)}$ between U_i and U_j with event id h, with U_i initiating the request.

- U_i **is corrupted**:$\mathcal{A}$ sends a channel close request $(close, id_{i,j}, h)$ to Sim. It checks $\mathcal{L}$ for an entry $(id_{i,j}, \beta', t', h)$. If this value exists then it sends $(close, id_{i,j}, h)$ to $\mathcal{F}$. Else the process aborts.
- U_j **is corrupted**: Sim receives $(close, id_{i,j}, h)$ from $\mathcal{F}$. It notifies $\mathcal{A}$ of the closing of the channel $id_{i,j}$.

payChannel($\{(id_{i,j}, val_{i,j}, t_{i,j}) : (U_i, U_j) \in \mathbb{PC}\}, val$) : We analyse it for each of the entity: *when the sender U_0 is corrupt, when an intermediate party U_i is corrupt* and *when the receiver U_n is corrupt.*

- U_0 **is corrupt**: Adversary $\mathcal{A}$ samples $m_{U_i} = (\{(R_{U_k,U_i}, val_{U_k,U_i}) : U_k \in V, (U_k, U_i) \in \mathbb{PC}\}, t_{U_i}, x_{U_i}, \{(R_{U_i,U_j}, t_{U_i,U_j}, val_{U_i,U_j}) : U_j \in V, (U_i, U_j) \in \mathbb{PC}\}, X_{U_i})$ for each $U_i \in V \setminus \{U_0, U_n\}$ and $m_{U_n} = (\{(R_{U_d,U_n}, val_{U_d,U_n}) : U_d \in V, (U_d, U_n) \in \mathbb{PC}\}, t_{U_n}, x_{U_n}, \phi, X_{U_n})$ for U_n. It sends this value to Sim. For each of the nodes U_i, Sim parses m_{U_i} and checks that terms of its incoming contract and outgoing contract are related to each other. If this holds, it checks whether for each of its outgoing neighbour U_j, $t_{U_i,U_j} = t_{U_i} - \Delta$ and the channel (U_i, U_j) has enough capacity to channelize the flow val_{U_i,U_j}. If all the condition holds true, Sim sends the tuple $(pay, \{(id_{k,i}, val_{k,i}, t_{k,i}) : (U_k, U_i) \in \mathbb{PC}, U_k \in V\}, \{(id_{i,j}, val_{i,j}, t_{i,j}) : (U_i, U_j) \in$

Procedure 3: Payment Protocol for intermediate node U_i

Input : $(m, decision)$
if *(decision is forward)* **then**
  parse m to get $\{(R_{U_k,U_i}, val_{U_k,U_i}) : U_k \in V, (U_k, U_i) \in \mathbb{PC}\}, t_{U_i}, x_{U_i}, \{(R_{U_i,U_j}, t_{U_i,U_j}, val_{U_i,U_j}) : U_j \in V, (U_i, U_j) \in \mathbb{PC}\}, X_{U_i}$
  Call **TimeLockContractCreate Phase** on U_i with m as the input
  Calculate $\tilde{x}_{U_i} = x_{U_i} + \Sigma_{(U_k,U_i)\in\mathbb{PC}}\tilde{x}_{U_k}$
  if *all the conditions of incoming and outgoing contract hold true* **then**
    for $U_j \in V, (U_i, U_j) \in \mathbb{PC}$ **do**
      if $(val_{U_i,U_j} \leq remain(U_i, U_j)) \wedge (t_{U_i,U_j} = t_{U_i} - \Delta)$ **then**
        wait for a time t_θ
        for $U_k \in V, (U_k, U_i) \in \mathbb{PC}$ **do**
          if $(!isContract(U_k, U_j))$ **then**
            abort the process
          end
        end
        $remain(U_i, U_j) = remain(U_i, U_j) - val_{U_i,U_j}$
        Generate a random message $m' \leftarrow \{0,1\}^*$
        U_j sends the message (key-gen-ecdsa, $sid, ssid, U_i$) to $F_{ECDSA-Lock}$, receives a public key (key-gen, $sid, ssid, pk$)
        query $\mathcal{F}_{ECDSA-Lock}$ on $Lock(sid, ssid, m', R_{U_i,U_j}, pk)$
        if $\mathcal{F}_{ECDSA-Lock}$ *returns* $(sid, ssid, (r, s))$ **then**
          contract$(U_i, U_j, (r, s), val_{U_i,U_j}, t_{U_i,U_j})$
          Send $\tilde{x}_{U_i}$ to U_j
        end
        else
          for $U_k \in V, (U_k, U_i) \in \mathbb{PC}$ **do**
            Send $((\{(R_{U_d,U_k}, val_{U_d,U_k}) : U_d \in V, (U_d, U_k) \in \mathbb{PC}\}, t_{U_k}), \perp)$ to U_k
          end
          abort the process
        end
      end
      else
        for $U_k \in V, (U_k, U_i) \in \mathbb{PC}$ **do**
          Send $((\{(R_{U_d,U_k}, val_{U_d,U_k}) : U_d \in V, (U_d, U_k) \in \mathbb{PC}\}, t_{U_k}), \perp)$ to U_k
        end
        abort the process
      end
    end
  end
  else
    for $U_k \in V, (U_k, U_i) \in \mathbb{PC}$ **do**
      Send $((\{(R_{U_d,U_k}, val_{U_d,U_k}) : U_d \in V, (U_d, U_k) \in \mathbb{PC}\}, t_{U_k}), \perp)$ to U_k
    end
  end
end
else if *(decision is* $\perp$*)* **then**
  Parse m to get $\{(R_{U_k,U_i}, val_{U_k,U_i}) : U_k \in V, (U_k, U_i) \in \mathbb{PC}\}$
  for $U_k \in V, (U_k, U_i) \in \mathbb{PC}$ **do**
    $remain(U_k, U_i) = remain(U_k, U_i) + val_{U_k,U_i}$
    Send $((\{R_{U_d,U_k} : U_d \in V, (U_d, U_k) \in \mathbb{PC}\}, t_{U_k}), \perp)$ to U_k
  end
end

