Secure Off-chain Transactions in Blockchain-based Payment Channel Networks

A thesis submitted to Indian Statistical Institute
in partial fulfillment of the thesis requirements for the degree of
Doctor of Philosophy in Computer Science

Author: Subhra Mazumdar

under the guidance of

Dr. Sushmita Ruj
University of New South Wales, Sydney
&
Prof. Bimal Kumar Roy
Indian Statistical Institute Kolkata

Cryptology and Security Unit
Indian Statistical Institute
203 Barrackpore Trunk Road
Kolkata, West Bengal
India - 700 108

September 2022

Dedicated to the Cypherpunks

Declaration of Authorship

I, **Subhra Mazumdar**, a student of Cryptology and Security Research Unit, of the Ph.D. program of Indian Statistical Institute, Kolkata, hereby declare that the investigations presented in this thesis are based on my works and, to the best of my knowledge, the materials contained in this thesis have not previously been published or written by any other person, nor it has been submitted as a whole or as a part for any degree/diploma or any other academic award anywhere before.

Subhra Mazumdar

28.09.2022

Subhra Mazumdar
Cryptology and Security Research Unit,
Indian Statistical Institute, Kolkata 203 Barrackpore Trunk Road,
Kolkata, West Bengal, India - 700108

List of Pubications/Manuscript

1. Subhra Mazumdar, Sushmita Ruj. Book Chapter - *Layer-1 Scaling solutions for Blockchains* in *"Blockchains - A Handbook on Fundamentals, Platforms and Applications"*. Editors: Sushmita Ruj, Salil Kanhere, Mauro Conti. Springer. To Appear.

2. Subhra Mazumdar, Sushmita Ruj, Ram Govind Singh, and Arindam Pal. "HushRelay: A privacy-preserving, efficient, and scalable routing algorithm for off-chain payments." In 2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), pp. 1-5. IEEE, 2020. DOI: 10.1109/ICBC48266.2020.9169405

3. Subhra Mazumdar, and Sushmita Ruj. "CryptoMaze: Privacy-Preserving Splitting of Off-Chain Payments." In IEEE Transactions on Dependable and Secure Computing. Early Acces. Date of Publication: 07 February 2022, DOI: 10.1109/TDSC.2022.3148476

4. Subhra Mazumdar, Prabal Banerjee, and Sushmita Ruj. "Time is Money: Countering Griefing Attack in Lightning Network." In 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), pp. 1036-1043. IEEE, 2020. DOI: 10.1109/TrustCom50675.2020.00138

5. Subhra Mazumdar, Prabal Banerjee, and Sushmita Ruj. "Griefing-penalty: Disincentivizing Griefing Attack in Lightning Network." Computer Communications. Manuscript Submitted.

6. Subhra Mazumdar, Prabal Banerjee, Abhinandan Sinha, Sushmita Ruj and Bimal Kumar Roy. "Strategic Analysis of Griefing Attack in Lightning Network." IEEE Transactions on Dependable and Secure Computing. Manuscript Submitted.

Acknowledgements

I think words are not enough to express gratitude and acknowledge the support I have received throughout this journey. For me, this is not just a degree, it was an enriching experience and as the saying goes *"Life is the best teacher!"*.

I would to take this opportunity to thank my supervisor Dr. Sushmita Ruj for her continuous support, advice, encouragement, and motivation. Her guidance and knowledge helped me in pursuing good research and writing this thesis. It is a privilege to have a supervisor who has constantly supported me, both academically and personally. I would like to express my deep gratitude to my co-supervisor, Prof. Bimal Kumar Roy for his constant support and encouragement throughout my Ph.D. years.

I wish to offer my gratitude to my course teachers, Prof. Palash Sarkar, Prof. Rana Barua, Prof. Sushmita Sur Kolay, Prof. Mandar Mitra, Dr. Sourav Sengupta, Dr. Pinakpani Pal, Prof. Sandip Das, Prof. Mridul Nandi, Dr. Debrup Chakraborty, Dr. Arijit Bishnu, Dr. Sushmita Ruj, Prof. Sasthi Charan Ghosh, Prof. Nabanita Das, Dr. Anisur Rahman Molla, and Prof. Ansuman Banerjee. I am really fortunate to have had such mentors during my M.Tech and Ph.D. These courses will always remain precious in my academic journey. I would be forever indebted to Dr. Anisur Rahman Molla for his technical guidance and advice that helped me with my research. I am obliged to Dr. Sarbani Palit for the kind help I got from her on several occasions.

My Ph.D. thesis would have remained incomplete had I not received generous support from my collaborators. I owe a deep sense of gratitude to Dr. Arindam Pal. I was lucky to have him as PI at TCS Innovations Labs Kolkata. He has taught me the methodology to carry out research work and present it as clearly as possible. I convey my gratitude to Prabal Banerjee, Ram Govind Singh, and Dr. Abhinandan Sinha. I have bugged them at odd hours with questions but they were really kind enough to provide invaluable inputs and have discussions whenever required

I was fortunate enough to work under the supervision of Prof. Anupam Chattopadhyay at NTU Singapore as a part of the M.Tech Summer Internship. It was my first hands-on experience on how to work on a Blockchain project. I enjoyed all the great moments with Mayank Raikwar, Rishav Chatterjee, and Dr. Sourav Sengupta, one of the excellent team from whom I have learned a lot. My sincere thanks go to Dr. Binanda Sengupta, Dr. Laltu Sardar, Dr. Aparajita Khan, Nishant Dhanaji Nikam, Manish Kumar, Debasmita Chakraborty, and Debendranath Das for discussing research ideas and guiding me whenever I faced any problem. I am grateful to Dr. Pedro Moreno Sanchez for providing an opportunity to work with him during my Ph.D. He is an excellent mentor and I have learned a lot from him. I thank Dr. Sandip Chakrborty for his reviewing my paper and providing his invaluable feedback.

Life is incomplete without friends, especially the crazy ones. Lucky to have Debasmita, Sreyosi, Akanksha, Manish, Mohammad, Manabendra, and Snehal di as my constant support team. Thanks for those virtual meetups, endless discussion on any topic, and cheering me whenever I felt disappointed. Special thanks go to all of my friends and seniors in ISI Kolkata. Thanks to Sanjana, Subhadeep da, Ajay, Debleena, Riya, Aparajita di, Anindita di, Sebati di, Shilpa, Prabal, Nishant, Debendra, Laltu da, Binanda da, Animesh, Jyotirmoy, Nilanjan da, Avijit da, Ashwin da, Avishek da, Samir, Nayana, Pritam, Mostaf da, Diptendu da, Abhinandan da, Soumya da, Partha da, Gopinath da, Suprita di, Rakesh, Sourav, Bibhas, Shibam, Panchalika, Veeresh, and many more who helped me at different times in different ways. They made my stay in ISI Kolkata both inspiring and enjoyable. A healthy mind resides in a healthy body, and I could not have aced this journey without the support of the ladies' hostel fitness instructor Mrs. Bijaya Saha.

I express my heartfelt gratitude to the non-academic staff of ASU, CSRU, the accounts section, and the Dean's office, who have assisted me with several logistics and official work.

I wish to thank my teachers and friends from my alma maters, Steel Carmel School Durgapur, and National Institute of Technology Durgapur. This list would be incomplete without thanking Tanushree, Poulomi, Ipsita, Ria, Purnata, Nishka, Adil, and Keshaw. I have literally tortured them with my incessant complaints and how they kept on encouraging me and acted as an anchor through the tough days.

Last but not the least, I will remain indebted to my family for their continuous support, love, and help. Without their encouragement, inspiration, and motivation, I would not have got the mental strength to pursue a career in academics amidst the societal stereotypes and prejudices.

Date: 28th September, 2022 Subhra Mazumdar

Abstract

Cryptocurrencies enable users to execute a financial transaction without relying on any third party. The use of Blockchain technology guarantees the security and immutability of transactions. Despite these features, blockchain-based financial transactions fail to compete with conventional payment systems like Visa, and PayPal, in terms of scalability. Layer 2 protocols built on top of blockchain solve the scaling difficulties that are faced by the major cryptocurrency networks. Payment Channel Network or PCN is one of the most widely deployed *layer 2* protocols. Users are allowed to execute off-chain payments, leading to high throughput. PCN relies on the underlying blockchain for security. After studying the literature, we observed that routing and payment in PCN are the two most challenging tasks. The network is susceptible to attacks where malicious players can intentionally stall payments and eliminate their competitors from the network.

In this thesis, we propose an efficient privacy-preserving distributed routing algorithm HushRelay. Experimental analysis shows that our proposed routing algorithm has a higher success ratio and lower execution time compared to the state-of-the-art. Given a set of routes, we propose an atomic and privacy-preserving multi-path payment protocol, CryptoMaze. No honest intermediary loses funds in the process, ensuring balance security. We observe that CryptoMaze is quite efficient and the communication overhead is within feasible bounds. We discuss the griefing attack, a major vulnerability in Bitcoin's PCN, and propose an efficient countermeasure for the attack, termed Griefing-Penalty. The penalty charged compensates parties who incurred loss by locking funds. We propose a new payment protocol HTLC-GP or Hashed Timelock Contract with Griefing-Penalty that demonstrates the utility of the countermeasure. Finally, we have analyzed griefing attacks in the network from a game-theoretic point of view and observed that HTLC-GP is weakly effective in disincentivizing the attacker in certain conditions. To further increase the cost of attack, we introduce the concept of guaranteed minimum compensation and integrate it into HTLC-GP. This modified payment protocol, HTLC-GP$^{\zeta}$, considers the participants to act rationally. By experimenting on several real instances of PCN, we observed that HTLC-GP$^{\zeta}$ is better than HTLC-GP to counter griefing attacks.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Cryptocurrencies are gaining prominence as an alternative method of payment. Blockchain, a decentralized public ledger, forms the backbone of such currencies. The identity of transacting parties remains hidden. The records stored in this distributed ledger are immutable and can be verified by anyone in the network. When a new block is added to the network, it references the previous block via its hash. Secure linking of blocks ensures that no one can modify the transactions recorded in the blockchain. Certain participants, termed as *miners*, compete to create new blocks, where each block constitutes a set of transactions. Once a block is created, the rest of the miners check the correctness of the block. If the block is valid then the next block is mined on top of this block. The miner who had mined the valid block gets rewarded. This entire process is termed as *Nakamoto Consensus* [103]. Anyone can join the network and start mining in a permissionless blockchain like Bitcoin and Ethereum. A new peer gets connected to the network via the gossip protocol. In this protocol, a node propagates the information received to its neighbors and the process continues. A Sybil-resistant paradigm termed as *proof-of-work* [103, 106, 33] is used where participants in a permissionless blockchain can add blocks to the blockchain and earn rewards. Miner's influence in the network is bounded by the investment made in terms of computational resources, hence a single miner cannot create an unlimited number of fake participants. The drawback of *proof-of-work* is that it is computation-intensive and impedes scalability [46], [113]. Lack of scalability hinders widespread adoption of Blockchain-based payment solutions unlike conventional methods of electronic payments like Visa, PayPal [137]. In Bitcoin, every transaction needs to be verified for signature and correctness of the format before it gets mined. Many blocks end up in the queue, facing considerable waiting time before it gets verified due to increasing network size. Ethereum [140] suffers from scalability issues due to constraints on the gas limit for processing transactions.

1.1 Factors limiting scalability in Blockchain

A limit on the block size was introduced to allow the formation of a decentralized peer-to-peer network. Each node has low bandwidth and limited computational powers. The limit on the block size has put a constraint on the number of transactions being processed. It is difficult to ensure that all the nodes will be in sync in terms of the latest view of the blockchain in a decentralized network. Once a valid block gets added to the blockchain, the next set of blocks is mined on top of it to ensure finality. We discuss certain factors which decrease the throughput [74]:

- **Latency.** It is defined as the time taken to confirm that a transaction has been included in the Blockchain. For Bitcoin, the average time to create and secure a block in the chain takes around 10 minutes and for Ethereum, it takes around 12.5 seconds.
- **Block Size.** This is the total size of the transaction that can enter the block. The limit of the block size is around 4 MB for Bitcoin.

Both these factors, lead to a decrease in the number of transactions processed in a unit of time. Other factors responsible are *Bootstrap time* and *Cost per Confirmed Transaction* [46]. The time taken for a new node to download the entire blockchain to validate the current system state is the Bootstrap time. The time factor is generally linear in the size of the chain. The cost required to confirm each transaction in Blockchain involves mining cost - resource (in terms of hardware, electricity) expended by miners in generating PoW for each block, validation cost - computation needed to check the correctness of input, output, and other cryptographic verification like signature verification, etc., bandwidth needed for transmission and receipt of blocks and the storage cost involved in downloading the entire blockchain history as well as a set of unspent transactions.

We discuss how relaxing the limit on the block size or latency for achieving scalability is not a solution and state the drawbacks:

- Increasing Block Size Limit: Greater the block size, the more the number of transactions processed. But this means more bandwidth and storage are required for processing each block. Nodes with greater processing power will be able to scale up the transactions leading to centralization in the network. Larger blocks propagate slowly through the network leading to an increase in the number of orphan blocks, and an increased probability of a successful double-spend [9].
- Decreasing Latency: The faster the blocks are produced, the higher the chance that the node produces a stale block. Given the block confirmation time in Bitcoin is 600 sec, if every 12 sec a new block is produced, then 12/600 or 0.02 stale blocks are produced per valid block. If the time is reduced to 60 sec, then 12/60 or 0.2 stale blocks are produced per valid block. This impacts the finality of transactions, may lead to soft forks, and reduces the security of the network [66], [41].

1.1.1 Scalability Trilemma

It is hard to achieve *Scalability, Security* and *Decentralization* at the same time in a distributed network. Improving one of the factors might lead to comprising at least one of the other two factors [115], [18]. For example, Bitcoin is decentralized and uses a secure public blockchain, but cannot process the order of hundreds of thousands of transactions per second. Permissioned blockchains like Hyperledger's Fabric have high transactional throughput but are centralized in the sense that only a few nodes are involved in achieving consensus in the network. We discuss each component of the trilemma.

Decentralization

A decentralized system means the system is controlled by multiple owners, and one does not overpower the other. More decentralized a system, the more secure it is. Ethereum and Bitcoin uses PoW [33], [106] for mining new blocks. This consumes a lot of energy and reduces speed and efficiency. Additionally, there is no central moderator to resolve the dispute.

Security

This property allows blockchain to be resilient to external attacks as well as internal disruption. As seen previously, the more decentralized a system, the less risk the system is susceptible to centralized attacks. For security, the blockchain network relies on a consensus mechanism like PoW where miners are required to solve the complex hash puzzle to include blocks. Consequently, this reduces throughput and increases network latency, losing out on speed and efficiency offered by centralized networks in processing transactions.

Scalability

The higher the scalability of the system, the greater is its capacity. In terms of blockchain, scalability ensures that a large number of the transaction gets processed. However, scalability is ensured at the cost of security. Ethereum intends to opt for an alternate mining mechanism like Proof of Stake [84]. However, it compromises on decentralization. Further, if the difficulty level of puzzles used in Proof-of-Work or PoW [33] is reduced for a faster validation process then this would compromise the security aspect.

Increasing the throughput is a challenging task. Any modification made to the underlying blockchain or any new protocol used for enhancing scalability must not introduce vulnerability in the system or compromise privacy. We discuss some of the scaling solutions that are adopted widely by the blockchain community in the next section.

1.2 Scaling Solutions in Blockchain

We categorize the solutions developed as *on-chain scaling* and *off-chain scaling*:

- On-Chain Solutions: On-chain refers to the *Blockchain layer* or *Layer 1*. Such solutions aim at making changes in the underlying blockchain for achieving scalability. This involves changing the rules of the protocol or increasing the amount of data in each block, or reducing the latency for increasing overall network throughput. An increase in block size will lead to the processing of more transactions. However, this would require more storage capacity of a full node to store the entire history, as well as centralization [9]. Later, SegWit or Segregated Witness was suggested. Since the signature of the transaction data occupied around 65% of the block storage, SegWit proposed segregating the signature script part of each transaction from a block and storing them into an extended block or witness [124]. This allowed packing more transactions into a single block.

  As stated previously, *Proof of Work* consumes too much computation resource as well as hinders scalability. Proof of Stake [84], [85] holds promise as it involves participants depositing a certain amount of cryptocurrency as a commitment to behaving honestly. Participants who have a certain stake in the network can validate a new block. If any validator produces an invalid block, the stakes deposited in the network get slashed. Other forms of consensus are proof of burn - the miner must ”burn” tokens i.e., they must be sent to an address so that these tokens are no longer accessible, to gain eligibility for mining a block. As more and more miners want to mine, they need to consistently burn more and more tokens to stay competitive [3]. In proof of capacity, miners dedicate available space of their storage device to store the list of possible solutions before the mining commences. If a solution matches with the hash value in the block, then it gets the mining reward [20]. Proof of elapsed time is used primarily in permissioned blockchain. Each participant node in the network generates a random waiting time and goes to sleep for the duration [19]. The node with the shortest waiting time wakes up and commits a new block to the blockchain. A new consensus protocol Prism [31, 142] achieves a throughput of over 70,000 transactions at low latency.

  Sharding is gaining a lot of importance in recent years. In this solution, the blockchain is divided into many units or shards. The workload of the master blockchain gets distributed in each unit, and they act independently [91]. Parallel computation in each of these shards enables faster computation and increases scalability. Shards can communicate with each other but report directly to the master blockchain. However, sharded blockchain is susceptible to 51% attack, and a security failure in one of the shards can fail the entire blockchain network.

On-chain scaling solutions may involve a hard fork in the system, and these are not easy to implement. The amount of scalability achieved falls short of the expected level of throughput provided by conventional payment systems.

- Off-Chain Solutions: Off-chain scaling means creating alternative protocols on top of layer 1. Hence, this is called *Layer 2*. Some solutions like payment channels rely on the blockchain for the security and resolution of a dispute. Other solutions like sidechain, and commit chain communicates with the underlying blockchain. However, each child chain follows its own rules to realize secure off-chain payments. Off-chain solutions massively cut down data processing on the blockchain by running computations off-chain. The trust assumption or the consensus mechanism used in the underlying blockchain remains unchanged, and the amount of data stored in the base layer gets minimized. It cuts down on the expensive consensus mechanism since it relies on layer one to resolve the dispute and reach an agreement. Since none of the transactions gets recorded in the blockchain, factors hindering scalability are the communication bandwidth and latency of the participants.

  Payment Channels [48], [113] stood out as a practically deployable solution. It is modular, without requiring any fundamental changes in the protocol layer. Two parties with some deposit made in the Blockchain network can mutually open a payment channel by locking their funds for a certain period. The funds locked in the channel enable several off-chain payments to be carried out between these two parties, by locally agreeing on the new deposit balance. Instead of recording all the transactions on-chain, the transaction is executed *off-chain*. We show in Fig. 1-1, how payments can be carried out off-chain between Alice and Bob by constructing payment channels. Disputes arising out of such transactions can be tracked and resolved on-chain. Nodes that are willing to transact but do not share a channel send the coins via an existing set of channels. This set of interconnected payment channels forms a *Payment Channel Network* or PCN.

1.2.1 Routing in Payment Channel Network

Routing of payment in PCN must be efficient but at the same time, it must preserve the privacy of payments. None of the intermediate parties must be able to identify the payer and payee. The state-of-the-art routing algorithms mainly focused on finding a single path for routing a transaction. But it may not be feasible to route high-valued payments via a single path. The channels in the network may not have sufficient balance for transferring the payment to the intended recipient. It is better to split the total amount into several partial payments and route it across multiple paths.

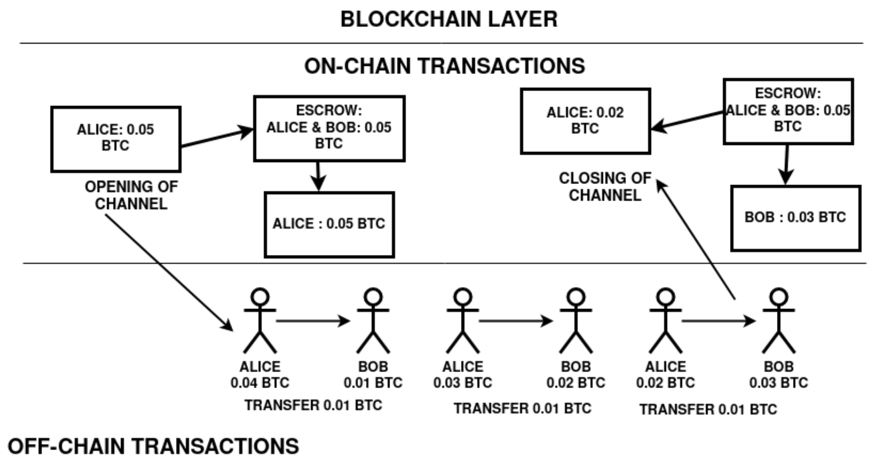


Figure 1-1: An instance of off-chain payment between Alice and Bob

The topology of PCN is publicly available since the opening of a payment channel is recorded on the Blockchain. Only the initial balance of the channel is known. Parties in the network may carry out several off-chain transactions that lead to a change in the residual balance of the channel. Channel's residual balance is kept private by the local nodes sharing a channel. We explain the problem with an example. The network instance in Fig. 1-2 receives a payment request $(S, R, 7$ units$)$. S had opened channels with A and B having an initial balance of 15 units and 10 units respectively. After execution of several off-chain payments, the residual balance of channel SA is 7 units, and the residual balance of SB is 8 units. S observes channel SA has the required balance but the residual balance of channels AC and CR is not known. In Fig.1-3, we represent the residual balance of individual channels based on the knowledge of each node. A knows that the residual balance of channel AC is 6 units and C knows the residual balance of channel CR is 6 units. Had S communicated with A and C, it would have figured out that only 6 units could be routed through path $SA \rightarrow AC \rightarrow CR$. The rest of 1 unit could be routed through path $SB \rightarrow BD \rightarrow DR$.

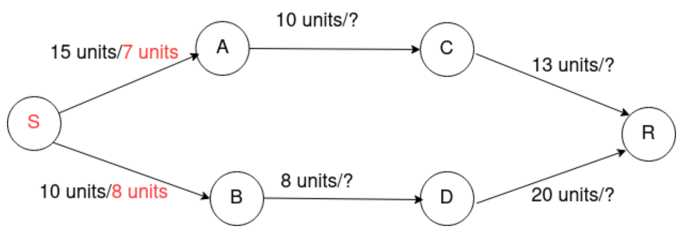


Figure 1-2: S knows the topology but lacks information of the residual balance of channels not directly connected to it.

We infer that none of the conventional centralized routing algorithms would work for routing

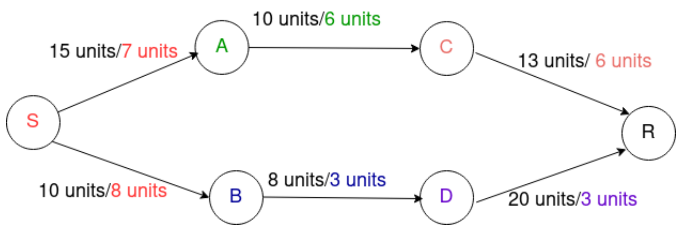


Figure 1-3: Complete information of residual balance in the network based on local information possessed by each node

high-valued payments in PCN. Distributed routing algorithm will be applicable where each node in the network makes a decision based on the information received from its neighbors. We discuss some of the challenges faced while designing such a routing protocol.

1.2.1.1 Challenges Faced

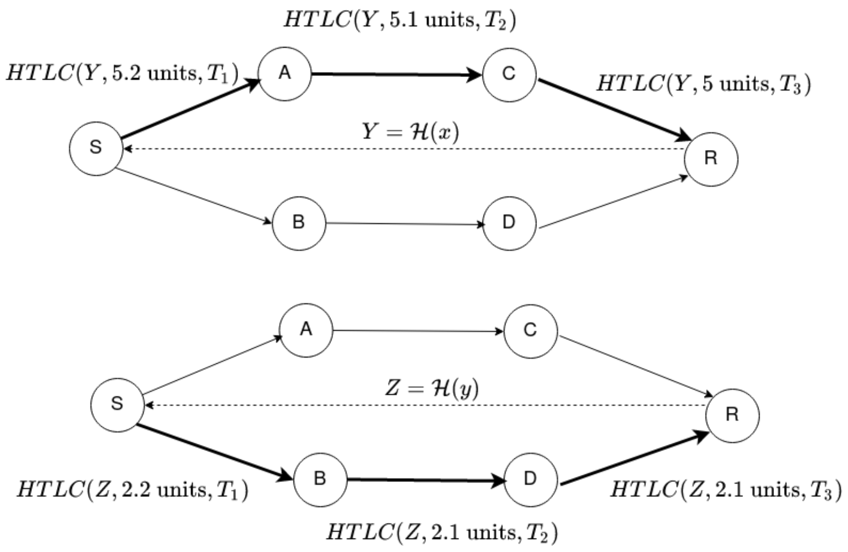
State-of-the-art mentions several distributed routing algorithms for PCN, but they suffer from various disadvantages. Elias et al. [120] state that a single node maintains a list of active vertices for executing the push relabel algorithm on single source-sink pair. Flare [114] violates privacy as intermediate users reveal the payment channel's residual capacity to the sender. The latter uses this information for the computation of the maximum flow that can be routed through a given path. Canal [136] entrusts a single node for computing maximum flow in a graph. Landmark-based routing algorithms [92], [122] decide the number of landmarks by trial and error. If the total number of landmarks is k, then the payment value is split into k microtransactions randomly without considering the nature of the graph. Such a myopic approach for routing each microtransaction may fail as it does not allow optimal utilization of the available capacities present across multiple paths.

It was first mentioned in Elias et al. [120] that push relabel fits better as a routing algorithm for PCN because it proceeds locally, taking into account the residual capacity of each payment channel. However, the push relabels algorithm used for single source-sink pair [120] is not decentralized in nature. A distributed version of the algorithm was implemented in their paper for multiple source-sink pairs but it is not well defined. It is not clear how many payment transfers can be allowed at a time through a channel. Further, it was assumed that each payment value for a source-sink pair is unsplittable. This assumption does not work in real life since the payment value might be higher than the bottleneck capacity of a single path. Deciding feasible routes even for a single payment transfer is an involved process in a distributed network.

1.2.2 Multi-Path Payments in PCN

Once the set of routes is provided, it is not possible to simply transfer the payment from S to R. There is a risk that any intermediate party may just withhold the payment and steal coins from S. Certain cryptographic primitives are used to construct conditional payments to avoid such a problem. *Conditional payments* are a type of off-chain contract instantiated between two parties connected by a payment channel. Such a contract is not recorded on the blockchain. If a party P_1 intends to transfer l coins to party P_2 via the channel P_1P_2, the former constructs a puzzle and shares it with P_2. The conditional payment states that if P_2 can solve the puzzle within a given amount of time, then P_2 can claim l coins from P_1. If P_2 fails to respond within the given time, P_1 settles the transaction on-chain after the period elapses. The latter withdraws l coins locked in the conditional payment and closes the channel with P_2.

We define a conditional payment based on a hash function used for forwarding payment in a single path across multiple hops. This type of conditional payment is called *Hashed Timelock Contract* or *HTLC*.



$HTLC(Y, 5.1 \text{ units}, T_2)$
$HTLC(Y, 5.2 \text{ units}, T_1)$
$HTLC(Y, 5 \text{ units}, T_3)$
$Y = \mathcal{H}(x)$
S
A
C
R
B
D
$Z = \mathcal{H}(y)$
$HTLC(Z, 2.2 \text{ units}, T_1)$
$HTLC(Z, 2.1 \text{ units}, T_3)$
$HTLC(Z, 2.1 \text{ units}, T_2)$

Figure 1-4: Forwarding payment using HTLC across paths $SA \rightarrow AC \rightarrow CR$ and $SB \rightarrow BD \rightarrow DR$

In Fig. 1-3, we observed that payment is routed via paths $SA \rightarrow AC \rightarrow CR$ and $SB \rightarrow BD \rightarrow DR$. Until now, we did not discuss the incentive offered to an intermediate party for routing a payment. Each intermediate party reserves its capacity for routing the payment, hence the payer must pay a processing fee for the service offered. We assume that each intermediate party charges a processing fee of 0.1 units. We explain the conditional payment *HTLC* with respect to the instance shown in Fig.1-4. R samples a random value x and computes the hash $Y = \mathcal{H}(x)$. The value Y is shared with S. The latter initiates the conditional payment for the path $SA \rightarrow AC \rightarrow CR$. S constructs a conditional payment *HTLC(Y, 5.2 units, T_1)* and forwards it to A. It means that if A can provide the preimage of Y within time T_1, it can claim 5.2 units from S. A deducts 0.1 units and forwards a conditional payment *HTLC(Y, 5.1 units, T_2)* to C, reusing the hash value. Finally C deducts 0.1 units and forwards *HTLC(Y, 5 units, T_3)* to R. Now R has the preimage of Y so it can easily initiate the payment process by releasing x. Similarly, R will sample another random value y, construct $Z = \mathcal{H}(y)$, and share it with S. The latter will generate a conditional payment using Z for the path $SB \rightarrow BD \rightarrow DR$. It looks like a straightforward extension of single-path payment protocol to multi-path payment protocol. However, there are quite a lot of challenges faced while designing payment protocols for multipath payment that we discuss next:

1.2.2.1 Challenges Faced

- **Atomicity of Payments**: Atomicity of payment means a payer is either successful in transferring the full amount to the payee, or the payment fails in its entirety. In Fig. 1-4, if the payment fails in the path $SB \rightarrow BD \rightarrow DR$, then R ends up receiving the partial amount from S. Thus extending the single path payment protocol to multi-path payment may not ensure atomicity.

- **Privacy Violation**: If all the intermediate nodes decide to settle the payment on-chain, then with a high probability identity of the sender and recipient of each partial payment gets revealed. Since the same hash value is used across all the channels routing the payment, it is quite easy to correlate once the transaction is recorded in the blockchain.

- **Stealing Fee of Honest Intermediaries**: Assume that two parties B and R are controlled by an adversary. Once R receives the conditional payment, instead of releasing the secret x to D, it reveals the secret directly to B and cancels the conditional payment forwarded by D. So B receives 5.2 units from S upon releasing x. D is under the impression that the payment got canceled, but S considers the payment successful upon receipt of x. It transfers the coins to B. Thus B and R have successfully stolen D's processing fee. This is called *wormhole attack* and is a major vulnerability faced in HTLC.

Designing an efficient multi-path payment protocol that addresses these challenges is a non-trivial task.

1.2.3 Problem of Griefing Attack

In the payment instance discussed in Fig.1-4, R has to release the preimage x within time T_3 and claim the payment from C. If R does not respond, then after elapse of T_3 unit of time, D settles the transaction on-chain and closes the channel CR. But in the process, none of the intermediate parties can utilize the coins locked in the conditional payment for the next T_3 units of time. If T_3 is around 24 hrs, then R manages to seize the coins in all the three channels SA, AC, and CR for a single day, as shown in Fig.1-5. This attack is called *Griefing Attack*. If R had taken a decision instantly, then S, A and C could have used their channel balance for routing other payments and gained a processing fee.

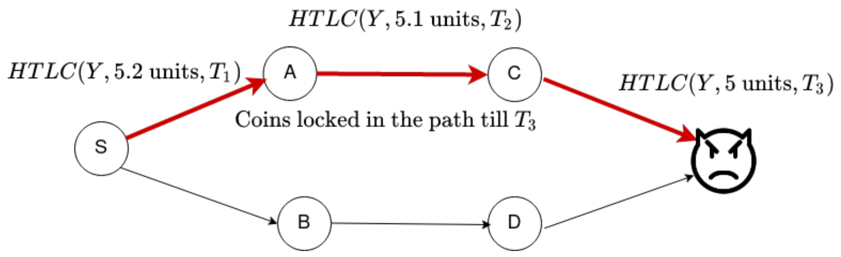

$HTLC(Y, 5.1 \text{ units}, T_2)$
$HTLC(Y, 5.2 \text{ units}, T_1)$
A
C
$HTLC(Y, 5 \text{ units}, T_3)$
Coins locked in the path till T_3
S
B
D

Figure 1-5: Griefing Attack in the path $SA \rightarrow AC \rightarrow CR$

An adversary may initiate an arbitrary number of HTLC payments to a node under her control [121]. This node may then simply ignore incoming HTLC requests, forcing the involved nodes to wait for the time locks to expire. Upon expiry, the entire state is rolled back, circumventing fee deduction. A griefing attack allows an adversary to claim channel capacity temporarily, free of charge. One can eliminate certain edges in the network by forcing channel closure. An adversary can even eliminate competing nodes by mounting such an attack and stealing the profits they could have earned by processing payments. The attack is quite effective for stalling the network, reducing the network throughput, and liquidity, and isolating certain nodes from the network. Griefing attack remains an unsolved problem, persisting in all the time-locked-based payment protocols. We aim to propose a countermeasure that will disincentivize the attacker from mounting the attack.

1.3 Our Contribution and Organization of the Thesis

The rest of the chapters of the thesis are organized as follows. In Chapter 2, we discuss the notations, cryptographic primitives, and background needed to understand the thesis. In Chapter 3, we provide a detailed survey of the existing literature on payment channel networks. We summarize our contribution in other chapters of this thesis as follows:

- **HushRelay: A Scalable Routing Algorithm for Off-Chain Payments**. In Chapter 4, we discuss the problems associated with existing routing schemes for PCN. We propose an efficient privacy-preserving routing algorithm that takes into account the funds left in each channel while splitting the transaction value across several paths. The proposed routing algorithm is modular and can be combined with any other privacy-preserving payment protocol.

- **CryptoMaze: Privacy-Preserving Splitting of Off-Chain Payments**. In Chapter 5, we state the problem associated with splitting high-valued payments across multiple paths. We discuss state-of-the-art payment protocols that either fail to achieve atomicity of payments across multiple paths or violate privacy. As an alternate, we propose a *secure* and *privacy-preserving* payment protocol, *CryptoMaze*, which guarantees *atomicity* and *unlinkability* of payments split across multiple paths. No honest intermediary loses funds in the process, ensuring *balance security*. We define the model in the *Universal Composability* framework and discuss the security of our construction.

- **Griefing-penalty: Countermeasure for Griefing Attack in Lightning Network**. In Chapter 6, we discuss *Griefing Attack*, a major vulnerability in Lightning Network, whereby an adversary intentionally exhausts the channel capacity of the network. Mitigating *Griefing Attack* remains an open problem. We have proposed an efficient countermeasure for the attack, known as *Griefing-Penalty*. Mounting such an attack requires the attacker to pay a penalty proportional to the *collateral cost* of executing a payment. The penalty charged compensates parties who incurred loss by locking funds. To realize it, we propose a new payment protocol Hashed Timelock Contract with Griefing-Penalty or *HTLC-GP*. It not only preserves privacy but also ensures that an attacker cannot ascribe blame to any honest party present in the path of transferring the payment.

- **Strategic Analysis of Griefing Attack in Lightning Network**. In Chapter 7, we discuss that HTLC-GP works under the classical assumption of participants being either honest or malicious but fails for rational participants. The security proof of the countermeasure does not analyze the attacker's behavior with the increasing cost of mounting the attack. To address the gap, we introduce a game-theoretic model for griefing attacks in Hashed Timelock Contract or HTLC. Using the same model, we analyze another payment protocol Hashed

Timelock Contract with Griefing-Penalty, or HTLC-GP, that claims to counter griefing attacks. We find that HTLC-GP is weakly effective in disincentivizing the attacker in certain conditions. To further increase the cost of attack, we introduce the concept of guaranteed minimum compensation that is used for controlling the maximum allowed path length for routing payment. We integrate it into HTLC-GP and propose a modified payment protocol HTLC-GP$^{\zeta}$. By experimenting on several instances of Lightning Network, we observed that HTLC-GP$^{\zeta}$ is better than HTLC-GP to counter griefing attack

1.4 List of Manuscripts and Publications

1. Subhra Mazumdar, Sushmita Ruj. Book Chapter - *Layer-1 Scaling solutions for Blockchains* in *"Blockchains - A Handbook on Fundamentals, Platforms and Applications"*. Editors: Sushmita Ruj, Salil Kanhere, Mauro Conti. Springer. To Appear.

2. Subhra Mazumdar, Sushmita Ruj, Ram Govind Singh, and Arindam Pal. "HushRelay: A privacy-preserving, efficient, and scalable routing algorithm for off-chain payments." In 2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC). DOI: 10.1109/ICBC48266.2020.9169405

3. Subhra Mazumdar, and Sushmita Ruj. "CryptoMaze: Privacy-Preserving Splitting of Off-Chain Payments." In IEEE Transactions on Dependable and Secure Computing. DOI: 10.1109/TDSC.2022.3148476

4. Subhra Mazumdar, Prabal Banerjee, and Sushmita Ruj. "Time is Money: Countering Griefing Attack in Lightning Network." In 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom). DOI: 10.1109/TrustCom50675.2020.00138

5. Subhra Mazumdar, Prabal Banerjee, and Sushmita Ruj. "Griefing-penalty: Disincentivizing Griefing Attack in Lightning Network." arXiv preprint arXiv:2005.09327 (2020). *Under Submission* .

6. Subhra Mazumdar, Prabal Banerjee, Abhinandan Sinha, Sushmita Ruj and Bimal Kumar Roy. "Strategic Analysis of Griefing Attack in Lightning Network." arXiv preprint arXiv:2203.10533 (2022). *Under Submission*.

Chapter 2

Preliminaries and Background

In this chapter, we define the notations, formally define cryptographic primitives, and discuss the background relevant to this thesis. Some notations and tools specific to a chapter are described in the respective chapter.

2.1 Notations

We describe the notations we use throughout this thesis in Table 2.1.

2.2 Probabilistic Polynomial-time (PPT) algorithm

An algorithm is said to be *polynomial time* if for all inputs x, the algorithm has a run-time bounded by some polynomial in $|x|$, where $|x|$ is the length of x when represented as a binary string.

When an algorithm makes a random choice from two outcomes, where each choice is selected with probability $\frac{1}{2}$, such algorithms are called *probabilistic algorithm*. The choice made is viewed as algorithmic coin tosses. A probabilistic algorithm $\mathcal{A}$ on input x may have multiple outputs depending on the outcome of coin tosses and $\mathcal{A}(x)$ denotes the probability distribution over all possible outputs.

Combining these two concepts, an algorithm $\mathcal{A}$ is said to be *probabilistic polynomial time (PPT)* if, for any input x, the expected run time taken over all possible coin tosses is polynomial in $|x|$, regardless of the outcome of coins tosses [65].

2.3 Negligible Function

In cryptography, a negligible function is defined with respect to the security parameter λ [35]. A function $g : \mathbb{N} \rightarrow \mathbb{R}$ is called negligible if for all $c \in \mathbb{N}$, there is an integer λ_c such that $g(\lambda) \leq \lambda^{-c}$ for all $\lambda \geq \lambda_c$.

Table 2.1: Notations used

Notation	Description
λ	the security parameter
$\mathcal{A}, \mathcal{Z}$	probabilistic polynomial-time (PPT) algorithms
Sim	Simulator, also a PPT algorithm
$a \xleftarrow{\$} S$	a is chosen from a set S uniformly at random
$\mathbb{N}$	the set of natural numbers
$\mathbb{R}$	the set of real numbers
q	A large prime number where $q \in \mathbb{N}$
$\mathbb{Z}$	the set of integers
$\mathbb{Z}_\mathrm{q}$	the set of congruence classes of integers modulo q (or the ring of integers modulo q)
$\mathbb{Z}_q^*$	the subset of $\mathbb{Z}_q$ whose elements are coprime to q
$\mathbb{G}(\mathbb{Z}_\mathrm{q})$	Elliptic curve over $\mathbb{Z}_q$
$\mathcal{G}$	Base point of elliptic curve $\mathbb{G}$
sk	Secret key
pk	Public key corresponding to secret key sk
$G := (V, E)$	Representation of the Payment Channel Network as a bidirected graph
V	Set of nodes in G
E	Set of edges in G where $E \subset V \times V$
(S, R, α)	Transaction request in G
S	Payer/Sender, $S \in V$
R	Payee/Receiver, $R \in V$
α	Amount to be transferred from S to R
P	Path connecting S to R
n	Length of the path P
$U_i \in V, i \in [0, n]$	Nodes in $P, U_0 = S, U_n = R, (U_i, U_{i+1}) \in E$
α_i	Amount to be transferred from U_i to U_{i+1}, $\alpha_{n-1} = \alpha$
$locked(U_i, U_j)$	Amount of funds locked by U_i in the payment channel (U_i, U_j)
$remain(U_i, U_j)$	Net balance of U_i that can be transferred to U_j via off-chain transaction
$f(\alpha_i)$	Processing fee charged by U_i for forwarding the payment $\alpha_i, i \in$ [1,n-1]
$\mathcal{H}\{0,1\}^* \to \{0,1\}^\lambda$	Standard Cryptographic Hash function
Δ	Worst-case confirmation time when a transaction is settled on-chain
γ	Rate of griefing penalty (per minute)

2.4 Cryptographic Primitives

2.4.1 Hash Function

Cryptographic hash function $\mathcal{H}$ is a one-way function that maps binary strings of arbitrary length to binary strings of a fixed length λ. It is represented as $\mathcal{H} : \{0,1\}^* \to \{0,1\}^\lambda$ and it must have the following properties [81]:

- *Collision-resistance*: A hash function $\mathcal{H}$ is said to be collision resistant if it is infeasible for a PPT adversary $\mathcal{A}$ to find a pair of values x and x' such that $\mathcal{H}(x) = \mathcal{H}(x')$. In other words, if for any PPT adversary $\mathcal{A}$ there exists a negligible function $g(\lambda)$ s.t.:

$$Pr[\mathcal{H}(x) = \mathcal{H}(x') \text{ and } x \neq x' : (x, x') \leftarrow \mathcal{A}(\lambda)] \leq g(\lambda) \tag{2.1}$$

  The property of collision resistance is a strong security requirement.

- *Second preimage resistance*: A hash function $\mathcal{H}$ is second preimage resistant if given x it is hard for a PPT adversary $\mathcal{A}$ to find x' such that $\mathcal{H}(x) = \mathcal{H}(x')$.