else if *(decision is OK)* **then**
  Parse m to get $\{(R_{U_k,U_i}, val_{U_k,U_i}) : U_k \in V, (U_k, U_i) \in \mathbb{PC}\}, t_{U_i}, x_{U_i}, \{(R_{U_i,U_j}, t_{U_i,U_j}, val_{U_i,U_j}) : U_j \in V, (U_i, U_j) \in \mathbb{PC}\}, X_{U_i}$
  for $U_j \in V, (U_i, U_j) \in \mathbb{PC}$ **do**
    Retrieve $m', (r, s), pk$ used in the contract
    Call **TimeLockContractRelease Phase** on U_i with input R_{U_i,U_j}
    Get r_{U_i,U_j} and query $\mathcal{F}_{ECDSA-Lock}$ on $Verify(sid, ssid, m', r_{U_i,U_j}, (r, s), pk)$
    if ($\mathcal{F}_{ECDSA-Lock}$ *returns* $(\perp, sid, ssid)$) **then**
      for $U_k \in V, (U_k, U_i) \in \mathbb{PC}$ **do**
        Send $((\{(R_{U_d,U_k}, val_{U_d,U_k}) : U_d \in V, (U_d, U_k) \in \mathbb{PC}\}, t_{U_k}), \perp)$ to U_k
      end
      abort the process
    end
  end
  for $U_k \in V, (U_k, U_i) \in \mathbb{PC}$ **do**
    Send $((\{(R_{U_j,U_k}, val_{U_j,U_k}) : U_j \in V, (U_j, U_k) \in \mathbb{PC}\}, t_{U_k}, x_{U_k}, \{(R_{U_k,U_d}, t_{U_k,U_d}, val_{U_k,U_d}) : U_d \in V, (U_k, U_d) \in \mathbb{PC}\}, X_{U_k}), OK)$ to U_k
  end
end
else
  for $U_k \in V, (U_k, U_i) \in \mathbb{PC}$ **do**
    Send $((\{(R_{U_d,U_k}, val_{U_d,U_k}) : U_d \in V, (U_d, U_k) \in \mathbb{PC}\}, t_{U_k}), \perp)$ to U_k
  end
end

Procedure 4: Payment Protocol for receiver

Input: $\mathbb{PC}, U_n, val_{U_n}, t_{U_n}, X_{U_n}, \{\tilde{x}_{U_k} : (U_k, U_n) \in \mathbb{PC}, \forall U_k \in V\}$
if $(t_{U_n} > t' + \Delta) \wedge (val_{U_n} = val)$ **then**
  $\tilde{x}_{U_n} = \Sigma_{(U_k,U_n)\in\mathbb{PC}} x_{U_k}, \forall U_k in V$
  if $X_{U_n} \stackrel{?}{=} \tilde{x}_{U_n} \mathcal{G}$ **then**
    for $U_k \in V, (U_k, U_n) \in \mathbb{PC}$ **do**
      Send $((\{(R_{U_j,U_k}, val_{U_j,U_k}) : U_j \in V, (U_j, U_k) \in \mathbb{PC}\}, t_{U_k}, x_{U_k}, \{(R_{U_k,U_i}, t_{U_k,U_i}, val_{U_k,U_i}) : U_i \in V, (U_k, U_i) \in \mathbb{PC}\}, X_{U_k}), OK)$ to U_k
    end
  end
end
else
  for $U_k \in V, (U_k, U_n) \in \mathbb{PC}$ **do**
    Send $((\{(R_{U_d,U_k}, val_{U_d,U_k}) : U_d \in V, (U_d, U_k) \in \mathbb{PC}\}, t_{U_k}), \perp)$ to U_k
  end
end

$\mathbb{PC}, U_j \in V\}$) to $\mathcal{F}$. For the receiver U_n, it checks it $X_{U_n} = \tilde{x}_{U_n}\mathcal{G}$. If this holds true, it sends $(pay, \{(id_{k,n}, val_{k,n}, t_{k,n}) : (U_k, U_n) \in \mathbb{PC}, U_k \in V\})$ to $\mathcal{F}$.
Sim confirms the payment for a payment channel (U_i, U_j) only when it receives from the user U_j an $r_{U_i,U_j} = \Sigma_{(U_j,U_l)\in\mathbb{PC}} r_{U_j,U_l}$ such that $R_{U_i,U_j} = (r_{U_i,U_j} + x_{U_j} + e_{U_i,U_j} sk_{U_j})\mathcal{G}$. In case $\mathcal{A}$ outputs an r^* for (U_j, U_k) such that $R_{U_j,U_k} = r^*\mathcal{G}$ but $r_{U_i,U_j} \neq \Sigma_{(U_j,U_l)\in\mathbb{PC}, U_l \neq U_k} r_{U_j,U_l} + r^*$ then *Sim* aborts. But finding an $r*$ is equivalent to breaking discrete logarithm hardness. Probability of this event is $\frac{1}{q}, q = |\mathbb{G}|$, which is negligible since q is a large prime number. If the receiver is honest then *Sim* confirms the payment if the amount val_{U_n} corresponds to what was agreed with the sender, provided $X_{U_n} = \tilde{x}_{U_n}\mathcal{G}$. If payment is confirmed for a channel (U_i, U_j), then an entry $(id_{i,j}, v'_{i,j} - val_{i,j}, t_{i,j}, h_{i,j})$ to $\mathcal{L}$ is added to $\mathcal{L}$ where $v'_{i,j}$ is the last updated capacity of channel $id_{i,j}$.

- U_n **is corrupt**: *Sim* receives $(\overrightarrow{I_{in,U_n}}, \perp, t_{prev,n})$ from $\mathcal{F}$. *Sim* samples a random $x' \xleftarrow{\$} \mathbb{Z}_q$ and returns to $\mathcal{A}$ the tuple $(x', x'\mathcal{G}, val_{U_n})$. If the adversary returns $x_{\tilde{U_n}} : x' = \tilde{x}_{U_n}$ then *Sim* return T to $\mathcal{F}$, otherwise it aborts.
- $\forall U_i \in V \setminus \{U_0, U_n\}, U_i$ **is corrupt**: *Sim* receives a tuple $(\overrightarrow{I_{in,U_i}}, \overrightarrow{I_{out,U_i}}, t_{prev,i})$ from $\mathcal{F}$, which corresponds to the corrupted user U_i. *Sim* samples $x_i \xleftarrow{\$} \mathbb{Z}_q, X_{U_i} \xleftarrow{\$} \mathbb{Z}_q$ and $r_{U_i,U_j}, \forall U_j \in V, (U_i, U_j) \in \mathbb{PC} : R_{U_i,U_j} = r_{U_i,U_j}\mathcal{G}$. It forms $R_{U_i} = \Sigma_{(U_i,U_j)\in\mathbb{PC}} R_{U_i,U_j}$. For each $(U_k, U_i) \in \mathbb{PC}$, it queries Random Oracle at point $(x_i\mathcal{G} + R_{U_i} || id_{U_k,U_i})$ and gets the value H_{U_k,U_i}. It then computes $R_{U_k,U_i} = x_i + H_{U_k,U_i} R_{U_i}, \forall U_k \in V, (U_k, U_i) \in \mathbb{PC}$ and forms the message $m_{U_i} = (\{(R_{U_k,U_i}, val_{U_k,U_i}) : U_k \in V, (U_k, U_i) \in \mathbb{PC}\}, t_{U_i}, x_{U_i}, \{(R_{U_i,U_j}, t_{U_i,U_j}, val_{U_i,U_j}) : U_j \in V, (U_i, U_j) \in \mathbb{PC}\}, X_{U_i})$. $\mathcal{A}$ gets m_{U_i}. If it can output r^* such that $R_{U_i} = r^*G$ then *Sim* aborts. Probability of finding such an r^* is is negligible, given that $\mathcal{A}$ is a probabilistic polynomial time algorithm and finding an r^* is equivalent to breaking discrete logarithm problem. Probability of this event is $\frac{1}{q}, q = |\mathbb{G}|$, which is negligible since q is a large prime number. Thus the probability that *Sim* aborts is also negligible. If *Sim* had been queried at $(\{h_{j,i} : U_j \in V, (U_j, U_i) \in \mathbb{PC}\}, \{h_{i,k} : U_k \in V, (U_i, U_k) \in \mathbb{PC}\})$, then it would have returned r' to $\mathcal{A}$ on behalf of all the outgoing neighbours of U_i. Then $\mathcal{A}$ could have easily constructed $r_{U_k,U_i} : (U_k, U_i) \in \mathbb{PC}, \forall U_k \in V$. *Sim* would have send T to $\mathcal{F}$ and added $(id_{k,i}, v'_{k,i} - val_{k,i}, t_{k,i}, h_{k,i})$ to $\mathcal{L}$, where $\forall U_k \in V, (U_k, U_i) \in \mathbb{PC}, v'_{k,i}$ is the last updated capacity of channel $id_{k,i}$.