- *Preimage resistance*: A hash function $\mathcal{H}$ is preimage resistant if given some y it is hard for a PPT adversary $\mathcal{A}$ to find a value x' such that $\mathcal{H}(x') = y$. This is the property of one-way function where it is computationally infeasible to obtain the input, given the output.

2.4.2 Elliptic Curve Groups

Given a large prime number q, an elliptic curve $\mathbb{G}$ is an equation of variables x and y, having the form [81]:

$$y^2 = x^3 + Ax + B \; mod \; q \tag{2.2}$$

where $A, B \in \mathbb{Z}_q$ are constants with $4A^3 + 27B^2 \neq 0 \; mod \; q$. Let $\mathbb{G}(\hat{\mathbb{Z}_q})$ denote the set of pair of $(x, y) \in \mathbb{Z}_q \times \mathbb{Z}_q$ satisfying Eq. 2.2. The elements of set $\mathbb{G}(\mathbb{Z}_q) = \mathbb{G}(\hat{\mathbb{Z}_q}) \cup \{\mathcal{O}\}$, where $\mathcal{O}$ is the point at infinity, are called points on the elliptic curve $\mathbb{G}$.

2.4.3 Elliptic Curve Discrete Logarithm Problem

A polynimial time algorithm Gen on input 1^λ, generates the elliptic curve $\mathbb{G}$ over $\mathbb{Z}_\mathrm{q}$, with base point $\mathcal{G}$, q is large prime. the elliptic curve discrete logarithm problem (ECDLP) is defined as follows:

Definition 2.1. *(Discrete Logarithm Problem).* Given points $\mathcal{G}, Q \in \mathbb{G}(\mathbb{Z}_\mathrm{q})$, find an integer a such that $Q = a\mathcal{G}$, if a exists. [58].

This computational problem is called the *Elliptic Curve Discrete Logarithm Problem* which forms the the fundamental building block for elliptic curve cryptography. The discrete logarithm problem is said to be hard relative to Gen if for PPT algorithms $\mathcal{A}$, there exists a negligible function g such that

$$\Pr[\mathrm{DLog}_{\mathcal{A},Gen}(Q) = a] \leq g(\lambda) \tag{2.3}$$

2.4.4 Elliptic Curve Digital Signature Algorithm (ECDSA)

The Elliptic Curve Digital Signature Algorithm (ECDSA) is the elliptic curve analog of the Digital Signature Algorithm (DSA) [78]. Given a collision-resistant hash function $\mathcal{H}$, a private and public key pair is generated by sampling a random value x and corresponding public key $Q = x\mathcal{G}$ where $\mathcal{G}$ is the base point of the elliptic curve $\mathbb{G}(\mathbb{Z}_q)$, the signature algorithm over a message m proceeds as follows:

- Sample a random value k, construct $R = k\mathcal{G}$ 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 $S' = \frac{e\mathcal{G}+r.Q}{s}$. Let x co-ordinate of S' be s_x. If $r \overset{?}{=} s_x \ mod \ q$ then return 1 else return 0.

2.4.5 Homomorphic One-Way Function

A function $\hat{g} : X \rightarrow Y$ is one-way if, given a random element $y \in Y$, it is hard to compute a $x \in X : \hat{g}(x) = y$. A function $\hat{g}$ is homomorphic if X and Y are two abelian groups and for each pair $(x_1, x_2) \in X^2$, it holds that $\hat{g}(x_1 \circ x_2) = \hat{g}(x_1) \circ \hat{g}(x_2)$, where $\circ$ denotes the group operation [94]. On multiplying a scalar quantity, say j, with $\hat{g}(x)$ results $\hat{g}(jx)$.

2.5 Universal Composability Model

For modeling the security and privacy definition of payment across several payment channels under concurrent execution of an instance of a protocol described in Chapter 5, we take the help of the Universal Composability framework, first proposed by Canetti et al. [43]. Notations and assumptions used for ideal functionality are similar to [93]. The nodes of the network are modeled as interactive Turing machines, denoted by $\mathbb{U} = \{\mathbb{U}_\mathrm{i}\}$, $0 \leq i \leq r$, $r \in \mathbb{N}$, U_0 denotes the initiator of protocol and U_r denotes the receiver, which communicates with an ideal functionality $\mathcal{F}$, defined later, 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.

Given a real protocol Π, which is also termed as the real-world execution, where parties interact amongst themselves, we need to define an ideal functionality $\mathcal{F}$ that mimics the execution of protocol in a trusted environment. Both worlds receive their inputs and send their output to the environment $\mathcal{Z}$. In static corruption, adversary $\mathcal{A}$ can corrupt parties before execution of the protocol. We denote $REAL_{\Pi}^{\mathcal{Z},\mathcal{A}}(\lambda, x)$ as the output of the real-world execution of protocol Π with input x. In the ideal world, each party in set $\mathbb{U}$ is a dummy party. Such parties forward their inputs to the ideal functionality $\mathcal{F}$. This same as the honest parties communicating via secure authenticated communication channels in the real world. The ideal functionality $\mathcal{F}$ mimics the execution of protocol Π in a trusted environment. The security properties of the protocol Π are defined and proved for the ideal world execution. The ideal world adversary, defined Sim, attacks the ideal functionality $\mathcal{F}$ and receives input and sends output to $\mathcal{Z}$. $IDEAL_{\mathcal{F}}^{\mathcal{Z},Sim}(\lambda, x)$ denotes the output of the ideal world execution. The task of $\mathcal{Z}$ is to try to distinguish between the interactions of the real world and the ideal world. A protocol Π is said to be *UC-secure* if any PPT environment $\mathcal{Z}$ can distinguish with negligible probability whether it is interacting with the ideal or real world. We formally define this as follows:

Definition 2.2. ***UC Definition of Security***. *Given that λ is the security parameter, a protocol Π, UC-realizes an ideal functionality $\mathcal{F}$ if for all computationally bounded adversary $\mathcal{A}$ attacking Π there exist a probabilistic polynomial time ($\mathcal{PPT}$) simulator Sim such that for all $\mathcal{PPT}$ environment $\mathcal{Z}$ such that $IDEAL_{\mathcal{F}}^{\mathcal{Z},Sim}(\lambda, x)$ and $REAL_{\Pi}^{\mathcal{Z},\mathcal{A}}(\lambda, x)$ are computationally indistinguishable.*

One can use some ideal functionalities like $\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_m$ as subroutines in the protocol Π as per the universal composition theorem [42]. Such a protocol Π is said to be UC-realize the ideal functionality $\mathcal{F}$ in the hybrid world [49]. In other words, any PPT environment $\mathcal{Z}$ can distinguish its interaction with the ideal world and hybrid world with negligible probability. Later the ideal functionalities in protocol Π are replaced with the respective protocol, and it must be shown that the environment $\mathcal{Z}$ cannot distinguish whether it is interacting with the hybrid world or the real world.

2.6 Game Theory

Game defines the interaction between rational players, where each player chooses his or her strategies to maximize a well-defined payoff. Each player is considered intelligent enough to compute his or her best strategies knowing that all other players are also trying to obtain their respective best possible outcomes [95]. *Game theory* is the logical and mathematical analysis that models the interaction between the rational players and describes a method for predicting the result of the interactions among the players using equilibrium analysis [104].

An *extensive form representation of a game* provides a detailed representation of a game, including the sequence of moves and information beneficial to players. Before formally defining the extensive form representation, we explain the concept of *information set* of a player [104].

Definition 2.3. *(Information Set). An information set of a player describes a collection of all possible distinguishable circumstances based on which a player chooses to make a move. Each decision node corresponds to a unique sequence of actions from the root node of the game tree to the decision node, thus each information set of a player consists of all proper subhistories relevant to that player which are indistinguishable to that player.*

Definition 2.4. *(Extensive Form Game)[104]. "An extensive form game Γ is a tuple $\Gamma = \langle N, (A_i)_{i \in \mathbb{N}}, \mathbb{H}, P, (\mathbb{I}_i)_{i \in \mathbb{N}}, (u_i)_{i \in \mathbb{N}} \rangle$ where :*

- $N = \{1, 2, \ldots, n\}$ *is a finite set of players.*
- A_i *for* $i = 1, 2, \ldots, n$ *is the set of actions available to player* i.
- $\mathbb{H}$ *is the set of all terminal histories where a terminal history is a path of actions from the root to a terminal node. Note that a particular terminal history is not a proper subhistory of any other terminal history.* $S_{\mathbb{H}}$ *is the set of all proper subhistories (including the empty history* ϵ*) of all terminal histories.*
- $P : S_{\mathbb{H}} \to N$ *is a player function that associates each proper subhistory to a certain player.*
- $\mathbb{I}_i$ *for* $i = 1, 2, \ldots, n$ *is the set of all information sets of player* i
- $u_i : \mathbb{H} \to \mathbb{R}$ *for* $i = 1, 2, \ldots, n$ *gives the utility of player* i *corresponding to each terminal history."*

Definition 2.5. *(Game with perfect information and game with imperfect information) A game with perfect information is one where the players are fully informed about the entire history. By* *entire history, we mean that each player is aware of the past moves of all other players as well as his past moves before he makes a move. If the player is partially informed, then it is called a game with imperfect information [104].*

Definition 2.6. *(Game with complete information and incomplete information) A game with incomplete information is one in which some players have private information about the game that other players do not know before the players to begin their first move. The initial private information that a player has, just before making a move in the game, is called the* type of the player *[104]. In a game with complete information, every aspect of the game is common knowledge.*

For the work done in Chapter 7, we specifically focus on *dynamic games of incomplete information*.

2.6.1 Dynamic Games of Incomplete Information or Sequential Bayesian Games

In this class of games, players move in sequence, with at least one player being uncertain about another player's payoff. To approach these games, it is necessary to define a belief system and a player's behavioral strategy. We define a *type space* for a player that is the set of all possible types of that player. A *belief system* in a dynamic game describes the uncertainty of that player of the types of the other players. A *behavioral strategy* of a player i is a function that assigns to each of $i's$ information set a probability distribution over the set of actions to the player i at that information set, with the property that each probability distribution is independent of every other distribution. A dynamic game of incomplete information consists of [62]:

- A set of players $\mathcal{I}$;
- A sequence of histories H^m at the m^{th} stage of the game, each history assigned to one of the players (or to Nature/Chance);
- An information partition. The partition determines which of histories assigned to a player are in the same information set;
- A set of pure strategies for each player i, denoted as S_i;
- A set of types for each player $i : \theta_i \in \Theta_i$;
- A payoff function for each player $i : u_i(s_1, s_2, \ldots, s_l, \theta_1, \theta_2, \ldots, \theta_l)$;

- A joint probability distribution $p(\theta_1, \theta_2, \dots, \theta_l)$ over types.

To analyze dynamic games with incomplete information, a new equilibrium concept *perfect Bayesian equilibrium* is used, which is defined as follows:

Perfect Bayesian Equilibrium: A perfect Bayesian equilibrium in a dynamic game of incomplete information is a strategy profile s and belief system θ such that:

- s is *sequentially rational* given beliefs θ. It means, given the beliefs and other players' strategies, no player can improve his or her payoffs at any stage of the game.
- θ is consistent with s, i.e., at information sets both on and off the equilibrium path, beliefs are determined by *Bayes' rule* and player's strategy profile s.

2.7 Background

2.7.1 Bitcoin

A secure, distributed currency must guarantee that only the owner of a coin can decide when and how to transfer a coin, and at the same time, prevent someone spends a coin twice (double spending). With the inception of Bitcoin by Satoshi Nakamoto in 2008 [103], a financial revolution began. It is the first open distributed system that allows users to transact securely and efficiently. In other words, Bitcoin enables users to join and leave the system on-the-fly without the need for identities (open/permissionless), does not require a trusted third party (decentralized), and guarantees that all transactions will be permanently written in an accessible and verifiable transaction ledger via an efficient consensus mechanism that is resilient to participants that deviate arbitrarily from the protocol specification, called Byzantine (secure). To achieve all these properties, Bitcoin employs a combination of tools, which we analyze below.

Addresses in Bitcoin. Users may generate any number of private/public key pairs. A Bitcoin address is derived from a public key and is used as a pseudonymous identity [23]. A private/public key pair can uniquely identify the owner of funds of an address. The addresses are used to send and receive bitcoins.

Transactions in UTXO model We define the *unspend transaction output* or UTXO model [27] before we go into the details of inserting conditions in the Bitcoin script. In this model, the output of Bitcoin transaction is defined as θ consisting of tuple (cash, ϕ). $\theta.\text{cash}$ denotes the number of coins held in the output and $\theta.\phi$ denotes the criteria that need to be fulfilled for spending the coins, encoded in the scripting language. If one single party has ownership of the coins, $\theta.\phi$ consists of a digital signature verification script. Similarly, if multiple parties have ownership of coins in output then a multi-signature forms the condition for spending the output. The signature can be verified using the party's public key. The signature scheme used in Bitcoin is the Elliptic Curve Digital Signature Algorithm (ECDSA).

Changing the ownership of output can be realized via transactions. A transaction is defined as `tx=(id,input,output)`. Transaction input is the existing outputs assigned to an owner who has the right to spend the coins. `tx.id` is the hash of `tx.input` and `tx.output`. Here we denote `tx.output=`$(\theta_1, \theta_2, \ldots, \theta_n)$ where each θ_i consists of coins and condition needed to spend these coins. A transaction is considered valid if it possesses a valid witness for every input which has not been spent previously. Such a transaction gets added to the blockchain only after it is accepted following the underlying consensus mechanism. The upper bound on the time taken to finalize a transaction is defined as Δ.

2.7.2 Blockchain

Blockchain is an ordered list of blocks. Each block contains a set of transactions. The first block in the blockchain is called the genesis block. It is the common parent of all the blocks in the blockchain. *Height of the blockchain* refers to the distance from the genesis block. Each block contains the hash of the parent block inside its block header. Sequences of hashes linking each block to its parent create a chain. A block has one parent but can have more than one child. Such a situation is termed as *fork* in the network. The issue is resolved when one of the children becomes part of the blockchain following the network's consensus [23].

Records in the blockchain are immutable. The property comes from the fact that a change in one block results in a change in the hash of the block. This results in a change in the child's hash that triggers changes in the grandchild's block, which in turn changes its grandchild. Recalculation of the hash value for a trail of blocks is computation-intensive. The longer the chain, the better is the security.

Consensus in Blockchain. Nakamoto's consensus ensures that all the independent nodes in the blockchain agree on the validity of a given block without relying on a third party. It involves the following steps [23]:

- Each transaction is verified by a node, i.e., the correctness of input and outputs, signature, and whether is any double-spending of a particular input.

- Once transactions are verified they are aggregated into a block by a miner. All the miners compete to demonstrate their computation power by solving a cryptographic hard puzzle through a *proof-of-work* algorithm, which we will explain later.

- A miner that adds the puzzle's solution in the block before anyone else emerges as the winner. It then broadcasts the block in the network. Participants verify the block independently, and the block is added to the chain if found valid.

Mining and Proof-of-Work

Mining ensures the security of the bitcoin system and enables participants to reach a consensus in a decentralized network without involving a third party [23]. Any node in the network can become a miner. The latter's job is to validate new transactions and record them in the blockchain. If a transaction becomes part of the blockchain then it is considered confirmed. Owners of the output of such a transaction can spend those coins. If a block is confirmed, then miners receive a block mining reward and transaction fees for all transactions in the block. A transaction fee is a difference between the transaction inputs and transaction outputs. To mine a block, miners compete to solve a cryptographic hash puzzle with a specified difficulty. Once a solution to the puzzle is discovered, it is included in the block. This process is called *Proof-of-Work*. It prevents a node from generating unlimited Sybil nodes in the network as they need to have sufficient computation power to win the mining race. This idea forms the basis of Bitcoin's security. When a miner has solved the puzzle, it receives a block mining reward for expending computation effort.

Due to the complexity of verification and *proof-of-work* algorithm, transactions in blockchain are slower compared to traditional payment systems.

2.7.3 Off-Chain Scaling Solutions

Off-chain scaling solutions address the problem of scalability in blockchain. These solutions, also known as *Layer 2* protocols, sit on top of the Blockchain layer. Parties mutually agree to execute transactions without recording them in the blockchain. However, the parties are not trusted and they could deviate from the protocol. In case of such disputes, second-layer solutions use the underlying blockchain to prevent a malicious party from stealing an honest party's funds. We discuss a layer 2 protocol, *payment channels* in the next section.

2.7.3.1 Bidirectional Payment Channel

The payment layer, also termed as Lightning Network for Bitcoin, proposed by Poon and Dryja [113] enables a large amount of off-chain payments without recording the same in the blockchain, leveraging on the security of the underlying Bitcoin Blockchain. The bidirectional channel between two parties allows the participants to *lock* funds in a joint multi-sig account, termed as *creation of channel*. Parties can then shift funds between themselves as per the transaction request. Such parties are also called *lightning nodes*. Such a set of nodes can be interconnected to form a network that can route payment between any parties. In the following sections, we describe the process of opening the channel, updating the channel balance, and closing the channel.

Opening of Channel

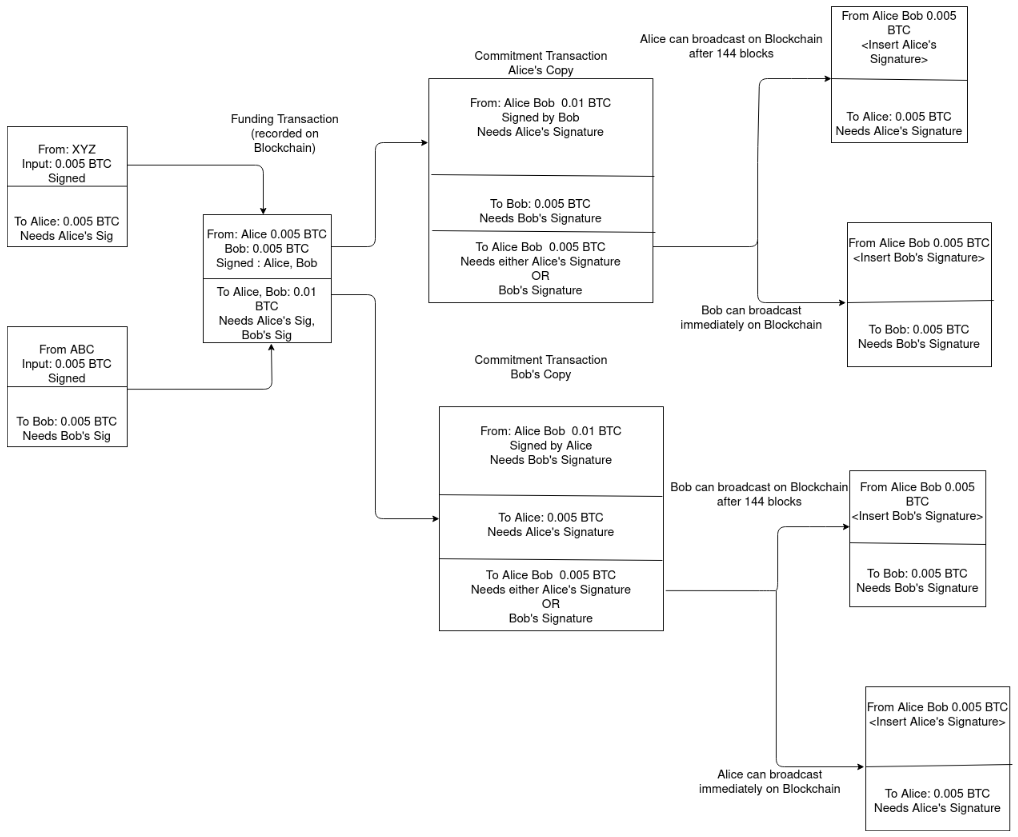

From: XYZ Input: 0.005 BTC Signed
To Alice: 0.005 BTC Needs Alice's Sig
From ABC Input: 0.005 BTC Signed
To Bob: 0.005 BTC Needs Bob's Sig
Funding Transaction (recorded on Blockchain)
From: Alice 0.005 BTC Bob: 0.005 BTC Signed : Alice, Bob
To Alice, Bob: 0.01 BTC Needs Alice's Sig, Bob's Sig
Commitment Transaction Alice's Copy
From: Alice Bob 0.01 BTC Signed by Bob Needs Alice's Signature
To Bob: 0.005 BTC Needs Bob's Signature
To Alice Bob 0.005 BTC Needs either Alice's Signature OR Bob's Signature
Alice can broadcast on Blockchain after 144 blocks
From Alice Bob 0.005 BTC <Insert Alice's Signature>
To Alice: 0.005 BTC Needs Alice's Signature
Bob can broadcast immediately on Blockchain
From Alice Bob 0.005 BTC <Insert Bob's Signature>
To Bob: 0.005 BTC Needs Bob's Signature
Commitment Transaction Bob's Copy
From: Alice Bob 0.01 BTC Signed by Alice Needs Bob's Signature
To Alice: 0.005 BTC Needs Alice's Signature
To Alice Bob 0.005 BTC Needs either Alice's Signature OR Bob's Signature
Bob can broadcast on Blockchain after 144 blocks
From Alice Bob 0.005 BTC <Insert Bob's Signature>
To Bob: 0.005 BTC Needs Bob's Signature
Alice can broadcast immediately on Blockchain
From Alice Bob 0.005 BTC <Insert Alice's Signature>
To Alice: 0.005 BTC Needs Alice's Signature

Figure 2-1: Setting up Bidirectional channel of 0.01 BTC between Alice and Bob

With mutual consent from both parties, they can deposit funds in a 2-out-of-2 multi-sig address to initiate a funding transaction. A *multi-sig address* is a bitcoin address that requires multiple private keys or the signature of the owners of such private keys to spend the bitcoins from the address. Once the amount is sent to the multi-sig address, the parties create the first commitment transaction before exchanging the signature for the funding transaction.

What if the opening transaction is signed before the creation of the first commitment transaction? Suppose both the parties exchange signature for the funding transaction. Then either of the parties has the right to broadcast the transaction and record the channel opening on-chain. If they do so, the funds can remain locked forever if either of the parties doesn't cooperate while spending from the funding transaction.

In the example illustrated below in Fig. 2-1, parties Alice and Bob send 0.005 BTC in the 2-out-of-2 multi-sig address but don't broadcast the funding transaction as of now. They create a first commitment transaction which allows each party to spend 0.005 BTC each before the funding transaction gets broadcasted. Alice and Bob both create a secret and exchange the hash H_{Alice} and H_{Bob}.

Alice creates a copy of the first commitment transaction for Bob where Alice sends 0.005 BTC to herself and the rest of the amount to Bob to a second multi-sig address. But Bob is allowed to spend the money sent to the second multi-sig address after a relative lock time from the time it got broadcasted in Blockchain. This is realized in the output script using a CheckSequenceVerify-Lock or CSV-lock. Alice can spend the amount from the second multi-sig address if Bob shares the preimage of the hash H_{Bob}. Alice signs her copy of the commitment transaction and sends it to Bob. Bob creates a similar copy of the commitment for Alice providing his signature. It contains output of 0.005 BTC which can be immediately spent by Bob and the rest of the output of 0.005 BTC for Alice to a second multi-sig address. Alice cannot spend this output before the elapse of the relative timelock period. Bob can spend the output of the second multi-sig address, provided Alice has provided the preimage of the hash H_{Alice}. With the half-signed transaction in place, both parties now exchange their signature and broadcast the funding transaction, which confirms the opening of channel [135].

Updating Channel Balance

Before creating a fresh commitment transaction, the parties exchange the preimage of the hashes exchanged for the previous commitment transaction. This procedure is known as *invalidating a transaction*. In the next step, both of them update the balance as per the terms of the transaction. Say if Alice wants to transfer 0.001 BTC to Bob. Both of them create fresh hash values H_{Alice1} and H_{Bob1} and exchange them. Then Alice creates a fresh commitment transaction with Alice sending 0.004 BTC to herself and 0.006 BTC to a second multi-sig address which can spend either by Bob after a relative timelock period or if Alice gets the preimage of the new hash value H_{Bob1}. Similarly, Bob creates a similar copy of the transaction and shares it with Alice. If either of the parties broadcasts an older transaction, then the counterparty uses the preimage of the stale hash value, H_{Alice}/H_{Bob}, and spends the output of such transaction immediately. The malicious party gets penalized since all the fund it had deposited into the channel is slashed from its account.

Closing Channel

Both parties can mutually decide to close the transaction by creating a new transaction reflecting the final balance based on the latest state of the channel. Alice and Bob put their signature on the closing transaction so that they can immediately spend the output of the transaction. If a party doesn't co-operate, then the counterparty can unilaterally close the channel by broadcasting its copy of the final transaction where it has to wait for the relative period before spending the output.

2.7.3.2 Payment Channel Network

Instead of opening a new channel, a party can find a path connecting it to a destination via some intermediaries. This saves the cost of locking fresh collateral in the channel as well as recording another transaction on the blockchain. This set of interconnected channels forms the *Payment Channel Network*. To send money from a payer to a payee via a set of nodes, it must be ensured that the intermediaries do not cheat and extort money from the payer, and at the same time, the payee receives the intended amount. We describe a very simple payment protocol that uses the hash as the cryptographic primitive in each of the off-chain contracts instantiated for forwarding conditional payment.

Hashed Timelock Contract or HTLC

Off-Chain contracts are smart contracts where the logic encoded in the contract is not run by the miners. It is mutually executed by the participants involved in instantiating the contract. The advantage of having off-chain contracts is that computation-intensive tasks can be executed without involving blockchain as long as participants behave honestly. An individual player can prove the correct contract state independently. Cheating is prevented as the state of the contract is signed by all the players. If a party misbehaves by broadcasting a wrong state in blockchain, the counterparty can raise a dispute and publish the valid accepted state. Hashed Timelock Contract or HTLC [113] is one such example used in PCN for forwarding conditional payments in the network. The logic used is a hash function, where players need to provide the preimage of the hash to claim coins.

Suppose Alice wants to transfer 0.001 BTC to Charlie via Bob, the latter claiming a processing fee of 0.001 BTC. Charlie samples a random value x and computes $H = \mathcal{H}(x)$ and sends it to Alice, as shown in Fig. 2-2. Alice uses onion routing for forwarding the conditional payment across the path $Alice \rightarrow Bob \rightarrow Charlie$. She forwards a conditional payment of 0.002 BTC to Bob. She sends the money to a different multi-sig address with the hash H. The output from the multi-sig address can be spent in two ways. Either Bob can provide the preimage x, provide his signature, and claim 0.002 BTC from Alice. Else after a certain period, say t units, Alice can sign and unlock this money. Note that this period is not a relative timelock but an absolute timelock (mentioned in the script as CheckLockTimeVerify or CLTV). Either Bob can settle the transaction off-chain with Alice by updating the channel balance upon providing the preimage x. But Bob still has not got the preimage. Upon decrypting the encrypted onion blob, he finds that the conditional payment has to be forwarded to Charlie. Hence, he will create a similar contract with Charlie, whereby it transfers 0.001 BTC to a new multi-sig address. The time period within which Charlie must provide the preimage is t_1 where $t_1 < t$, as shown in Fig. 2-3. This will allow Bob to get the preimage x and claim money from Alice [134], [107].

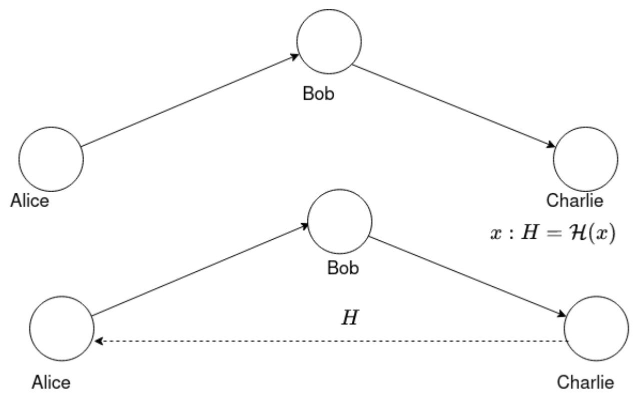


Figure 2-2: Setup Phase of HTLC

Let us define the procedure in detail. Let the initial balance of the channel between Alice and Bob be 0.01 BTC each, with either of the parties depositing 0.005 BTC. Alice creates a new multi-sig address sending 0.002 BTC with two options of spending the amount as mentioned before. Alice has a balance of 0.003 BTC, Bob has a balance of 0.005 BTC, and the money locked in hashed timelock contract is 0.002 BTC. Bob can use the preimage to claim the amount. If he

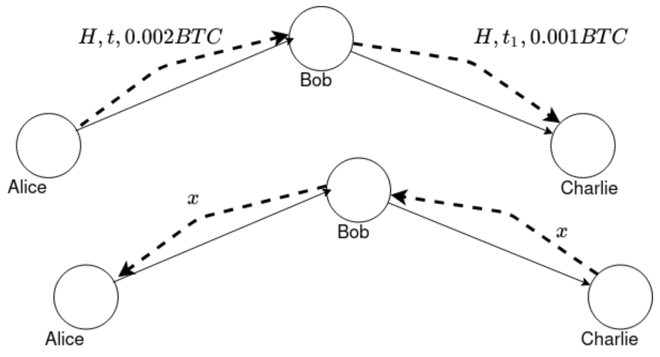


Figure 2-3: Routing payment from Alice to Carol using HTLC

goes on-chain and broadcasts the transaction then Alice can spend 0.003 BTC immediately. Bob can spend his output of 0.005 BTC after a relative timelock period. He can also claim the money locked in hashed timelock contract by providing his preimage and signature. However, again a relative timelock is added to this output as well where Bob can spend a transaction after a certain period has elapsed. Had Alice broadcasted the commitment transaction, then Bob could have spent immediately 0.005 BTC as well as the money locked in the hashed timelock contract.

Apart from Lightning Network, hashed timelock contract is used for routing payments in Raiden Network as well, a payment channel network for Ethereum.

Protection Mechanisms in Lightning Network

It might be possible that the path selected for routing payment may suffer from the failure of intermediate nodes. This might hinder the payment but we state two countermeasures to handle such situations [113]. We discuss certain protection mechanisms to prevent an honest node from losing money upon routing payments.

- *Settling on-chain*. While resolving a conditional payment, if a party finds its counterparty has stopped responding then it goes on-chain, reveals the preimage to claim payment, and closes the channel. For example, in Fig. 2-4, if Bob doesn't respond, Charlie broadcasts its transaction with preimage and signature and closes the channel formed with Bob. Charlie claims the money and Alice withdraws the money locked in the contract with Bob after elapse of lock time.

- *Rerouting*. An intermediate can stop forwarding the conditional payment to the next neighbor. For example, in Fig. 2-5, in the path $Alice \rightarrow Bob \rightarrow Charlie \rightarrow Dave$, Charlie

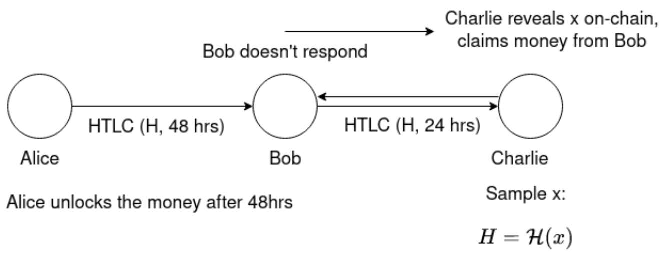


Figure 2-4: Settling of Payment on-chain

may not forward the payment. Ultimately the contract doesn't reach the recipient Dave. Alice may not wait for an indefinite period and start forming HTLCs across another path $Alice \rightarrow Bob \rightarrow Mathew \rightarrow Dave$. Simultaneously a new path connecting Dave to Alice is searched for so that in case of the HTLC via path $Alice \rightarrow Bob \rightarrow Charlie \rightarrow Dave$ gets established, the money can be transferred back to Alice via path $Dave \rightarrow Eric \rightarrow Bob \rightarrow Alice$. This ensures a risk-free payment via another path.

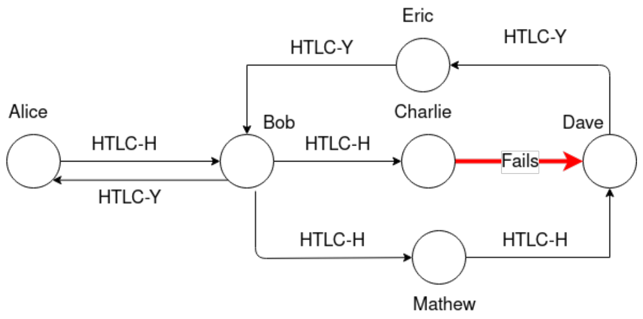


Figure 2-5: Rerouting payment from Alice to Dave via $Alice \rightarrow Bob \rightarrow Mathew \rightarrow Dave$ using condition H. If Charlie establishes HTLC with Dave, then the same must be refunded via path $Dave \rightarrow Eric \rightarrow Bob \rightarrow Alice$ using condition Y.

2.7.3.3 Vulnerabilities in PCN

We state the plausible attacks on Lightning Network which lead to loss of funds, loss of privacy, or even jamming of the network rendering it unsuitable for processing any payment.

Wormhole Attack

The Hashed Timelock Contract (HTLC) Protocol uses the same commitment across the path routing the payment. For example, if Alice wants to transfer 1 unit of the coin to Dave via Bob, Charlie, and Eve, then Eve constructs a hash value H and shares it with Alice. We assume that each party charges 0.1 unit as a processing fee. Thus Alice will transfer 1.3 coins via this path. After the conditions of payment reach Dave, he releases the preimage to Eve. However, as discussed before, this is susceptible to *Wormhole Attack* [94]. Eve and Bob can collude. Upon seeing that both have received the same condition for payment, instead of sharing the preimage with Charlie, she shares it directly with Bob. Just before elapse of lock time, Eve cancels the contract with Charlie and Charlie cancels the contract with Bob. But Bob and Eve manage to steal Charlie's processing fee, as shown in Fig. 2-6.

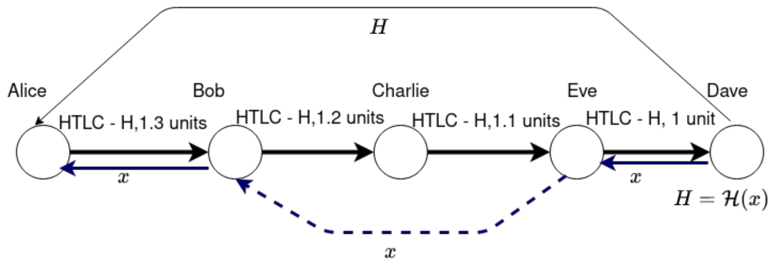


Figure 2-6: Wormhole Attack, Bob and Eve steal Charlie's fee

Griefing Attack

Given a PCN, consider a payment α has to be made via a path from payer to payee consisting of n payment channels, where $n \in \mathbb{N}$, $n > 1$. Let us index each channel by i where $i \in \{1, 2, \ldots, n\}$. The channel connected directly to the payer is indexed 1 and the channel ending with the payee is indexed n. For executing the payment, an amount α is locked in each of the payment channel i for a period of $(n - i + 1)\Delta$, which is also termed as *locktime*. $\Delta > 0$ is the worst-case confirmation time when the transaction is settled on-chain. Once the amount gets locked, it cannot be utilized before the elapse of the lock time. If an adversary controlling channel j, where $j \in [1, n]$, refuses to resolve the contract off-chain and raises a dispute, then the time taken to resolve it will be $(n - j + 1)\Delta$. It manages to lock α coins each in the preceding j payment channels, just by locking α coins in $(j + 1)^{th}$ channel. If the adversary is controlling the receiver then without incurring any cost for mounting the attack, it locks $(n - 1)\alpha$ for Δ unit of time. If an adversary manages to capture any node present in the middle of the path, then the attack results in worst-case collateral damage. The amount of fund locked is approximately $\frac{n}{2}\alpha$ for a time period of $\mathcal{O}(n\Delta)$, thus total loss incurred is $\mathcal{O}(n^2\Delta\alpha)$.

Example 1. *A wants to transfer funds to a node R. It leverages the existing payment channels AB, BC, CD, and DR for relaying funds from A to R, as shown in Fig. 2-7. A locks fund with B for 4*

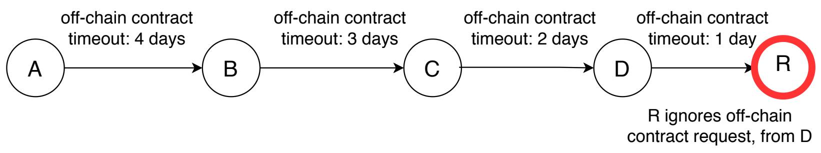

off-chain contract timeout: 4 days
off-chain contract timeout: 3 days
off-chain contract timeout: 2 days
off-chain contract timeout: 1 day
A
B
C
D
R
R ignores off-chain contract request, from D

Figure 2-7: Griefing Attack when R ignores *HTLC* request

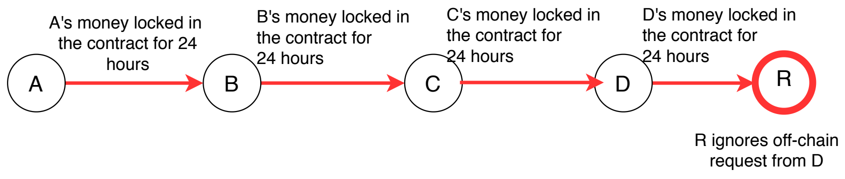

A's money locked in the contract for 24 hours
B's money locked in the contract for 24 hours
C's money locked in the contract for 24 hours
D's money locked in the contract for 24 hours
A
B
C
D
R
R ignores off-chain request from D

Figure 2-8: Funds locked in the path for 1 day

days, B locks its fund with C for 3 days, C locks funds with D for 2 days and ultimately D locks fund with R for 1 day. R ignores the request and refrains from releasing the payment condition, as shown in Fig. 2-8. The payment fails and D rolls back to its previous state after one day. D terminates its off-chain contract with C, and C cancels its contract with B, B, and A following suit as well. Hence R manages to lock coins in each payment channel for one entire day. None of the channels can utilize the locked amount before the contract established in the channel DR expires.

Chapter 3

Literature Survey

3.1 Routing Algorithms in Payment Channel Network

Routing payments from sender to receiver via a set of intermediate nodes without trusting any of them is a crucial problem in PCN. The difficulty lies in discovering channels with sufficient capacity for routing a payment. The opening balance of the channel is known publicly, the residual capacity of a channel might be different due to several off-chain payments. The routing algorithm must not reveal the identity of the payer or payee. We discuss a few routing algorithms and the security guarantee it offers.

"Flare: An approach to routing in lightning network" by Prihodko, Zhigulin, Sahno, Ostrovskiy, and Osuntokun (2016)

This routing algorithm for Lightning Network uses a trustless source routing scheme. It can also be scaled to the network size of at least several hundred or thousand nodes [114]. The algorithm comprises two different stage: *Proactive* and *Reactive*.

- Route Discovery (Proactive Part): Each node in Lightning Network constructs a routing table. The routing table is a subset of channels and helps in finding a path to the recipient. If the information is not sufficient to find a route, the table is used to determine the beacon nodes that assist in finding the route. In the initial stage, a node has a clear view of the neighboring nodes. With the help of beacon nodes, the visibility of the network for a given node is increased beyond its local neighborhood by incorporating random nodes. Hence with high probability, a node determines whether it can reach a particular node or not.

- Route Selection (Reactive Part): When a node receives a request to route a payment to a particular payee, it uses the routing table and the routing table of the payee to figure out a route from the payer to the payee. If no such routes exist, the beacon node corresponding to the payee and other nodes can pitch in with their routing table in constructing an appropriate path. If there exist several such routes, they can be ranked accordingly. The ranking is done based on some predetermined cost function. The cost function is defined based on static information like route length and dynamic information, like fees to use channels.

Based on the ranks assigned to the routes, the sender selects the best feasible path for routing its payment.

In this algorithm, each node keeps track of all the neighbors that are at a hop length of k from itself, i.e, $k - neighbourhood$, and figures out the weight of the links in this $k - neighborhood$. Privacy gets violated. Additionally, any credit change needs to be communicated in the $k - neighborhood$, leading to flooding of messages in the network, which is quite inefficient.

"SilentWhispers: Enforcing Security and Privacy in Decentralized Credit Networks" by Malavolta, Moreno-Sanchez, Kate and Maffei (2017)

SilentWhispers [92] proposes a distributed protocol for finding routes in a credit network that can transfer payment between a pair of sender and receiver. It applies a distributed landmark routing-based approach [136] for the credit network design [57]. Also, this is the first payment system that does not require the use of blockchain to ensure the integrity of the transactions. The steps of the protocol are as follows:

i. *Routing*: Given the assumption that the credit network is a connected graph and the set of landmarks is fixed, each landmark periodically constructs a spanning tree to account for the change in the network. To create the spanning tree, the landmark executes two instances of the Breadth-First-Search or BFS algorithm rooted in itself. The first instance calculates the shortest path from the landmark to each node, termed as *arborescence*, and the second instance calculates the shortest path from each node back to the landmark, termed as *anti-arborescence*.

ii. *Finding credit on a path*: *Multi-Party Computation* is used for secure computation of the credit available in a certain path. The landmarks receive a share of the credit on each link from the sender to the receiver. Given this information, the set of landmarks jointly computes the credit on the entire path. None of the landmarks learn about the result of the computation nor the credit on each link.

iii. *Path construction*: For the construction of the path, a user routing payment from sender to receiver uses a short-term verification key along with the long-term key. A user first signs a fresh verification key with her long-term key and creates a chain of signatures.

iv. *Accountability*: To resolve the dispute between two users of a link, the logs from the user are used to get the real value of the link.

"Settling payments fast and private: Efficient decentralized routing for path-based transactions" by Roos, Moreno-Sanchez, Kate and Goldberg (2018)

SilentWhisper has a lot of computation overhead, and the path returned by the algorithm may not be optimal. SpeedyMurmur [122] overcomes the problem by taking into account the availability of funds in the channel and the proximity of a neighbor to a given destination. This results in an efficient algorithm with flexible path selection. A set of *landmark nodes* is identified. Generally, a node that is highly connected forms the landmark. A payer traces a route to a landmark, and a payee traces a route to a landmark. Concatenating both, a route from payer to payee gets constructed. We describe the main subroutines as follows:

- *setRoutes*: Assuming k such landmarks have been selected, iterate over each landmark and assign a BFS-based coordinate to each of the nodes per landmark. Once a landmark is selected, an empty vector is assigned as its coordinate. The landmark selects all its neighbors and probes whether the links to such nodes have enough credit and have not been assigned any coordinate. If such a node is found, it is added to the spanning tree. The coordinate assigned to such a node is the concatenation of the parent's coordinate along with a random $b - bit$ string. In an asynchronous setting, an upper bound on the time limit is set by a node. If a neighbor of a node has to be added to the spanning tree, it must send a message within this time-bound. Once such a request is received, the node assigns this neighbor as its parent. Initially, a node adds all those neighbors having bidirectional non-zero capacity links. Later, all nodes having unidirectional non-zero capacity links get added.