From analysis each of the cases, it is clear that the distinguishing event of execution of protocol in the real world from that in the ideal world is whenever *Sim* aborts. This is possible only if $\mathcal{A}$ can output the discrete logarithm of the commitment given in the contract, without querying *Sim*. However, this event is possible with negligible probability as per the assumption of hardness of discrete logarithm.

4 Performance Analysis

4.1 Experimental Setup

In this section, we define the experimental setup. The code for *CryptoMaze* is available in [1]. System configuration used is : `Intel Core i5-8250U CPU, Kabylake GT2 octa core processor, frequency 1.60 GHz`, OS : *Ubuntu-18.04.1 LTS* (64 bit). The programming language used is C, compiler - gcc version 5.4.0 20160609. The library *igraph* was used for generating random graphs of size ranging from 50 to 25000, based on Barábasi-Albert model [6], [10]. Payment Channel Network follows the scale free network where certain nodes function as hub (like central banks), having

higher degree compared to other nodes [27]. For implementing the cryptographic primitives in both *CryptoMaze* and *Multi-hop HTLC*, we use the library *OpenSSL*, version-1.0.2 [48] and SHA-256 has been modeled as a random oracle. For constructing the zero-knowledge proof for *Multi-hop HTLC*, we have used C-based implementation of ZKBoo[5]. The number rounds for running the protocol is set to 136, which guarantees soundness error of 2^{-80} for the proof and witness length is set to 32 bytes. For implementing *CryptoMaze*, we have considered the elliptic curve secp224r1.

4.2 Evaluation

Fig. 7: Time taken for Payment

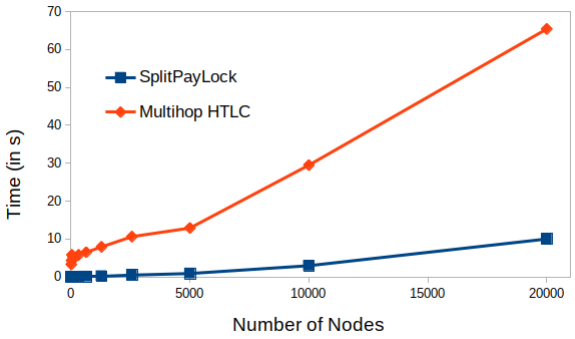
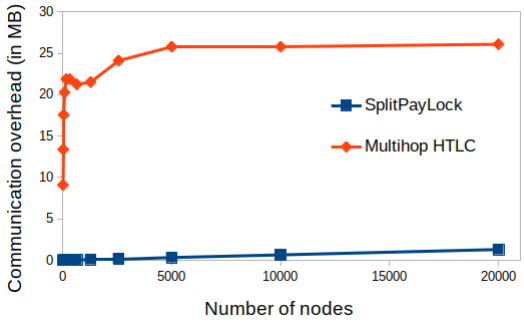


Fig. 8: Communication overhead



Following metrics are used to compare the performance of the payment protocol, *CryptoMaze* with *Multi-Hop HTLC* [34]:

– TTP (*Time taken for payment*) : Given set of paths for payment transfer, it is the time taken for construction of hashed time-lock contract across all the edges in the path and completion of payment upon successfully fulfilling the criteria set in the contract.
– Communication Overhead: For the given payment protocol, the number of message packets exchanged between the nodes in terms of *bytes.*

For the graph given in Fig. 4, in order to transfer a value of 10 Satoshis from S to R, two paths $S \rightarrow B \rightarrow C \rightarrow E \rightarrow R$ and $S \rightarrow B \rightarrow D \rightarrow E \rightarrow R$ are obtained, each carrying 5 Satoshis. Multi-hop HTLC is applied to each path, one at a time. The time taken to complete the payment protocol is 1.53s and communication overhead is 6.483 MB, considering each path having 5 users each. On executing *CryptoMaze* for the same payment, the execution time taken is 1.9ms and communication overhead is 1.087 KB.

Testing on Real Instances. We test our protocol on a Ripple network [33], comprising around 20000 nodes. Our proposed payment protocol takes around 10s to complete the payment with a communication overhead of less than 1.5 MB. Multi-hop HTLC takes around 65s to complete the protocol and incurs a communication overhead of 26 MB. Considering an instance of Lightning Network as stated in [46], comprising 2500 nodes. It takes around 485ms and communication overhead of 0.16 MB as compared to 10.6s and communication overhead of 24 MB by Multi-hop HTLC.

Testing on Simulated Instances. We consider synthetically generated graphs, with the number of nodes ranging from 10 to 20000. We vary the source-sink pair and use value 40 for transfer from payer to a payee for all the instances. Apart from simulated graphs, Overall, the result demonstrates the benefit of considering all the split simultaneously instead of one path at a time in terms of scalability and efficiency in terms of computation cost and resource utilization.

5 Related Works

Payment Channel Network is a peer-to-peer, path-based transaction (PBT) network where each party operates independently of other parties. Several P2P path-based transaction networks such as such as Lightning Network for Bitcoin [41], Raiden Network for Ethereum [4], SilentWhispers [33], InterLedger [49], Atomic-swap [3], TeeChain [30] etc. have been developed over the years.

In Table 1, we compare the properties of *CryptoMaze* with existing payment protocols. Privacy guarantee offered by PCN and its challenges has been extensively discussed in [7], [25], [22]. A payment along a path must be atomic - either it succeeds fully or it is aborted. Partial satisfaction of a transaction may lead to loss of funds. As a solution, Hashed Time-lock Contract [41] was proposed for Lightning Networks. It is compatible with the Bitcoin script but has its demerits. Bolt [21] states about a hub-based payment construction retaining payment anonymity but it is restricted to just two-hop payment. TumbleBit [23] follows a similar approach assuring payer/payee privacy but suffers from the same shortcoming. Malavolta et al. [34] had proposed a secure version of payment for multi-hop path based on zero-knowledge proof system `ZK-Boo` [20]. It uses Multi-hop HTLC, working on one path at a time. *Anonymous Multi-Hop Locks*, defined in [35], are compatible with vast majority of cryptocurrencies. It is generic as well as interoperable, supporting both script and scriptless support for PCN. An efficient privacy-preserving payment protocol based on Chameleon Hash Function [52] was proposed which is devoid of complex key management and zero-knowledge proof. But in this protocol, honest intermediaries lying on a path are susceptible to

Table 1: Comparison among the existing Payment Protocols in PCN

Algorithm	Atomic single path/multiple path payment	Privacy Violation/Other Disadvantage	Wormhole Attack
Hashed Timelock Contract [41]	Atomic single path payment	No Balance Security, Payment Correlation possible, can identify sender and receiver with some observation.	Susceptible
Sprites [37]	Strong atomic single path payment	No relationship anonymity, sender and receiver identity revealed as well.	Not applicable
SilentWhisper [33]	Multiple Path payment but atomicity guaranteed for single path	Costly multi-party computation for determining credit available on each path, privacy leakage due to knowledge of minimum funds available on each channel	Not possible
Multi-hop HTLC [34]	Atomic single path payment	Too much communication overhead, use of complex zero knowledge proofs	Not possible
Anonymous Multi-hop Lock [35]	Atomic single path payment	Privacy Preserving but applicable for single path payment	Not possible
Atomic Multi Channel Update with Constant Collateral [18]	Strong atomic single path payment	Violates relationship anonymity, practically not yet realized	Not applicable
Atomic Multi-path Payment [2]	Atomic multi path payment	High Latency	Each path is susceptible to attack
Boomerang [8]	Atomic multi path payment	Too much redundancy in order to reduce latency	Each path is susceptible to attack
CryptoMaze	Atomic multi path Payment	Efficient and Privacy Preserving	Not possible