  The process is repeated for the k landmarks resulting in k such spanning trees and coordinates based on each of the spanning tree. This is termed as *prefix embedding*. Let the coordinate with respect to landmark L_1 for node u and v be $id_1(u)$ and $id_1(v)$; then the distance between node u and v is

  $$dist(id_1(u), id_1(v)) = |id_1(u)| + |id_1(v)| - 2cpl(id_1(u), id_1(v)) \tag{3.1}$$

  $id_1(u)$ denotes coordinate length, $cpl(id_1(u), id_1(v))$ denotes the length of the common prefix i.e. when the id is assigned to each node u and v, this is the portion of the string matching in both. Such common prefix signifies the shared ancestor. To avoid double counting, this value needs to be deducted.

- *setCred*: A pair of nodes may need to change the capacity of the link (or the channel) present between them. Later, they check if such a change of capacity leads to a change in the coordinate. We describe the situations when there would be a possibility that the coordinates might change:
  - New non-zero link: Given a pair of nodes, if one of them does not belong to the tree then it chooses the other node as its parent.
  - New non-zero link: Given a pair of nodes say (u, v), if u has a parent with a unidirectional link and later, v forms a bidirectional link with u, then u must change its parent to v.
  - Removed link: If a node loses its link to its existing parent (i.e. credit in the link drops to 0), it should find a new parent.

    If a node changes its parent then its descendants must undergo a coordinate change as well. They inform their neighbors about the deletion of the existing coordinates. Later a new parent is chosen, and it undergoes a coordinate reassignment. If a node receives several requests from neighbors who can be its potential parent, it chooses the one having the shortest route to the landmark. The configuration change is recorded for each of the landmark nodes.
- *routePay*: For payment of value v from payer to payee, the payer splits the total transaction value into $k - shares$ i.e. $v_1, v_2, \ldots, v_k$ for each of the k landmarks. k such paths needs to discovered for transmitting each of the payment and k different addresses of the receiver is generated i.e. $dest_i(recv), i \in [1, k]$. For the share v_i, the path needs to be found out based on the coordinate assigned using landmark $L_i, i \in [1, k]$ as the reference. Starting from the payer, a neighbour, say m, is selected such that the distance to the destination node is shortest i.e. $d(id_i(m), dest_i(recv))$ is minimum and the link has capacity at least v_i. This is repeated for rest of the $k - 1$ shares as well. If the routing fails, then the route-search procedure is repeated for a different split of the transaction value.

“Coinexpress: A fast payment routing mechanism in blockchain-based payment channel networks” by Yu, Ruozhou, et al. (2018)

Landmark routing assumes a small set of trusted landmark users who controls the entire routing process. This may lead to the centralization of the P2P network. Coinexpres alleviates the problem by taking a distributed approach for PCN routing. It uses distributed Ford-Fulkerson algorithm and applies a locking technique for resolving concurrency among multiple simultaneous requests in a particular channel [144]. The drawback of the algorithm is that it is not privacy-preserving, as intermediate parties routing payment come to know about the channels and their residual balance in such a path.

"Flash: efficient dynamic routing for off-chain networks" by Wang, Xu, Jin, and Wang (2019)

SpeedyMurmur suffers from low transaction throughput and is susceptible to failure because the split in transaction value doesn't take into consideration the change in the capacity of each link. Flash deals with the problem by differentiating large payments, termed as *elephant payments*, from small-valued payments, termed as *mice payments* [138]. For elephants payments, a modified max-flow algorithm based on Edmond-Karp [54] is used for finding k such a set of good paths with sufficient capacity for routing large valued payments. Given the set of k paths, to route the payment using a minimum fee, an optimization program is designed to split the payments over k paths. For mice payments, each node maintains a routing table that figures out the path to the receiver. Each payer figures out m such shortest path where $m < k$. If such paths don't exist, they can be computed dynamically using the local topology and existing routing information. The routing table is periodically refreshed based on the change in the topology of the payment channel network. Once m such paths are found, the sender tries to route the full payment value via one path. If it doesn't succeed, it checks the amount remaining to be routed and tries to route it via the next available path.

"High throughput cryptocurrency routing in payment channel networks" by Sivaraman et al. (2020)

In landmark-based routing, dynamic update of the topology and link balances is a challenging task. To avoid such problems, a high throughput routing algorithm, *Spider* [125], has been proposed that packetizes transactions, splits them into several transaction units, and sends them via different paths at different rates. Each transaction unit can be transmitted using onion encryption, to hide the full route from intermediate routers. Breaking up payments into packets allows completing even large transactions on low-capacity payment channels over time. Congestion control across multiple paths ensures balance utilization of channels and fairness across flows.

Each payment channel has a router that maintains price variables, updated periodically based on the current arrival rate of transaction units in the channel, available channel balance, and the number of transactions currently queued up in the router. Transaction queues up in the router present in each payment channel whenever the channel lacks funds to forward them immediately. Queuing of the transaction indicates that either the transaction is being processed at a faster rate and the rate needs to be brought down or the channel lacks capacity. If the node uses a congestion control protocol that controls queues, it could detect both capacity and imbalance violations. Any standard congestion control algorithm used for routing in the network can be adapted in PCN. The routers monitor the time that each packet spends in its queue. If the time spent exceeds a certain threshold, the packet is marked. If the packet is already marked then these are forwarded. Once the receiver receives the packet, it sends an acknowledgment back to the sender that can interpret the existence of a route.

“RobustPay$^+$: Robust Payment Routing With Approximation Guarantee in Blockchain-Based Payment Channel Networks” by Zhang, Yuhui, and Dejun Yang (2021)

Spider [125] ignores the impact of constraints like transaction fees and timeliness of payment. Zhang et al. [147] had proposed *CheaPay* which aims to minimize the fee for routing payment across a single path. The drawback is that the algorithm is not resilient to transaction failures. [146] has proposed an efficient routing algorithm that is resistant to transaction failure but minimizes the worst-case transaction fee for a given payment. The protocol consists of three stages:

i. *Payment Path Construction*: Two node-disjoint paths are constructed for a given payment to guarantee robustness. The rationale behind this is that if one path fails, payment can be routed via the other path. A distributed 2-approximation algorithm is used here to minimize the worst-case transaction fee for routing payment.

ii. *Hashed Timelock Contract (HTLC) Establishment*: *HTLC* is modified to handle cancelation of payment. If the conditional payment is canceled or not satisfied in one of the paths, the coins locked in the conditional payment via the second path are refunded to the original sender. The modification is made by the use of two different preimages on each of the paths for each of the hash values.

iii. *Payment Forwarding*: After the *HTLC establishment* phase, if the payment is transferred successfully via one of the paths, the conditional payment via the other path must be invalidated. In case the receiver tries to claim both payments, the sender initiates a refund on one of the paths.

Another work [45] proposes an algorithm that returns the most economical path for a given payment by designing a transaction fee model.

Privacy-Preserving Route Discovery (2021)

The lightning network maintains a set of trampoline nodes to enable efficient and scalable discovery of routes [128]. These nodes may act selfishly and leak information about routes, frequency of transactions in a particular channel, etc. The state-of-the-art routing algorithm discussed till now might reveal critical information, violating privacy. In [105], the authors have discussed how *gossiping* and *probing* can be used to leak the residual balance of a channel and proximity of the recipient while the discovery of the route. To counter these problems, Pietrzak et al. have proposed *LightPIR* [112], a privacy-preserving discovery of the shortest path connecting source to destination. The algorithm relies on a simple hub labeling heuristic combined with private information retrieval. Previously, the algorithms were executed under the assumption that the network is fully known by at least one participant. Avarikioti et al. [29] was the first to propose an efficient route discovery algorithm considering the payment channel network topology is partially known. Instead of using expensive multi-party computation (MPC) on the entire network, the sender and receiver propagate gossip messages across the network involving a small fraction of nodes. A path is discovered when a node receives both messages.

3.2 Payment Protocol in Payment Channel Network

We discuss some construction for single path and multi-path payment that is privacy-preserving and efficient.

3.2.1 For single-path payment

"Concurrency and Privacy with Payment-Channel Networks" by Malavolta et al. (2017)

Prior to this work, *TumbleBit* [71] and *Bolt* [67] proposed off-chain path-based payments via a single hub or tumbler. However, these protocols worked for single-hop payments, and there is no mention of how these protocols can be extended for multi-hop payments. *HTLC* used in Lightning Network is susceptible to *wormhole attack*. Since the conditional payment forwarded on the path uses the same hash value, malicious parties can collude and prevent honest users from gaining fees by processing off-chain transactions. Also, *HTLC* lacks any rigorous security and privacy analysis. We discuss a smart contract construction termed as *Multi-Hop HTLC* which preserves the privacy of payment and hides the identity of payer and payee [93]. Later, we discuss two protocols that deal with concurrency and privacy of payments in PCN.

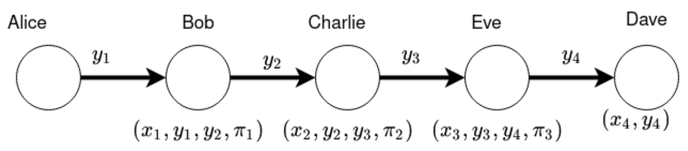


Figure 3-1: Mutli-Hop HTLC construction for payment from Alice to Dave

Multi-Hop HTLC. In the Fig. 3-1, Alice samples 4 random numbers x_1, x_2, x_3 and x_4. It constructs $y_4 = \mathcal{H}(x_4)$ where $\mathcal{H}$ is any standard one-way hash function. Next it constructs $y_3 = \mathcal{H}(x_3 \oplus x_4)$ and a zero knowledge proof π_3 for the statement *"given y_3 and y_4, there exists an $x : y_4 = \mathcal{H}(x)$ and $y_3 = \mathcal{H}(x_3 \oplus x)$"*. Similarly, it constructs $y_3 = \mathcal{H}(x_2 \oplus x_3 \oplus x_4)$ and a zero knowledge proof π_2 for the statement *"given y_2 and y_3, there exists an $x : y_3 = \mathcal{H}(x)$ and $y_2 = \mathcal{H}(x_2 \oplus x)$"*. It constructs $y_1 = \mathcal{H}(x_1 \oplus x_2 \oplus x_3 \oplus x_4)$ and a zero knowledge proof π_1 for the statement *"given y_1 and y_2, there exists an $x : y_2 = \mathcal{H}(x)$ and $y_1 = \mathcal{H}(x_1 \oplus x)$"*. It sends the value (x_1, y_1, y_2, π_1) to Bob, (x_2, y_2, y_3, π_2) to Charlie, (x_3, y_3, y_4, π_3) to Eve and (x_4, y_4) to Dave via a secure anonymous channel.

(a) Contract Creation Phase. Bob, Charlie, and Eve check whether the zero-knowledge proof received is correct or not. Alice forms the contract with Bob using condition y_1. If π_1 is correct, Bob accepts the payment, else he will abort. Bob forwards the payment to Charlie using the condition y_2. Charlie forwards the payment to Eve using the condition y_3 and Eve does the same to Dave using the condition y_4.

(b) Contract Release Phase. Upon receiving the conditional payment, Dave checks if $y_4 \overset{?}{=} \mathcal{H}(x_4)$. If this holds true, Dave send x_4 to Eve and claims payment. Eve calculates $x_3 \oplus x_4$, sends it to Charlie and claims payment. Charlie computes $x_2 \oplus x_3 \oplus x_4$, claims payment from Bob upon releasing this key. Bob computes $x_1 \oplus x_2 \oplus x_3 \oplus x_4$ and claims payment from Alice. None of the intermediate participants can correlate the payments as every channel uses a different condition.

Concurrency Issue in PCN. It might be the case that several concurrent payments might enter a deadlock due to a lack of channel capacity. Consider the example shown in Fig. 3-2. Alice tries to send a payment of 1 unit to Eve via path $Alice \rightarrow Bob \rightarrow Charlie \rightarrow Dave \rightarrow Eve$ and simultaneously, Mathew wants to send a payment of 1 unit to Charlie via path $Mathew \rightarrow Dave \rightarrow Eve \rightarrow Bob \rightarrow Charlie$. Assuming each channel has a capacity of 1 unit, the payment from Mathew to Charlie gets blocked because the link Bob-Charlie has forwarded the payment request sent by Alice. Payment from Alice to Eve gets blocked because of the link Dave-Eve has forwarded the payment request by Mathew.

The problem can be solved by either using the concept of *Blocking Payments* or *Fulgor* or the

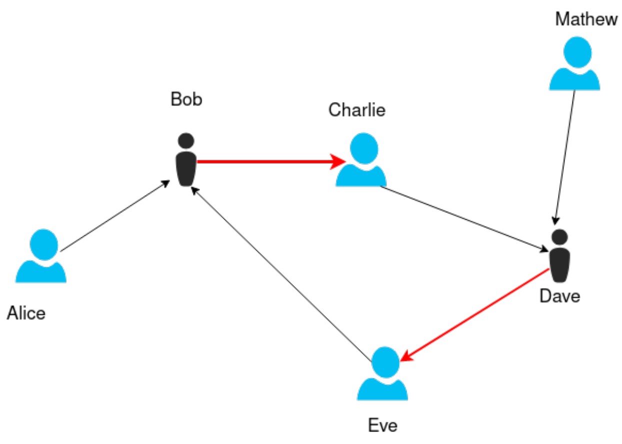


Figure 3-2: Payment from Alice to Eve and from Mathew to Charlie enter a deadlock

concept of *Non-Blocking Payments* or *Rayo*. In Fulgor, both the payments stated in Fig. 3-2 is allowed to fail. The money locked in each contract is sent back to the party forwarding conditional payment. Each sender (Mathew and Charlie in this context) waits for a random amount of time before reattempting the payment. A decrease in throughput is observed. In Rayo, at least one of the concurrent payments is allowed to complete. A global ordering of payment is maintained. A user can queue payments with identifiers higher than the payment currently being served and ignore all such payments with a low valued identifier. Rayo provides non-blocking progress of payment and higher throughput but at cost of less anonymity guarantee. On the other hand, Fulgor provides stronger anonymity.

"Anonymous Multi-Hop Locks for Blockchain Scalability and Interoperability" by Malavolta et al. (2018)

Multi-Hop HTLC [93] requires exchanging non-trivial amounts of data and computation of complex zero-knowledge proof during the setup phase. It lacks interoperability since it requires that any cryptocurrency using it must support HTLC. Anonymous Multihop Locks or AMHL [94] can be used for both script-based settings or even scriptless setting leveraging on Schnorr signature and ECDSA signature. It reduces the transaction size, and blockchain load and guarantees interoperability.

We state a generic script-based construction of AMHL in Fig. 3-3. Given a party P_0 wants to transfer an amount to party P_n via path $\mathcal{P} = \langle P_0, P_1, , P_n \rangle$ and a homomorphic one-way function $g : \mathcal{D} \rightarrow \mathcal{R}$, in the setup phase P_0 samples n values $(y_0, y_1, \ldots, y_{n-1})$. Using anonymous

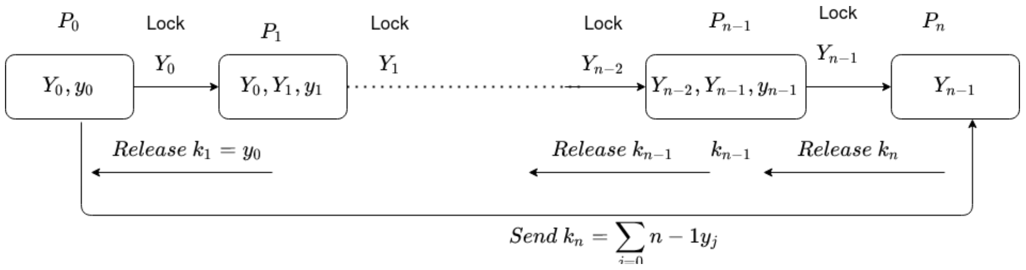


Figure 3-3: Using AMHL for secure payment from P_0 to P_n

communication channel, P_0 sends $(g(\sum_{j=0}^{i-1} y_j), g(\sum_{j=0}^{i} y_j), y_i)$ to each of the party $P_i, i \in [1, n-1]$ along with a zero knowledge proof π_i for the statement "$\exists y : g(y + y_i) = g(\sum_{j=0}^{i} y_j)$". P_0 sends a key $k_n = \sum_{j=0}^{n-1} y_j$ to P_n. Each party checks whether the received message is consistent using homomorphic properties of g. A pair of party P_i and P_{i+1} sharing a channel agrees on the common value $Y_i = g(\sum_{j=0}^{i} y_j)$ serving as the *lock*. Rest of the parties does the same. Once P_n receives the conditions of payment from P_{n-1}, it checks whether $Y_{n-1} \overset{?}{=} g(k_n)$. If this holds true, then it releases k_n to P_{n-1}. P_{n-1} upon receiving k_{n-1}, computes $k_{n-1} = k_n - y_{n-1}$ where $Y_{n-2} = g(k_{n-1}$ and sends this value to P_{n-3}. In general, this can be stated as party P_i and P_{i+1} opens the lock by using $g(k_{i+1} - y_i) = g(\sum_{j=0}^{i} y_j - y_i) = g(\sum_{j=0}^{i-1} y_j) = Y_i$.

We can use either Schnorr or ECDSA (compatible with Bitcoin) signature scheme for scriptless construction. We discuss the schnorr signature scheme.

Scriptless Schnorr Construction: Given a message $m \in \mathcal{M}$, for a given public key pk, any two-party can construct an *incomplete signature* over message m. This incomplete signature acts as the lock. Completion of the signature acts like the release of the lock. For an elliptic curve group $\mathbb{G}$ having generator $\mathcal{G}$ of prime order q, we discuss the steps in detail:

- Two parties P_i and P_{i+1} have public key $PK_i = x_0\mathcal{G}$ and $PK_{i+1} = x_1\mathcal{G}$ where $x_0 \overset{\$}{\leftarrow} \mathbb{Z}_q$ and $x_1 \overset{\$}{\leftarrow} \mathbb{Z}_q$.

- Using the common lock value $Y_i = \sum_{j=0}^{i} y_j\mathcal{G}$, they chooses random value R_0 and R_1 where $R_0 = r_0\mathcal{G}$, created by P_i and $R_1 = r_1\mathcal{G}$, created by P_{i+1}.

- Both the parties generate $R = R_0 + R_1 + Y_i$ and construct $s = r_0 + r_1 + e(x_0 + x_1)$ where $e = \mathcal{H}((PK_i + PK_{i+1})||R||m)$, $\mathcal{H}$ is a standard hash function. The pair (R, s) forms the lock. It is not a valid signature on m, since it lacks the term $y : Y_i = y\mathcal{G}$.

- Once y is revealed, P_{i+1} can construct $s^* = s + y$ and complete the signature.
- Note that the lock between P_i and P_{i+1} is conditioned on the lock between P_{i+1} and P_{i+2}. Once these parties complete their signature and release the lock only then P_{i+1} can complete the signature on P_i and P_{i+1}.

"MAPPCN: Multi-hop Anonymous and Privacy-Preserving Payment Channel Network" by Tripathy and Mohanty (2020)

In AMHL, the sender had to construct the random values for all the intermediate nodes. It was assumed that the sender transmits these values using anonymous secure communication channels to all the intermediate nodes during the setup phase. MAPPCN [131] does not require the sender to communicate with each intermediate party. The computational overhead of the sender is very less. The protocol offers a better privacy guarantee and is resistant to several attacks. We provide a high level overview of the protocol where S routes v coins to recipient R via intermediaries $u_1, u_2, \ldots, u_n$.

- A base point $\mathcal{G}$ of an elliptic curve group is chosen.
- The sender S generates a random number r and shares $(r, \mathcal{G})$ with the receiver via a secure channel.
- S sends $\langle S, u_1, v_1, \beta_1, \alpha_1, t_1 \rangle$ to its next neighbor u_1 where β_1 and α_1 are a pair of random numbers, t_1 is the lock time of the off-chain contract between S and u_1 and v_1 is the coins locked in the contract.
- If we state in general, each user u_i generates a random number l_i and computes $\beta_{i+1} = l_i \beta_i$ and $\alpha_{i+1} = l_i \alpha_i$. It sends $\langle \beta_{i+1}, \alpha_{i+1} \rangle$ to its next neighbor u_{i+1}.
- Finally, R receives $\langle \beta_{n+1}\, \alpha_{n+1} \rangle$ from neighbor u_n. It now verifies whether $\alpha_{n+1} \overset{?}{=} r\beta_{n+1}\mathcal{G}$, computes $\Gamma_{n+1} = r\beta_{n+1}$ and returns it to u_n. The latter verifies if $\Gamma_{n+1}\mathcal{G} \overset{?}{=} \alpha_{n+1}$ and releases the coins to R.
- In general, a node u_i computes $\Gamma_i \overset{?}{=} l_i^{-1}\Gamma_{i+1}$ to u_{i-1}, where $i \in [0, n]$, and claims the coins. This continues till $u_0 = S$ releases the coins to u_1.

Another work, *n-HTLC* [101] offer better privacy and faster than AMHL. The protocol uses garlic routing that is compatible with cryptocurrencies like Monero [133], Verge [126]. The authors have also proposed another symmetric key encryption-based payment protocol, kTLC. It is compatible with Lightning Network and Raiden Network [13].

In [87], the authors have proposed Generalized Multi-Hop Locks (GMHL) that are used to design a general PCN. The construction has a lightweight setup, and unlike AMHL, the computation load is borne by all the participants. GMHL is built on a Guillou-Quisquater-based adaptor signature and a proposed novel RSA-based randomizable puzzle. Thus GMHL can be applied to any blockchain.

3.2.2 For multi-path payment

“Atomic MultiPath Payment” by Osuntokun (2018)

For high-valued payments, it might not be feasible to find a single path in a network to route a payment. Instead, it is advisable to split such payments into several micropayments. Routing each micropayment can be accomplished by using HTLC where the same hash value is reused across all the paths. The downside of an approach is that the payments can be correlated, it is susceptible to wormhole attacks. Even if the payment fails in one of the paths, the receiver might still be able to claim the full payment as the same preimage can solve all the conditional payments.

Atomic MultiPath Payment or *AMP* [10] resolves these issues guaranteeing atomicity, either all payments succeed or none succeeds. Once the receiver receives all the conditional payments from different routes, it can claim the payment. The payment hash used across different routes is different, preventing any correlation. The setup is non-interactive where the payer need not coordinate with the payee.

Construction. Alice needs to send an amount v to Bob. It figures n paths $P_1, P_2, \ldots, P_n$, with each path transferring $v_1, v_2, \ldots, v_n : v = \sum\limits_{i=1}^{n} v_i$ as shown in Fig. 3-4. Alice samples secret $s_1, s_2, \ldots, s_n$. The master secret $s = s_1 \oplus s_2 \oplus \ldots \oplus s_n$ can be generated. Using s, she generates the condition of payment for each path P_i as follows: $H_i = \mathcal{H}(s||i), i \in [1, n]$. For each path P_i, the conditions of payment H_i is forwarded using onion routing, where the the tuple (s_i, i) is send as an encrypted onion blob or EOB which can only be decrypted by Bob. Upon receiving all the conditions from n paths, Bob computes s and constructs the preimage $s||i$ for each path P_i in order to claim payment. If any of the path fails, then Bob will not be able to claim payment.

The drawback of the construction is the latency involved. The time taken by the receiver to

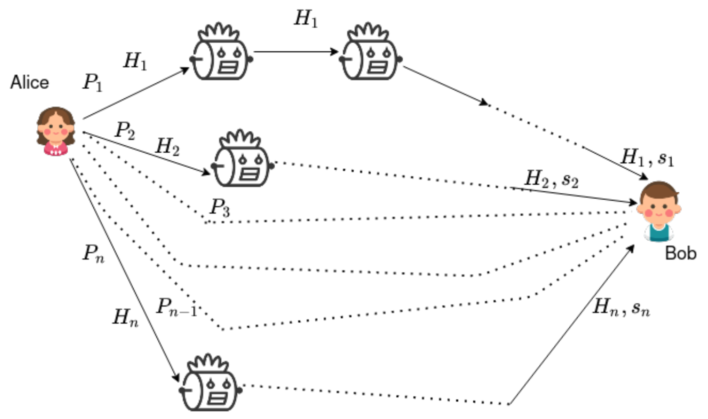


Figure 3-4: Atomic MultiPath Payment from Alice to Bob

receive all the conditional payments is slowed down by the path having the slowest propagation rate. Another multipath payment, *Boomerang* [32], addresses these issues by enabling redundancy. Alice can use more than the transaction value v for forwarding her payment via k such paths where $k > n$. Later, she can reclaim the funds that exceed v. If some of the paths fail, still the success rate of payment increases as the rest of the paths might be successful in propagating the desired value. The disadvantage of Boomerang is that excess collateral gets locked up due to the use of redundant paths that could have been utilized for routing other payments.

"Split payments in payment networks" by Piatkivskyi and Nowostawski (2018)

In *AMP* [10], either the entire payment must succeed or fails. This may result in payment remaining unresolved if any of the partial payment remains stuck and funds remain locked. In this work [111], the issue is resolved by allowing partial payments to be spread evenly across a given execution time window, called as *time taken to live or TTL*. Since each partial payment is executed through a subsequent amount of time, they might eventually succeed due to a change in the channel's residual balance over time. If the payment is fully transferred within *TTL*, then it is considered to be successful. The success ratio of payment is higher than *AMP*.

"Splitting Payments Locally While Routing Interdimensionally" by Eckey, Faust, Hostáková and Roos (2020)

In this protocol [53], instead of the sender specifying the path from sender to receiver and computing for the setup phase, each party independently decides on the split of the payment value. Initially, the receiver sends the hash of a secret preimage x_R, to the sender. Let it be denoted as

$H : H = \mathcal{H}(x_R)$. Note that the hash function used here is an additive homomorphic one-way function. The sender splits the amount based on the channel's residual capacity. It decides on the conditional payment for each split by sampling different random values. If sender splits the payment t into k parts, then it samples $x_1, x_2, \ldots, x_k$. The hash of the receiver's preimage and the hash of the sender's preimage for each split is added, $H + \mathcal{H}(x_i), i \in [1, k]$ and forwarded to the neighbor along with the address of the receiver and encrypted value of each random value, denoted as $Enc_{HE}(x_i)$. The encryption used here is homomorphic, using the public key of the receiver. If two encrypted values are added, then upon decryption we get the summation of these two values. The neighbor upon receiving the packet decides upon the next neighbor which can forward the packet to the receiver. It performs the same step as done by the sender and combines its encrypted random value with that received from the sender. When the receiver receives the conditional payments for all the splits, it decrypts the encrypted value and adds the preimage it had sampled initially. It now claims the payment by releasing this preimage. The sender can generate a valid receipt of the payment provided the sender receives the secret preimage sampled by the receiver. For routing, it uses a new algorithm, *Interdimensional SpeedyMurmur*. The main advantage of this protocol is that it combines both routing and payment, guaranteeing higher throughput.

Another work based on *Non-Atomic Payment Splitting (NAPS)* [51] allows the intermediate parties routing payment to split recursively into several partial payments. However, the protocol does not ensure atomicity and considers the partial success of payment valid. The advantage of this protocol is that it does not use computation-intensive cryptographic primitives like homomorphic encryption, unlike [53].

"Spear: fast multi-path payment with redundancy" by Rahimpour and Khabbazian (2021)

In the previous work, a payment fails or gets delayed for an indefinite period even if one of the partial payments fails or delays. This problem was mitigated to some extent in Boomerang [32], however, the protocol still suffers from higher latency and computational complexity. Spear [116] is more efficient in terms of computation cost and requires half the maximum lock time of Boomerang. Given that *Alice* needs to make a payment α to *Bob*, she follows the following steps:

i. *Bob* samples a hash digest and shares it with *Alice*.

ii. She divides α across k such paths, where the partial payment may not be equally divided. For each path, she samples a new hash digest, whose preimage is unique to that path. Note that summation of partial payment across k path is greater than α to allow redundancy.

iii. Once *Bob* receives all the partial payment, he informs *Alice*. Now *Alice* reveals the preimage to a subset of hash digest such that summation of the subset of partial payment is α.

iv. Upon getting the preimage from *Alice*, *Bob* ensures that he has enough time left to claim the payment and it cancels the rest of the redundant payments.

3.3 Griefing Attack in Lightning Network

A vulnerability on mainnet Lightning [76], the victim does not lose any money but is forced to pay for an expensive channel close. An adversary to disrupt competitors or jam a significant portion of the network will mount this attack.

3.3.1 Attacking Strategy

In this section, we discuss a few papers that define the attacking strategy for stalling the network using a griefing attack and the damage inflicted on the network.

"Discharged Payment Channels: Quantifying the Lightning Network's Resilience to Topology-Based Attacks" by Rohrer, Malliaris and Tschorsch (2019)

Rohrer et al. [121] have discussed the topology of Lightning Network. Upon analysis, the network was found to follow a small world, scale-free structure. The authors have also investigated the resilience of the network towards random failures and targeted attacks. Several attack vectors have been defined that cause exhaustion of the payment channel's capacity and isolation of nodes in the network. The impact of the damage gets amplified due to payment griefing. The adversary initiates an arbitrary number of HTLC payments to a node under its control. The former instructs the corrupted node to ignore such payment requests forcing other nodes in the path to wait for the HTLC timelocks to expire. No cost is involved in mounting the attack as the payment balance is restored after the contract timeouts. However, intermediate parties cannot use their channel capacity for the designated period, allowing the adversary to freeze the channel's liquidity.

Various attacking strategies have been discussed where the adversary either targets highly central nodes in the network or applies the highest-ranked minimum cut strategy. Nodes with high centrality mean they are important to the network. Such nodes either have a high degree or high betweenness or high eigenvector centrality. Removing these nodes might lead to the partitioning of the network or an abrupt reduction of liquidity. A minimum cut set of a graph is a set of edges with minimal cumulative capacity that partitions the graph if removed from the graph. An adversary with a limited budget chooses a minimum cut strategy but to inflict substantial damage, a powerful adversary with a substantial budget will choose to remove highly central nodes.

In [109], an adversary conducts self-payment blocking the outgoing capacity of the channel, similar to a channel exhaustion attack. When the victim runs an LND implementation, 80% of the capacity of the victim can be locked during 287 blocks (almost two days) in any balance distribution tested.

"Congestion Attacks in Payment Channel Networks" by Mizrahi and Zohar (2020)

Paralyzing the network for multiple days by overloading each channel with maximum unresolved HTLCs has been studied in [100]. The attack is mainly dependent on the limits imposed on the number of unresolved payments a channel can have. The authors have discussed three versions of the attack - blocking several high liquidity channels, disconnecting many pairs of nodes, and isolating several nodes from the network. The attacker either chooses the longest feasible route or selects the hub node. The paper discusses two proof of concept implementations for mounting the attack. In both cases, the attacker sends several self-payment requests. After sending such requests, it can either wait for the HTLC timelock to elapse or it can cancel the payment just before elapse of lock time. In the first case, the channel connected to the attacker who has stopped responding is closed and the honest party is forced to settle on-chain. In the second case, the payment is canceled leading to the failure of HTLCs along the entire route but none of the channels gets closed. The attacker can continue locking channel liquidity without opening fresh channels with the victim nodes. It reduces the success rate of payment and increases the average attempts made for routing payments.

"Bank run Payment Channel Networks" by Lu, Han, and Yu (2020)

Bankrun attacks exploit payment griefing to bank-run PCNs. In this attack, the attacker generates Sybil nodes, establishes channels with hubs in the network, makes payments between his nodes, and starts griefing simultaneously [89]. If the adversary has sufficient coins, he can lock a high percentage of coins in the PCN, so that the PCN may no longer handle normal payments. This leads to repeated attempts for payment. Payers may resort to longer paths increasing the fee per transaction. The evaluation result shows that using channels with 1.5% high capacity nodes, the attacker can lock 83% of the capacity in the entire network. If the attacker forms connections to these nodes, then by investing just 77 BTC, the former can lock up to 45% (around 267 BTC) of coins in the entire network for an entire day, or longer as per the HTLC timeout period.

In [90], a *general congestion attack* was introduced, which generalizes the existing congestion attack in terms of attacking strategies and targeted metrics. The strategies defined here for mounting the attack focus on the network's liquidity.

"Route Hijacking and DoS in Off-Chain Networks" by Tochner, Zohar and Schmid (2020)

In this paper [130], an adversary establishes a set of edges in a topologically important location i.e., increasing its centrality. It announces a very low fee thereby attracting parties to select the adversary as an intermediate node for routing. Once it receives the conditional payment, it drops the payload and stops responding. The payment does not take place and the sender waits for the HTLC timeout period before attempting to resend the payment. If the attacking node has very high centrality, it will be able to hijack a large number of paths. The attack may be further amplified if the attacker sets a higher timeout value for the HTLCs. The coins locked in the contract cannot be utilized and can be unlocked once the period elapses. With the help of the attack, residual channel balance is disclosed and the network's liquidity reduces.

In another work, [139], a denial-of-service attack on SpeedyMurmurs is performed in two ways. The first method is the same as discussed before where an attacker attracts payments to be routed through itself and then drops them. The second method performs griefing by delaying the payment.

3.3.2 Countermeasure for Griefing Attack

Several ideas have been proposed for countering the griefing attack. A limit on the number of incoming channels and the channel capacity was proposed in [121] as a countermeasure for node isolation attacks. However, the attacker may split the funds over multiple identities and channels to bypass the restrictions imposed. Faster resolution of *HTLC* [100] was proposed as another method to avoid the disadvantage of having staggered lock time across payment channels. However, such a feature would violate the purpose of having *HTLC* timeout which acts as a safety net against other possibly malicious activities. All these payment protocols had a staggered lock time over each channel responsible for routing the payment. The collateral cost incurred for staggered lock time protocols is substantial. We discuss several countermeasures for griefing attacks based on (i) construction of constant lock time payment protocols (ii) virtual payment channel construction and lastly, (iii) penalization mechanism.

3.3.2.1 Constant Locktime Payment Protocols

"Atomic Multi-Channel Updates with Constant Collateral in Bitcoin-Compatible Payment-Channel Networks" by Egger, Moreno-Sanchez and Maffei (2019)

Payment protocols are susceptible to *Griefing Attack* where a receiver can stop responding. If the path routing payment from payer to payee is of length n and the time taken to settle a transaction on-chain is Δ, then in the worst case, if all parties go on-chain and settle the payment, the collateral across the path is locked for $\mathcal{O}(n\Delta)$. To reduce the worst-case collateral locked, a variant of payment channel construction supporting Ethereum-styled smart contracts, Sprites [99] was proposed. Sprites support partial withdrawals and deposits, and the channel can continue to operate without interruption while such operations are being performed.

In *Sprites*, the authors had made a conjecture that the locking of constant collateral cannot be achieved in Bitcoin scripts. The conjecture was refuted in the work by Egger et al. [55]. In this protocol, a solution has been proposed that uses MIMO transactions. A MIMO transaction supports multiple inputs and multiple outputs in a transaction. It enables synchronizing the off-chain updates of multiple payment channels while ensuring that each payment channel can function independently after the protocol is executed. The drawback of the protocol is that it violates relationship anonymity and relies on a coordinator to control and keep track of the initiation and completion of each phase. The identity of all the parties must be known to ensure whether a payment fails or succeeds.

"Blitz: Secure Multi-Hop Payments Without Two-Phase Commits" by Aumayr, Moreno-Sanchez, Kate and Maffei (2021)

We discussed previously that [55] violates relationship anonymity by revealing the identity of each participant to one another. Thus, another single round multi-hop payment protocol *Blitz* [27] was proposed that ensures the privacy of participants, resilient to wormhole attack. The protocol locks constant collateral like the atomic multi-channel update. This ensures that the PCNs are far more robust against griefing attacks and have a higher transaction throughput.

Blitz builds on standard PCN and makes use of timelocks and signatures as the cryptographic primitives. It can be widely deployed across cryptocurrencies that do not support hash locks or scripts like Monero, Stellar, etc. To allow fast tracking of payments, the authors have extended the

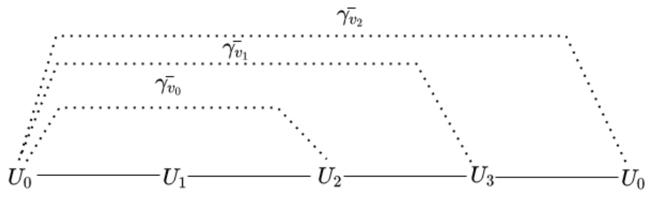


Figure 3-5: Construction of self virtual channel for mounting griefing attack

protocol to Blitz-FT with an optional second round of communication. Fast tracking of payments allows the honest users on the path to update their channel when the coins have been transferred from the payer to the payee.

The problem with *Blitz* is that it supports only path-based payments. Another work by Aumayr et al. [25] presents an off-chain protocol that enables atomic multi-channel updates across generic topologies and is not just restricted to path-based payments. It also achieves value privacy where channel owners need not disclose the payment amount and they can still update their channel.

3.3.2.2 Virtual Payment Channel Constructions for Bitcoin Blockchain

Dziembowski et al. [50, 52] had proposed *virtual state channels* that is used for executing applications that are more complex than just payments. Such state channels eliminate the need to interact with any intermediary nodes unlike in HTLC where intermediate parties are paid some processing fee for forwarding the payment. Two parties not sharing a state channel can perform a transaction as if they are directly connected. The creation of virtual state channels requires blockchain with smart-contract capability, thus they cannot be extended to Bitcoin-based PCNs.

Jourenko et al. propose a UTXO based lightweight virtual payment channel construction for blockchains such as Bitcoin [79]. The cryptographic primitive used is the same as *Blitz*. The construction consists of three phases: `Open_VC, Close_VC` and `Enforce_VC`. `Open_VC` and `Close_VC` is used for setup and tear down of virtual channels. `Enforce_VC` is used to resolve any dispute that arises while transactions are executed off-chain. To construct virtual payment channels across multiple hops, the protocol is applied iteratively. Another work, Elmo [83], mentions a recursive virtual channel construction, offering optimal round complexity even in the pessimistic case. However, such construction suffers from *virtual griefing attack*. In this attack, a payer opens a virtual channel to itself as shown in Fig. 3-5. The channel is kept open before the timelock of payment expires. Funds remain locked at each virtual channel in the underlying recursion layer. The problem with this construction is that only a fixed number of transactions can be executed within the virtual channel's lifetime. A generalized construction of Bitcoin compatible virtual channel [26] eliminates the problem, and it can be used to run any application off-chain, supported by Bitcoin script.

The construction of a lightweight virtual payment channel is re-used for a low-collateral payment tree protocol [80]. The damage due to the griefing attack is less in this case, however, the protocol requires all the parties to remain online and interact with each other, leaking sensitive information. Recently, Aumayr et al. proposed a one-round virtual channel construction *Donner* [28] that addresses the problems discussed above. The payment channel construction is efficient, privacy-preserving, and resistant to virtual griefing attacks.

3.3.2.3 Penalization Mechanism

State-of-the-art discusses alternate mitigation strategies by incentivizing or punishing nodes. The use of up-front payment was first proposed in [16]. A party has to pay a fee to the other party for accepting the *HTLC* in up-front payment. An excess fee paid is returned to the sender upon successful payment. This introduces a lot of economic barriers where up-front payment may exceed the transaction fee. For small valued payments, a large up-front payment adds a burden on the sender. In [11], the concept of reverse-bond was proposed that is similar to our proposed strategy. The counterparty accepting the *HTLC* will have to pay a hold fee on a per unit interval basis as if it has rented the *HTLC*. However, it has not been stated formally how this can be realized. There is no way to track per unit interval in a decentralized asynchronous setup. Up-front payments have also been used to disincentivize an attacker from griefing in atomic swaps [72]. In [8], Proof-of-Closure of channels was proposed, where each *HTLC* will have a hard timeout and a soft timeout period. However, a malicious node can set up several Sybil nodes just for this purpose so that channel closure doesn't affect its normal activity in the network.

Chapter 4

HushRelay: A Scalable Routing Algorithm for Off-Chain Payments

Payment channel networks (PCN) are used in cryptocurrencies to enhance the performance and scalability of off-chain transactions. Except for the opening and closing of a payment channel, no other transaction requests accepted by a PCN get recorded in the Blockchain. Only the parties which have opened the channel will know the exact amount of funds left at a given instant.

For routing payment to a particular node in the network, a party need not open a direct channel. It can route the payment along a path in the network via several intermediaries. However, it might be difficult to find a single path with sufficient capacity for transferring a high-valued payment. In such cases, it is better to split the amount and send partial payments via multiple paths. While there exist several approaches for multi-path routing of transactions, they are either quite inefficient or leaks sensitive information [92]. For example, in SpeedyMurmur [122], the decision taken on the number of splits at the initial phase of the routing algorithm might not be accurate. We observed that finding feasible routes even for a single payment in PCN is a challenging task. Such factors motivated us to design a new routing algorithm for PCN that is privacy-preserving, efficient, and scalable.

4.1 Our Contributions

We have made the following contributions in this chapter :

- We have proposed a privacy-preserving distributed routing algorithm, HushRelay, in the payment channel network.
- We have implemented the scheme and compared the performance of HushRelay with Speedy-Murmur [122] in terms of *success ratio* and *time taken to route (TTR)* a payment. We used a snapshot of Ripple Network and Lightning Network and our results show that HushRelay attains a success ratio of 1 in both cases having an execution time of 2.4s and 0.15s. However, Speedy-Murmur attains a success ratio of 0.98 and 0.91 provided the number of landmarks is 6. The

execution time is higher, around 4.7s and 1.9s for the snapshot of Ripple Network and Lightning Network, respectively. Our observation shows that HushRelay is efficient and scalable. The code is provided in [6].

- The proposed routing algorithm is modular and can be combined with any other privacy-preserving payment protocol.