key exposure attacks. However, all such payment protocols deal with routing transactions via single path. All these works assumes a staggered locktime across the path, involving high collateral cost. Later, Sprites [37], an ethereum styled payment network, first proposed the idea of using constant locktime for resolving payment. If at least one channel reports successful payment transfer, all the channels involved in relaying payment must update their state. Privacy was violated as the path information, identity of sender and receiver was known by all participants involved in routing the payment. Similar concept of reducing collateral cost using constant locktime contracts was proposed for Bitcoin-compatible payment networks in [18]. Even if one party misbehaved, the payment failed entirely. However, it violated relationship anonymity and the proposed protocol is yet to be realized practically. Other protocols for cross-chain payment [24] have been studied but there is substantial leakage of information violating transaction privacy.

SilentWhisper proposed multi-path payment but at the cost of substantial computation overhead. Also, it failed to capture atomicity. This might lead to partial transfer of funds, as the possibility of payment failing in certain paths exist. State-of-the-art on atomic multi-path payment by Osuntokun [2] captures atomicity by using linear secret sharing across the various paths routing partial payments but it lacks security model. A particular path uses same payment hash, formed with a secret share, across multiple hops for relaying the partial payment. Hence intermediate nodes in that path become susceptible to *Wormhole attack* [35]. It suffers from high latency as well where the receiver has to wait for all the paths to complete the formation of off-chain contracts. In the event of failure even in one path, the contracts has to be canceled across all the remaining paths. The problem of latency is claimed to be solved by another payment protocol, Boomerang [8]. To

increase throughput at reduced latency, payment is split across k paths and sender uses $k+n, n > 0$ paths for relaying transaction, so that success is guaranteed even if some path fails. If k such paths have formed their contract, receiver must cancel the microtransactions on the remaining n redundant paths. However the redundant paths are susceptible to *Griefing attack* [18], [44] as intermediate nodes may withhold the cancel message from being propagated to the sender. Recently, a new technique based on *Dynamic Internal Payment Splitting* [17] recursively splits payments across multiple intermediaries in state channel network and receiver aggregates such payment receipts for claiming payment. But this protocol does not strictly adhere to the requirement of atomic transfer of payment and it will not work for the Bitcoin-based payment channel network.

6 Conclusion

In this paper, we have proposed a novel privacy-preserving, off-chain payment protocol for Payment Channel Network, *CryptoMaze*, guaranteeing atomicity, i.e. either the payment succeeds fully or fails entirely. It transfers funds across across several payment channels involved in routing in a breath-first fashion, instead of considering each path individually. This ensures faster resolution of payment. ECDSA based scriptless locking can be incorporated for establishing timelocked contract, reducing space overhead. We analysed the performance of the protocol on some real instances like Lightning Network and Ripple Network. From the results, it was inferred that our proposed payment protocol has less execution time and low communication overhead as compared to existing payment protocols like Multi-hop HTLC [34]. It is efficient and scalable as the setup phase doesn't require any complex computation. Our protocol instance has been defined for a transaction between a payer and payee but it can be extended to handle multiple transactions by enforcing blocking protocol or non-blocking protocol to resolve deadlocks in concurrent payments [34].

References

1. Cryptomaze. `https://www.dropbox.com/sh/x9pngj005dxh87b/ AAAJNt-WquV0JZTspnijEXNVa?dl=0` (2019)
2. Amp: Atomic multi-path payments over lightning. `https://lists.linuxfoundation.org/pipermail/lightning-dev/2018-February/000993.html` (February 2018)
3. Atomic cross-chain trading. `https://en.bitcoin.it/wiki/ Atomic_cross-chain_trading` (July 2017)
4. Raiden network. `http://raiden.network/` (July 2017)
5. Source code : C based implementation of zkboo. `https://github.com/Sobuno/ZKBoo/` (October, 2016)
6. Albert, R., Barabási, A.L.: Statistical mechanics of complex networks. Reviews of modern physics **74**(1), 47 (2002)
7. Atlas, K.: The inevitability of privacy in lightning networks, 2017. URL https://www. kristovatlas. com/the-inevitability-of-privacy-in-lightning-networks/.[Online
8. Bagaria, V., Neu, J., Tse, D.: Boomerang: Redundancy improves latency and throughput in payment networks. arXiv preprint arXiv:1910.01834 (2019)
9. Bano, S., Sonnino, A., Al-Bassam, M., Azouvi, S., McCorry, P., Meiklejohn, S., Danezis, G.: Consensus in the age of blockchains. arXiv preprint arXiv:1711.03936 (2017)
10. Barabási, A.L., Bonabeau, E.: Scale-free networks. Scientific american **288**(5), 60–69 (2003)
11. Camenisch, J., Lysyanskaya, A.: A formal treatment of onion routing. In: Annual International Cryptology Conference. pp. 169–187. Springer (2005)

12. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on. pp. 136–145. IEEE (2001)
13. Croman, K., Decker, C., Eyal, I., Gencer, A.E., Juels, A., Kosba, A., Miller, A., Saxena, P., Shi, E., Sirer, E.G., et al.: On scaling decentralized blockchains. In: International Conference on Financial Cryptography and Data Security. pp. 106–125. Springer (2016)
14. Decker, C., Russell, R., Osuntokun, O.: eltoo: A simple layer2 protocol for bitcoin. White paper: https://blockstream. com/eltoo. pdf (2018)
15. Decker, C., Wattenhofer, R.: A fast and scalable payment network with bitcoin duplex micropayment channels. In: Symposium on Self-Stabilizing Systems. pp. 3–18. Springer (2015)
16. Dziembowski, S., Eckey, L., Faust, S.: Fairswap: How to fairly exchange digital goods. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 967–984. ACM (2018)
17. Dziembowski, S., Kędzior, P.: Ethna: Channel network with dynamic internal payment splitting
18. Egger, C., Moreno-Sanchez, P., Maffei, M.: Atomic multi-channel updates with constant collateral in bitcoin-compatible payment-channel networks. In: 26th ACM Conference on Computer and Communications Security. ACM (2019)
19. Galbraith, S.D., Gaudry, P.: Recent progress on the elliptic curve discrete logarithm problem. Designs, Codes and Cryptography **78**(1), 51–72 (2016)
20. Giacomelli, I., Madsen, J., Orlandi, C.: Zkboo: Faster zero-knowledge for boolean circuits. In: USENIX Security Symposium. pp. 1069–1083 (2016)
21. Green, M., Miers, I.: Bolt: Anonymous payment channels for decentralized currencies. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 473–489. ACM (2017)
22. Gudgeon, L., Moreno-Sanchez, P., Roos, S., McCorry, P., Gervais, A.: Sok: Off the chain transactions. IACR Cryptology ePrint Archive **2019**, 360 (2019)
23. Heilman, E., Alshenibr, L., Baldimtsi, F., Scafuro, A., Goldberg, S.: Tumblebit: An untrusted bitcoin-compatible anonymous payment hub. In: Network and Distributed System Security Symposium (2017)
24. Herlihy, M.: Atomic cross-chain swaps. In: Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing. pp. 245–254. ACM (2018)
25. Herrera-Joancomartí, J., Pérez-Solà, C.: Privacy in bitcoin transactions: new challenges from blockchain scalability solutions. In: Modeling Decisions for Artificial Intelligence. pp. 26–44. Springer (2016)
26. Hoenisch, P., Weber, I.: Aodv–based routing for payment channel networks. In: International Conference on Blockchain. pp. 107–124. Springer (2018)
27. Javarone, M.A., Wright, C.S.: From bitcoin to bitcoin cash: a network analysis. In: Proceedings of the 1st Workshop on Cryptocurrencies and Blockchains for Distributed Systems. pp. 77–81. ACM (2018)
28. King, S., Nadal, S.: Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. self-published paper, August **19** (2012)
29. Li, W., Andreina, S., Bohli, J.M., Karame, G.: Securing proof-of-stake blockchain protocols. In: Data Privacy Management, Cryptocurrencies and Blockchain Technology, pp. 297–315. Springer (2017)
30. Lind, J., Eyal, I., Kelbert, F., Naor, O., Pietzuch, P., Sirer, E.G.: Teechain: Scalable blockchain payments using trusted execution environments. arXiv preprint arXiv:1707.05454 (2017)
31. Lindell, Y.: Fast secure two-party ecdsa signing. In: Annual International Cryptology Conference. pp. 613–644. Springer (2017)
32. Luu, L., Narayanan, V., Zheng, C., Baweja, K., Gilbert, S., Saxena, P.: A secure sharding protocol for open blockchains. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 17–30. ACM (2016)
33. Malavolta, G., Moreno-Sanchez, P., Kate, A., Maffei, M.: Silentwhispers: Enforcing security and privacy in decentralized credit networks. In: Network and Distributed System Security Symposium (2017)
34. Malavolta, G., Moreno-Sanchez, P., Kate, A., Maffei, M., Ravi, S.: Concurrency and privacy with payment-channel networks. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 455–471. ACM (2017)