4.1.1 Organization

Section 4.2 provides the background needed for understanding the chapter. Section 4.3 defines the problem statement and section 4.4 discusses the protocol HushRelay with section 4.4.1 dealing with generic construction and section 4.4.2 providing the proof of correctness. We provide the privacy analysis in section 4.5. We have compared the performance of HushRelay with the state-of-the-art SpeedyMurmur in section 4.6.

4.2 Background

In this section, we provide the background for understanding this chapter. The terms source/payer means the sender node. Similarly, sink/payee/destination means the receiver node, and transaction means payment transfer.

4.2.1 Payment Channel Network

Definition 4.1. *A Payment Channel Network (PCN) [93] is represented as a bidirected graph $G := (V, E)$, where V is the set of nodes and $E \subset V \times V$ is the set of edges connecting the nodes in V, also termed as payment channels. Only the opening and closing of payment channels get recorded on blockchain. Disputes arising in an off-chain transaction are settled by broadcasting the transaction on blockchain.*

Basic operations of PCN [93]-

- `openPaymentChannel`(v_1, v_2, α, t, m) : For a given pair of accounts $v_1, v_2 \in V$, channel capacity α (initial balance escrowed), timeout value of t and processing fee charged m, `openPaymentChannel` creates a new payment channel $(id_{(v_1,v_2)}, \alpha, t, m) \in E$, where $id_{(v_1,v_2)}$

is the channel identifier, provided both v_1 and v_2 has authorized to do so and the funds contributed by each of them sum up to value α.

- `closePaymentChannel`$(id_{(v_1,v_2)}, \tilde{\alpha})$: Given a channel identifier $id_{(v_1,v_2)}$ with balance $\tilde{\alpha}$, `closePaymentChannel` removes the channel from G provided it is authorized to do so by both $v_1, v_2 \in V$. The balance $\tilde{\alpha}$ gets written on blockchain and this amount is distributed between v_1 and v_2 as per the net balance recorded.

- `payVal`$(p(s,r), val)$: $p(s,r)$ denotes a path between sender s and receiver r. It is defined by a set of identifiers $id_{(s,v_1)}, id_{(v_1,v_2)}, \ldots, id_{(v_n,r)}$, $s, v_1, v_2, \ldots, v_n, r \in V$, having enough credit to allow transfer of val from s to r, if for each payment channel denoted by $id_{(v_i,v_{i+1})}$ has capacity of at least $\beta \geq val'_i, val'_i = val'_{i+1} + f(val'_{i+1}), 0 \leq i \leq n-1, val'_{n-1} = val, v_0 = s \, and \, v_{n+1} = r$, where $f(val'_i)$ is the processing fee charged by each intermediate node v_i for forwarding val'_i coins in path $p(s,r)$. A successful `payVal` operation leads to a decrease of capacity of each payment channel $id_{(v_i,v_{i+1})}$ by val'_i. Else the capacity of the channel remains unaltered.

4.2.2 Payment Flow problem

Consider a directed graph $G := (V, E) : \; n = |V|, m = |E|, m \geq n-1$, having two distinguished vertices, source $s \in V$, sink $r \in V, s \neq r$, as a *flow network*. For a pair of vertices v, w, distance from v to w in graph G is defined by $d_G(v,w)$, the minimum number of edges on the path from v to w; if there is no path from v to w, $d_G(v,w) = \infty$. A positive real-valued capacity $c(v,w)$, defined by $c : E \rightarrow \mathbb{R}$, is the maximum amount of funds that can be transferred across a payment channel. For every edge $(v,w) \in E$; if $(v,w) \notin E$, then $c(v,w) = 0$. A flow fl on G is a real-valued function on vertex pairs satisfying the constraints [63], [110] :

$$\begin{aligned} fl(v,w) \leq c(v,w),\ \forall (v,w) \in V \times V \text{ (capacity)}, \\ fl(v,w) = -fl(w,v),\ \forall (v,w) \in V \times V \text{ (antisymmetry)}, \\ \Sigma_{u \in V} fl(u,v) = 0 \; \forall v \in V - \{s,r\} \text{ (flow-conservation)}, \end{aligned} \tag{4.1}$$

The net flow into the sink is given by f, where:

$$f = \Sigma_{v \in V} fl(v,r) \tag{4.2}$$

A payment channel network is mapped to a flow network with channels forming the edges and funds locked on each channel becoming the edge capacity. Finding the maximum flow value from source to sink for a flow network is termed as the *Maximum Flow problem.* In the context of PCN, given a payment value val, one has to find a feasible flow from payer to payee, which is termed here as *Payment Flow problem*. Any max-flow algorithm with subtle modifications can be applied here, taking into account the pre-flow fl of each vertex (except the source and sink) on the network. A pre-flow is a real-valued function on a vertex pair that satisfies the first two constraints of Eq. 4.1 and a weaker form of the third constraint :

$$\Sigma_{u\in V} fl(u,v) \geq 0,\ \forall v \in V - \{s,r\} \text{ (non-negativity constraint),} \tag{4.3}$$

A residual capacity of an edge $(v,w) \in E$ is the amount of capacity remaining after the preflow fl, i.e. $c(v,w) - fl(v,w)$ and it is denoted by $r_{fl}(v,w)$. A residual graph $G_{fl} = (V, E_{fl})$ for a preflow fl is the graph whose vertex set is V and edge set E_{fl} is the set of residual edges $(v,w) \in E\ :\ r_{fl}(v,w) > 0$. The *flow excess e(v)* of a vertex v is the net balance of funds in node v denoted by $\Sigma_{u\in V} fl(u,v)$. The algorithm ends with all vertices except s and r having zero excess flow. If the sink is unreachable or if the network does not have adequate capacity for transferring the payment val, then the excess value is pushed back to source s.

4.3 Problem Statement

It is not always possible to route the transaction across a single path as the value may be quite high compared to minimum capacity of the designated path. Hence it is better to find set of paths such that the total amount to be transferred is split across each such path. We define the problem as follows -

Problem 1. *Given a payment channel network $G(V,E)$, a transaction request (s,r,val) for a source-sink pair (s,r), the objective is to find a set of paths $p_1, p_2, \ldots, p_m$ for transferring the fund from s to r such that p_1 transfers val_1, p_2 transfers $val_2, \ldots, p_m$ transfers $val_m : val = \Sigma_{i=1}^{m} val_i$ without violating transaction level privacy i.e. neither the sender nor the receiver of a particular transaction must be identified as well as hiding the actual transaction value from intermediate parties.*

4.4 Our Proposed Construction

In this section, we provide a detailed overview of the routing algorithm, HushRelay. The payment network comprises set of payment channels denoted by channel identifier $id_{(i,j)}, (i,j) \in E$. We describe the model and state the assumptions.

Network Model and its Assumptions

- The network is static i.e. no new channels get added or an existing channel is removed during the execution.
- The topology of the network is known by all the nodes in the network since any opening or closing of a channel gets recorded on the blockchain.
- The residual balance of each payment channel is not known publicly.
- Users sharing a payment channel use secure and authenticated channels (such as TLS) for communication.

4.4.1 Generic Algorithm

Since we consider PCN as a flow network, for solving the *payment flow problem* in the given network for executing a transaction request (s, r, val), we propose a routing algorithm inspired from distributed push relabel algorithm stated in [63], [110]. The algorithm proceeds locally by the exchange of messages between neighboring nodes. No single entity controls the flow in the network.

Before discussing the algorithm, we briefly describe the *Push Relabel* algorithm for a single source-sink pair (as stated in [63]):

- The instruction *push* redirects the excess flow of a vertex to the sink via its neighbouring vertices. The amount of excess flow that can be *pushed* from a vertex v to one of its neighbouring vertex w is $\delta = min(e(v), r_{fl(v,w)})$, where $r_{fl(v,w)}$ is the residual capacity of edge (v, w). The value δ is added to the preflow value $fl(v, w)$ (subtracted from $fl(w, v)$) and subtracted from $e(v)$. Any push which results in zero residual capacity of the edge is said to be *saturating*.
- A valid labeling function $d : V \rightarrow I^+ \cup \{0, \infty\}$ is used for estimating the distance of a vertex v from sink r. $d(s) = n, d(r) = 0$ and $d(v) \leq d(w) + 1$ for every residual edge (v, w). The label $d(v) < n$ forms the lower bound on the actual distance from v to r in the residual graph G_{fl} and if $d(v) \geq n$, then $d(v) - n$ is a lower bound on the actual distance from r in the residual graph.

- A relabeling operation is triggered when a vertex with the excess flow has a label value less than or equal to that of the neighboring vertex. Once relabeling is done, it can initiate a push operation. So one can think of labels to denote the potential level, where the flow can occur from a region of higher potential to a region of lower potential.

- A vertex v is defined as active $v \in V \setminus \{s, r\}, d(v) < \infty$, and $e(v) > 0$. The maximum-flow algorithm is initialized with preflow value f, which is summation of the edge capacities of all edges incident from the source vertex s and rest all other edges have zero flow.

For our distributed algorithm, HushRelay, the basic operations are *Push* and *Relabel*, with all the nodes acting as individual processing units in parallel. The network model considered for the payment channel network is asynchronous. Synchronization across all the nodes is achieved via use of *acknowledgements* [63]. A vertex v tries to push excess flow to one of its neighbouring vertex w if and only if, as per the information maintained by v, label $d(v) = d(w)+1$. It first sends a request message with the information $(v, \delta, d(v), e(v))$. Vertex w can either accept the push by sending an acknowledgement or it may reject it by sending a negative acknowledgement (NAK). If $d(v) = d(w)+1$, then w sends to v a message of the form $(accept, w, \delta, d(w))$ and v initiates the push. Otherwise, if $d(w) \geq d(v)$ or $d(v) < d(w) + 1$, then it sends a message $(reject, w, \delta, d(w))$ where $d(w)$ is the updated distance label of w. A reject message will cause v to update the value of $d(w)$. When a distance label of the vertex increases, it sends the information of new label to all its neighbouring nodes.

As seen in *Push Relabel* algorithm [63], the label initially set for source and sink node reveals the identity of payer and payee. To obfuscate their identity from other intermediate nodes in the network, we use a dummy source vertex s' for node s and a dummy sink vertex r' for node r. Note that the existence of dummy node is known only by the source and sink. In the initialization phase of HushRelay, a directed virtual edge from s' to s and from r to r' is established. Since s', r' are virtual entities, introduction of edge (s', s) and (r, r') is not recorded in the blockchain. The capacity is initialized to $c(s', s) = val, c(r, r') = val$ and the label is set as $d(s') = n + 2, d(s) = 0, d(r) = 0$ and $d(r') = 0$. The flow $fl(s', s)$ is set to val, $fl(r, r') = 0$, excess flow $e(s) = val$, $e(r) = e(r') = 0$. For all vertices $v \in V - \{s, r\}$, $d(v) = 0, e(v) = 0$, $fl(w, v) = 0, (w, v) \in E, w, v \in V - \{s, r\}$. We mention the procedure of *Push*, *Push-request* and *Relabel* for a vertex in Procedure 1, 2 and 3 respectively. The algorithm terminates when there are no active vertex left (except the dummy source and dummy sink) in the graph.

Example 2. *Consider a network given in Fig. 4–5. Sender S intends to make a payment of 15 units to receiver R. Dummy vertices S' and R' is added to the network with edges (S', S) and (R, R').*

Procedure 1: Push(v,w,d)

```
Input : Active vertex $v \in V, e(v) > 0$, vertices $w : (v, w) \in E$
if $v \neq r'$ and $v \neq s'$ then
    Set $find\_neighbour$=0
    while neighbour $w$ of $v$ : $e(v) > 0, r_{fl}(v, w) > 0$ and $d(w) < d(v)$ do
        $v$ generates a push of the value $\delta = min(e(v), r_{fl}(v, w)$
        $fl(v, w) = fl(v, w) + \delta$
        $e(v) = e(v) - \delta$
        $r_{fl}(v, w) = c(v, w) - fl(v, w)$
        Send Push-request(w,v,$fl(v, w), \delta$) to node $w$
        if NAK received then
            Update information $d(w)$
            $fl(v, w) = fl(v, w) - \delta$
            $e(v) = e(v) + \delta$
            $r_{fl}(v, w) = c(v, w) + fl(v, w)$
        end
        else
            Set $find\_neighbour = 1$
        end
    end
    if $find\_neighbour = 0$ then
        Call Relabel function.
    end
end
```

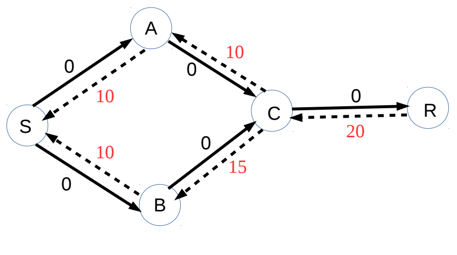
Procedure 2: Push-request(v,w,$fl(w, v)$,δ)

```
Input : Active vertex $w \in V, e(w) > 0$, vertex $v : (w, v) \in E$
if $d(v) < d(w)$ then
    $r_{fl}(v, w) = fl(w, v)$
    $e(v) = e(v) + \delta$
    if $v \neq t'$ and $v \neq s'$ then
        send Push request to its neighbouring nodes.
    end
    Send push request accepted message to node $w$.
end
else
    Send negative acknowledgement (NAK) message and current value of $d(v)$ to node $w$.
end
```

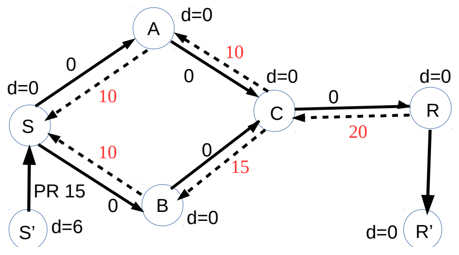
Procedure 3: Relabel(v,w,d)

```
Input : Active vertex $v \in V, e(v) > 0$, vertices $w : (v, w) \in E, r_{fl}(v, w) > 0, d(v) \leq d(w)$
    1. Update $d(v) = min(d(w), (v, w) \in E) + 1$
    2. Inform all the neighbours of vertex $v$ about the updated label $d(v)$.
```

The edge capacities are as follows : $c(S', S) = 15, c(S, A) = 10, c(S, B) = 10, c(A, C) = 10, c(B, C) = 15, c(C, R) = 20$ and $c(R, R') = 15$. HushRelay is implemented on the network to obtain a feasible flow of value 15 from source node S to sink node R. The initial state is given in Fig. 4–5 (a). In the initialization phase, each nodes is assigned a label of 0 except dummy vertex S' where d(S') is the count of the number of nodes (except S') in the network. As given in Fig. 4–5 (b), S' sends a push request of 15 units to S.

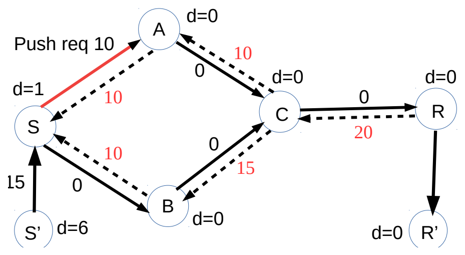


(a) Initial state

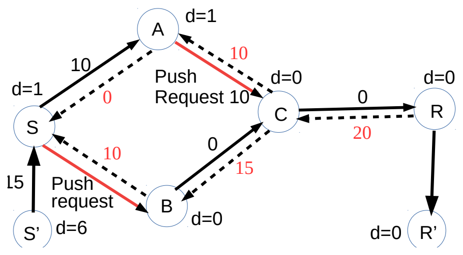


(b) After adding dummy vertices

In Fig. 4–5 (c), S accepts the push request as $d(S) < d(S')$ and an excess flow of 15 units is assigned to S. S changes its label calling relabel *function and changes $d = 1$. Now S sends a push request of 10 units to A, bounded by the capacity of payment channel SA. In Fig. 4–5 (d), A accepts the push request as $d(A) < d(S)$ and gets an excess flow of 5 units. $d(A)$ is changed to 1. S still has an excess flow of 5. It sends a push request to B. Simultaneously, A sends a push request of 10 units to C.*

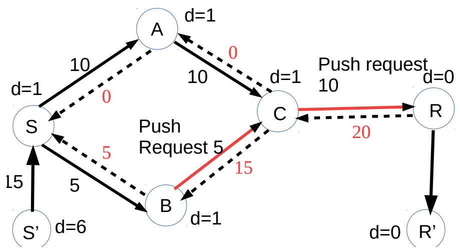


(c) Push request to A

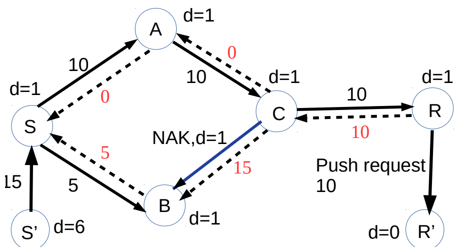


(d) Push request to C

In Fig. 4–5 (e), B accepts the push request as $d(B) < d(S)$ *and has excess flow of 5 units.* $d(B)$ *is changed to 1. Similarly, C accepts push requests as* $d(C) < d(A)$ *and has an excess flow of 10 units.* $d(C)$ *is changed to 1. B sends a push request to C, and C sends a push request to R. In Fig. 4–5 (f), upon receipt of a request from B, C finds that* $d(C) = d(B)$ *and hence it sends a NAK to B. R accepts the push request from C and gets an excess flow of 10. It changes its label to* $d(R) = 1$*, and consequently, it sends a push request of 10 units to R'. In Fig. 4–5 (g), B changes*

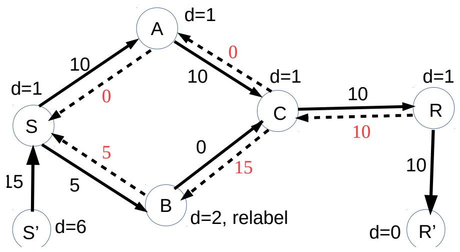


(e) Push request to C and R

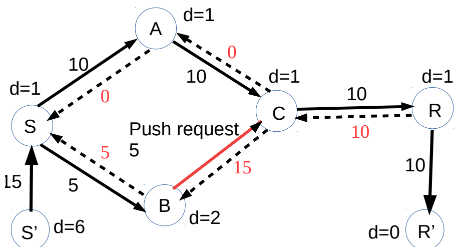


(f) NAK to B

its label to $d(B) = 2$*. R' accepts the push request. In Fig. 4–5 (h), B sends a push request of 5 units to C.*

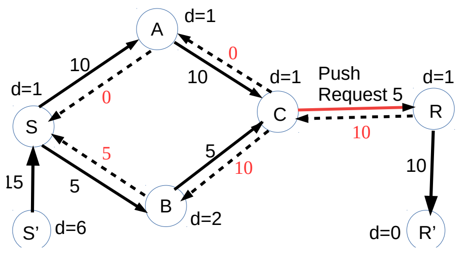


(g) Relabeling of B

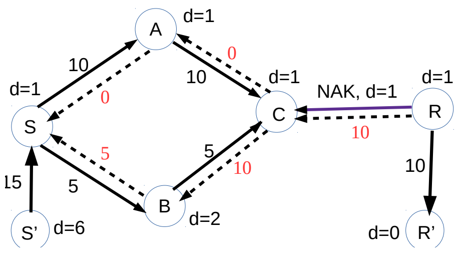


(h) Push request to C after relabeling

In Fig. 4–5 (i), C accepts the push request and changes it label to 1. It sends a push request of 5 units to R. In Fig. 4–5 (j), R finds that $d(R) = d(C)$ *and hence it sends a NAK to C.*

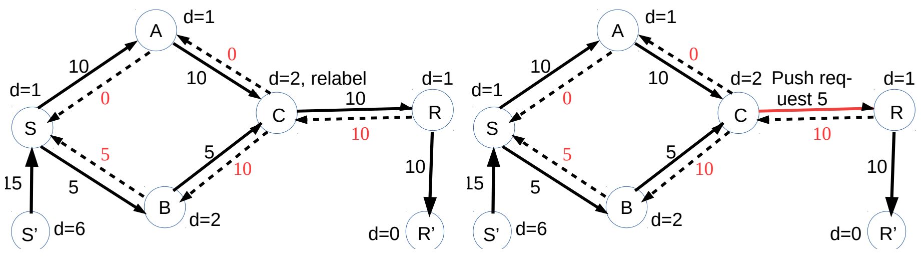


(i) Push request to R after relabeling



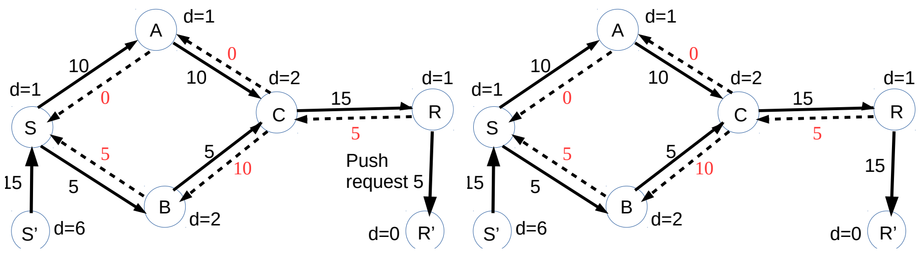
(j) NAK to C

In Fig. 4–5 (k), C undergoes a relabeling operation and $d(C)$ is changed to 2. In Fig. 4–5 (l), C again sends a push request of 5 units to R. In Fig. 4–5 (m), R accepts the push request and



(k) Relabeling of C (l) Push Request to R

it sends a push request of 5 units to R'. In Fig. 4–5 (n), the algorithm terminates transferring 15 units to R'



(m) Push request to R' (n) Final State

Figure 4–5: Execution of HushRelay

4.4.1.1 Propagating the flow information to source node

We describe the procedure by which the sink propagates the routing information back to the source.

- Participants of an edge $e \in E$ involved in the transfer of payment from source to sink generates a temporary key k_e. The key is used for encrypting the flow message to be propagated back to the source node.
- The sink node r generates a key k_r as well some random message $\mathcal{Y}$, equivalent to the size of the packet or its multiple. Each such packet contains flow information that is shared with a predecessor node.
- Sink r constructs a message m' containing the information of identity of preceding vertex w, the non negative flow $fl(w, r) > 0$ along with key k_{wr}. It encrypts the packet with k_r,

$E' = Enc_{k_r}(w, fl(w,r), k_{wr})$, and concatenates the randomly generated message rm with the encrypted packet to construct message $E'||\mathcal{Y}$.

- Sink shares this information with w. If w is honest, it will construct a similar message m', for its neighbour say u containing the identity of u, flow $fl(u,w)$, key k_{uw}. It is encrypted with k_{wr} to get E''. The encrypted message is concatenated with the message received from its successor i.e. $E''||E'||\mathcal{Y}$.

- The process is repeated till all the packets reach the source vertex s. The sink vertex shares k_r and a set of randomly generated message $\mathcal{Y}$ with source vertex s via a secure communication channel.

- s discards $\mathcal{Y}$ from the received message and starts decrypting, beginning with the message encrypted by the sink. Upon decryption, the source retrieves the flow information, identity of vertex, and key with which it will decrypt the next encrypted packet. All duplicate information on flow is discarded, and the rest is used for reconstructing the flow across the network. The routing information is denoted by $\mathcal{P}$.

4.4.2 Proof of correctness of the HushRelay

We state the following lemmas which justify the correctness of our routing algorithm.

Lemma 4.1. *If $val \leq$ maximum flow in G, then s can successfully transfer funds to r.*

Proof of Lemma 4.1. Given that $\tilde{d} <$ maximum flow, let us assume that transaction from s to r fails due to non existence of sufficient capacity from sender to receiver. Now we execute the distributed push relabel algorithm which will return the maximum flow in the graph. Let us denote it by f_m. But our algorithm was able to find augmenting path for flow till $\tilde{d} - \gamma$, where $\gamma > 0$ is an integral value, since our transaction failed. If there exists no more augmenting paths, then $f_m = \tilde{d} - \gamma$, which implies $f_m < \tilde{d}$. This contradicts the fact that $f_m > \tilde{d}$. Hence our assumption was wrong.

Lemma 4.2. *For $v \in V \setminus \{s', r'\}, e(v) = 0$ on termination.*

Proof of Lemma 4.2. Assume that there exist one vertex $\hat{v} \in V \setminus \{s', r'\} : e(\hat{v}) > 0$ after termination. But since termination condition has been reached, it means that vertex s is not reachable from vertex $\hat{v}$. Let the set of vertices not reachable from $\hat{v}$ be V' and those reachable from $\hat{v}$ be $V - V'$. Thus $s \in V'$.

$$\begin{aligned} e(\hat{v}) &= \Sigma_{k \in V, (\hat{v},k) \in E} fl(k, \hat{v}) \\ &= \Sigma_{k \in V', (\hat{v},k) \in E} fl(k, \hat{v}) + \Sigma_{k \in V-V', (\hat{v},k) \in E} fl(k, \hat{v}) \\ &= \Sigma_{k \in V', (\hat{v},k) \in E} fl(k, \hat{v}) \\ &(\because \Sigma_{k \in V-V', (\hat{v},k) \in E} fl(k, \hat{v}) = 0, \text{flow conservation constraint}) \end{aligned} \tag{4.4}$$

Since $e(\hat{v}) > 0$, $\Sigma_{k \in V', (\hat{v},k) \in E} fl(k, \hat{v}) > 0$. This implies that there exists some augmenting path from s to $\hat{v}$. The result contradicts our assumption.

Lemma 4.3. *For all edges* $(v, w) \in E, v, w \in V$*,* $fl(v, w) \leq c(v, w)$*.*

Proof of Lemma 4.3. In the algorithm *Push*, for a given edge $(v, w) \in E$, the flow value $fl(v, w)$ from vertex v to vertex w is decided by $\min(e(v), r_{fl}(v, w))$. Since $r_{fl}(v, w) = c(v, w) - fl(v, w)$, $fl(v, w) \leq c(v, w)$ and $e(v) \leq \tilde{d}$. Flow is either bounded by $\tilde{d}$, if $\tilde{d} < c(v, w)$ or $c(v, w)$.

Table 4.1: SpeedyMurmur vs HushRelay - Performance Analysis on Real Instances

Network/Algorithm	SpeedyMurmur								HushRelay	
	Success Ratio				Time taken (in s)				Success Ratio	Time taken
	Number of Landmarks				Number of Landmarks					
	1	2	4	6	1	2	4	6		
Ripple Network	0.38	0.7	0.92	0.98	1.66	2.2	3.23	4.74	1	2.4
Lightning Network	0.42	0.65	0.83	0.91	0.61	0.69	0.83	1.94	0.99	0.15

4.5 Privacy Analysis

Theorem 4.1. *Given a transaction request* (s, r, val) *for a source-sink pair* $(s, r) \in V \times V, s \neq r$ *to be routed through the network G, any node* $v \in V \setminus \{s, r\}$ *must not know the identity of payer and payee.*

Proof Sketch. In push relabel algorithm, labels assigned are $d(s) = n$, where $n = |V|$, $d(r) = 0$ and $d(v) \leq d(w) + 1$ for every residual edge $(v, w) \in E_{fl}$. This labeling leaks the identity of the source and sink. Thus a pair of dummy vertices s' and r' is introduced where s' is connected to s and r is connected to s. The label assigned to s' is same as the label assigned to the source vertex involved in executing the push relabel algorithm, thus s' acts as dummy source. r' acts as the dummy sink and r has a non-zero label after the execution of an instance of HushRelay. Thus the label assigned to s and r gives the impression that they are intermediate nodes routing the payment without revealing their actual role in the network.

Even while propagating the information $\mathcal{P}$ to the source s, since the random information $\mathcal{Y}$ is multiple of single message size, even if some of the users are corrupt they will get information of the subpath rather than the full path.

Theorem 4.2. *PCN achieves value privacy if corrupted users outside the payment path must not have any information regarding the payment value in a pay operation involving only honest users.*

Proof Sketch. In HushRelay, the algorithm is initiated by the source s when it gets a payment request from sink or payee r. Therefore, an adversary will not get any information about the payment value, if it is neither controlling the payer nor the payee. Any communication between honest intermediaries routing the payment happens via a secure communication channel. Value privacy is preserved.

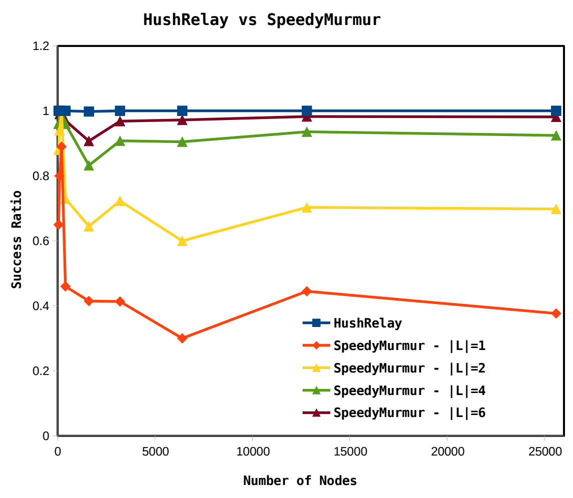
4.6 Performance Analysis

4.6.1 Experimental Setup

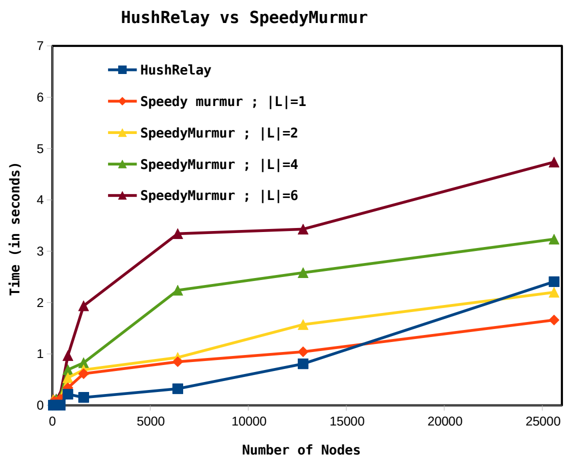
In this section, we define the experimental setup. The code for HushRelay is available in [6]. System configuration used is : `Intel Core i5-8250U CPU, Kabylake GT2 octa core processor, frequency 1.60 GHz`, OS : *Ubuntu-18.04.1 LTS* (64 bit). The programming language used is C, compiler - gcc version 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 [22], [34]. PCN follows the scale free network where certain nodes function as hubs, having higher degree compared to other nodes [75]. For implementing the cryptographic primitives, we use the library `Libgcrypt`, version-1.8.4 [7], which is based on code from GnuPG.

4.6.2 Evaluation

We use the following metrics to compare the performance of the routing algorithm, HushRelay with SpeedyMurmur [122]



(a) Success Ratio vs Number of Nodes



(b) Time To Route vs Number of Nodes

Figure 4–4: Comparative Analysis of HushRelay and SpeedyMurmur for simulated instances

- Success Ratio: It is the ratio of the number of successful payments to the total number of payment transfer requests submitted in an epoch.
- TTR *(Time Taken to Route)*: When a transaction request arrives in the network, it is the time taken from the start of routing protocol till its completion (returning the set of feasible paths).

We allow just one trial (i.e. $a = 1$) of SpeedyMurmur since HushRelay executes just once. The number of landmarks varies between 1,2,4 and 6.

- Real Instances- We execute HushRelay and SpeedyMurmur on snapshots of Ripple Network [93] and Lightning Network [123]. The results are recorded in Table 4.1. We observe that HushRelay attains a success ratio of 1 with an execution time of 2.4s and SpeedyMurmur [122], attains a success ratio of 0.98 when the number of landmarks used is 6 but the execution time is around 4.8s for Ripple Network. In the case of the Lightning network, we observe that HushRelay attains a success ratio of 0.99 with an execution time of 0.15s. However, SpeedyMurmur attains a success ratio of 0.91, with an execution time of 1.9s when the number of landmarks increases.
- Simulated Instances- The capacity of each payment channel is set between 20 to 100, and each transaction value ranges between 10 to 80. For each synthetic graph, we execute a set of 2000 transactions, each execution is done on the original state of the graph. The source code for SpeedyMurmur is available in [15]. It is written in Java and makes use of the graph analysis tool

GTNA[1]. From the graphs plotted in Fig. 4–4 a), we observe that as the number of landmarks increases, SpeedyMurmur gives a better success ratio. However, Fig. 4–4 b), we observe that execution time for routing increases with an increase in the number of landmarks. On the other hand, HushRelay achieves a better success ratio having less execution time.

4.6.3 Discussion

Run time Complexity. Time complexity of HushRelay is $\mathcal{O}(n^2)$ for the asynchronous case. For SpeedMurmur, the execution time is $\mathcal{O}(l^2)$, where l is the height of the breadth-first search or BFS tree rooted at a landmark node. The upper bound on the height of the tree is equal to the diameter of the network D where $D \leq n$.

We infer from the results that splitting payment value without any knowledge of the channel's residual balance may lead to failure in routing. The success ratio for SpeedyMurmur is higher when the number of landmarks is high but at the cost of higher execution time. On the other hand, HushRelay has a higher success ratio since the flow is split on the fly as per the residual balance of the channels. The run time of SpeedyMurmur is higher than HushRelay when the number of landmarks increases. This is because for finding each path in SpeedyMurmur, a BFS tree rooted at the landmark needs to be explored.

Communication Complexity. The number of messages exchanged (*for push request, push accepted/NAK, height updation*) is also bounded. The communication complexity and termination condition of the algorithm in the given graph are stated in [110]. $\mathcal{O}(n^2m)$ messages are exchanged in the asynchronous implementation. The overhead lies in the interprocessor communication between a vertex and its neighbors. SpeedyMurmur has $\mathcal{O}(m)$ message complexity. The communication complexity is lower than HushRelay, but the latter is more efficient with a higher success ratio.

[1]https://github.com/BenjaminSchiller/GTNA

Chapter 5

CryptoMaze: Privacy-Preserving Splitting of Off-Chain Payments

Payment Channel Networks or PCNs solve the problem of scalability in Blockchain by executing payments off-chain. Designing privacy-preserving routing and payment protocols for such networks is a big challenge. Most of the routing algorithms focus on finding a single path for routing a transaction. However, routing high-valued payments via a single path may not be feasible. After several payments get executed in the network, channels in a path may not have sufficient balance to relay the funds. In such circumstances, it is better to split high-valued payments across multiple paths to increase the success rate of transactions.

Several single path payment protocols like Hashed Time-Lock Contract or *HTLC* [113], Multi-Hop HTLC [93], Anonymous Multi-Hop Lock or *AMHL* [94] have been proposed that works for single-path payment. However, a direct extension of such protocols into multi-path payment may fail to guarantee atomicity. Sprites [99] was proposed for Ethereum-styled PCN guarantees atomicity of payments and locks constant collateral. In [55], [27], a similar construction has been proposed Bitcoin-compatible PCN. However, such protocols work for single-path and lack discussion on multi-path settings.

The multi-path payment was first discussed in SilentWhisper [92], but at the cost of substantial computation overhead. The protocol was not atomic. Osuntokun [10] was the first to propose a protocol that guarantees the atomicity of split payments. It uses linear secret sharing of the commitments shared across the multiple paths routing partial payments. But this protocol is susceptible to *wormhole attack* and high latency. In [111], *split payments* was proposed where the payment is split into unit-amounts and routed through the same or different routes. However, the authors state that their protocol does not stress achieving atomicity. Partial satisfaction of payment is considered a favorable outcome. The problem of latency and throughput in *AMP* is addressed by another payment protocol, Boomerang [32]. However, the protocol suffers from the problem of *wormhole attack* and requires locking of excess collateral. In [86], a payment protocol termed D-HTLC was proposed for multiple paths. However, the protocol relies on the atomicity of payments using a penalization mechanism. Levying penalty is not a good method since honest nodes might lose coins without any fault. A protocol *Non-Atomic Payment Splitting (NAPS)* that recursively splits payment is discussed in [51]. However, the protocol does not aim for atomicity, and partial payment is considered a valid outcome. Eckey et al. [53] had proposed an atomic payment protocol that allows intermediaries to split payments dynamically by adapting to the local condition. The protocol is atomic, privacy-preserving, and not susceptible to wormhole attacks. However, each node forwarding payment uses homomorphic encryption to encrypt the payment information. Such an operation is quite computation-intensive. The receiver's public key is forwarded to all the nodes routing partial payments. Though the authors claim that partial payments remain unlinkable, colluding parties can link payments if they observe that the same public key is circulated. It is a non-trivial task to design a protocol for multi-path payment, and we discuss the challenges faced.

Challenges faced in multi-path payments

- *Atomicity of payments*: Several distributed routing algorithms [120, 114, 92, 122, 136, 144, 73, 138, 98, 86] have been proposed for relaying transaction across multiple paths. Applying existing payment protocols like Hashed Timelock Contract [113], [102], `BOLT` [67], Sprites [99], [93], Anonymous Multihop Lock or `AMHL` [94], individually on each of the paths might not guarantee atomicity. It is quite possible that an instance of the protocol might fail in a particular path due to resource constraints or malicious behavior of nodes [55], [121]. The payment must be *atomic* - either all the micro-transactions succeed, and the receiver receives the full amount or it fails completely. If funds get transferred partially, the sender has to make several attempts for the residual amount.

- *Susceptible to wormhole attack*: Existing multi-path payment protocols like *AMP*[10], *Boomerang* [32] achieve atomicity. Each path forwarding the partial payment uses the same commitment, making it susceptible to *wormhole attack* [94]. Malicious parties in a given path may collude and steal an honest party's processing fee.

- *Multiple off-chain contracts on shared channels*: Multiple paths routing a single payment may not be edge-disjoint. In Fig. 5-1, M wants to transfer 5.1 $units$ to N. The payment is split across two paths $p_1 = \langle MA \rightarrow AB \rightarrow BD \rightarrow DN \rangle$ and $p_2 = \langle MA \rightarrow AC \rightarrow CD \rightarrow DN \rangle$ into 2.6 units and 2.5 units respectively. Each intermediate parties charge a processing fee of 0.1 units. The two paths share channels MA and DN. Two distinct off-chain contracts are created on these shared channels for routing partial payment. Also, nodes A and D get paid twice for forwarding each partial payment. This creates an additional cost overhead on the sender M. Instead of multiple off-chain contracts, it is better to construct one off-chain contract on shared payment channels for a single payment instance.

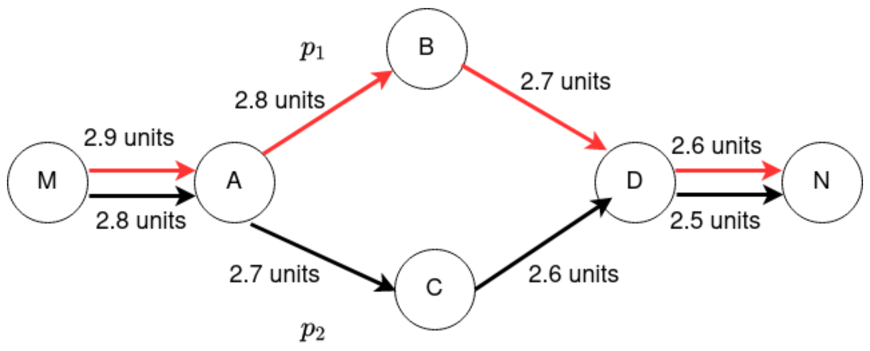


Figure 5-1: Paths p_1 and p_2 shares channels MA and DN

- *Linkability between partial payments*: A given node will be willing to route full payment instead of partial payments [53]. If a partial payment fails in one of the paths, then the entire payment rolls back. Fearing a lower success rate on routing split payments, if colluding parties can link partial payments, they will tend to reject such requests and preserve their channel capacity for routing the full amount. Unlinkability must be ensured to prevent censoring split payments.

Our goal is to construct a payment protocol that addresses all the shortcomings discussed above.

5.1 Contributions

- We propose CryptoMaze, an efficient, privacy-preserving, atomic multi-path payment protocol. Our protocol optimizes the setup cost by avoiding the formation of multiple off-chain contracts on a channel shared by partial payments. To date, no other protocol has been able to achieve this optimization.

- Our protocol ensures balance security, i.e., honest intermediaries do not lose coins while forwarding the payment.

- Our protocol description ensures unlinkability between partial payments.

- We have modeled CryptoMaze and defined its security and privacy notions in the Universal Composability or UC framework.

- Experimental Analysis on several instances of Lightning Network and simulated networks show that our proposed payment is as fast as Atomic Multi-path Payment [10]. The run time is around 11s for routing a payment of 0.04 BTC in a network instance of 25600 nodes.

Notation	Description
$C : E \times \mathbb{N} \rightarrow \mathbb{R}^+$	Capacity function
$id_{i,j}$	Identifier of payment channel $(U_i, U_j) \in E$
$\mathbb{B}$	Blockchain
U_0	Payer, a node in set V
U_r	Payee, a node in set V
$gain : V \rightarrow \mathbb{R}^+$	Function defining coins gained by a node
$\mathbb{PC}$	Set of payment channels, created by U_0
T_i	Timestamp at which node U_i receives its first incoming contract request.
δ	latency
$\mathcal{F}$	Ideal functionality for payment in PCN
$\mathcal{F}_{\mathbb{B}}$	Ideal functionality for Blockchain $\mathbb{B}$
$\mathcal{F}_{smt}$	Ideal functionality for secure message transmission

Table 5.1: Notations used in the chapter

The communication overhead is within feasible bounds, being less than 1MB. The code is available in [6].

5.1.1 Organization

The rest of the chapter has been organized as follows: Section 5.2 provides the background concept needed for understanding the chapter. Our proposed protocol has been described in detail in Section 5.3. We discuss the security of our protocol in Universal Composability (UC) framework in Section 5.4 and provide the security analysis. The experimental observation has been provided in Section 5.5.

5.2 Background

In this section, we provide the required background for understanding our protocol. The terms source/payer means the sender node. Similarly, sink/payee/destination means the receiver node. A payment channel has been referred to as an edge. Table 5.1 states the notations used in the chapter.