35. Malavolta, G., Moreno-Sanchez, P., Schneidewind, C., Kate, A., Maffei, M.: Multi-hop locks for secure, privacy-preserving and interoperable payment-channel networks. In: Network and Distributed System Security Symposium (2019)
36. Mazumdar, S., Ruj, S., Singh, R.G., Pal, A.: Hushrelay: A privacy-preserving, efficient, and scalable routing algorithm for off-chain payments. arXiv preprint arXiv:2002.05071 (2020)
37. Miller, A., Bentov, I., Kumaresan, R., McCorry, P.: Sprites: Payment channels that go faster than lightning. In: Twenty-Third International Conference on Financial Cryptography and Data Security 2019 (2019)
38. Moreno-Sanchez, P., Kate, A., Maffei, M., Pecina, K.: Privacy preserving payments in credit networks. In: Network and Distributed Security Symposium (2015)
39. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
40. O'Dwyer, K.J., Malone, D.: Bitcoin mining and its energy footprint (2014)
41. Poon, J., Dryja, T.: The bitcoin lightning network: Scalable off-chain instant payments. See https://lightning. network/lightning-network-paper. pdf (2016)
42. Prihodko, P., Zhigulin, S., Sahno, M., Ostrovskiy, A., Osuntokun, O.: Flare: An approach to routing in lightning network. White Paper (bitfury.com/content/5-white-papers-research/whitepaper _flare_an_approach_to_routing_in_lightning_n etwork_7_7_2016. pdf) (2016)
43. Rohrer, E., Laß, J.F., Tschorsch, F.: Towards a concurrent and distributed route selection for payment channel networks. In: Data Privacy Management, Cryptocurrencies and Blockchain Technology, pp. 411–419. Springer (2017)
44. Rohrer, E., Malliaris, J., Tschorsch, F.: Discharged payment channels: Quantifying the lightning network's resilience to topology-based attacks. In: 2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW). pp. 347–356. IEEE (2019)
45. Roos, S., Moreno-Sanchez, P., Kate, A., Goldberg, I.: Settling payments fast and private: Efficient decentralized routing for path-based transactions. In: Network and Distributed System Security Symposium (2018)
46. Seres, I.A., Gulyás, L., Nagy, D.A., Burcsi, P.: Topological analysis of bitcoin's lightning network. arXiv preprint arXiv:1901.04972 (2019)
47. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612âĂŞ613 (Nov 1979). https://doi.org/10.1145/359168.359176, https://doi.org/10.1145/359168.359176
48. The OpenSSL Project: OpenSSL: The open source toolkit for SSL/TLS (10 September, 2019), www.openssl.org
49. Thomas, S., Schwartz, E.: A protocol for interledger payments. URL https://interledger. org/interledger. pdf (2015)
50. Viswanath, B., Mondal, M., Gummadi, K.P., Mislove, A., Post, A.: Canal: Scaling social network-based sybil tolerance schemes. In: Proceedings of the 7th ACM european conference on Computer Systems. pp. 309–322. ACM (2012)
51. Vlastelica, R.: Why bitcoin wont́ displace visa or mastercard soon. https://www.marketwatch.com/story/ why-bitcoin-wont-displace-visa-or-mastercard-soon-2017-12-15 (December 2017)
52. Yu, B., Kermanshahi, S.K., Sakzad, A., Nepal, S.: Chameleon hash time-lock contract for privacy preserving payment channel networks. In: International Conference on Provable Security. pp. 303–318. Springer (2019)
53. Yu, R., Xue, G., Kilari, V.T., Yang, D., Tang, J.: Coinexpress: A fast payment routing mechanism in blockchain-based payment channel networks. In: 2018 27th International Conference on Computer Communication and Networks (ICCCN). pp. 1–9. IEEE (2018)