5.2.1 Payment Channel Network (PCN)

A Payment Channel Network is modeled as a bidirected graph $G = (V, E)$ where V represents the participants in the network and $E \subseteq V \times V$ denotes the payment channels existing between parties [113], [48]. Opening a payment channel (U_i, U_j) is equivalent to the opening of two unidirectional payment channels (U_i, U_j) and (U_j, U_i). The channel identifier for (U_i, U_j) is denoted as $id_{i,j}$. The underlying blockchain, denoted as $\mathbb{B}$, acts like a trusted append-only ledger recording the opening and closing of payment channels. A capacity function, defined as $C : E \times \mathbb{N} \to \mathbb{R}^+$, denotes the balance of each party in the channel at a given time. For example, $C((U_i, U_j), t)$ denotes the balance of party U_i in the channel $id_{i,j}$ at time t. We define the fee charged by each node as $f : V \to \mathbb{R}^+$. The fee calculated is proportional to the coins a particular node is routing through its channel. If a party U_i receives a request to transfer val coins at time $t_{current}$ to a node U_j, it checks locally whether there exist payment channels connected to U_i and $C((U_i, U_j), t_{current}) \geq val$.

5.2.2 Off-chain Contracts

Off-Chain contracts are smart contracts where the logic encoded in the contract is not run by the miners. It is mutually executed by the participants involved in instantiating the contract. The advantage of having off-chain contracts are that computation-intensive tasks can be executed without involving blockchain as long as participants behave honestly. An individual player can prove the correct contract state independently. Cheating is prevented as the state of the contract is signed by all the players. If a party misbehaves by broadcasting a wrong state in blockchain, the counterparty can raise a dispute and publish the valid accepted state. Hashed Timelock Contract or HTLC [113] is one such example used in PCN for routing payments in the network. The logic used is a hash function, where players need to provide the preimage of the hash to claim coins.

5.2.3 Random Oracle Model

The random oracle model is used to in place of hash functions in a variety of applicatons. Proving security of protocols in this model are often very simple and efficient.

The hash function $\mathcal{H}$ is modeled as a truly random function O. If the hash function $\mathcal{H}$ maps a message in message space $\mathcal{M}$ to fixed length tag in tag space $\mathcal{T}$, then function O is sampled uniformly from the set $\text{Funs}[\mathcal{M}, \mathcal{T}]$. Here, $\text{Funs}[\mathcal{M}, \mathcal{T}]$ denotes the set of all possible random functions $f : \mathcal{M} \to \mathcal{T}$.

The function O is called random oracle and proof in this setting is said to be secure in the random oracle model [38]. The random oracle works like a box that returns a binary string as output when the input supplied is a binary string. Both the honest and malicious parties can interact with the random oracle. The working of random oracle is consistent, i.e., if $\mathcal{O}$ outputs y on input x, then the answer returned is same whenever a query is made on input x. It is assumed that the input to the random oracle is private [81].

5.3 Proposed construction

We provide a detailed overview of our protocol using a payment instance. U_0 wants to transfer val coins to U_r efficiently via the PCN $G = (V, E)$, where $U_0 \in V, U_r \in V$. None of the nodes in the network must learn the identity of the payer, payee, or the coins transferred. Any honest party must not lose coins while routing the payment. We discuss the system requirements, security, and privacy goals, followed by a formal description of the proposed protocol for realizing the payment.

5.3.1 System model

Given the PCN $G = (V, E)$, a function $gain : V \rightarrow \mathbb{R}^+$ is defined to quantify the coins any node has gained or lost while running an instance of the protocol. If we assume that the protocol starts at time t_0 and ends at time t' then, for a node v, $gain(v) = \sum\limits_{u \in V, (v,u) \in E} C((v,u), t') - C((v,u), t_0)$. The global ideal functionality for blockchain $\mathcal{F}_{\mathbb{B}}$ [93] maintains $\mathbb{B}$. An arbitrary condition can be specified in the contract in order to execute a transaction in $\mathbb{B}$. $\mathcal{F}_{\mathbb{B}}$ is entrusted to enforce fulfillment of the contract before the corresponding transaction is executed. t_{end} is the least timeout period set for an off-chain contract. Δ is the worst-case time taken for a transaction to settle on-chain. Each node $U_i \in V$ has its pair of the private key and public key. A pair of honest users sharing a payment channel communicate using ideal functionality for secure message transmission $\mathcal{F}_{smt}$ [43]. U_i send $(sid, \text{instruction}, U_i, U_j, m)$, containing the secret message m, to U_j via $\mathcal{F}_{smt}$. $(sid, \text{instruction}, U_i, U_j, |m|)$ is leaked to an adversary, where $|m|$ is the message length.

System Assumption. Any user can get information of the network topology by sending a *read* instruction to $\mathcal{F}_{\mathbb{B}}$. The latter replies by sending the whole transcript of $\mathbb{B}$. The residual balance of each payment channel is kept private by the users sharing a payment channel. Payment fee charged by a user is publicly known. All the nodes know each other's public keys. We do not discuss other problems occurring in the network like individual channel congestion, blocking of nodes, etc. These issues are orthogonal to the problem addressed in this chapter. Problems arising due to concurrent payments can be addressed with the solutions proposed in [93].

Communication Model. We consider the bounded synchronous communication model [24]. In this model, time corresponds to the number of entries of $\mathbb{B}$, denoted by $|\mathbb{B}|$, divided into fixed communication rounds. If a user sends a message in a round, the intended recipeint gets the message within a bounded time. If no message is received in a round then it indicates there was no communication.

5.3.2 Security and Privacy goals

We identify the following security and privacy notions:

- *Correctness*: Given all the nodes routing the payment are honest, $gain(U_0) = -(val + \sum_{(U_i,U_j)\in\mathbb{PC}:U_i\notin\{U_0,U_r\}} f(val_{i,j}))$, $gain(U_r) = val$ and $gain(U_i) = f(val_i), \forall U_i \in V \setminus \{U_0, U_r\}$ where $f(val_i) = \sum_{\forall U_j\in V,(U_i,U_j)\in\mathbb{PC}} f(val_{i,j})$.

- *Consistency*: No intermediate node $U_i \in V \setminus \{U_0, U_r\}$ can provide the decommitment for the preceding off-chain contracts before the release of the decommitment in at least one of the succeeding off-chain contracts. If this holds, then no wormhole attack is possible as intermediate nodes cannot be bypassed.

- *Balance Security*: Honest intermediary does not lose coins, i.e., for any honest $U_i \in V \setminus \{U_0, U_r\}$, $gain(U_i) \geq 0$.

- *Value Privacy*: Corrupted users outside the payment path must not have any information regarding the payment value in a pay operation involving only honest users.

- *Unlinkability*: Given a node U_i splits the payments val into k parts $val_1, val_2, \ldots, val_k$ among the k neighbors $U_{i,1}, U_{i,2}, \ldots, U_{i,k} : (U_i, U_{i,j}) \in E, j \in [1, k]$. If all the neighbors collude, they cannot figure out whether they are part of the same payment or different payment.

- *Relationship Anonymity*: Given two simultaneous successful pay operations of the form (U_0, U_r, val) and (U_0', U_r', val), using the same set of intermediate nodes and payment channels for routing payment, with at least one honest intermediate user U_i, corrupted intermediate users cannot determine whether the payment is from U_0 to U_r or from U_0' to U_r' with a probability greater than $\frac{1}{2}$.

- *Atomicity*: If all the nodes preceding U_r have forwarded their partial payments, then only the receiver can start claiming payments. Even if one of the nodes fails to forward the payment, then $gain(U_r) = 0$ and $gain(U_i) = 0, \forall U_i \in V \setminus \{U_0\}$.

5.3.3 Mapping a set of paths into a set of edges

In the example shown in Fig. 5-2, M wants to transfer an amount 5.1 units to N. Each intermediate node charges 0.1 unit as a processing fee. Initially, the set of routes must be realized by M. Any known routing algorithm like [122, 144, 86, 138] or [98] can be used. The paths returned are $p_1 = \langle id_{M,A} \rightarrow id_{A,B} \rightarrow id_{B,D} \rightarrow id_{D,N} \rangle$ and $p_2 = \langle id_{M,A} \rightarrow id_{A,C} \rightarrow id_{C,D} \rightarrow id_{D,N} \rangle$. Given that there are *four* intermediate nodes, M forwards 5.5 units to A, the latter will deduct 0.1 units, split the amount, and forwards 2.7 units each to channels $id_{A,B}$ and $id_{A,C}$. Node B and C charge 0.1 units each and forwards 2.6 units to channels $id_{B,D}$ and $id_{C,D}$ respectively. D deducts 0.1 unit and forwards 5.1 units to N. In the paths p_1 and p_2, the channels $id_{M,A}$ and $id_{D,N}$ are shared. Instead of considering each path individually, a union of all the edges present in p_1 and p_2 is taken and set $\mathbb{PC}$ is constructed. The channels are inserted into the set in breadth-first order, starting from M. The set $\mathbb{PC} = \{id_{M,A}, id_{A,B}, id_{B,D}, id_{A,C}, id_{C,D}, id_{D,N}\}$ is used as the protocol's input. Thus, mapping a set of paths into a set of edges allows a shared edge to appear not more than once in $\mathbb{PC}$.

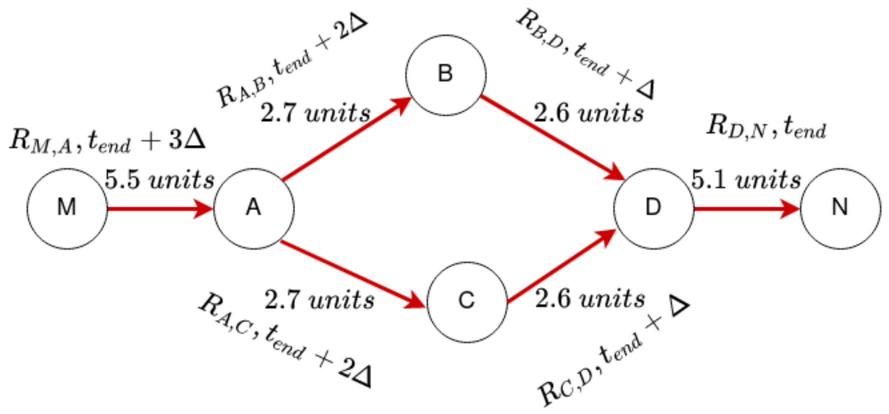

$R_{M,A}, t_{end} + 3\Delta$
5.5 $units$
$R_{A,B}, t_{end} + 2\Delta$
2.7 $units$
$R_{B,D}, t_{end} + \Delta$
2.6 $units$
$R_{D,N}, t_{end}$
5.1 $units$
M
A
B
C
D
N
2.7 $units$
$R_{A,C}, t_{end} + 2\Delta$
2.6 $units$
$R_{C,D}, t_{end} + \Delta$

Figure 5-2: CryptoMaze executed on the network for routing payment from M to N

5.3.4 Formal definition of the protocol

The protocol involves three phases: *Preprocessing Phase*, *Contract Forwarding Phase* and *Release Phase*. U_0 forms the set $\mathbb{PC}$ and uses it as an input for *Preprocessing Phase*. We define each phase in detail.

5.3.4.1 Preprocessing phase

U_0 extracts out the set of edges from $\mathbb{PC}$. We divide the phase into sub-phases, explained as follows:

(i) *Secret value for claiming Payment.* The payee U_r samples a random number $x_{\acute{r}}$ and sends $X_{\acute{r}} = x_{\acute{r}}\mathcal{G}$ to U_0 via a secure communication channel. U_0 checks the number of incoming channels sending partial payments to U_r. If there are k such channels, U_0 samples $y_i \in \mathbb{Z}_q$. The latter constructs the condition for each off-chain contract in reverse order, starting from node U_r. For any channel $id_{b,r} \in \mathbb{PC}, \mathbb{U} \in \mathbb{V}$, $R_{b,r} = e_{b,r} \sum\limits_{i=1}^{k} y_i\mathcal{G} + X_{\acute{r}}$. $R_{b,r}$ is the condition encoded in the off-chain contract formed on the channel $id_{b,r}$. $e_{b,r}$ is blinding factor for hiding the secret value $y = \sum\limits_{i=1}^{k} y_i$. It is defined as $e_{b,r} = \mathcal{H}(\sum\limits_{i=1}^{k} y_i || id_{b,r})$. U_r needs to provide the discrete logarithm of $R_{b,r}$ for claiming coins from U_b.

(ii) *Conditions for off-chain contracts.* If any intermediate node U_i is forwarding payment to a single node, U_0 samples independent strings $x_i \in \mathbb{Z}_q$ for the node. If a node U_i forwards payments to multiple neighbors, then it must be ensured that U_i does not lose coins when one of the neighbors fail to release the decommitment of an off-chain contract. To avoid this problem, our protocol uses a *1-out-of-m* policy where even if one of the outgoing neighbors of U_i responds, the latter can claim the coins. We first explain the procedure for computing secrets for a node that splits the payment value and forwards it to multiple neighbors with an example.

In Figure 5-2, the condition used in the contracts established on each of the channels are denoted as follows: $R_{M,A}$ for $id_{M,A}$, $R_{A,B}$ for $id_{A,B}$, $R_{A,C}$ for $id_{A,C}$, $R_{B,D}$ for $id_{B,D}$, $R_{C,D}$ for $id_{C,D}$, and $R_{D,N}$ for $id_{D,N}$. Node A splits the payment and sends it to nodes B and C. The condition $R_{M,A}$ must be constructed so that the secrets provided by either B or C helps A in claiming the amount from M. If A establishes the same contract R with nodes B and C, then $R_{M,A} = R + e_{M,A}x_A\mathcal{G}$. If B and C collude, they can link their payments. The situation is shown in Figure 5-3. To avoid the problem, two different conditions $R_{A,B}$ and $R_{A,C}$ are assigned to off-chain contracts on channels $id_{A,B}$ and $id_{A,C}$. A adjusts the value by adding $x_{A,B}\mathcal{G}$ to $R_{A,B}$ and $x_{A,C}\mathcal{G}$ to $R_{A,C}$ to ensure equality. Thus, we have $R_{M,A} = R_{A,B} + e_{M,A}x_A\mathcal{G} + x_{A,B}\mathcal{G}$ where $R_{A,B} + x_{A,B}G = R_{A,C} + x_{A,C}\mathcal{G}$.

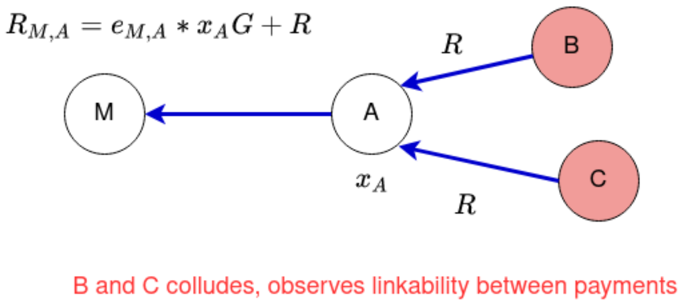

$R_{M,A} = e_{M,A} * x_AG + R$
M
A
B
C
R
R
x_A
B and C colludes, observes linkability between payments

Figure 5-3: Problem of linkability between partial payments

Let $Z = R_{A,B} + x_{A,B}\mathcal{G} = R_{A,C} + x_{A,C}\mathcal{G}$. If we fix the discrete logarithm of Z to $x : Z = xG = R_{A,B} + x_{A,B}\mathcal{G} = R_{A,C} + x_{A,C}\mathcal{G}$, we can calculate the values $x_{A,B}$ and $x_{A,C}$. Again, $x_A = x_{A,B} + x_{A,C}$. Even if the off-chain contracts $R_{A,B}, R_{A,C}$ and $R_{M,A}$ are settled on-chain, still a miner cannot establish linkability between the three. If $x_{A,B}$ and $x_{A,C}$ is not known to the miner, then it can establish a relationship between discrete logarithm of $R_{M,A}$ and discrete logarithm of $R_{A,B}$ or $R_{A,C}$ with negligible probability.

Summarizing the procedure, for a pair of channels $id_{i,j}$ and $id_{j,k}$, having conditions $R_{i,j}$ and $R_{j,k}$ where $U_j \neq U_r$, $e_{i,j} = \mathcal{H}(x_j||id_{i,j})$:

(a) If U_j forwards payment to only one neighbor U_k, the condition $R_{i,j}$ is defined as follows:

$$R_{i,j} = e_{i,j}x_j\mathcal{G} + R_{j,k} \tag{5.1}$$

(b) If U_j splits the payment and forwards it to one of the neighbour U_k, $R_{i,j}$ is defined as follows:

$$R_{i,j} = e_{i,j}x_j\mathcal{G} + R_{j,k} + x_{j,k}\mathcal{G} \tag{5.2}$$

where $U_k \in \text{neighbor}(U_j), id_{j,k} \in \mathbb{PC}$.

To compute $x_{j,k}$ for $R_{j,k}$, U_0 generates a random value $\hat{x}$ such that $\hat{x}\mathcal{G} + X_{\acute{r}} = R_{j,k} + x_{j,k}\mathcal{G}, \forall U_k \in V, id_{j,k} \in \mathbb{PC}$. Fixing discrete logarithm as $\hat{x}$ helps U_0 to calculate $x_{j,k}$ for each channel $id_{j,k}$ corresponding to node U_k. The expression can be rewritten as follows:

$$x_{j,k}\mathcal{G} = X_{\acute{r}} + \hat{x}\mathcal{G} - R_{j,k} \tag{5.3}$$

The discrete logarithm of $X_{\acute{r}} + \hat{x}\mathcal{G} - R_{j,k}$ is known to U_0, i.e. $x_{j,k} = \hat{x} - \text{dlog}(R_{j,k} - X_{\acute{r}})$, where dlog is the discrete logarithm. Once each $x_{j,k}$ gets computed, U_0 computes $x_j = \sum\limits_{U_k \in V, id_{j,k} \in \mathbb{PC}} x_{j,k}$. Thus, for any node U_i forwarding payments to multiple neighbors, the discrete logarithm for $R_{i,j}$ can be supplied by any of the outgoing neighbors of U_j. Substituting the value of $x_{j,k}\mathcal{G}$ obtained from (5.3) in (5.2), we have:

$$R_{i,j} = e_{i,j} x_j \mathcal{G} + X_{\acute{r}} + \hat{x}\mathcal{G} \tag{5.4}$$

(iii) *Setting timeout period.* The least timeout period assigned to the all incoming contract of U_r is denoted as t_{end}. Starting from this point, the timeout period of all the preceding contracts get decided. For time-locked contracts established with any channel $id_{i,j}, U_j \neq U_r$, assign $t_{i,j} = \max\limits_{\forall U_k \in V, id_{j,k} \in \mathbb{PC}} \{t_{j,k}\} + \Delta$ as the timeout period of the contract on payment channel $id_{i,j}$.

5.3.4.2 Contract Forwarding Phase

Each node U_i uses shared variable $flag_i$ and T_i, both initialized to 0. The variable $flag_i$ is set to 1 if the node U_i has received all the incoming contracts. T_i is set to the current time when U_i receives its first incoming contract request. U_i waits for time $T_i + \delta$ to receive all the incoming contract requests, where $\delta > 0$ is the latency. If the time elapsed is greater than $T_i + \delta$ but $flag_i$ is still 0, then U_i sends abort to its preceding contracts, canceling the payment.

Starting from node U_0, any node $U_i \neq U_r$ sends the request $(R_{i,j}, val_{i,j}, t_{i,j})$ for forming contracts to all its neighbor via $\mathcal{F}_{smt}$, once $flag_i$ is set to 1. For ease of analysis, we explain the procedure for one of its neighbors, say U_j. If the latter accepts the request, it gets the encrypted message $Z_{i,j}$. Upon decryption, it gets $M_j = \{(val_{j,k}, x_{j,k}, R_{j,k}, t_{j,k}, Z_{j,k}) : \forall k \in V, id_{j,k} \in \mathbb{PC}\}$, where $Z_{j,k}$ is the encrypted message to be forwarded to the node U_k. U_j checks the consistency of incoming contracts with the terms stated for an outgoing contract by calling the subroutine **Time-LockContractForward**, described in Module 4. The checks mentioned in this subroutine ensure the integrity of the phase. If the subroutine returns failure, then U_j cancels all the off-chain contracts formed with preceding nodes. Else, U_j waits for all preceding contracts such that the total value from the incoming contract is the summation of the fee charged by U_j and the coins it needs to lock in all the outgoing contracts specified in M_j. After U_j receives all the contracts within time $T_j + \delta$, then it begins forwarding the payment to its neighbor. The steps are defined in Procedure 6. The execution time is determined by the degree of the node and thus the time complexity of the procedure is $O(\frac{|E|}{|V|})$ where $|E|$ is the number of edges and $|V|$ is the number of vertices in G.

A node can identify its predecessor if the former obtains similar messages upon decryption. The node can forward one such message and discard the rest. This phase continues till all partial payments reach U_r. Even if there is one off-chain contract that did not get instantiated in a payment channel belonging to $\mathbb{PC}$, U_r cannot compute the secret y for claiming coins. Satisfying this constraint implies that all the partial payments have been combined properly, guaranteeing *atomicity*. Once the receiver has received all the partial payments within a bounded amount of time, it triggers the *Release Phase*.

5.3.4.3 Release Phase

U_r gets the secret share from all the incoming off-chain contracts forwarding the payment. The former can compute the secret y as described in Procedure 7. Upon computing the secret, U_r calls the subroutine **TimeLockContractRelease** defined in Module 5. The module returns the solution for the condition encoded in the incoming contracts forwarded by its neighbor. If the solution is correct, U_r sends a decision of acceptance to its predecessor along with the secret. Else, it sends an abort message to the neighbors and the payment fails. The abort process is mentioned in Procedure 9. Any intermediate node, involved in forwarding conditional payment can claim the coins if at least one of the neighbors responds. The steps followed by an intermediate node for claiming payment have been defined formally in Procedure 8. Time complexities of Procedure 7 and Procedure 8 are $O(\frac{|E|}{|V|})$ each.

Scriptless lock based on a two-party ECDSA signature [94] can be easily integrated into our framework for conditional payments. We have discussed the construction in the next section.

5.3.5 Use of scriptless lock in Cryptomaze

We leverage the use of scriptless scripts, where a signature scheme can be used simultaneously for authorization and locking. The crux of a scriptless locking mechanism is that the lock can consist only of a message m and a public key pk of a given signature scheme, and can be released only with a valid signature σ of m under pk. We next define how scriptless ECDSA signature can be used as a locking mechanism, the construction is similar to the one defined in [94]. The main idea used here is that the locking algorithm is initiated by two users U_i and U_j who agree on a message m, for our purpose we consider $m = id_{i,j}$, and on the value $R_{i,j} = r_{i,j}\mathcal{G}$ of unknown discrete logarithm. The two parties then generate a random number k and agree on a randomness $R = kR_{i,j}$. The shared *ECDSA signature* is computed by *"ignoring"* the $R_{i,j}$, since the parties are unaware of its discrete logarithm. The signature computed is (r_x, s) where it can be written as

Module 4: TimeLockContractForwardPhase for node U_j

```
Input : $(info)$
Parse $info$ to get $D_j, t_{i,j}, R_{i,j}$.
Parse $D_j = \{(id_{j,k}, x_{j,k}, R_{j,k}, t_{j,k}) : \forall k \in V, id_{j,k} \in E\}$
Compute $x_j = \sum_{\forall k \in V, id_{j,k} \in E} x_{j,k}$
Compute $e_{i,j} = \mathcal{H}(x_j || id_{i,j})$
if $|D_j| > 1$ then
    for $k \in V : id_{j,k} \in E$ do
        if $R_{i,j} \overset{?}{=} e_{i,j} x_j \mathcal{G} + R_{j,k} + x_{j,k} \mathcal{G}$ and $t_{i,j} \overset{?}{\geq} t_{j,k} + \Delta$ then
            continue
        end
        else
            return failure
        end
    end
end
else
    if $R_{i,j} \neq e_{i,j} x_j \mathcal{G} + R_{j,k}$ or $t_{i,j} < t_{j,k} + \Delta$ then
        return failure
    end
end
return success
```

Module 5: TimeLockContractReleasePhase for node U_j

```
Input : $(info)$
Parse $info$ to get $x_j, id_{i,j}, r_{j,k}$.
Compute $e_{i,j} = \mathcal{H}(x_j || id_{i,j})$
Compute $r_{i,j} = e_{i,j} x_j + r_{j,k}$
return $r_{i,j}$
```

KeyGen

Upon receiving $(sid, \text{keygen}, U_j)$ from U_i and $(sid, \text{keygen}, U_i)$ from U_j:

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

Lock

Upon receiving $(sid, lock, m, R_{i,j}, pk)$ from both U_i and U_j:

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

Verify

Upon receiving $(sid, verify, m, r', z', pk)$ from both U_i and U_j, where $R_{i,j} = r_{i,j}\mathcal{G}$:

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

Figure 5-4: Interface of ideal world functionality $\mathcal{F}_{ECDSA-Lock}$

$(r_x, s'r_{i,j})$. The signature (r_x, s') is a valid *ECDSA signature* on m. Once $r_{i,j}$ is released by node U_j, it is used for completing the signature.

We define this as an ideal functionality $\mathcal{F}_{ECDSA-Lock}$ in Fig. 5-4, which has access to a Random Oracle [44]. 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 and Verify Phase have been discussed previously. CryptoMaze accesses this ideal functionality $\mathcal{F}_{ECDSA-Lock}$ for forming the lock and releasing it as well.

5.4 Security definition of CryptoMaze

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. [43]. Notations used here are similar to [93].

5.4.1 Attacker model & Assumptions

The real world execution of the protocol is attacked by an adversary $\mathcal{A}$, a PPT or *probabilistic polynomial-time* algorithm. We assume that only static corruption is allowed, i.e., the adversary must specify the nodes it wants to corrupt before the start of the protocol [49], [94]. Once a node is corrupted, $\mathcal{A}$ gets access to its internal state and controls any transmission of information to and from the corrupted node. The attacker is provided with the internal state of the corrupted node. Also, the incoming and outgoing communication of such a node gets routed through $\mathcal{A}$.

5.4.2 Ideal World Functionality

Notations. We define an ideal functionality $\mathcal{F}$ for payment in PCN. Honest nodes in the network are modeled as interactive Turing machines. Such nodes are termed as dummy parties and they can communicate with each other via $\mathcal{F}$. U_0 denotes the initiator of the protocol and U_r denotes the receiver. The latter internally access the global ideal functionality $\mathcal{F}_{\mathbb{B}}$, defined in Section 5.3.1. Any payment channel existing in $\mathbb{B}$ is denoted by $(id_{i,j}, v_{i,j}, t'_{i,j}, f_{i,j})$, 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 $f_{i,j}$ is the associated fee charged for the channel $id_{i,j}$. $\mathcal{F}$ 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 $\mathcal{L}$ [93]. Upon

Procedure 6: Contract Forwarding Phase for node $U_j \in V$

```
Upon input $(forward, m)$ from $U_i$, parse $m$ to get $R_{i,j}, val_{i,j}, t_{i,j}$.
Initialize proceed=0
$val_j = val_j + val_{i,j}$
Form contract with $U_i$ using condition $R_{i,j}$, receive $Z_{i,j}$ from $U_i$.
if $U_j \neq U_r$ then
    Decrypt $Z_{i,j}$ to get $M_j = \{(id_{j,k}, val_{j,k}, x_{j,k}, R_{j,k}, t_{j,k}, Z_{j,k}) : \forall k \in V, id_{j,k} \in E\}$
    Form set $D_j = \{(id_{j,k}, x_{j,k}, R_{j,k}, t_{j,k}) : \forall k \in V, id_{j,k} \in E\}$
    Call TimeLockContractForward Phase with $(D_j, t_{i,j}, R_{i,j})$ as the input
    if (receives success) then
        Set proceed=1
        if $T_j = 0$ then
            Set $T_j = T_{current}$
        end
    end
    if proceed=1 then
        if $val_j < \sum_{\forall U_k \in V, id_{j,k} \in E} val_{j,k} + f(val_{j,k})$ then
            Wait for timeperiod of $T_j + \delta$
            if timeperiod has elapsed and $flag_j = 0$ then
                Set proceed=0
            end
        end
        else
            Set $flag_j = 1$
            for $U_k \in V : id_{j,k} \in D_j$ do
                Send (forward, $R_{j,k}, val_{j,k}, t_{j,k}$) to $U_k$, receive response from $U_k$
                $C(U_j, U_k) = C(U_j, U_k) - val_{j,k}$
                Set Contract($id_{j,k}$)=1, send $Z_{j,k}$ to $U_k$
            end
        end
    end
    if proceed=0 then
        for $U_m \in V : id_{m,j} \in E$ do
            if $isContract(id_{m,j}) = 1$ then
                Send $(abort)$ to $U_m$.
            end
        end
    end
end
```

```
else
    if $T_r = 0$ then
        Set $T_r = T_{current}$
    end
    if $val_r < val$ then
        Wait for timeperiod of $T_r + \delta$.
        if timeperiod has elapsed and $flag_r = 0$ then
            for $U_b \in V : id_{b,r} \in E$ do
                if $isContract(id_{b,r}) = 1$ then
                    Send $(abort)$ to $U_b$
                end
            end
        end
    end
    else
        Set $flag_r = 1$.
        Call Release Phase defined in Procedure 7
    end
end
```

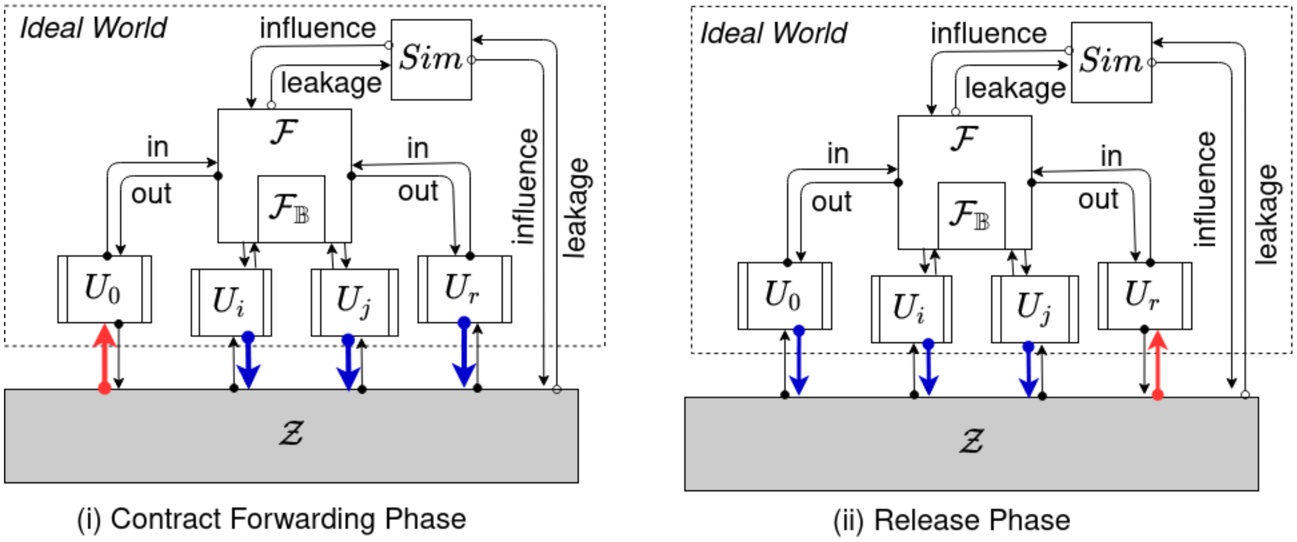


Figure 5-5: Execution of $\mathcal{F}$ with dummy parties U_0, U_r representing payer and payee, U_i,U_j representing intermediaries routing payment

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 residual capacity of the channel and $t_{i,j}$ is the expiration time of the payment, $h'_{i,j}$ is the event identifier. When a channel $id_{i,j}$ is closed on-chain, it is entered into the list $\mathcal{C}$. Payment channels

Procedure 7: Release Phase for receiver U_r

```
Set $stop = 0$
for $U_b \in V : id_{b,r} \in E$ do
    Decrypt $Z_{b,r}$ to get $\{y_{b,r}, t_{end}\}$
    if $t_{b,r} = t_{end}$ then
        $y = y + y_{b,r}$
    end
    else
        $stop$=1
        break from the loop
    end
end
if $stop$=0 then
    for $U_b \in V : id_{b,r} \in E$ do
        Call TimeLockContractRelease Phase with $(y, x_{\acute{r}}, id_{b,r})$ as input, gets $r_{b,r}$.
        if $R_{b,r} \neq r_{b,r}\mathcal{G}$ then
            $stop$=1
            break from the loop
        end
        else
            Store $r_{b,r}$.
        end
    end
end
if $stop = 1$ then
    for $U_b \in V : id_{b,r} \in E$ do
        if $isContract(id_{b,r}) = 1$ then
            Send $(abort)$ to $U_b$
        end
    end
end
else
    for $U_b \in V : id_{b,r} \in E$ do
        Send $(accept, r_{b,r})$ to $U_b$
    end
end
```

Procedure 8: Release Phase for node $U_j \in V \setminus \{U_r\}$

```
Upon receiving input $(U_i, accept, m)$, parse $m$ to get $r_{j,i}$
$C(U_i, U_j) = C(U_i, U_j) + val_{j,i}$
**if** $release_j$=*0* **then**
    Set $release_j$=1
    **if** $U_j$ *had forwarded payment to more than one node* **then**
        $r_{j,i} = r_{j,i} + x_{j,i}$
        $x_j = \sum_{\forall U_i \in V : id_{j,i} \in E} x_{j,i}$
    **end**
    **for** $U_m \in V : id_{m,j} \in E$ **do**
        Call **TimeLockContractRelease Phase** with $(r_{j,i}, x_j, id_{m,j})$ as input, gets $r_{m,j}$.
        Send (accept, $r_{m,j}$) to $U_m$.
    **end**
**end**
```

Procedure 9: Abort for node $U_j \in V$

```
Upon receiving input $(U_i, abort)$
Set $flag = 0$
$C(U_j, U_i) = C(U_j, U_i) + val_{j,i}$
**for** $id_{j,k} \in M_j$ **do**
    **if** $isContract(id_{j,k}) = 1$ **then**
        $flag = 1$
        **break from the loop**
    **end**
**end**
**if** $flag = 0$ **then**
    **for** $U_m \in V : id_{m,j} \in E$ **do**
        Send $(abort)$ to $U_m$.
    **end**
**end**
```

forwarding the payment from U_0 to U_r are put in set $\mathbb{PC}$, added serially upon breadth-first traversal of the network, starting from U_0. The flow in each channel $id_{i,j}$ present in $\mathbb{PC}$ is denoted by $val_{i,j}$.

5.4.2.1 Operations

We describe the operation PAY in the ideal world. $\mathcal{F}$ initializes a pair of local empty lists $(\mathcal{L}, \mathcal{C})$. Each session is denoted by a session identifier sid. The phase is initiated by U_0, sending the payment value val to be paid to U_r, the least timeout period of off-chain contract t_{end}. The other inputs are the set of payment channels $\mathbb{PC}$ along with the flow in each channel $val_{i,j}$ and the timeout period of off-chain contracts established on each channel, denoted as $t_{i,j}$, $\forall id_{i,j} \in \mathbb{PC}$. $\mathcal{F}$ initializes the variable $contract(sid, id_{i,j}) = 0$, $\forall id_{i,j} \in \mathbb{PC}$ to indicate that till now no off-chain contract got established in this session. PAY is divided into two phases: (i) *Contract Forwarding Phase* and (ii) *Release Phase*, defined in Fig. 5-6.

(i) The *Contract Forwarding Phase* is triggered after U_0 sends the *pay* instruction along with the set $\mathbb{PC}$ and the value of the payment. The inputs provided from environment $\mathcal{Z}$ are marked in red, as shown in Fig. 5-5(i). Each node $U_i \neq U_r, U_i \in \mathbb{PC}$ is visited in breadth-first fashion and the nodes are inserted in the queue Q_{pay}. Before sending the request to U_j, $\mathcal{F}$ checks whether an open channel $id_{i,j}$ exists in $\mathbb{B}$. Next, it checks whether the channel $id_{i,j}$ has enough capacity for forwarding the payment. The consistency of the timeout period for incoming and outgoing contracts is checked as well. If any of the conditions fail, $\mathcal{F}$ removes any entry for off-chain payments in $\mathcal{L}$ and aborts. If all the criteria hold, $\mathcal{F}$ forwards the partial payment to node U_j, output arrow marked in blue, shown in Fig. 5-5(i). If all preceding contracts of U_j got established, then it becomes a candidate for forwarding the payment. U_j is thus inserted into Q_{pay}. If U_j sends abort, then all the entries in $\mathcal{L}$ are removed and $\mathcal{F}$ aborts.

(ii) Once the payment reaches U_r, it triggers the *Release Phase* by sending a *response* to $\mathcal{F}$, input arrows marked in red, shown in Fig. 5-5(ii). If U_r sends abort, then the payment is considered to have failed. All the entries are removed from $\mathcal{L}$ and $\mathcal{F}$ aborts. If U_r responds with success, then $\mathcal{F}$ sends a success message to predecessors of U_r, updates the entry in $\mathcal{L}$. The output of the intermediate parties sent to environment $\mathcal{Z}$ is marked as blue arrows in Fig. 5-5(ii). If the predecessor sends an abort message, then such a node is marked as *visited* and the entry is pushed in $Q_{failure}$. Else, that node is considered as the candidate for forwarding the success message to its predecessors and marked as *visited*, if it has not been visited before. Nodes in $Q_{failure}$ are dealt with later after all the successful payments get settled. Each of these nodes sends an abort message to its predecessor. If a predecessor has not visited before, then it is pushed in $Q_{failure}$ and the process continues.

5.4.2.2 Discussion

The operation PAY defined in ideal functionality $\mathcal{F}$ satisfies privacy properties of CryptoMaze in the following ways:

- Correctness: In *Contract Forwarding phase*, each intermediate node U_i gets instructions for forwarding payment from $\mathcal{F}$ on behalf of node U_j, provided $\sum\limits_{U_k \in V, id_{k,j} \in \mathbb{PC}} val_{k,j} = \sum\limits_{U_m \in V, id_{j,m} \in \mathbb{PC}} val_{j,m} + f(val_{j,m})$. U_r triggers the release phase and responds with success, provided it has received the amount val. If all the parties have behaved honestly and U_r responds with success in the *release phase*, then $\mathcal{F}$ updates in $\mathcal{L}$ the channels present in $\mathbb{PC}$. Thus, U_0 can successfully complete the payment by forwarding $val + \sum\limits_{(U_i,U_j) \in \mathbb{PC}: U_i \notin \{U_0, U_r\}} f(val_i)$, where each node $U_i \in V \setminus \{U_0, U_r\}$ gains $f(val_i)$ and U_r gets the amount val.

- Consistency: Release Phase defined in Fig. 5-6 shows that U_i is pushed into the queue $\mathcal{T}$ only if there is a successor U_j that had resolved the off-chain contract forwarded by U_i. Once U_i enters into $\mathcal{T}$, then it will be popped out of the queue for resolving its preceding contracts. If all the neighbors of U_i have sent abort, then none of the preceding contracts forwarded to U_i will get resolved.

- Balance Security: Any intermediate node U_i can claim payment from its preceding neighbors if at least one of the outgoing neighbors of U_i accepted the payment. If U_i receives abort from all the successors, it will abort as well. The total balance of U_i either remains unchanged or it gains a processing fee $f(val_i)$ where $f(val_i) = \sum\limits_{\forall U_j \in V, (U_i,U_j) \in \mathbb{PC}} f(val_{i,j})$.

- Value Privacy: The ideal functionality $\mathcal{F}$ does not contact any user that does not belong to the set $\mathbb{PC}$, hence they learn nothing about the transacted value.

- Unlinkability: For all the neighbors U_j of node U_i, $\mathcal{F}$ samples a random identifier $h'_{i,j}$. Even if the neighbors collude, they cannot find any correlation amongst the payment identifiers.

- Relationship Anonymity: Follows from unlinkability. If there exist at least one honest intermediate node U_i, then it receives a unique event identifier from $\mathcal{F}$ for each payment over any of its outgoing payment channels. Since all the event identifiers are independently generated, if at least one honest user U_i lies in a payment path, any two simultaneous payments getting routed over the same set of payment channels for the same value val are indistinguishable to the outgoing neighbors of U_i receiving the request for forwarding the payments. This implies that any corrupted node cannot distinguish between the payments (U_0, U_r, val) and (U_0', U_r', val) with probability greater than $\frac{1}{2}$.

- Atomicity: If U_r triggers the release by responding with *success*, it means that it has received all the partial payments. If U_r fails to receive even one partial payment, then it will send *abort* signaling a failed payment.

5.4.3 Universal Composability (UC) Security

The ideal functionality $\mathcal{F}$ can be *attacked* by an ideal world adversary called a simulator or *Sim*, a $\mathcal{PPT}$ algorithm. An additional special party called the environment $\mathcal{Z}$ which observes both the real world and the ideal world, provides the inputs for all parties, and receives their outputs. $\mathcal{Z}$ can use the information leaked by adversary $\mathcal{A}$ or actively influence the execution. Adversary $\mathcal{A}$ can corrupt any party before the protocol starts. However, the former doesn't get any information from communication occurring between honest parties. Let $REAL_{\Pi,\mathcal{A},\mathcal{Z}}$ be the ensemble of the outputs of the environment $\mathcal{Z}$ when interacting with the attacker $\mathcal{Z}$ and users running protocol Π,

Definition 5.1. ***UC Security**. Given that λ is the security parameter, a protocol Π UC-realizes an ideal functionality $\mathcal{F}$ if for all computationally bounded adversary $\mathcal{A}$ attacking Π there exist a probabilistic polynomial time ($\mathcal{PPT}$) simulator Sim such that for all $\mathcal{PPT}$ environment $\mathcal{Z}$, $IDEAL_{\mathcal{F},Sim,\mathcal{Z}}$, and $REAL_{\Pi,\mathcal{A},\mathcal{Z}}$ are computationally indistinguishable.*

5.4.4 Security analysis

From Definition 5.1, a protocol Π is said to be *UC*-secure if $\mathcal{Z}$ cannot distinguish whether it is interacting with the ideal world or real world even in presence of a computationally bounded adversary $\mathcal{A}$. Since our protocol execution in real world relies on ideal functionalities $\mathcal{F}_{smt}$ and $\mathcal{F}_{\mathbb{B}}$, we define our protocol in the hybrid world [49] instead of real world.

Theorem 5.1. *Given λ is the security parameter, elliptic curve group of order q is generated by the base point $\mathcal{G}$, the protocol CryptoMaze UC-realizes the ideal functionality $\mathcal{F}$ in the $(\mathcal{F}_{\mathbb{B}}, \mathcal{F}_{smt})$-hybrid world.*

Proof. We design *Sim* for the ideal world execution for the following cases: either the sender is corrupt or the receiver is corrupt, or one of the intermediate node is corrupt. The only event which distinguishes hybrid world from ideal world is when the *Sim* aborts in ideal world.

PAY

(ii) **Contract Forwarding Phase** U_0 invokes $\mathcal{F}$ with message $(sid, pay, U_r, val, t_{end}, \{(id_{i,j}, val_{i,j}, t_{i,j}) : id_{i,j} \in \mathbb{PC}\}, \mathbb{PC})$.

- For each $id_{i,j} \in \mathbb{PC}$, set $contract(sid, id_{i,j}) = 0$.
- $\mathcal{F}$ forms a set $V_{\mathbb{PC}} = \{U_i\}$ such that $U_i \in V$ and has a channel in $\mathbb{PC}$.
- Initialize an empty queue Q_{pay}. Push U_0 into queue Q_{pay}.
- While Q_{pay} is not empty:
  - Pop U_i from Q_{pay}.
  - For each $U_j \in V_{\mathbb{PC}} : id_{i,j} \in \mathbb{PC}$:
    - If U_j sends $(sid, abort)$ to $\mathcal{F}$ then it removes all entries such entries from $\mathcal{L}$ added in this phase, cancel their contracts by resetting the variable to 0, and abort.
    - $\mathcal{F}$ checks $isChannel(id_{i,j}) = 1$. If the check fails, then remove all entries d_i from $\mathcal{L}$ added in this phase and abort.
    - Create $z_{i,j} = \{(id_{j,k}, val_{j,k}, t_{j,k}) : \forall U_k \in V_{\mathbb{PC}}, id_{j,k} \in \mathbb{PC}\}$, if $U_j \neq U_r$. Else $z_{i,r} = \{val, t_{end}\}$.
    - $\mathcal{F}$ checks $t_{i,j} \overset{?}{\geq} \max\limits_{U_k \in V_{\mathbb{PC}}, id_{j,k} \in z_{i,j}} \{t_{j,k}\} + \Delta$ and $val_{i,j} \leq \sum\limits_{U_k \in V_{\mathbb{PC}}, id_{j,k} \in z_{i,j}} val_{j,k} + f(val_{j,k})$. If any of the checks fail, then remove all entries from $\mathcal{L}$ added in this phase, cancel their contracts by resetting the variable to 0, and abort.
    - $\mathcal{F}$ checks whether for $(id_{i,j}, v'_{i,j}, ., .) \in \mathcal{L}$, if $v'_{i,j} \geq val_{i,j}$. If that is the case, then add $d_{i,j} = (id_{i,j}, v'_{i,j} - val_{i,j}, t_{i,j}, \perp)$ to $\mathcal{L}$, where $(id_{i,j}, v'_{i,j}, ., .) \in \mathcal{L}$ is the entry with the lowest $v'_{i,j}$. If the conditions are not met, $\mathcal{F}$ removes all entries from $\mathcal{L}$ added in this phase and abort.
    - If the conditions are met, set $contract(sid, id_{i,j}) = 1$. Sample an identifier $h'_{i,j}$ and send request $(sid, \text{forward}, U_i, id_{i,j}, val_{i,j}, t_{i,j}, h'_{i,j}, z_{i,j})$ to U_j.
    - If $isPred(sid, U_j, \mathbb{PC}, \mathbb{V}_{\mathbb{PC}})$ returns $success$, push U_j to Q_{pay}.

Figure 5-6: Ideal World Functionality for payment in PCN

- U_0 is corrupted: $\mathcal{A}$ acts like the sender U_0, and forms packet $(R_{i,j}, val_{i,j}, t_{i,j}, Z_{i,j})$, for each $id_{i,j} \in \mathbb{PC}, U_i \neq U_r$. The encrypted message $Z_{i,j}$ upon decryption gives $M_j = \{(id_{j,k}, val_{j,k}, x_{j,k}, R_{j,k}, t_{j,k}, Z_{j,k}) : \forall k \in V, id_{j,k} \in \mathbb{PC}\}$, when $U_j \neq U_r$ and $M_j = \{(y_{i,j}, t_{end})\}$, when $U_j = U_r$. $\mathcal{A}$ forwards the packet to *Sim*.

  For each node $U_i \in V, U_i \neq \{U_0, U_r\}$, *Sim* does the following:

(ii) <u>Release Phase</u> U_r invokes $\mathcal{F}$ with message $(sid, response)$.

- For each $U_j \in V_{\mathbb{PC}}$:
  - Set $visited(U_j) = 0$.
- Initialize $flag_{abort} = 0$ and initialize empty queues $\mathcal{T}$ and $Q_{failure}$.
- If $response = \perp$, then set $flag_{abort} = 1$.
- If $flag_{abort} = 0$, push U_r in T.
- While $\mathcal{T}$ is not empty:
  - Pop node U_j from $\mathcal{T}$.
  - For each $U_i \in V_{\mathbb{PC}} : id_{i,j} \in \mathbb{PC}$ and $contract(sid, id_{i,j}) = 1$:
    - Update $d_{i,j} \in \mathcal{L}$ to $(-, -, -, h'_{i,j})$, send $(sid, success, h'_{i,j})$ to U_i and U_j, set $contract(sid, id_{i,j}) = 0$.
    - If U_i sends $(sid, abort)$ then $visited(U_i) = 1$, push U_i in $Q_{failure}$.
    - Else if $visited(U_i) = 0$ and $U_i \neq U_0$, set $visited(U_i) = 1$ and push U_i in $\mathcal{T}$.
- If $flag_{abort} = 1$, then :
  - Push U_r to $\mathcal{T}$.
  - While $\mathcal{T}$ not null:
    - Pop node U_j from $\mathcal{T}$.
    - If $U_j \neq U_0$, go to the next step, else go back to previous step and continue.
    - For each $U_i \in V_{\mathbb{PC}} : id_{i,j} \in \mathbb{PC}$ and $contract(sid, id_{i,j}) = 1$:
      - set $contract(sid, id_{i,j}) = 0$. Remove $d_{i,j}$ from $\mathcal{L}$, send $(sid, \perp, h'_{i,j})$ to U_i and U_j.
    - If $U_i \notin \mathcal{T}$, push U_i in $\mathcal{T}$.
- Else:
  - While $Q_{failure}$ is not empty:
    - Pop node U_j from $Q_{success}$
    - For each $U_i \in V_{\mathbb{PC}} : id_{i,j} \in \mathbb{PC}$ and $contract(sid, id_{i,j}) = 1$:
      - set $contract(sid, id_{i,j}) = 0$. Remove $d_{i,j}$ from $\mathcal{L}$, send $(sid, \perp, h'_{i,j})$ to U_i and U_j.
      - If $visited(U_i) = 0$, set $visited(U_i) = 1$, push U_i in $Q_{failure}$.

Figure 5-6: Ideal World Functionality for payment in PCN (Continued)

$isChannel(id_{i,j})$:

- $\mathcal{F}$ sends $id_{i,j}$ to $\mathcal{F}_{\mathbb{B}}$. The latter checks for an entry in $\mathbb{B}$ of the form $(id_{i,j}, v_{i,j}, t'_{i,j}, f_{i,j})$.
- If the entry does not exist, then return 0.
- If the entry exists, then check if there is an entry $id_{i,j}$ in $\mathcal{C}$. If it is true, then return 0, else return 1.

$isPred(sid, U_i, \mathbb{PC}, \mathbb{V}_{\mathbb{PC}})$:

- For each $U_k \in V_{\mathbb{PC}} : id_{k,i} \in \mathbb{PC}$:
  - If $contract(sid, id_{k,i}) = 0$, then return *failure*.
- Return *success*

Figure 5-7: Submodules used in $\mathcal{F}$

- Form set $D_i = \{(id_{i,k}, x_{i,k}, R_{i,k}, t_{i,k}) : \forall k \in V, id_{i,k} \in \mathbb{PC}\}$.
- For each $U_j \in V : id_{j,i} \in \mathbb{PC}$:
  - Get $(R_{j,i}, t_{j,i})$, call **TimeLockContractForward Phase** with input $(D_i, t_{j,i}, R_{j,i})$ as input. If it returns failure, then abort.

Sim checks $\sum_{U_j \in V : id_{j,i} \in \mathbb{PC}} val_{j,i} = \sum_{U_k \in V : id_{i,k} \in \mathbb{PC}} val_{i,k} + f(val_{i,k})$. If the check fails, abort.

If the process didn't abort, *Sim* sends $(sid, pay, U_r, val, t_{end}, \{(id_{i,j}, val_{i,j}, t_{i,j}) : id_{i,j} \in \mathbb{PC}\}, \mathbb{PC})$ to $\mathcal{F}$. *Sim* has already checked the flow consistency for the intermediate honest nodes, as well as consistency of terms of incoming and outgoing contracts before it forwards the conditional payment.

In the *release phase*, if U_r aborts, then the process aborts as well. If U_r has released the secret, then *Sim* checks that any node U_i claiming payment from U_j has released the discrete logarithm for $R_{j,i}$. We consider that an honest intermediate node U_m splits the transaction value across multiple payment channels, and a partial value gets routed via channel $id_{m,i}$. We identify a bad event E_1: if adversary $\mathcal{A}$ has released $r_{m,i}$ for $R_{m,i} : r_{m,i}\mathcal{G} = R_{m,i}$ but $\exists U_k$ where $id_{k,m} \in \mathbb{PC}$, $r = r_{m,i} + e_{k,m}x_m + x_{m,i}$ and $R_{k,m} \neq r\mathcal{G}$ then *Sim* aborts the simulation.

Claim 5.1.1. *The probability of E_1 is 0.*

Proof. *Sim* checks the relation $R_{k,m} \overset{?}{=} R_{m,i} + e_{k,m}x_m\mathcal{G} + x_{m,i}\mathcal{G}, \forall U_i \in V, id_{m,i} \in \mathbb{PC}$ at the start. If $R_{m,i} = r_{m,i}\mathcal{G}$ but $R_{k,m} \neq r\mathcal{G}$, then $r \neq r_{m,i} + e_{k,m}x_m + x_{m,i}$, which contradicts event

E_1. Hence, the probability is 0.

- An intermediate party U_m is corrupted: If $\mathcal{A}$ is able to release the discrete logarithm of the statement used in the incoming channel's contract of node U_m before the secret is revealed, *Sim* aborts.

  When *Sim* gets $(sid, \text{forward}, U_j, id_{j,m}, val_{j,m}, t_{j,m}, h'_{j,m}, z_{j,m})$ from $\mathcal{F}$ on behalf of all incoming nodes U_j of node U_m, it samples $x_{m,k}$ for each $U_k \in z_{j,m}$, computes $x_m = \sum_{U_k \in V, id_{m,k} \in \mathbb{PC}} x_{m,k}$. *Sim* sends $(forward, R_{j,m}, val_{j,m}, t_{j,m})$, $Z_{j,m}$ to $\mathcal{A}$ on behalf of all U_js. $\mathcal{A}$ sends $(R_{m,k}, val_{m,k}, t_{m,k})$ to *Sim* for all such U_ks, on behalf of U_m. *Sim* checks whether $\forall U_j \in V$, $R_{j,m} \overset{?}{=} x_m\mathcal{G} + R_{m,k} + e_{j,m}x_{m,k}\mathcal{G}$ and $t_{j,m} \overset{?}{=} \Delta + \max_{U_k \in V, id_{m,k} \in \mathbb{PC}} \{t_{m,k}\}$ and $\sum_{U_j \in V, id_{j,m} \in \mathbb{PC}} val_{j,m} \overset{?}{=} \sum_{U_k \in V, id_{m,k} \in \mathbb{PC}} val_{m,k} + f(val_{m,k}), \forall U_k \in V, id_{m,k} \in \mathbb{PC}$. If any of the checks fail, then it sends abort to $\mathcal{F}$.

  Consider U_t as the incoming node forwarding payment to U_m and U_h as the outgoing neighbor of U_m. *Sim* samples r^* such that $R_{m,h} = r^*\mathcal{G}$ and $R_{t,m} = x_{m,h}\mathcal{G} + e_{t,m}x_m\mathcal{G} + R_{m,h}$. We identify another bad event E_2: if $\mathcal{A}$ releases r' such that $R_{t,m} = r'\mathcal{G}$ without querying *Sim* on the event identifier $h'_{m,h}$, *Sim* aborts.

  Claim 5.1.2. *The probability of E_2 is $\frac{1}{q}$, where $\mathbb{G}$ is an elliptic curve group with large order q i.e. $|\mathbb{G}| = q$.*

  Proof. Follows from the discrete logarithm hardness assumption, given a random point $h \in \mathbb{G}$, it is possible to guess the value $log_G h$ with probability $\frac{1}{q}$.

- U_r is corrupted: *Sim* receives $(sid, \text{forward}, U_j, id_{j,r}, val_{j,r}, t_{j,r}, h'_{j,r}, z_{j,r})$ on behalf of all incoming nodes U_j of node U_r from $\mathcal{F}$. *Sim* gets $X_{\acute{r}}$ from $\mathcal{A}$ and samples y_j, creates $R_{j,r} = X_{\acute{r}} + e_{j,r}y\mathcal{G}$, where $y = \sum_{U_j \in V : id_{j,r} \in \mathbb{PC}} y_j$, for all the incoming neighbors U_j of node U_r. It sends $(forward, R_{j,r}, val_{j,r}, t_{j,r})$, $Z_{j,r}$ to $\mathcal{A}$ on behalf of all U_js. We identify another bad event E_3: if there exists a node $U_k : id_{k,r} \in \mathbb{PC}$ such that $\mathcal{A}$ releases x' such that $R_{k,r} = x'\mathcal{G}$ without querying *Sim* on the event identifier $h'_{k,r}$, *Sim* aborts the simulation.

  Claim 5.1.3. *The probability of E_3 is $\frac{1}{q}$, where $\mathbb{G}$ is an elliptic curve group with large order q i.e. $|\mathbb{G}| = q$.*

  Proof. $\mathcal{A}$ knows dlog$(X_{\acute{r}})$, but it doesn't know y. Hence, $\mathcal{A}$ can guess dlog$(R_{k,r})$ with probability $\frac{1}{q}$.

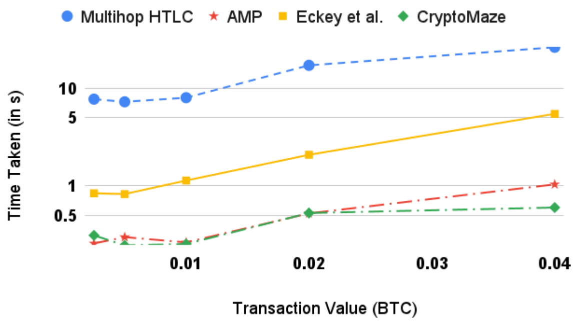
□

Indistinguishability from the ideal world. The simulator *Sim* designed is efficient since it runs a polynomially-bounded algorithm. To argue that $\mathcal{Z}$'s view in simulation is indistinguishable from the execution protocol in the hybrid-world protocol, we consider the occurrence of a bad event in `PAY`:

(i) When U_0 is corrupted, the random values sampled by *Sim* and the values are chosen by an honest U_0 follow the same distribution. Similarly, when U_r is corrupted or an intermediate node is corrupted, the random values sampled by *Sim* remain indistinguishable from the data used in honest execution.

(ii) Indistinguishability breaks when *Sim* aborts in the ideal world. We infer from Claim 5.1.1, Claim 5.1.2, and Claim 5.1.3, that bad events occur with negligible probability and hence *Sim* aborts with negligible probability.

Thus, we have proved that our protocol CryptoMaze *UC*-realizes the ideal functionality $\mathcal{F}$ in the $(\mathcal{F}_{\mathbb{B}}, \mathcal{F}_{smt})$-hybrid world. If the security and privacy goals stated in Section 5.3.2 are realized by $\mathcal{F}$, then as per *UC Definition of Security* stated in Definition 5.1 these security notions are satisfied by our protocol as well.



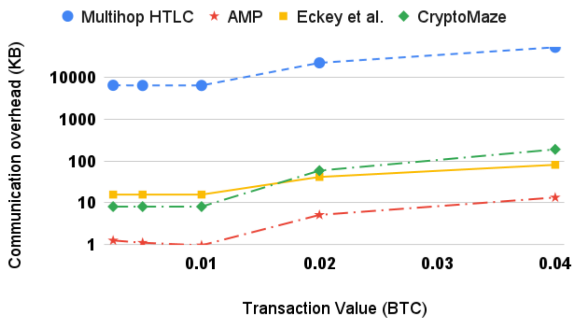


Figure 5-8: LN snapshot March 2020, Figure on the left is (a) TTP vs transaction value and on the right (b) Communication overhead vs transaction value

5.5 Experimental analysis

We choose to compare our protocol with *Multi-Hop HTLC* [93], *Atomic Multi-path Payment* [10] and *Eckey et al.* [53]. *Multi-Hop HTLC* is a single-path payment protocol, and we show how extending it to a multiple-path payment would work. In this protocol, a node forwarding payment in a given path gets a tuple (y, h_1, h_2) along with the non-interactive zero-knowledge proof Π for the statement "$\exists x' : h_1 = \mathcal{H}(x')$ *and* $h_2 = \mathcal{H}(y \oplus x')$". *Atomic Multi-path Payment* or *AMP* is the

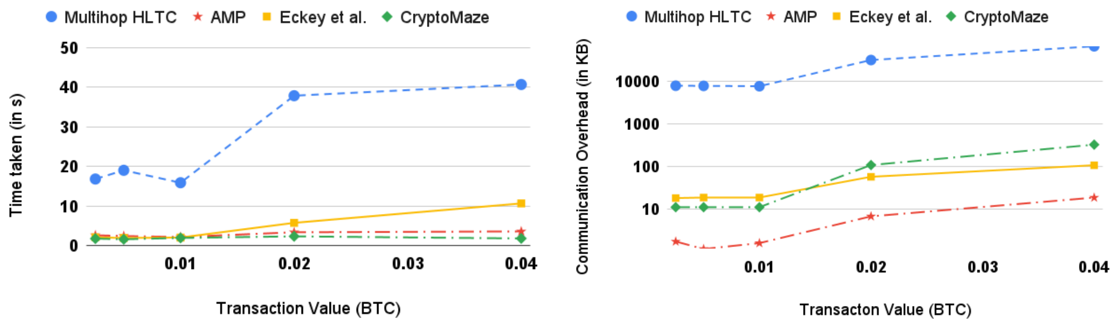


Figure 5-9: LN snapshot May 2021, Figure on the left is (a) TTP vs transaction value and on the right (b) Communication overhead vs transaction value

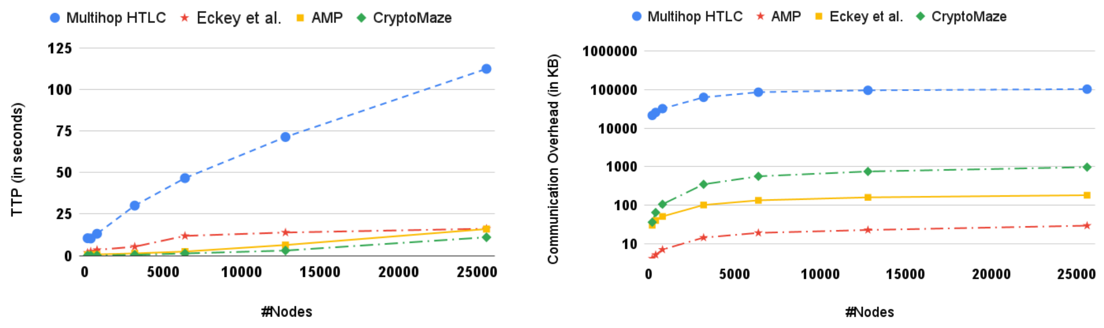


Figure 5-10: Simulated Network, Figure on the left is (a) TTP vs number of nodes and on the right (b) Communication overhead vs number of nodes

most efficient protocol in the existing state-of-the-art in terms of run time (considering best-case run time) as well as communication cost. If there are n paths in *AMP*, then the payer generates secret shares $x_1, x_2, \ldots, x_n$ for each path from the master secret $x : x = x1 \oplus x_2 \oplus \ldots \oplus x_n$. We find the objective of the protocol proposed in *Eckey et al.* similar to ours. However the payment split is decided on the fly and sharing of public key leads to linkability between partial payments.

5.5.1 Evaluation methodology

In this section, we define the experimental setup. The code is available in [6]. System configuration used is `Intel Core i5-8250U CPU`, Operating System: *Kubuntu-20.04.1*, and Memory: 7.7 GiB of RAM. The programming language used is C, compiler - gcc version 5.4.0 20160609. For implementing the cryptographic primitives in CryptoMaze, Atomic Multi-path Payment or *AMP* and *Multi-Hop HTLC*, we use the library *OpenSSL*, version-1.0.2 [127]. For constructing the zero-knowledge proof for *Multi-Hop HTLC*, we have used C-based implementation of ZKBoo[17] and libgcrypt version-1.8.4 [1]. The number of rounds for ZKBoo is set to 136. This guarantees a soundness error of 2^{-80} for the proof and witness length is set to 32 bytes. For elliptic curve operations in CryptoMaze and *Eckey et al.*, we have considered the elliptic curve secp224r1. For homomorphic encryption using *Paillier Cryptosystem* in *Eckey et al.*, *libhcs* is used [12]. It is a C library implementing a number of partially homomorphic encryption schemes [47].

5.5.1.1 Metric Used

The following metrics are used to compare the performance of *CryptoMaze* with other state-of-the-art protocols.

- TTP (*Time taken for payment*): It is the time taken for searching of eligible paths for routing a payment, formation of off-chain payment contracts, and completion of payment upon successfully fulfilling the criteria set in the contract. It is measured in seconds or *s*.
- Communication Overhead: For the given payment protocol, the number of messages exchanged between the nodes while searching for a set of paths and execution of the payment protocol, measured in kilobytes or *KB*.

5.5.2 Observations

We use the distributed routing algorithm *HushRelay* [98] for our protocol, *Atomic Multi-path Payment* and *Multi-Hop HTLC*, that returns the set of paths. Based on this set of paths as input, we run each instance of the payment protocol. Since the time taken for routing is taken into account while estimating TTP, it can be further optimized by using a more efficient distributed routing algorithm.

5.5.2.1 Evaluation on Real Instances

We select two snapshots of Lightning Network taken on March 2020 [14] and May 2021[2]. The first instance has *6329 nodes* and the second instance has *11072 nodes*. The payment amount is varied between 0.0025 BTC - 0.04 BTC.

(a) *Optimization in terms of off-chain contracts.* Before stating the observation in terms of execution time and communication cost, we analyze the saving in terms of off-chain contracts

[1] https://gnupg.org/software/libgcrypt/index.html
[2] https://www.dropbox.com/s/fkq7kh5xyu3l33t/LN_25_05_2021.json?dl=0

established in shared edges. Since state-of-the-art protocols instantiate multiple contracts on shared edges, we choose anyone as a representative and compare it with our protocol. We run 20000 payment instances for a given transaction value and state our results for the two Lightning Network instances.

- *LN instance, March 2020*: When the transaction value was increased from 0.0025 BTC to 0.04 BTC, the payment instances that had multiple routes sharing payment channels increased from 1% to 38%. The number of payment channels shared in a single payment instance increased from 33% to 55%. The number of times a particular payment channel got shared increased from 2 to 5. The total number of off-chain contracts per payment instance increased from 24% to 68.75% for state-of-the-art.

- *LN instance, May 2021*: Payment instances sharing channels for a single payment increased from 0.04% to 38.6%. Channels shared for a given instance increased from 33% to 42.8%. A channel gets shared not more than 4 times. The total number of off-chain contracts per payment instance increased up to 54.5% for state-of-the-art.

(b) *Computation and Communication Cost.* We analyze the efficiency of CryptoMaze compared to state-of-the-art in terms of the metric stated, when executed on a single payment instance.

- TTP for CryptoMaze is equivalent to *Atomic Multi-path Payment*, not exceeding 0.39s on average as shown in Fig. 5-8 (a) and it is around 1.85s in Fig. 5-9 (a) for the second snapshot. Our protocol is approximately 3 times faster than *Eckey et al.* and 17.5 times faster than *Multi-Hop HTLC* for both instances.

- The communication overhead in Fig. 5-8 (b) is 53.18KB and in 5-9 (b) is 93.203KB, on average. The overhead is 14.5 times greater than that of *Atomic Multi-path Payment* and 2 times more than that of *Eckey et al.* for both instances. The communication overhead of *Multi-Hop HTLC* is 297 times more than CryptoMaze.

5.5.2.2 Evaluation on Simulated Instances

Payment Channel Networks follow a small-world, scale-free structure [121]. For generating synthetic graphs of size ranging from 200 to 25600 based on Barábasi-Albert model [22], [34], library *igraph* was used. Optimization in terms of off-chain contracts are not analyzed since these are synthetic graphs. The topology of the synthetic graph may not be able to mimic the execution of multiple payment instances in the Lightning Network. We make the following observations based on executing a single payment instance:

- TTP for CryptoMaze increases gradually with the increase in the size of the network. The execution time does not exceed 11s upon execution on an instance of size 25600. Run time of *AMP* is 1.7 times of CryptoMaze, that of *Eckey et al.* and *Multi-Hop HTLC* being 3.5 times and 18 times that of our protocol on an average. The plot is given in Fig. 5-10(a).
- The communication overhead in Fig. 5-10 (b) increases with an increase in the size of the network, with the communication overhead not exceeding 1000KB or 1MB on an instance of size 25600. On average, the communication overhead of CryptoMaze is 5 times of *Eckey et al.* and 33 times of *Atomic Multi-path Payment*. However, the overhead is 105 times less compared to *Multi-Hop HTLC*.

5.5.3 Discussion

(i) *Optimization in terms of off-chain contracts*: When the transaction amount per payment was increased, the liquidity of channels decreased. Payments were split into smaller amounts and routed via multiple paths. Thus, we observed that the number of instances where the routes were not edge-disjoint increased. With the increase in transaction amount, the number of paths routing a payment increased due to the increase in the split. The number of off-chain contracts established per payment increased for the state-of-the-art protocols. When the size of the network increases, the higher the chance of finding routes with higher capacity, the more options of edge-disjoint routes. Hence, a decrease in the number of off-chain contracts is observed.

CryptoMaze combines the conditions for each of the partial payments routed via shared edges and form a single off-chain contract, our protocol saves around $50\% - 60\%$ compared to state-of-the-art in terms of setup cost. Also, it does not have to pay a node more than once for routing payment, thus saving on the processing fee.

(ii) *Efficiency in terms of computation and communication cost*: We discuss our observation in terms of the metric used.

- Time taken to execute CryptoMaze is comparable to *AMP*, sometimes even lower than the latter. The reason is the mapping set of routes into a set of edges before establishing the off-chain contracts. All the previous protocols considered each route individually, increasing the setup time. *Eckey et al.* have a higher run time due to the use of homomorphic encryption. In *Multi-*

Hop HTLC, generating zero-knowledge proofs for the preimage of a given hash value is an expensive process in terms of computation cost.

Overall, the time taken to execute the payment protocol increases slightly with an increase in the transaction amount and an increase in the network size. The higher is the transaction amount, the higher the chance of the payment being split into multiple partial payments. When the network size increases, the time taken to process the network for searching paths for routing the payment increases as well.

- It is observed that *AMP* has the lowest communication overhead because each node forwards just a single commitment to its neighbor in the path routing payment. However, each path is susceptible to wormhole attack. *Eckey et al.* have a higher communication overhead. Here, each node forwards the public key and an encrypted message to its neighbor. In CryptoMaze, each node forwards a set of conditions and a set of secret values to its neighbor. The communication overhead is slightly greater than *Eckey et al.*. However, the surge in communication overhead is to some extent compensated in the shared channels, where a single off-chain contract instead of multiple off-chain contracts. *Multi-Hop HTLC* has the highest communication cost. The zero-knowledge proof Π forwarded to each node has a significant size, plus multiple off-chain contracts are formed on shared edges, increasing the communication overhead.

The result demonstrates that our proposed protocol is efficient and scalable in terms of computation cost and resource utilization.

	AMP [10]	[111]	[32]	NAPS [51]	[53]	This work
At	✓	✗	✓	✗	✓	✓
WA	✓	✓	✓	✗	✗	✗
Li	✗	✗	✗	✓	✓	✗
M-OC	✓	✓	✓	✓	✓	✗

Table 5.2: Comparative Analysis of CryptoMaze with existing Multi-path payment protocols in terms of atomicity (At), wormhole attack (WA), Linkability (Li) and multiple off-chain contracts on shared edges (M-OC)

We provide a comparative analysis of our protocol with the state-of-the-art multi-path payment in Table 5.2. Our protocol is atomic, wormhole attack resistant and guarantees unlinkability between partial payments. None of the shared edges require multiple off-chain contracts for a single payment instance. A new protocol, xLumi [143] was proposed for blockchain systems. This protocol creates unidirectional channels. Unlike Lightning Network, xLumi drastically reduces the number of interactions and complexity of opening a payment channel. Users are not required to store a new secret for every off-chain transaction. However, xLumi has not been expanded to bidirectional channels and payment channel networks. It would be interesting to see how CryptoMaze can be adapted in xLumi based PCN.

Chapter 6

Griefing-penalty: Countermeasure for Griefing Attack in Lightning Network

Lightning Network can execute an unlimited number of off-chain payments, without incurring the cost of recording each of them in the blockchain. If a sender and a recipient of payment do not share a channel, such payments make use of Hashed Timelock Contracts or HTLCs [113]. Since the payment gets routed via several intermediaries, a specific condition is imposed via off-chain contracts to prevent cheating. Payments are contingent on the fulfillment of this condition. We describe with an example how conditional payments get executed between parties not directly connected by payment channels in Lightning Network. Suppose *Alice* wants to transfer p coins to *Bob* via path comprising payment channels *Alice-Dave, Dave-Charlie* and *Charlie-Bob*, as shown in Fig. 6-1. Each intermediate node charge a processing fee of p'. *Alice* forwards a conditional payment to *Dave*, forming an off-chain contract, denoted as *Contract*$(p + 2p', t + 2\Delta)$, locking $p + 2p'$ coins for a time period $t + 2\Delta$. Here Δ is the worst-case confirmation time for settling a transaction on-chain. *Dave* deducts p' coins from the amount and forwards the payment to *Charlie* by forming a off-chain contract, locking $p + p'$ coins for $t + \Delta$. Finally, *Charlie* deducts p' coins from the payment amount and locks p coins with *Bob* for a time period t. In order to claim p coins from *Charlie*, *Bob* must resolve the payment within time period t. If the period elapses, *Charlie* goes on-chain to claim a refund, closes the channel *Charlie-Bob* and unlocks the money from the contract. Using the information released by *Bob*, the rest of the intermediaries resolve the payment as well, each claiming a processing fee of p'.

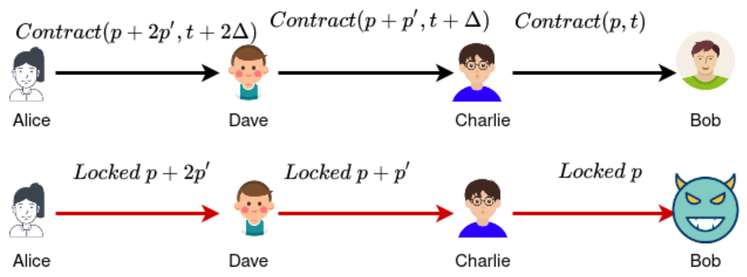


Figure 6-1: *Bob* mounts Griefing Attack

Suppose *Bob* stops responding. *Charlie* can go on-chain and withdraw the coins locked in the contract only after the elapse of the contract's lock time. Thus *Bob* manages to lock $\mathcal{O}(p)$ coins in each of the preceding payment channels for a period of t units, without investing any coin. t could be of the order of *24 hours*. In that case, none of the parties can utilize the amount locked in their respective off-chain contracts for an entire day. Thus, *Bob* has mounted a *griefing attack*, illustrated in Fig. 6-1. Griefing attack was first mentioned in [119]. Paralyzing the network for multiple days by overloading each channel with maximum unresolved HTLCs has been studied in [100], [129]. In [89], Sybil nodes initiate several payments via multiple paths and griefs them simultaneously.

Motive behind Griefing Attack By mounting a griefing attack, an adversary may try to achieve either of the objectives:

- Stalling network using self-payment: The adversary controls the sender and receiver of several payment requests, blocking multiple intermediaries from accepting any other payments to be routed through it [89], [8]. To decrease the network throughput, an adversary may set up several *Sybil nodes* at strategic positions across the PCN and amplify the damage by submitting several payment requests.

- Eliminating a competitor from the network: The adversary tries to eliminate a competitor and block all its existing channel's outgoing capacity [55], [8]. The adversary sets the victim as an intermediate node in the path carrying out the self-payment. The transaction value of self-payment is equivalent to the victim's outgoing channel capacity, jamming all the channels of the victim node. The victim cannot utilize the fund until the adversary claims the payment. As a result, the former cannot process several payment requests could due to a lack of channel liquidity. Adversary reaps the indirect economic benefit by claiming the processing fee for routing such transactions. For example, B has outgoing channel with A and C, each of capacity *0.1 BTC* (each party having a balance of *0.05 BTC*), shown in Fig. 6-2. Node D has channel with *A* and *C*, each of capacity *0.2 BTC*. It conducts self-payment of *0.05 BTC*, in each direction. By griefing for *24 hrs*, D has managed to block B from accepting any transaction request. A and C, having residual outgoing capacity of *0.1 BTC* each in channel *AD* and *CD*, are now forced to route all the payments via D.

- Stalling network using intermediary: The adversary controls a node with a high degree of centrality and broadcasts its processing fee to be extremely low to ensure multiple payments get routed through such nodes [121]. It later ignores all the payments by not forwarding the message to outgoing neighbors. Funds remain locked across multiple paths, affecting a large portion

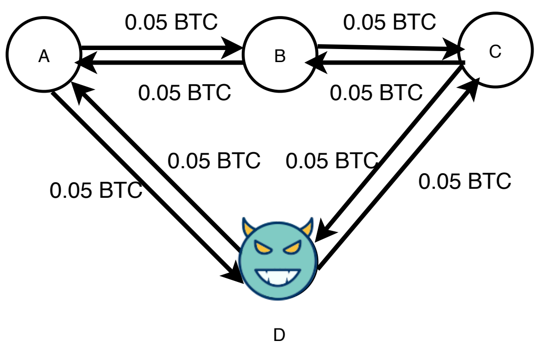


Figure 6-2: Eliminating a competitor

of the network.

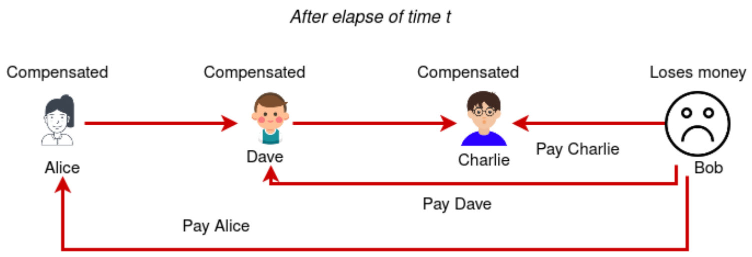


Figure 6-3: *Bob* is penalized

6.1 Our Goal

Griefing attack in Lightning Network cannot be prevented as long as a malicious node has nothing to lose or there is *nothing at stake*. If the attacker is penalized, it will be discouraged from mounting the attack. The amount deducted from the adversary's balance must be able to compensate all the parties which got affected by the attack. A high-level idea of the countermeasure is represented in Fig. 6-3. *Alice* forwards the payment to *Bob* via some intermediaries. Each party accepting the off-chain contract is supposed to lock an amount, which gets deducted if the party fails to resolve the payment before the contract timeout period. *Bob* doesn't respond intentionally, allowing the timeout period of the contract to elapse. As per the terms of the contract, he gets penalized and the funds slashed from his account are used to compensate *Alice, Dave* and *Charlie*.

6.2 Our Contributions

In this paper, we have made the following contributions:

- We propose a countermeasure for mitigating griefing attack in Lightning Network, known as *Griefing-Penalty*. To illustrate the benefit of the proposed countermeasure, we propose a new payment protocol, called *HTLC-GP* or Hashed Timelock Contract with Griefing-Penalty. The penalty deducted is a fraction of the coins locked by the attacker per unit of time. This fraction is termed as *rate of griefing penalty*.

- We propose a construction for secure multihop payment using HTLC-GP and provide security analysis. It proves that our protocol is privacy-preserving and mitigates loss due to griefing attacks by compensating the honest nodes.

- We study two attacking strategies for eliminating competitor nodes from the network. Upon mounting the griefing attack following either of the strategies, we compare the profit made by the attacker in *HTLC* and *HTLC-GP*, by executing the protocols on several snapshots of Lightning Network. The profit is termed as *Return on Investment (RoI)*. It is observed that RoI is negative for *HTLC-GP* compared to a positive RoI in *HTLC*, hence disincentivizing a node from mounting griefing attack due to substantial loss incurred.

- We compare the investment made by the adversary for mounting the attack in both *HTLC-GP* and *HTLC* upon varying *path length* and *rate of griefing-penalty*. The budget needed for mounting a griefing attack in *HTLC-GP* is 4 times more than the budget needed for HTLC when the path length is set to 4. The ratio increases to 12 for a path length of 20, the maximum hop count allowed in Lightning Network. For a fixed path length, the ratio increases up to 500 for the rate of griefing penalty exceeding 10^{-3}.

- We suggest a suitable range for selecting the rate of griefing-penalty, from the perspective of an honest payer and an honest payee. We observe that selecting the rate of griefing penalty in the range $(10^{-5}, 10^{-3}]$ ensures that the cumulative penalty to be locked by the recipient of a given payment is neither too high nor too low. At the same time, the collateral locked upon mounting a griefing attack in HTLC-GP is lower than that in HTLC for a given attacker's budget.

6.2.1 Organization

We discuss the necessary background by defining payment channel networks in Section 6.3. We provide a high-level overview of our proposed countermeasure, *Griefing-Penalty*, in Section 6.4. Based on this idea, we have proposed a new payment protocol, *HTLC-GP* or *Hashed Timelock Contract with Griefing-Penalty* in Section 6.5. We provide a detailed construction of Multi-hop payment using HTLC-GP in Section 6.6. Security analysis of the proposed multi-hop payment protocol has been provided in Section 6.7. We divide the *Performance Evaluation* into two parts in Section 6.8. Firstly, we analyze the profit earned by eliminating competitors in Section 6.8.1, demonstrating the efficiency of *HTLC-GP* over *HTLC* in countering the griefing attack. Next, we discuss in Section 6.8.2 the impact of certain parameters on the investment made by an attacker in HTLC-GP. In Section 6.9, we suggest bounds on the amount of griefing-penalty charged that works for practical purposes.

6.3 Background

6.3.1 Payment Channel Network

A Payment Channel Network or PCN is defined as a bidirected graph $G := (V, E)$, where V is the set of nodes and E is the set of payment channels opened between a pair of nodes. 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, the final balance is credited to each user's wallet as per the recent state of the payment channel. Every node $v \in V$ charges a processing fee $f(val)$, for relaying val coins to its neighbor with which v shares a payment channel. Correctness of payment across each channel is enforced cryptographically by hash-based scripts [113] or scriptless locking [94]. Each payment channel (v_i, v_j) has an associated capacity $locked(v_i, v_j)$, denoting the amount locked by v_i and $locked(v_j, v_i)$ 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 sender S, which is node v_0, wants to transfer amount α to R, which is node v_n through a path $v_0 \rightarrow v_1 \rightarrow v_2 \ldots \rightarrow v_n$. If $remain(v_i, v_{i+1}) \geq \alpha_i : \alpha_i = \alpha_{i+1} + f(\alpha_{i+1}), i \in [0, n-1], \alpha_{n-1} = \alpha$, then funds can be relayed across the channel (v_i, v_{i+1}). The residual capacity is updated as follows : $remain(v_i, v_{i+1}) = remain(v_i, v_{i+1}) - \alpha_i$ and $remain(v_{i+1}, v_i) = remain(v_{i+1}, v_i) + \alpha_i$. Each processing node v_i charges a processing fee $f(\alpha_i), i \in [1, n-1]$. *Lightning Network* (LN) [113] is the most widely accepted Bitcoin-compatible PCN.

6.4 Key Idea of Griefing-Penalty

Designing fair protocols on Bitcoin, where an adversary is forced to pay a mutually predefined monetary penalty to compensate for the loss of honest parties was first introduced by Bentov et al. [36]. Inspired by this idea, we propose a countermeasure for griefing attack, termed *Griefing-Penalty*. The griefing penalty imposed on an adversary for mounting a griefing attack on a path of length, n is proportional to the summation of collateral cost of each payment channel involved in routing. *Collateral cost per payment channel* is defined as the product of the amount locked in the off-chain contract and the expiration time of the contract. The amount deducted per unit time from the adversary's balance is a fraction of the collateral locked. This fraction is termed as *rate of griefing penalty* or γ. We account for the expiration time of the contract while calculating the griefing penalty to compensate participants for the lost opportunity cost. In the next section, we discuss how to incorporate griefing-penalty into the existing payment protocol, Hashed Timelock Contract or *HTLC*. Note that the use of *Griefing-Penalty* is independent of the cryptographic primitive used for the underlying payment protocol.

6.4.1 A Simple Protocol for countering Griefing Attack: *HTLC1.0*

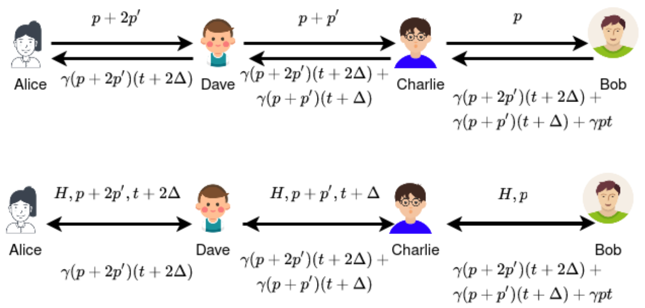

$p + 2p'$
$p + p'$
p
Alice
$\gamma(p + 2p')(t + 2\Delta)$
Dave
$\gamma(p + 2p')(t + 2\Delta) +$
$\gamma(p + p')(t + \Delta)$
Charlie
Bob
$\gamma(p + 2p')(t + 2\Delta) +$
$\gamma(p + p')(t + \Delta) + \gamma pt$
$H, p + 2p', t + 2\Delta$
$H, p + p', t + \Delta$
H, p
Alice
Dave
Charlie
Bob
$\gamma(p + 2p')(t + 2\Delta)$
$\gamma(p + 2p')(t + 2\Delta) +$
$\gamma(p + p')(t + \Delta)$
$\gamma(p + 2p')(t + 2\Delta) +$
$\gamma(p + p')(t + \Delta) + \gamma pt$

Figure 6-4: Formation of contract in *HTLC1.0*

We incorporate *Griefing-Penalty* into *HTLC* [113]. Let us rename the modified payment protocol as *HTLC1.0*. A payment instance using *HTLC1.0* for multihop payment is shown in Fig.6-4. *Alice* wants to transfer p coins to *Bob*. *Bob* shares the hash H with *Alice* offline. This is used as the condition for the off-chain contracts established in the route forwarding the payment. Given the rate of griefing-penalty as γ per unit of time, $0 \leq \gamma < 1$, and locktime of the contract being $(t + 2\Delta)$, *Dave* is expected to lock $\gamma(p + 2p')(t + 2\Delta)$ coins as griefing-penalty in the off-chain contract, $(p + 2p')(t + 2\Delta)$ being the collateral cost in channel *Alice-Dave*. If *Dave* provides the preimage of H within this period, he will claim $p + 2p'$ coins and withdraw $\gamma(p + 2p')(t + 2\Delta)$ coins locked in the contract. *Dave* forwards a conditional payment of $p + p'$ coins to *Charlie* by forming similar off-chain contract using payment hash H and locktime $(t + \Delta)$. Upon griefing, *Charlie* must pay a compensation of $\gamma(p + p')(t + \Delta)$. However, this amount is not sufficient to compensate both *Dave* and *Alice*. Hence he has to lock a cumulative griefing-penalty $\gamma(p + 2p')(t + 2\Delta) + \gamma(p + p')(t + \Delta)$ in the contract. This cumulative griefing-penalty is the summation of collateral cost in channel *Alice-Dave* and *Dave-Charlie*. *Charlie* forwards a conditional payment of p coins to *Bob* by forming an off-chain contract for locktime of t units. *Bob* has to lock $\gamma(p+2p')(t+2\Delta)+\gamma(p+p')(t+\Delta)+\gamma pt$ coins. This amount is the cumulative penalty to be distributed among *Alice, Dave* and *Charlie*, if Bob griefs.

Suppose *Bob* griefs and refuses to release the preimage of H, waiting for time t to elapse. He will pay a compensation of $\gamma(p + 2p')(t + 2\Delta) + \gamma(p + p')(t + \Delta) + \gamma pt$ coins to *Charlie*, as per the terms of the contract. After the timelock t expires, *Charlie* goes on-chain. He closes the channel, unlocks p coins and claims $\gamma(p + 2p')(t + 2\Delta) + \gamma(p + p')(t + \Delta) + \gamma pt$ coins as the compensation. He requests *Dave* to cancel the off-chain contract offering a compensation of $\gamma(p+2p')(t+2\Delta)+\gamma(p+p')(t+\Delta)$. *Dave* cancels the contract off-chain, unlocks $p+p'$ coins from the contract and claims the compensation from *Charlie*. If *Charlie* decides to grief, *Dave* can claim the compensation by going on-chain and closing the channel. *Dave* requests *Alice* to cancel the contract by offering a compensation of $\gamma(p + 2p')(t + 2\Delta)$. Thus except *Bob*, none of the parties lose funds in order to compensate any of the affected parties.

6.4.2 Problem of Reverse-Griefing in HTLC1.0

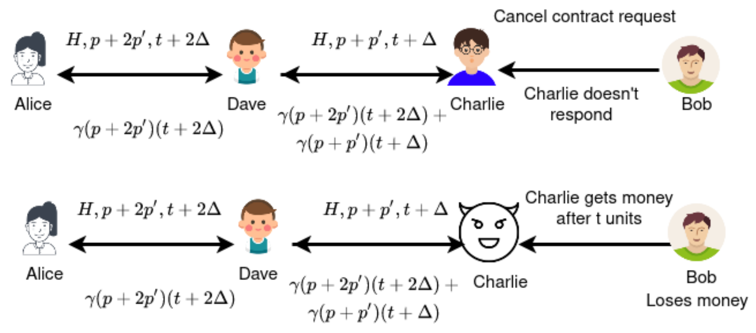


Figure 6-5: Reverse-Griefing attack by *Charlie*

A drawback of the protocol is that with the introduction of the griefing penalty, a malicious party can now ascribe the blame of griefing to an honest party as well. In the previous example, *Bob* cancels the payment under following conditions - if *Alice* uses a wrong hash value or the HTLC timeout period is closer to the current block height of the blockchain or the value of the transaction forwarded is less than the agreed value [2], then. However, *Charlie* can deny settling the contract off-chain. *Bob* has no way to prove his innocence. Ultimately, with elapse of lock time, *Charlie* goes on-chain, claiming *Bob's* coins, as shown in Fig. 6-5. This attack is termed *Reverse Griefing*. There exist several nodes in the network that earns either low or zero processing fees. Either they charge a very negligible amount of fee for large valued transactions [37], or they remain inactive for most of their lifetime in the network. Such nodes have a higher tendency to deviate as the profit earned by reverse-griefing is higher than the total anticipated processing fee.

6.5 Our Proposed Protocol using Griefing Penalty

It is observed in *HTLC1.0* that establishing a single contract with minimal changes to the script is not sufficient to protect a party from being cheated. We propose a new payment protocol for Lightning Network, termed as Hashed Timelock Contract with Griefing-Penalty or *HTLC-GP* to avoid the problem of reverse-griefing.

In this protocol, the locking of the penalty and the payment amount are executed in separate rounds. Instead of both parties locking their funds into a single contract, the payer locks funds in one contract, the *Payment Contract*, and the payee locks his penalty in a separate contract, the *Cancellation Contract*. The two contracts are bound together using two different hashes, termed as *Payment Hash* and *Cancellation Hash*. The payee can unlock the penalty deposited in *Cancellation Contract* either by providing the preimage to the first hash, i.e., the payment hash, or by releasing the preimage to the second hash, i.e., the cancellation hash. We change the order with the payee initiating the first round of the protocol. The former locks the penalty into the *Cancellation Contract* and forwards it to the payer. After the payer receives the contract, it initiates the second round. The payer locks the fund into the *Payment Contract* and forwards it to the payee. Since the payee has preimages corresponding to both the hashes, even if the payer denies forming the payment contract, it cannot mount a reverse-griefing attack on the payee. After waiting for a short duration, the payee will cancel the contract by releasing the cancellation hash.

6.5.1 Two party HTLC-GP

Alice has a payment channel with Bob where each party had deposits 5 msat in the channel. She intends to transfer 1 msat to Bob. The rate of griefing penalty is set to 0.001 per minute. Bob samples r and x, creates the cancellation hash $Y = \mathcal{H}(r)$ and payment hash $H = \mathcal{H}(x)$, and forwards both the hashes to Alice. The timeout period for the contract is set as 3 days or 72 hrs, and hence the penalty amount is $0.001 * 72 * 60 * 1 = 4.32$ msat. The condition for payment is as follows: *Given $H = \mathcal{H}(x)$ and $Y = \mathcal{H}(r)$ in the contract, Bob can claim a fund of 1 msat from Alice contingent on the knowledge of x, within 3 days and unlocks 4.32 msat from the contract. If Bob fails to do so, then after a timeout of 3 days, it pays a penalty of 4.32 msat to Alice and at the same time, Alice withdraws 1 msat. If Bob desires to cancel the contract, then it releases the preimage r. In that case, Alice withdraws 1 msat, and Bob withdraws 4.32 msat locked in the contract.* Initially, Bob locks the penalty by establishing an off-chain contract with Alice. After accepting the contract from Bob, Alice locks the payment value in an off-chain contract with Bob, as shown in Fig. 6-6.

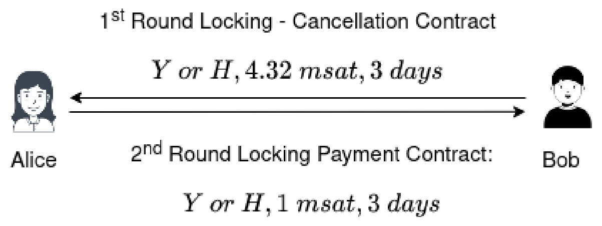

1^{st} Round Locking - Cancellation Contract
$Y\ or\ H, 4.32\ msat, 3\ days$
Alice
2^{nd} Round Locking Payment Contract:
Bob
$Y\ or\ H, 1\ msat, 3\ days$

Figure 6-6: Payment from Alice to Bob using HTLC-GP

In the first round of locking, Bob forwards the *cancellation contract* to Alice by locking 4.32 msat. Bob can withdraw the entire amount contingent on the knowledge of the preimage corresponding to the cancellation hash Y or payment hash H. If Bob fails to respond, Alice claims the entire amount after 3 days. The state of the channel is: Alice has a balance of 5 msat, Bob has a balance of 0.68 msat, coins locked in *HTLC-GP* is 4.32 msat. In the second round of Locking, Alice forwards the *payment contract* to Bob by locking 1 msat. The state of the channel is: Alice has a balance of 4 msat, Bob has a balance of 0.68 msat, coins locked in first *HTLC-GP* is 4.32 msat, and coins locked in second *HTLC-GP* is 1 msat. Bob can claim the coins from both contracts contingent on the knowledge of preimage corresponding to the payment hash H. If Bob reveals the preimage corresponding to cancellation hash Y, Alice withdraws 1 msat and Bob withdraws 4.32 msat from the contract. If Bob doesn't respond before the lock time expires, Alice claims the coins that are locked in both the contracts.

HTLC-GP Script

The structure of the script is as per the convention used in [1]. For the implementation of HTLC-GP in Lightning Network, we discuss how to design the output scripts for *Cancellation Contract* and *Payment Contract*.

Figure 6-7: Script Structure: Offered Cancellation Contract

```
OP_DUP OP_HASH160 ⟨ RIPEMD160 ( SHA256 ( revocationpubkey ))⟩
OP_EQUAL
OP_IF
  OP_CHECKSIG
OP_ELSE
  ⟨ remote_htlcgppubkey⟩ OP_SWAP OP_SIZE 32 OP_EQUAL
  OP_NOTIF
    OP_IF
      OP_HASH160 ⟨ RIPEMD160 ( payment_hash )⟩ OP_EQUALVERIFY
      2 OP_SWAP ⟨ local_htlcgppubkey ⟩ 2 OP_CHECKMULTISIG
    OP_ELSE
      OP_HASH160 ⟨ RIPEMD160 ( cancellation_hash) ⟩
OP_EQUALVERIFY
      2 OP_SWAP ⟨ local_htlcgppubkey ⟩ 2 OP_CHECKMULTISIG
    OP_ENDIF
  OP_ELSE
    OP_DROP ⟨ cltv_expiry ⟩ OP_CHECKLOCKTIMEVERIFY OP_DROP
    OP_CHECKSIG
  OP_ENDIF
OP_ENDIF
```

HTLC-GP Offered Cancellation Contract: Bob offers this script to Alice. This output sends funds to either the remote node after the HTLC-GP timeout or using the revocation key, or to an HTLC-GP -success transaction either with a successful payment preimage or cancellation preimage. The output is a P2WSH, with a witness script:

- Release the funds if the script is signed by the revocation key (*revocationpubkey*).
- If the above condition fails, then check if the HTLC-GP public key of the party not publishing the commitment (remote public key), i.e., of Alice, has been provided. Now check which of the condition holds:

- The publisher of the commitment, i.e., Bob, can publish the HTLC-GP-success by using the notif clause. It ignores the condition if the remote public key is not provided. Bob will earn coins by publishing the contract only if either of the condition is satisfied:
  - Bob can use the preimage of *cancellation hash*. Release the funds if the preimage is released and signed by both Alice and Bob,
  - Bob can use the preimage of *payment hash*. Release the funds if the preimage is released and signed by both Alice and Bob.
- If Bob didn't react, Alice can publish the HTLC-GP-timeout transaction.

The Bitcoin script structure is shown in Fig.6-7.

Figure 6-8: Script Structure: Offered Payment Contract

```
OP_DUP OP_HASH160 ⟨ RIPEMD160 ( SHA256 ( revocationpubkey ))⟩
OP_EQUAL
OP_IF
   OP_CHECKSIG
OP_ELSE
   ⟨ remote_htlcgppubkey⟩ OP_SWAP OP_SIZE 32 OP_EQUAL
   OP_NOTIF
      OP_DROP 2 OP_SWAP ⟨ local_htlcgppubkey ⟩ 2
OP_CHECKMULTISIG
   OP_ELSE
      OP_IF
         OP_HASH160 ⟨ RIPEMD160 ( cancellation_hash) ⟩
OP_EQUALVERIFY
         OP_CHECKSIG
      OP_ELSE
         OP_HASH160 ⟨ RIPEMD160 ( payment_hash )⟩ OP_EQUALVERIFY
         OP_CHECKSIG
      OP_ENDIF
   OP_ENDIF
OP_ENDIF
```

HTLC-GP Offered Payment Contract: Alice offers this script to Bob. This output sends funds to either an HTLC-timeout transaction after the HTLC-timeout or to the remote node using either the payment preimage or cancellation image or the revocation key. The output is a P2WSH, with a witness script:

- Release the funds if the script is signed by the revocation key (*revocationpubkey*).
- If the above condition fails, then check if the HTLC-GP public key of the party not publishing the commitment (remote public key), i.e., of Bob, was provided. Now check which of the condition holds:
  - The publisher of the commitment, i.e., Alice, can publish the HTLC-GP-timeout by using the notif clause.
  - Else, Bob can publish HTLC-GP success if any of the conditions holds:
    - Bob can use the preimage of *cancellation hash*. Release the funds if the preimage is released and signed by both Alice and Bob,
    - Bob can use the preimage of *payment hash*. Release the funds if the preimage is released and signed by both Alice and Bob.

The Bitcoin script structure is shown in Fig.6-8. In the next section, we provide an instantiation of multihop payment using *HTLC-GP*.

6.6 Multihop Payment using HTLC-GP

6.6.1 System Model

All the participants know the topology of the Lightning Network. Pairs of honest users, sharing a payment channel, communicate through secure and authenticated channels. An honest party willing to send funds to another party will adhere to the protocol. It will not change the terms of the off-chain contract and disrupt the protocol. We assume the communication model to be synchronous, with all the parties following a global clock for settling payments off-chain.

6.6.2 Objective

- **Guaranteed compensation for an honest node**: All the honest parties affected by the griefing attack get compensated by the griefer. Except for the griefer, none of them must lose funds to pay compensation to any affected parties.
- **Payer and Payee's Privacy**: None of the intermediate nodes involved in routing a payment must be able to identify its exact position in the path as well as figure out the identities of the sender and receiver of the payment.

6.6.3 Adversarial Model & Assumptions

An adversary introduces multiple Sybil nodes and places them strategically in the network to maximize the collateral damage. Such Sybil nodes may be involved in self-payment or transfer funds from one Sybil node to the other for mounting the griefing attack. An intermediate node routing payment may be a Sybil node. The adversary can perform the following arbitrary actions to keep funds locked in the network for a substantial amount of time: (i) it withholds the solution without resolving the incoming off-chain payment request, (ii) it may refrain from forwarding the off-chain payment request to the next neighbor, (iii) it just refuses to sign any incoming off-chain contract request.

We assume that at least one node in the path will be honest. So in the worst case, except for one node (either sender or receiver or any intermediate party), the rest all the parties may be corrupt and controlled by the adversary. We also assume that an honest party cannot be denied going on-chain by the adversary during the protocol. Using untrusted/semi-trusted third-party service provider, *WatchTowers*, prevents such attackers from mounting *time dilation attacks* [118] and censoring transactions. Recently, [88] have proposed a method for censoring attacks where shorter deadlines are used in off-chain contracts for the good case but the deadline can be extended, if congestion occurs. The countermeasure would prevent the attacker from censoring the HTLC-GP success transaction and mounting reverse-griefing.

6.6.4 Our proposed Construction

For secure transfers of funds from sender U_0 to the receiver, the former selects an optimal route in the Lightning network for transferring funds to the payee. Since the path length n, we index the receiver as U_n. Let the path be $P = \langle U_0, U_1, \ldots, U_n \rangle$, via which payer U_0 will relay fund of value α to payee U_n, each U_i is a node in the graph and $(U_i, U_{i+1}), i \in [0, n-1]$ denotes a payment channel. Each party $U_i, i \in [1, n-1]$ charge a service fee of $f(\alpha_i)$ for transferring α_i to U_{i+1} and $\alpha_{n-1} = \alpha$. Hence the total amount that U_0 needs to transfer is $\tilde{\alpha} = \alpha + \Sigma_{i=1}^{n-1} f(\alpha_i)$. We denote each $\alpha_i = \tilde{\alpha} - \Sigma_{j=1}^{i} f(\alpha_j), i \in [1, n-1], \alpha_0 = \tilde{\alpha}$. Each node U_i samples pair of secret key and public key (sk_i, pk_i), the public key of each node is used to encrypt the information of establishing contract with the neighbouring node.

Parameters used

- **Rate of Griefing-penalty** γ: It decides the amount to be deducted per unit of time as compensation from the balance of a node responsible for griefing. It is set as a system parameter with $0 \leq \gamma \leq 1$, measured in terms of per minute.

- **Routing Attempt Cost** ψ: U_0 has to figure out the path by probing channels that will be able to route the transaction. Finding a route has an extra computational and resource overhead on U_0. Thus U_0 adds the cost of routing attempt to the compensation withdrawn from the griefer. This is a variable quantity and the quantity is kept hidden from other nodes but generally, a sender sets the value $\psi t_0 \geq \alpha((k+1)t_0 + \sum_{l=1}^{k} l\Delta), k \in \mathbb{N}$ and preferably $k > 3$. Here k is the masking factor, t_0 is the period of the contract established between U_0 and U_1, and Δ is the time taken for a transaction to settle on-chain.

Computing Griefing-Penalty

U_0 shares $\phi(n)$ with U_n, where ϕ is a function used for blinding the exact value of n, $\phi(n).\alpha t_{n-1} \approx ((\psi + \alpha_0)t_0 + \Sigma_{j=1}^{n-1}\alpha_j t_j)$, adding the extra cost for routing to the compensation it must claim from U_1. Similar to *HTLC*, t_{n-1} is the least timeout period, assigned to the off-chain contract between U_{n-1} and U_n where $t_{n-1} > \Delta$. For rest of the off-chain contracts established between U_i and $U_{i+1}, i \in [0, n-2]$, the timeperiod of the contract $t_i > t_{i+1} + \Delta$. Adding the routing attempt costs to the compensation disallows U_1 from inferring the identity of the sender of a particular payment. It cannot distinguish whether ψ is routing attempt fee or the cumulative flow from any predecessors of U_0. Even other nodes must not be able to figure out their position in the path with the information of cumulative griefing-penalty.

The maximum compensation earned by $U_i, i \in [1, n-1]$ is $\gamma.\alpha_i t_i$, where α_i is the amount to be transferred to U_{i+1}, if the latter resolves the contract within t_i. If U_{i+1} does not respond, then it has to pay compensation to all the parties which got affected, starting from U_i till U_0. Hence compensation charged by each channel $(U_k, U_{k+1}), k \in [0, i]$, must be withdrawn from the faulty node U_{i+1}. The total griefing-penalty to be paid is $\gamma.(\Sigma_{j=1}^{i}(\alpha_j t_j) + (\alpha_0 + \psi)t_0)$, so that each party $U_m, m \in [1, i]$, gets a compensation of $\gamma.\alpha_m t_m$ and U_0 gets a compensation of $\gamma(\psi + \tilde{\alpha})t_0$.

6.6.4.1 Protocol Description

Our protocol involves the following three phases:

`Pre-processing Phase`

- U_n samples the preimages x and r, $x \neq r$ and constructs the two hashes: $H = \mathcal{H}(x)$ and $Y = \mathcal{H}(r)$.
- It shares H, Y with the payer, U_0. The payer uses standard onion routing [64] for propagating the information needed by each node $U_i, i \in [1, n]$, across the path P.
- The cumulative griefing-penalty for node U_0 is defined as $\text{tgp}_0 = \gamma(\psi + \tilde{\alpha})t_0$ and for any node $U_i, i \in [1, n-1]$ as $\text{tgp}_i = \gamma.(\Sigma_{j=1}^{i}(\alpha_j t_j) + (\alpha_0 + \psi)t_0)$.
- U_0 sends $M_0 = E(\ldots E(E(E(\phi, Z_n, pk_n), Z_{n-1}, pk_{n-1}), Z_{n-2}, pk_{n-2})\ldots, Z_1, pk_1)$ to U_1, where $Z_i = (H, Y, \alpha_i, t_{i-1}, \text{tgp}_{i-1}, U_{i+1}), i \in [1, n-1]$ and $Z_n = (H, Y, \alpha_{n-1}, t_{n-1}, \text{tgp}_{n-1}, null)$. Here $M_{i-1} = E(M_i, Z_i, pk_i)$ is the encryption of the message M_i and Z_i using public key pk_i, $M_n = \phi$.
- U_1 decrypts M_0, gets Z_1 and M_1. $M_1 = E(\ldots E(E(E(\phi, Z_n, pk_n), Z_{n-1}, pk_{n-1}), Z_{n-2}, pk_{n-2}), \ldots, Z_2, pk_2)$ is forwarded to the next destination U_2. This continues till party U_n gets $E(\phi, Z_n, pk_n)$.

`Two-Round Locking Phase` It involves two rounds: *establishing Cancellation Contract* and *establishing Payment Contract*.

- *Establishing Cancellation Contract*: Since the flow of griefing-penalty is in the opposite direction of the actual payment, it is logical for U_n to initiate this round.
  - U_n decrypts to get Z_n. It checks $\gamma\phi(n)t_{n-1} \overset{?}{\approx} tgp_{n-1}$ and $\alpha_{n-1} \overset{?}{=} \alpha$. If this holds true, it forms a contract with U_{n-1}, locking tgp_{n-1}.
  - For the rest of the parties, $U_i, i \in [1, n-1]$ first checks $tgp_i - \gamma\alpha_i t_i \overset{?}{=} tgp_{i-1}$ and then forms the off-chain contract with U_{i-1}, locking tgp_{i-1}.
  - The terms of the contract is defined as follows: 'U_{i+1} *can withdraw the amount* $tgp_i = \gamma.(\Sigma_{j=1}^{i}(\alpha_j t_j) + (\alpha_0 + \psi)t_0)$ *from the contract provided it reveals either* $x : H = \mathcal{H}(x)$ *or* $r : Y = \mathcal{H}(r)$ *within a period of* t_i *else* U_i *claims this amount as griefing-penalty after the elapse of the locktime.*'.

  Bad Case: If U_{i-1} denies signing the cancellation contract then U_i will abort. Since U_n locks a substantial amount as griefing-penalty, it will wait for a bounded amount of time for confirmation. We denote this time as $\delta : \delta \leq t_{n-1}$. If U_{n-1} stops responding after establishment of the cancellation contract, U_n releases on-chain the preimage r corresponding to cancellation hash after waiting for δ units of time and unlock the penalty tgp_{n-1} from the contract. The preimage r is now used by other parties $U_j, j \in [i+1, n]$ to cancel their respective off-chain contracts with U_{j-1}. So even if U_i aborts, U_{i+1} can go on-chain, close the channel and withdraw the amount locked in the contract.

The pseudocode of the first round of Locking Phase for U_n, any intermediate party $U_i, i \in [1, n-1]$ and payer U_0 is stated in Procedure 10, Procedure 11 and Procedure 12 respectively.

Procedure 10: Establishing Cancellation Contract: First Round of Locking Phase for U_n

```
Input: $(Z_n, \phi(n), \gamma, \alpha)$
$U_n$ parses $Z_n$ and gets $H', Y', \alpha', t', \text{tgp}_{n-1}$.
if $t' \geq t_{now} + \Delta$ and $\alpha' \overset{?}{=} \alpha$ and $\gamma(\phi(n)\alpha)t' \approx tgp_{n-1}$ and $H' \overset{?}{=} H$ and $Y' \overset{?}{=} Y$ and $remain(U_n, U_{n-1}) \geq tgp_{n-1}$ then
    Send Cancel_Contract_Request$(H, Y, t', \text{tgp}_{n-1}, \gamma)$ to $U_{n-1}$
    if acknowledgement received from $U_{n-1}$ then
        $remain(U_n, U_{n-1}) = remain(U_n, U_{n-1}) - \text{tgp}_{n-1}$
        establish $Cancel\_Contract(H, Y, t', \text{tgp}_{n-1})$ with $U_{n-1}$
        Record $t_n^{form} = current\_clock\_time$
    end
    else
        abort
    end
end
else
    abort.
end
```

- *Establishing Payment Contract*: U_0, upon receiving the cancellation contract, initiates the next round by establishing chain of contracts in the forward direction, till it reaches the payer U_n. This proceeds as normal *HTLC*.
  - Each node $U_i, i \in [0, n-1]$ forwards the terms of off-chain contract to U_{i+1}, locking α_i.
  - The off-chain contract is defined as follows: *'U_{i+1} can claim the amount α_i provided it reveals $x : H = \mathcal{H}(x)$ within a period of t_i. If not, then U_i withdraws the amount either contingent to the knowledge of $r : Y = \mathcal{H}(r)$ or after the elapse of locktime.'*

Bad Case: If U_{i+1} doesn't sign the payment contract, U_i aborts from the process. Similar to the first round of locking phase, if U_{n-1} doesn't form the payment contract within time δ, U_n releases the preimage r and unlocks the penalty tgp_{n-1} from the contract. U_i will not be able to

Procedure 11: Establishing Cancellation Contract: First Round of Locking Phase for U_i, $i \in [1, n-1]$

```
Input: $(H', Y', t', \text{tgp}_i, \gamma)$
$U_i$ parses $Z_i$ and gets $H, Y, \alpha_i, t_{i-1}, \text{tgp}_{i-1}$.
if $H' \overset{?}{=} H$ and $Y \overset{?}{=} Y'$ and $t' + \Delta \overset{?}{\leq} t_{i-1}$ and $tgp_i - \gamma\alpha_i t' \overset{?}{=} tgp_{i-1}$ and
$remain(U_i, U_{i+1}) \geq \alpha_i$ and $remain(U_i, U_{i-1}) \geq tgp_{i-1}$ and (current_time not close to
contract expiration time) then
    Sends acknowledgment to $U_{i+1}$ and waits for the off-chain contract to be established
    Send Cancel_Contract_Request$(H, Y, t_{i-1}, \text{tgp}_{i-1}, \gamma)$ to $U_{i-1}$
    if acknowledgement received from $U_{i-1}$ then
        $remain(U_i, U_{i-1}) = remain(U_i, U_{i-1}) - \text{tgp}_{i-1}$
        establish $Cancel\_Contract(H, Y, t_{i-1}, \text{tgp}_{i-1})$ with $U_{i-1}$
    end
    else
        abort
    end
end
else
    abort.
end
```

Procedure 12: Establishing Cancellation Contract: First Round of Locking Phase for U_0

```
Input: $(H', Y', t', \text{tgp}', \gamma)$
if $t' \overset{?}{=} t_0$ and $tgp' \overset{?}{=} tgp_0$ and $H' \overset{?}{=} H$ and $Y' \overset{?}{=} Y$ and $remain(U_0, U_1) \geq \alpha_0$ then
    Sends acknowledgment to $U_1$
    Confirm formation of penalty contract with $U_1$
    Initiate the second round, the establishment of payment contract
end
else
    abort.
end
```

reverse-grief U_{i+1} by aborting since the latter can go on-chain and withdraw the amount locked in the contract.

The pseudocode of the second round of Locking Phase for U_0 and any intermediate party $U_i, i \in [1, n-1]$ is stated in Procedure 13 and Procedure 14 respectively.

Release Phase: U_n waits for δ units of time before initiating this round. If the payment contract received from U_{n-1} is correct, it releases the preimage x or payment witness and resolves the contract off-chain. If U_{n-1} has not responded or the terms of the contract are not correct (wrong payment or penalty value, invalid lock time), U_n releases the cancellation preimage r. In case of dispute, U_n goes on-chain and releases either of the preimages for settling the contract. The process is repeated for other parties $U_i, i \in [1, n-1]$, which upon obtaining the preimage claims payment from the counterparty or withdraws funds from the contract.

If U_{i+1} griefs and refuses to release preimage to U_i, it has to pay the a griefing-penalty for affecting the nodes $U_k, 0 \leq k \leq i$, so that all the nodes obtain their due compensation. The pseudocode of the Release Phase for U_n and any intermediate party $U_i, i \in [1, n-1]$ is stated in Procedure 15 and Procedure 16 respectively.

Procedure 13: Establishing Payment Contract: Second Round of Locking Phase for U_0

```
**Input**: $(H, Y, \alpha_0, t_0)$
**if** *($U_1$ has agreed to form the contract) and (current_time not close to contract expiration time)* **then**
    $remain(U_0, U_1) = remain(U_0, U_1) - \alpha_0$
    establish $Payment\_Contract(H, Y, t_0, \alpha_0)$ with $U_1$
**end**
**else**
    abort
**end**
```

Safeguard against Reverse-Griefing. Any request for off-chain termination of the contract by a party $U_i, i \in [1, n-1]$, without providing a valid preimage, will not be accepted by U_{i-1} unless U_i is compensating it for the loss of time. If the party U_{i-1} mutually terminates the contract without the knowledge of any of the preimage before elapse of lock time, U_{i-2} may refuse to cancel the contract and wait for the contract to expire. This might lead to the problem of *reverse-griefing* where U_{i-1} loses funds. Hence to safeguard itself, a party will agree to terminate the contract off-chain either on receiving griefing-penalty or on receiving one of the preimages.

Procedure 14: Establishing Payment Contract: Second Round of Locking Phase for $U_i, i \in [1, n-1]$

```
Input: $(H, Y, \alpha_i, t_i)$
if $t_{i-1} \geq t_i + \Delta$ and $\alpha_{i-1} \overset{?}{=} \alpha_i + f(\alpha_i)$ and ($U_{i+1}$ has agreed to form the contract) and
(current_time not close to contract expiration time) then
    $remain(U_i, U_{i+1}) = remain(U_i, U_{i+1}) - \alpha_i$
    establish $Payment\_Contract(H, Y, t_i, \alpha_i)$ with $U_{i+1}$
end
else
    abort
end
```

6.7 Security Analysis

Theorem 6.1. ***(Guaranteed compensation for an honest node).*** *Given a payment request (U_0, U_n, α_0) to be transferred via path $P = \langle U_0, U_1, \ldots, U_n \rangle$, if at least one party $U_k, k \in [1, n]$ mounts griefing attack then any honest party $U_j \in P, j \in [0, k-1]$ will earn compensation, without losing any funds in the process.*

Proof: We consider the worst case in which we assume only a single node is honest in a path and the rest of the nodes act maliciously. We note that if fewer parties are corrupted, the honest nodes still interact with malicious neighbors and hence they get reduced to cases mentioned here. In particular, we analyze interactions between honest and dishonest parties in our system and ensure that honest parties do not get cheated and get their due.

- *Case 1 : U_0 is honest*

  In the `Pre-processing Phase`, the honest sender builds the onion packets containing terms of contract which are propagated through the nodes in the path. While the values in the packets are contingent on the values sent by R, honest S is in no position to verify it. At this point, S just follows the protocol.

  In the `Two Round Locking Phase`:

  – *Establishing the Cancellation Contract*: Since U_0 is the last party to receive the contract, in case any party $U_i, i \in [1, n]$ griefs, then it will end up paying a cumulative penalty of $\gamma.(\Sigma_{j=1}^{i-1}(\alpha_j t_j) + (\alpha_0 + \psi)t_0)$, whereby U_0 earns a compensation of $\gamma(\alpha_0 + \psi)t_0$.

  – *Establishing the Payment Contract*: U_0 may not be able to forward the contract to U_1 if there is discrepancy in the terms of the outgoing contract or if U_1 has stopped responding. Since

Procedure 15: Release_Phase for U_n

```
Input: Message $M$, time bound $\delta$
if $M \overset{?}{=} Payment\_Contract(H, Y, \alpha', t')$ and $current\_clock\_time - t_n^{form} \leq \delta$ then
    Parse $M$ and retrieve $(H, Y, \alpha', t')$
    if $t' \geq t_{now} + \Delta$ and $\alpha' = \alpha$ then
        $z = x$
    end
    else
        $z = r$
    end
end
else
    $z = r$
end
Release $z$ to $U_{n-1}$
if $current\_time < t_{n-1}$ then
    if $U_n$ and $U_{n-1}$ mutually agree to terminate Payment Contract and Cancellation Contract
    then
        if $z$=$x$ then
            $remain(U_n, U_{n-1}) = remain(U_n, U_{n-1}) + \alpha + \mathrm{tgp}_{n-1}$
        end
        else
            $remain(U_n, U_{n-1}) = remain(U_n, U_{n-1}) + \mathrm{tgp}_{n-1}$
            $remain(U_{n-1}, U_n) = remain(U_{n-1}, U_n) + \alpha$
        end
    end
    else
        $U_n$ goes on-chain for settlement by releasing preimage $z$.
    end
end
else
    $U_{n-1}$ goes on-chain for settlement, claims $(\alpha + \mathrm{tgp}_{n-1})$.
    $z = null$
end
Call Release_Phase($U_{n-1}, z$)
```

Procedure 16: Release_Phase $U_i, i \in [1, n-1]$

```
Input: $z$
Release $z$ to $U_{i-1}$
if $z \neq null$ and $current\_time < t_{i-1}$ then
    if $U_i$ and $U_{i-1}$ mutually agree to terminate Payment Contract and Cancellation Contract
    then
        if $z$=$x$ then
            $remain(U_i, U_{i-1}) = remain(U_i, U_{i-1}) + \alpha_{i-1} + \text{tgp}_{i-1}$
        end
        else
            $remain(U_i, U_{i-1}) = remain(U_i, U_{i-1}) + \text{tgp}_{i-1}$
            $remain(U_{i-1}, U_i) = remain(U_{i-1}, U_i) + \alpha_{i-1}$
        end
    end
    else
        $U_i$ goes on-chain for settlement by releasing preimage $z$.
    end
end
else
    $U_{i-1}$ goes on-chain for settlement after elapse of locktime $t_{i-1}$, claims $(\alpha_{i-1} + \text{tgp}_{i-1})$.
end
Call Release_Phase($U_{i-1}, z$)
```

U_n is dishonest, it will not release the preimage r for cancelling the contracts established in the first round. As per the terms of the contract, after the elapse of locktime t_{n-1}, it pays a cumulative griefing-penalty $\gamma.(\Sigma_{j=1}^{n-1}(\alpha_j t_j) + (\alpha_0 + \psi)t_0)$ to U_{n-1}. This money is used for compensating rest of the parties, starting from U_0 till U_{n-1}. Even if U_1 griefs, as per the terms of the contract it has to pay the required compensation $\gamma(\alpha_0 + \psi)t_0$ to U_0. Hence U_0 will not lose funds.

In the `Release Phase`, we make a similar argument. Assuming that two round locking phase got executed successfully if U_1 is not able to release the preimage corresponding to either cancellation hash or payment hash before the elapse of the contract's lock time, it will have to pay a penalty to U_0.

- *Case 2 : An intermediate node $U_i, i \in [1, n-1]$ is honest*

  For the `Pre-processing Phase`, if U_i does not receive the onion packet, the payment won't get instantiated.

  In the `Two Round Locking Phase`:

  - *Establishing the Cancellation Contract*: U_i may not be able to forward the contract to U_{i-1} if the latter stops responding. Since U_n is dishonest, it will not release the preimage r for cancelling the contracts established in the first round. As per the terms of the contract, after the elapse of locktime t_{n-1}, it pays a cumulative griefing-penalty $\gamma.(\Sigma_{j=1}^{n-1}(\alpha_j t_j) + (\alpha_0 + \psi)t_0)$ to U_{n-1}. Even if U_{i+1} griefs, as per the terms of the contract it has to pay the required compensation $\gamma((\alpha_0 + \psi)t_0 + \Sigma_{j=1}^{i}(\alpha_j t_j))$ to U_i. Since no contract has been established between U_i and U_{i-1}, U_i retains the entire compensation.
  - *Establishing the Payment Contract*: U_i may not be able to forward the contract to U_{i+1} if U_{i+1} stops responding. The same logic stated for *cancellation contract* holds true except now U_i can retain $\gamma\alpha_i t_i$ as compensation and forward the rest of the amount to node U_{i-1}. Even if U_{i-1} doesn't respond and wait for the locktime t_{i-1} to elapse, U_i will not lose funds.

  In the `Release Phase`, U_i can be griefed in the following ways:

  - U_{i+1} withholds the preimage (either cancellation or payment) from U_i and waits for the contract locktime to expire. In that case, U_{i+1} has to pay compensation of $\gamma((\alpha_0 + \psi)t_0 + \Sigma_{j=1}^{i}(\alpha_j t_j))$ to U_i. Even if U_{i-1} reverse-griefs, U_i will be able to compensate without incurring any loss.

- *Case 3 : U_n is honest*

  Receiver U_n initiates the release of preimage. It will resolve the payment within a bounded amount of time either by releasing the preimage for payment hash or cancellation hash, as per the situation. U_{n-1} cannot reverse-grief and force receiver to pay a griefing-penalty.

Theorem 6.2. ***(Payer and Payee's Privacy).*** *Given the information of griefing-penalty in the off-chain contract, an intermediate node cannot infer its exact position in the path for routing payment.*

Proof: For routing payment of amount α from U_0 to U_n via intermediaries $U_i, i \in [1, n-1]$, several instances of off-chain contract is established across the payment channels. The amount locked by party U_j and U_{j+1} in their off-chain contract is α_j and $\gamma((\psi + \alpha_0)t_0 + \Sigma_{k=1}^{k=j}\alpha_j t_j), j \in [0, n-1]$, respectively. Let us assume that there exists an algorithm τ which reveals the exact position of any intermediate node $D : D \in \{U_1, U_2, \ldots, U_{n-1}\}$ in the path. This implies that given the information of cumulative griefing-penalty mentioned in the contract, it can distinguish between the penalty charged by channel $(U_j, U_{j+1}), j \in [1, n-1]$ and penalty charged by channel (U_0, U_1), which is $\gamma((\psi + \alpha_0)t_0)$. However, the routing attempt cost ψ, was added by node U_0 as an extra compensation charged to cover up for routing attempt expense as well as hiding its identity from its next neighbour. This is information is private and not known by any node except U_0. Additionally, the value of ψ is set such that $\psi t_0 \geq \alpha((k+1)t_0 + \Sigma_{l=1}^{k} l\Delta), k \in \mathbb{N}$. Any number being selected from $\mathbb{N}$ being equiprobable, the probability of distinguishing becomes negligible.

Note: *In practical application, there is a limit on the routing attempt fee which a sender can charge. The set from which k is selected is significantly smaller compared to $\mathbb{N}$. But even under such circumstances, the best inference made by any intermediate node U_j about its location is that it is located at position $(j+k)$ where $\psi t_0 \geq \alpha.t_0 + \alpha.(t_0 + \Delta) + \alpha.(t_0 + 2\Delta) + \ldots + \alpha.(t_0 + k\Delta)$, k acts as the blinding factor.*

6.8 Performance Evaluation

6.8.1 Analysis of Profit earned by eliminating a Competitor from the Network

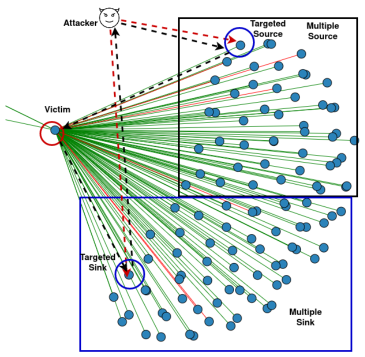
The motivation of the griefer is to eliminate a competitor. The attacker tries to exhaust all the channel capacity of the victim. Transaction requests in the future get routed through the attacker, enabling the latter to steal the victim's processing fee.

Return on Investment or *RoI* is the profit earned by the attacker with respect to the investment made in the network. Here investment means the liquidity utilized by the attacker for simulating an attack. In Lightning Network, the *RoI* of a node processing transaction request is calculated as follows:

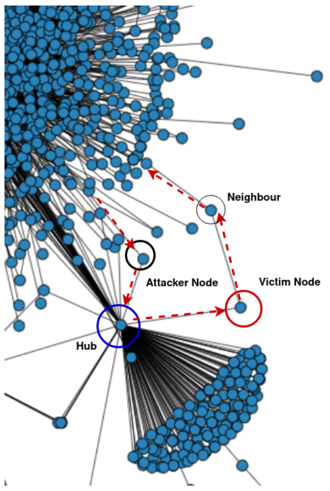
$$\begin{aligned} profit\_processed &= N_{tx}(base\_fee + fee\_rate * tx\_value) \\ RoI &= profit\_processed - total\_griefing\_penalty \end{aligned} \tag{6.1}$$

profit_processed is calculated based on [4], [5]. N_{tx} is the total number of transactions processed by the node and tx_value is the amount transferred from payer to payee. *total_griefing_penalty* is the penalty required to pay as compensation to the affected parties upon mounting griefing-attack. For *HTLC*, the *total_griefing_penalty* is 0 since there is no concept of penalizing the attacker. Hence the node always earns a non-negative *RoI*.

For *HTLC-GP*, if the node mounts a griefing attack, it has to pay a griefing penalty proportional to the collateral locked for the given period. If the *total_griefing_penalty* exceeds the *profit_processed*, the node incurs a loss . In the next section, we define two strategies that can be opted by the attacker. Based on these strategies, we compare the *Return on Investment* obtained for *HTLC* and *HTLC-GP* for a given budget.



(a) Attacker establishes two edges with a targeted source and targeted sink connected to the victim



(b) $Attacker$ uses existing channel for mounting the attack

Figure 6-9: Snapshot of the network on $19^{th} May, 2020$

6.8.1.1 Attacking Strategies

(a) Attacker establishes additional channels

Nodes with *high betweenness centrality* tend to act as intermediaries for routing payments. The attacker selects such nodes as its victim. We illustrate the situation by studying the structure of an instance of Lightning Network. The snapshot taken on 19th May, 2020, in Fig. 6-9(a). Nodes marked as `Targeted Source` and `Targeted Sink` routes their payment via node `Victim`. A malicious node establishes new channels with the `Targeted Source` and `Targeted Sink`. It selects the route `Attacker`→`Targeted Source`→`Victim`→`Targeted Sink`→`Attacker`, sends self-payment requests and mounts griefing attack. The path `Targeted Source`→`Victim` →`Targeted Sink` gets blocked. All the payments from `Targeted Source` gets routed through the path `Targeted Source`→ `Attacker`→`Targeted Victim`.

(b) Attacker uses existing channels

In the previous strategy, the attacker had to establish channels before mounting the attack. To avoid the cost of establishing new channels, the attacker makes use of its existing payment channels to block payments received by its competitor. Illustrating the attack on the same instance of Lightning Network, as shown in Fig. 6-9(b), we consider that the node marked as `Hub` routes all the payment request via `Victim node` and ignores sending any payment via `Attacker node`. In order to steal payments being routed via `Victim node`, it selects the route `Attacker Node`→`Hub`→`Victim Node`→`Neighbour...`→`Attacker Node` for self-payment and mounts griefing attack. The path `Hub`→`Victim Node`→`Neighbour...` →`Attacker Node` gets blocked. Now `Hub` will be forced to route such payment request through `Attacker Node`.

6.8.1.2 Experimental Analysis

Setup: For our experiments, we use Python 3.8.2 and NetworkX, version 2.4 [69] - a Python package for analyzing complex networks. System configuration used is Intel Core i5-8250U CPU, Operating System: Kubuntu 20.04, Memory: 7.7 GiB of RAM. The code for our implementation is available on GitHub [1]. From the dataset mentioned in [100], we took twelve snapshots of Bitcoin Lightning Network over a year, starting from *September 2019*. Each snapshot provides information regarding the public key of the nodes and the aliases used. The network is represented

[1]https://github.com/subhramazumdar/GriefingPenaltyCode

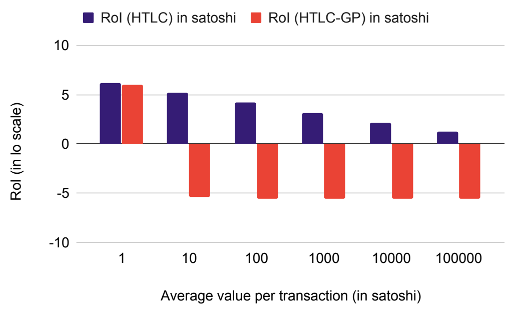
in the form of channels, a pair of public keys. The channel capacity and the channel identifier are mentioned as well. Each node of the channel follows a node policy that mentions the base fee in millisatoshi, fee rate per million (in millisatoshi), and time_lock_delta. The capacity denotes the coins deposited while the opening of the channel. Thus each snapshot of Lightning Network undergoes preprocessing where we filter out channels that are marked as disabled. Next, we select the largest connected component in the network. Since our proposed strategy for countering griefing attacks requires both parties to fund the channel, we divide the channel's capacity into equal halves and allocate each half as the balance of a counterparty. The preprocessed graphs are used for evaluating both *HTLC* and *HTLC-GP*.

Designing transaction set: The best way to approximate the maximum value routed through a node is to map it into a flow problem and compute the maximum flow [56] from multiple payers/sources to multiple payees/sinks. The flow across each channel is the upper bound of the number of transactions being processed. If the attacker manages to block at least one path connecting a payer and payee, then the payer will route its transaction via the attacker. Since it is easier to analyze the situation in a *hub-and-spoke* network, we select a subgraph of LN having a similar structure. Nodes with high *betweenness centrality* [121] have high probability of being a potential *hub node*. We select such a hub node where a subset of the pendant nodes connected to the hub forms the set of sources, and the rest of the neighbors form the sink. Once a maximum flow is computed, the flow through a channel connecting a source to the hub and through the channel from the hub to a sink forms the maximum valued payment through the path. The attacker targets such a source-sink pair with the hub acting as its victim. The attacker selects the victim and blocks its channels. Once the maximum flow across such source-sink pairs gets computed, the attacker checks the fraction of flow that gets routed through itself. To estimate the transaction set size, the attacker divides the flow by the amount per transaction. *Return on Investment* can be calculated for all such transactions based on Eq. 6.1.

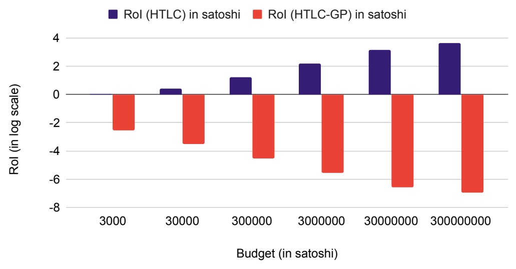
Data Used: We vary the range of transaction amount between *1 satoshi* to *100000 satoshi* [37], increasing the amount by multiple of 10. For the attack involving the establishment of new channels by the attacker, we vary the level of the budget of the adversary as *3000 satoshi, 30000 satoshi, 300000 satoshi, ..., 3 BTC*. For the attack involving the use of existing channels by the attacker, we vary the level of the budget of the adversary as *3000 satoshi, 30000 satoshi, 300000 satoshi, ..., 0.03 BTC*. Increasing the budget beyond 0.03 BTC is of no use since a substantial amount of the budget remains unutilized after this point. The budget allotted is utilized by the attacker for instantiating payment and locking cumulative griefing-penalty, ignoring the cost of establishing the channels in the network.

Simulation Result

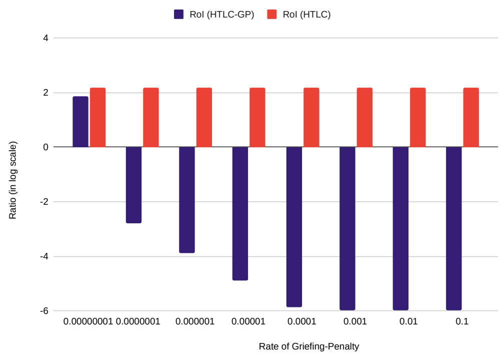
The *Return on Investment (RoI)* is measured in log-scale. For negative *RoI*, we use log-modulus transformation [77].



(a) RoI vs Average value per transaction: Fixed Budget - 0.03 BTC



(b) RoI vs Budget: Fixed Average value per transaction - 10000 satoshi



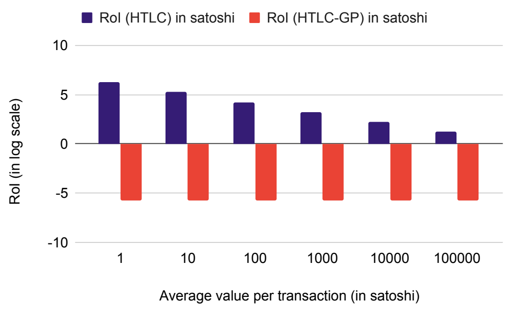
(c) RoI vs Rate of Griefing-Penalty: Fixed Average value per transaction - 10000 satoshi

Figure 6-10: When Attacker uses new channels for mounting the attack

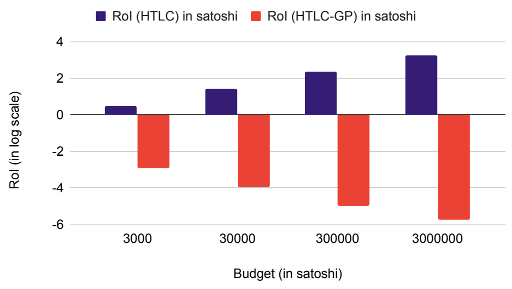
- *Using attacking strategy 1*: The first result *RoI vs Average value per transaction*, for a fixed budget of 0.03 BTC and fixed rate of griefing-penalty of 0.001 per minute, shows that as the average value of each transaction increases, the processing fee earned by the attacker decreases due to decrease in the maximum number of payments processed for *HTLC*. However, for *HTLC-GP*, as the average value per transaction increases, *RoI* becomes negative, shown in Fig. 6-10(a). The processing fee earned becomes negligible compared to the penalty incurred.

  The second result *RoI vs Budget*, for a fixed average value of transaction of 10000 satoshi and fixed rate of griefing-penalty of 0.001 per minute, shows that as the budget of the attacker increases, the processing fee earned by the attacker increases linearly. But a reverse trend is observed for *HTLC-GP*. *RoI* decreases linearly, as shown in Fig. 6-10(b). This is because is the amount of cumulative penalty is directly related to the total collateral locked by the attacker.

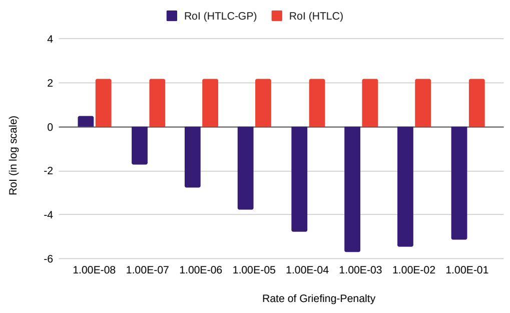
The third result *RoI vs Rate of Griefing-Penalty*, for a fixed average value of transaction of 10000 satoshi and fixed budget of 0.03 BTC, the return on investment for *HTLC* remains constant since the rate of griefing-penalty has no impact in this case. But the loss incurred increases with increase in γ, as observed for *HTLC-GP* in Fig. 6-10(c).



(a) RoI vs Average value per transaction: Fixed Budget - 0.03 BTC



(b) RoI vs Budget: Fixed Average value per transaction - 10000 satoshi



(c) RoI vs Rate of Griefing-Penalty: Fixed Average value per transaction - 10000 satoshi

Figure 6-11: When Attacker uses existing channels for mounting the attack

- *Using attacking strategy 2*: For a fixed budget of the attacker, loss incurred for *HTLC-GP* using second attacking strategy is higher than the first attacking strategy for all the three cases, as shown in Fig Fig. 6-11(a), Fig. 6-11(b) and Fig. 6-11(c). The reason being the average path length for self-payment is around 6.5 compared to the first attacking strategy, where the average path length remains fixed at 4.

6.8.2 Investment made by attacker for stalling the network

For a path of length n, the cumulative griefing-penalty is $\gamma((\psi+\alpha_0)t_0+\Sigma_{j=1}^{n-1}\alpha_j t_j)$ for transferring an amount of α_{n-1} from sender U_0 to receiver U_n. In case of *HTLC*, for blocking liquidity of at least α_{n-1} in each of the n channels, the attacker needs to invest α_0 and execute a self-payment. In case of *HTLC-GP*, in order to execute a self-payment of α_0, the attacker needs to invest $\alpha_0 + \gamma((\psi+\alpha_0)t_0+\Sigma_{j=1}^{n-1}\alpha_j t_j)$. If we take the ratio of the investment made for *HTLC* and investment made by attacker for *HTLC-GP* for a fixed transaction value,

$$\frac{\alpha_0}{\alpha_0+\gamma((\psi+\alpha_0)t_0+\Sigma_{j=1}^{n-1}\alpha_j t_j)} \leq \frac{1}{1+\gamma(t_0+\Sigma_{j=1}^{n-1}\frac{\alpha_j}{\alpha_0}t_j)} \tag{6.2}$$

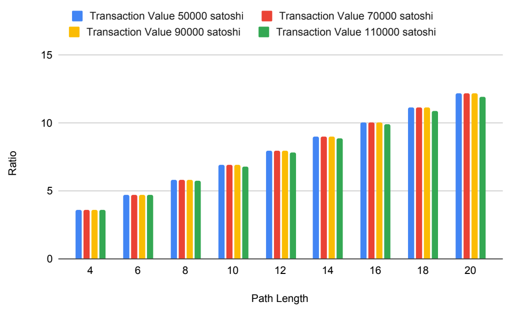
- *Path Length*: For fixed γ and α, the ratio will be strictly less than 1 for any $n > 1$, since $\gamma(t_0+\Sigma_{j=1}^{n-1}\frac{\alpha_j}{\alpha_0}t_j) > 0$.
- *Rate of Griefing-Penalty*: For fixed n and α_n fixed, the ratio will be strictly less than 1 for any value of $\gamma \in (0,1)$ since $\gamma(t_0+\Sigma_{j=1}^{n-1}\frac{\alpha_j}{\alpha_0}t_j) > 0$.

Evaluation. We use the same experimental setup and graph instances as in Section 6.8.1.2.

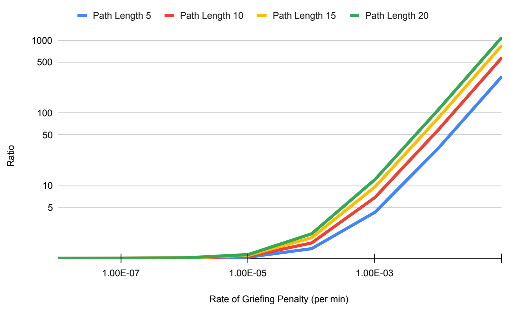
- *Impact of Path Length.* For a given transaction value and fixed rate of griefing-penalty set to 0.001 per minute, we vary the path length in the range from 4 to 20, and transaction value as *50000 satoshis, 70000 satoshis, 90000 satoshis, 110000 satoshis*. The ratio of the adversary budget needed for mounting a griefing attack in *HTLC-GP* and the adversary budget needed for mounting a griefing attack in *HTLC* is around 4.7 when the path length is 4 and around 12 when the path length is 20. The ratio increases linearly with an increase in path length, as observed in Fig. 6-12(a). Upon varying the transaction value, we do not observe any change in this trend.

- *Impact of Rate of Griefing-Penalty.* For a fixed transaction value of 50000 satoshi and given path length, we vary the rate of griefing-penalty γ in the range $\{10^{-8}, 10^{-7}, 10^{-6}, \ldots, 0.01, 0.1\}$ and the path length as *5,10,15,20*. The ratio of the adversary budget needed for mounting a griefing attack in *HTLC-GP* and *HTLC* increases exponentially with an increase in the rate of griefing-penalty. The rate of increase in the ratio is almost equal till γ is 10^{-4}, invariant of change in path length. When $\gamma > 10^{-3}$, the rate of increase in the ratio is the lowest for a path length of 5 and increases faster for a path length of 20, as shown in Fig. 6-12(b).

  The result shows that the investment made by the attacker for *HTLC-GP* is higher than the investment made by the attacker for *HTLC* thereby strongly disincentivizing the griefing attack.



(a) Impact of path length on the investment made by attacker for launching griefing attack (HTLC-GP vs HTLC)



(b) Impact of rate of griefing-penalty on the investment made by attacker for launching griefing attack (HTLC-GP vs HTLC)

Figure 6-12: Investment made by attacker (HTLC vs HTLC-GP)

6.9 Rate of Griefing-Penalty for Practical Purpose

In this section, we provide an estimated range for the rate of griefing penalty to be selected for practical purposes. The decision is taken from the perspective of an honest payer and honest payee. We assume that the transaction value or the penalty locked is less than the capacity of each payment channel.

6.9.1 From the perspective of an honest payer and honest payee

For a given payment instance, the amount of penalty locked by a recipient must not become either too low or too high. Given the amount to be transferred from payer to payee as α and the path length as n, the cumulative penalty to be locked is $\gamma\Sigma_{j=0}^{n-1}\alpha t_j$ (ignoring routing attempt fee and the processing fee charged by intermediate nodes for ease of analysis). We consider a model where the summation of the amount routed by the sender and the cumulative griefing penalty locked by the receiver is constant. Let this quantity be C. Thus we have the following equation:

$$\alpha + \gamma\Sigma_{j=0}^{n-1}\alpha t_j = C \tag{6.3}$$

where γ is the rate of griefing-penalty. From Eq. 6.3, α is $\frac{C}{(1+\gamma\sum\limits_{j=0}^{n-1} t_j)}$.

The ratio of the amount the sender can transfer in *HTLC-GP* and the amount the sender could have transferred in *HTLC*, denoted as $Ratio_{Sender}$, is $\frac{\alpha}{C}$. The ratio between the amount the receiver

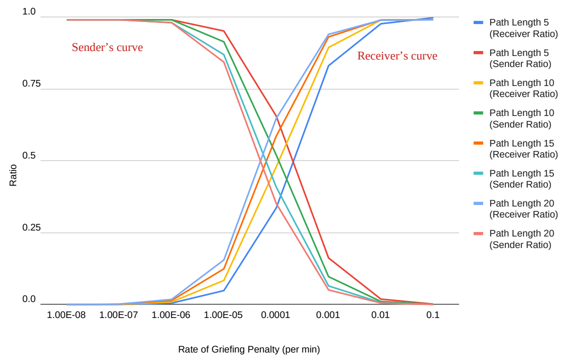


Figure 6-13: Plot of ratio of amount locked by the party and the total capital C, for both sender and receiver

needs to lock as cumulative penalty and the maximum cumulative penalty that the receiver incurs, denoted as $Ratio_{Receiver}$, is $\frac{\gamma \sum_{j=0}^{n-1} \alpha t_j}{C}$.

When the rate of griefing-penalty almost tends to 0, transaction value α is equivalent to C. The protocol functions like *HTLC* in terms of the payment routed from payer to payee without overburdening intermediaries. Throughput of payment will be almost equivalent to what has been observed for *HTLC*. If the incidence of the attack increases, the value of γ must be increased to protect the affected parties. The transaction value decreases, and cumulative griefing-penalty increases. Intermediate parties are forced to lock a substantial channel funds for a very small-valued transaction. Payments may not get routed because of a lack of liquidity in channels. Large-valued payments get discouraged, and the throughput of payment falls substantially.

Experimental Analysis. For a fixed value of C=*50,000 satoshi*, and a given path length, we vary the rate of griefing-penalty as $\{10^{-8}, 10^{-7}, \ldots, 0.01, 0.1\}$. We plot both $Ratio_{Sender}$ and $Ratio_{Receiver}$ for path length $\{5, 10, 15, 20\}$, depicted in Fig. 6-13. For γ between 10^{-8} to 10^{-5}, the transaction value lies between *37,500 satoshi - 48,000 satoshi*, rest of the amount being the cumulative griefing-penalty. When γ is $> 10^{-5}$ and $\leq 10^{-3}$, the transaction value routed and cumulative griefing-penalty locked becomes almost equivalent. Beyond this range, the transaction value falls abruptly with cumulative griefing-penalty increasing at an equal rate. To avoid this problem, a range $> 10^{-5}$ but $\leq 10^{-3}$ looks suitable for practical purpose.

6.10 Discussion

- *Other forms of griefing attack*: We do not consider the case where a counterparty resolves the contract just before the expiration of lock time. It is difficult to record the time elapsed in an asynchronous system. In such a system, time intervals may have different meanings to different nodes in the network. For example, a malicious party may manipulate the clock so that time elapses slower than that of other parties. Additionally, it is difficult to say whether a party is delaying intentionally or if it is due to network congestion. It might be the case the party who has to pay a penalty tampers the information of elapsed time and pays a lower penalty. No one can raise a dispute as the Bitcoin script cannot currently decide the penalty for each time interval in a single contract. The closest match is the CheckSequenceVerify opcode. However, this enables the broadcast of a transaction after a certain block height has been reached. But there is no way to execute a transaction like this: *If t' time units have elapsed, pay amount p. If t'+1 time units have elapsed, pay amount* $p + \delta$. CheckSequenceVerify imposed on the first condition of elapse of t' time unit makes it eligible for the broadcasting event after elapse of time t'+1. Hence imposing a penalty for the milder form of griefing has not been considered.

- *Rate of Griefing-Penalty* γ: If each node had a different rate of griefing-penalty, then it becomes difficult for the receiver to check the correctness of cumulative penalty using the modified path length $\phi(n)$ and lock time t_{n-1}. In that case, the sender has to send the entire set of $\gamma_i, \forall i \in [0, n-1]$, to U_n. Revealing γ_i may leak information regarding the identity of an intermediate node to the receiver. Hence, for ease of analysis, we keep the rate constant. Moreover, we envisage the rate to be determined by the market. If the rate is too steep, the entry barrier for new participants will be high and there will be liquidity concerns. On the other hand, if the rate is too low, the disincentivization will be minimal and griefing attacks will remain rampant. Hence, the rate will be adjusted by the community based on the current scenario of the network. The method is similar to the adjustment of the difficulty parameter of Bitcoin.

- *Censoring attacks on HTLC-GP success transaction*: The cost involved in censoring a transaction is too much. Rational participants may not be interested in mounting such an attack considering the long-term profits. There are several countermeasures like MAD-HTLC [132] where miners act as watchtowers and *Sliding Window Protocol* [88] protocol that defends from attacking miners. We can incorporate these countermeasures in *HTLC-GP* to prevent censoring of transactions.

Chapter 7

Strategic Analysis of Griefing Attack in Lightning Network

A griefing attack allows a malicious node to claim the network's liquidity for a certain duration. Sybil nodes submit several payments in the network and grief [89]. The network gets jammed, and the honest nodes cannot process payment. In Chapter 6, we discussed how HTLC-GP can be used to counter griefing attacks. However, such protocols work under the assumption that participants are either honest or malicious, but fail for a rational participant. A rational adversary will change its attacking strategy to avoid being penalized. Similarly, a rational non-attacking participant will follow the steps of the protocol only if it is profitable. The security proofs focus on the cryptographic aspect but do not capture the impact a penalty would have on the attacker's behavior. We address the gap by proposing a game-theoretic model for griefing attacks in Lightning Network. An attacker with a given budget will try to maximize the damage inflicted on the network. It targets certain nodes in the network and offers a bribe. Such nodes are corrupt, and they mount the griefing attack on behalf of the attacker. We model the strategic interaction between any two players in the network connected by a payment channel. The first player is assumed to have received the conditional payment, and the second player is the neighbor of the former to whom the conditional payment request must get forwarded. The latter can either be corrupt or uncorrupt. We use the model to analyze the countermeasure *HTLC-GP* and state our observations.

7.1 Contributions

We have made the following contributions:

- Given a fixed budget of the attacker, we model griefing attack in *HTLC*, as a two-player *dynamic game of incomplete information*.
- Based on the model, we formulate another two-player game to analyze griefing attacks in the payment protocol Hashed Timelock Contract with Griefing Penalty or *HTLC-GP* [96]. We observe that a corrupt player can mount a griefing attack without paying any penalty. We infer that it is infeasible to design a protocol that will compensate parties affected by a griefing attack.

- We observe that the introduction of penalty increases the cost of the attack, and thus, we measure the effectiveness of *HTLC-GP* in terms of the *capacity locked* by the attacker in the network. We claim that *HTLC-GP* is *weakly effective* in countering the attack when the rate of griefing-penalty is too low.

- We introduce the concept of *guaranteed minimum compensation*, ζ, to control the maximum allowed path length for routing and further increase the cost of the attack. We modify HTLC-GP to HTLC-GP$^\zeta$ by including the concept of ζ.

- We simulate the game model for griefing attacks in both HTLC and HTLC-GP and provide some interesting analysis. By experimenting on several instances of Lightning Network, we observe that if the guaranteed minimum compensation for each affected party is 2.5% of the transaction amount, the maximum allowed path length drops to 10 and the capacity locked by the attacker drops to 28% in HTLC-GP$^\zeta$. This quantity is 27% less than the capacity locked in *HTLC-GP*. We infer from the results that HTLC-GP$^\zeta$ is far more effective than *HTLC-GP*. The code is provided in GitHub[1].

7.1.1 Organization

Section 7.2 discusses the state-of-the-art. In Section 7.3, we propose a game-theoretical analysis of the payment protocol *HTLC*, and discuss griefing attack in this model. We discuss an existing countermeasure for griefing attack, *HTLC-GP*, in Section 7.4.1 and in Section 7.5, we analyze griefing attack in HTLC-GP and discuss the effectiveness of the protocol. In Section 7.6, we propose the concept of *guaranteed minimum compensation* ζ. We modify *HTLC-GP* into HTLC-GP$^\zeta$ by incorporating the concept of minimum compensation in Section 7.7. Experimental results obtained upon simulating the game models of *HTLC* and *HTLC-GP*, and measuring the efficiency of *HTLC-GP* and HTLC-GP$^\zeta$ is provided in Section 7.8. The notation used in the chapter has been defined in Table 7.1.

7.2 Related Works

We discuss some of the works that analyze the payments executed in Lightning Network from a game-theoretic point of view. In [145], a framework for formally characterizing the robustness of blockchain systems in the presence of Byzantine participants has been proposed. The paper defines

[1]https://github.com/subhramazumdar/Strategic_Analysis_Griefing

Notation	Description
D	Least HTLC Timeout period
t_i	HTLC Timeout period in channel $(U_i, U_{i+1}), i \in [0, n-1]$
M	Average mining fee for closing a payment channel
Γ_{HTLC}	Extensive form of a 2-party sequential Bayesian game in *HTLC*
γ	Rate of griefing penalty (per minute)
T	Lifetime of a channel
$t_{i,j}$	The time at which the channel between U_i and U_j was opened
$t_{contract\_initiate}$	Timestamp at which off-chain contract got initiated
$\Gamma_{HTLC-GP}$	Extensive form of a 2-party sequential Bayesian game in *HTLC-GP*
ζ	Guaranteed Minimum Compensation
$\tilde{n}$	Maximum path length in HTLC-GP$^\zeta$, $\tilde{n} \leq n$
k	Ratio of the cumulative penalty locked by payer and the payment value locked by payee
$\gamma^{\zeta,k}$	Rate of griefing-penalty in HTLC-GP$^\zeta$ for a given ζ and k

Table 7.1: Notations used in this chapter

the routing module *HTLC* as a game between three participants. However, they have considered that a payment can be accepted or rejected. This work doesn't capture the case when a malicious party can intentionally stop responding. It is not justified to consider a lack of response equivalent to rejecting a payment instantly since the parties cannot utilize the coins locked in the contract. Assigning a payoff of 0 for a failed payment doesn't account for the opportunity cost of coins locked in an off-chain contract. Another work [117] discusses the shortcoming of the game model of multi-hop payment proposed in [145]. They have improved the model that is capable of detecting *wormhole attacks* in the network, but the work does not discuss the griefing attack. In [141], a game-theoretic analysis of atomic cross-chain swaps using *HTLC* has been provided. They have studied the impact of token price volatility on the strategic behavior of the participants initiating the swap and suggested the use of collateral deposits to prevent parties from canceling the swap. Using premium for fairness in the atomic cross-chain swap was proposed in [70]. The authors have suggested penalty as a countermeasure for countering the griefing attack in the context of atomic swap. However, the paper lacks a detailed analysis of how the introduction of premiums might ensure a faster exchange of assets. Additionally, there is no mention of loss incurred due to the rise in the opportunity cost of locked assets.

This is the first attempt to model griefing attack in Lightning Network as a sequential Bayesian game. It provides us a better insight into how a participant will behave based on its type and helps design suitable countermeasures.

7.3 Analysis of Griefing Attack in HTLC

Before modeling griefing attack as a two-player *sequential Bayesian game* [62], we state the system requirements, attack model, and assumptions.

7.3.1 System Model

Lightning Network is modeled 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. Every node charges a processing fee for relaying funds across the network. Fee is defined by a function f, where $f : \mathbb{R}^+ \rightarrow \mathbb{R}^+$. Correctness of payment across each channel is enforced cryptographically using hash-locks and time-locks [113]. Each payment channel $(U_i, U_j) \in E$ is assigned an identifier $id_{i,j}$. The channel $id_{i,j}$ has an associated capacity $locked(U_i, U_j)$, denoting the amount locked by U_i and $locked(U_j, U_i)$ denoting the amount locked by U_j. Let us denote the amount as $val_{i,j} = locked(U_j, U_i) + locked(U_i, U_j)$. The transaction recorded in the blockchain is $(id_{i,j}, val_{i,j}, t_{i,j}, T)$ where $t_{i,j}$ is the timestamp at which the channel was opened. In the context of the Bitcoin blockchain, this will be the block height. T is the expiration time of the channel $id_{i,j}$, i.e., once a channel is opened, it is expected to remain active till T[2]. $remain(U_i, U_j)$ signifies the residual amount of coins U_i can transfer to U_j via off-chain transactions. M denotes the average fee for mining a Bitcoin transaction.

Given an instance of payment where payer wants to transfer α coins via maximum allowed path length n. The payer U_0 wants to transfer α coins to payee U_n through path $P = \langle U_0 \rightarrow U_1 \rightarrow U_2 \ldots \rightarrow U_n \rangle$. A node U_{i-1} can locks coins α_{i-1} in an off-chain contract formed with U_i if $remain(U_{i-1}, U_i) \geq \alpha_{i-1} : \alpha_{i-1} = \alpha_i + f(\alpha_i), i \in [1, n], \alpha_{n-1} = \alpha$, then funds can be relayed across the channel (U_{i-1}, U_i). Node U_{i-1} gets a processing fee $f(\alpha_{i-1})$. If U_i claims the coins, the residual capacity is updated as follows : $remain(U_{i-1}, U_i) = remain(U_{i-1}, U_i) - \alpha_{i-1}$ and $remain(U_i, U_{i-1}) = remain(U_i, U_{i-1}) + \alpha_{i-1}$. U_n generates a payment condition $H = \mathcal{H}(x)$ and shares it with U_0. The *HTLC* timeout period in the contract between U_{i-1} and U_i is set to $t_{i-1}, i \in [1, n]$.

[2]Each channel in Lightning Network has an infinite lifetime. However, we assume an upper bound on the channel lifetime for our analysis. Setting channel expiration time has been used in the literature as well [94]

7.3.1.1 System Assumption

All the nodes in the network are rational [30], [59][3]. Rational processes always seek to maximize their expected utility. They will deviate from a prescribed protocol if and only if doing so increases their expected utility. Uncorrupt nodes may or may not participate in the execution of the protocol, depending on the utility. This is different from the classical model, where an honest node always follows the steps of the protocol. We assume that a channel between U_{i-1} and U_i is unilaterally funded by $U_{i-1}, i \in [1, n]$, i.e., $locked(U_i, U_{i-1}) = 0$.

Functions used. We define a function $O : \mathbb{R}^+ \times \mathbb{W} \times \mathbb{R}^+ \rightarrow \mathbb{R}^+ \cup \{0\}$, where $O(r_U, t, val)$ is the expected revenue a node U would have earned had it utilized the amount val for processing transactions in period of t units given that r_U is the rate of payments processed by U per unit time. In other words, O defines the *opportunity cost* [40]. The primary source of revenue for a routing node in the Lightning Network is the fee obtained by processing transactions [108]. Also, the arrival of payment in a channel follows a Poisson process [4] [61], [68], [82]. U expects each transaction size to be per_tx_val. Given the number of coins locked is val, the number of transactions U expects to receive in period of t units is $J = \frac{val}{per\_tx\_val}$. Given X is the number of transactions in that interval or $X \sim Poisson(r_U t)$, we have.

$$P(X = x) = \frac{e^{-r_U t}(r_U t)^x}{x!} \tag{7.1}$$

where $0 \leq x \leq J$. Expected number of transactions in t unit of time

$$E(X) = \sum_{x=0}^{J} x P(X = x) \tag{7.2}$$

The fee earned by processing a transaction of size per_tx_val is defined as [4],[5]:

$$Fee_{per\_tx\_val} = base\_fee + fee\_rate \times per\_tx\_val \tag{7.3}$$

Thus, the revenue U expects to earn within period t or in other words, the opportunity cost

[3]For our model, we restrict it to just rational participants. Since the attacker has a fixed budget, it will not be able to bribe all the nodes in the network. However, the Lightning network may have Byzantine as well as altruistic nodes. We leave the analysis of griefing attack in the BAR model [21] as future work.

[4]All the papers assumed the arrival of transaction in the Bitcoin blockchain as a Poisson process but the validity of the assumption was not verified. It was first analyzed in [61, 60] and the authors have reported that transactions' inter-arrival times can be approximately fitted with an exponential distribution, which partially supports the Poisson arrival assumption but with noticeable deviation. In our paper, we consider the arrival of transactions in the Bitcoin blockchain following a Poisson process, and the same holds for transactions arriving in Lightning Network.

$O(r_U, t, val)$ is defned as:

$$O(r_U, t, val) = E(X) Fee_{per\_tx\_val} \tag{7.4}$$

7.3.2 Attacker Model & Assumptions

An attacker with budget $\mathcal{B}_{EX}$ has the objective of disrupting the network by jamming the network [30][5]. Given the budget, the attacker will be able to incentivize a certain number of nodes in the network to mount the griefing attack. We define the model and assumptions as follows:

- If a node has accepted the bribe, then it implies that the expected earning by cooperating with other nodes is lower than the bribe. Such nodes act as per the instructions received from the attacker.
- All corrupt nodes know each other and the rest of the uncorrupt nodes in the network. The uncorrupt nodes remain unaware of a given node's nature.
- Corrupt nodes may open additional payment channels in the network just for the attack.

(i) *Method for mounting Griefing Attack.* If a node is corrupt, it executes a self-payment of amount α (where it acts as both payer and payee, i.e. $U_0 = U_n$) via a route of the maximum allowed path length. The timeout period for each HTLC is defined in a fashion such that the least timeout period $t_{n-1} \approx D$. After the conditional payment reaches the payee U_n, it stops responding, locking collateral of $(n-1)\alpha$ for D units in the path routing payment.

(ii) *Bribe offered per attack.* In the system model, we consider that each payment is of value α. The attacker fixes the bribe offered to a node to L coins. The amount L is $\alpha + I_{D,\alpha} + C$, where C is the cost of routing payment and $I_{D,\alpha}$ is used to compensate the node for keeping α coins unutilized for the next D units of time. $I_{D,\alpha} \approx 2O(r_U, D, \alpha), \forall U \in V$ so that once the corrupt node locks an amount α for time D, even after losing $O(r_U, D, \alpha)$, the net gain would still be at least $O(r_U, D, \alpha)$. This ensures that the corrupt nodes will not switch to being altruistic.

7.3.3 Game Model

We model the interaction between any two entities present in path P. Assuming that every incidence of griefing attack gets reported in the network, U_{i-1} form a belief, defined as θ_i, where

[5]In this work, we consider the incentives at play to be external to the system. It is standard practice, and several works have adhered to this model [59].

$\theta_i \in [0, 1]$. U_i can be *corrupt* with probability θ_i or *uncorrupt* with probability $1 - \theta_i$.

(i) *Belief Model*: Any uncorrupt player is unaware of the attacker's budget. Since an attacker wants to corrupt as many nodes as possible for a given budget, it will select nodes in the network that either has a low centrality measure or a meager expectation of earning by processing transaction. If the node has a high degree of betweenness centrality, it is difficult to corrupt such a node as it would prefer to act honestly rather than accept the bribe. A rational player forms a belief about the expected behavior of other nodes based on their position in the network and their reputation.

(ii) *Choice of players*: The interaction can be modeled for any pair of node U_{i-1} and $U_i, i \in [1, n]$. However, a corrupt node will execute a self-payment via maximum path length for mounting the attack. Other intermediate nodes can choose to grief, but the attacker will not pay a node that is not hijacking as many channels as possible. Thus griefing will be a loss-making strategy for a rational intermediate node routing the payment.

We discuss the action set of a pair of nodes U_{i-1} and $U_i, i \in [1, n]$:

- *U_{i-1}'s action*: The action space of U_{i-1} is defined as follows: it can either *forward (F)* the conditional payment to U_i or it can choose to *not forward (NF)*. If it chooses to forward the payment, it forms a contract with U_i, locking the designated amount in the channel $id_{i-1,i}$ for time t_{i-1}, which is the HTLC timeout period. If $i > 1$, then U_{i-1} gets a fee of $f(\alpha_{i-1})$ from U_{i-2}, if U_i resolves the payment and claims the coins. If $i = 1$, then U_0 will be satisfied if the payment succeeds. In this case, we consider $f(\alpha_0)$ as the level of satisfaction. If U_i delays, then opportunity cost of coins locked in the off-chain contract increases, and a loss is incurred. If U_i doesn't respond within the deadline, then U_{i-1} closes the channel and withdraws its coins from the contract.

- *U_i's action*: If U_{i-1} has forwarded the payment, then U_i can choose its action from the following: *accept the payment* or *Ac*, *reject the payment* or *Rt*, *wait and then accept* or *W & Ac*, *wait and then reject* or *W & Rt*, and *grief* or *Gr*.

  - An *uncorrupt* U_i's expected behavior is to resolve the payment instantly, i.e., *Ac* or *Rt*. Still, there is a probability that it ends up delaying or griefing. These actions lead to a loss in terms of the opportunity cost of locked coins and the closure of the channel.

  - A *corrupt U_i's* (when $i = n$) expected behavior is to grief, *Gr* or wait and reject the payment, *W & Rt*, just before contract timeout. The attacker offers a bribe to such a node and instructs to mount the griefing attack.

We model interaction between U_{i-1} and U_i as a sequential *Bayesian game* Γ_{HTLC}. The game begins with Nature (**N**) choosing the type of U_i, either *corrupt* or *uncorrupt*, respectively. U_{i-1} believes that a corrupt U_i will be selected with probability θ_i, whereas an uncorrupt U_i will be selected with probability $1-\theta_i$. After **N** makes its move, U_{i-1} selects its strategy based on the belief of U_i's type. Finally, U_i chooses its strategy, provided U_{i-1} has forwarded the payment.

Definition 7.1. *The extensive-form game Γ_{HTLC}, represented in Fig.7-1, is defined as tuple $\Gamma_{HTLC} = \langle N, (\Theta_{U_{i-1}}, \Theta_{U_i}), (S_{U_{i-1}}, S_{U_i}), p_{U_{i-1}}, (u_{U_{i-1}}, u_{U_i})\rangle$ [104]:*

- *The set of players $N = \{U_{i-1}, U_i\}$ where $i \in [1, n]$.*
- *Since player U_{i-1} has received the payment request, it has just has one type, i.e., rational. Thus $\Theta_{U_{i-1}} = \{rational(r)\}$. Since, it is a singleton set, we ignore this factor while defining payoff.*
- *The type of player U_i defined as $\Theta_{U_i} = \{$Corrupt(co), Uncorrupt (uco)$\}$*
- *The set of actions for player U_{i-1}, $S_{U_{i-1}} = \{F, NF\}$*
- *The set of actions for player U_i, $S_{U_i} = \{$Ac, Rt, W & Ac, W & Rt, Gr$\}$*
- *Probability function p_{U_i} is a function from $\Theta_{U_{i-1}}$ into $p(\Theta_{U_i})$, where the $p(\Theta_{U_i})$ denotes the set of probability distribution over Θ_{U_i}. Since U_{i-1} has just one type. $p_{U_{i-1}}$ specifies a probability distribution $p_{U_{i-1}}(|r)$ over the set Θ_{U_i} representing what player U_{i-1} beliefs about the type of player U_i. Here $p_{U_{i-1}}(Corrupt) = \theta_i$, $p_{U_{i-1}}(Uncorrupt) = 1-\theta_i$.*
- *The payoff function $u_k : \Theta \times S \rightarrow \mathbb{R}$ for any player $k \in \{U_{i-1}, U_i\}$, where $\Theta = \Theta_{U_i}$ and $S = S_{U_{i-1}} \times S_{U_i}$, is such that for any profile of actions and any profile of types $(\hat{\theta}, s) \in \Theta \times S$, specifies the payoff the player k would get, if the player's actual type were all as in $\hat{\theta}$ and the players all chose their action as in s.*

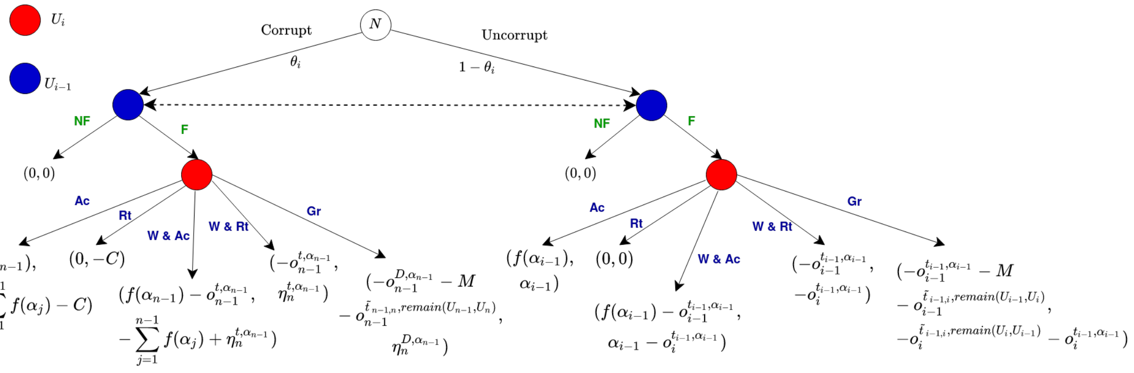

U_i
U_{i-1}
N
Corrupt
θ_i
Uncorrupt
$1-\theta_i$
NF
F
$(0,0)$
Ac
Rt
W & Ac
W & Rt
Gr
$(f(\alpha_{n-1}), -\sum_{j=1}^{n-1} f(\alpha_j) - C)$
$(0, -C)$
$(f(\alpha_{n-1}) - o_{n-1}^{t,\alpha_{n-1}}, -\sum_{j=1}^{n-1} f(\alpha_j) + \eta_n^{t,\alpha_{n-1}})$
$(-o_{n-1}^{t,\alpha_{n-1}}, \eta_n^{t,\alpha_{n-1}})$
$(-o_{n-1}^{D,\alpha_{n-1}} - M - o_{n-1}^{\tilde{t}_{n-1,n},remain(U_{n-1},U_n)}, \eta_n^{D,\alpha_{n-1}})$
NF
F
$(0,0)$
Ac
Rt
W & Ac
W & Rt
Gr
$(f(\alpha_{i-1}), \alpha_{i-1})$
$(0,0)$
$(f(\alpha_{i-1}) - o_{i-1}^{t_{i-1},\alpha_{i-1}}, \alpha_{i-1} - o_i^{t_{i-1},\alpha_{i-1}})$
$(-o_{i-1}^{t_{i-1},\alpha_{i-1}}, -o_i^{t_{i-1},\alpha_{i-1}})$
$(-o_{i-1}^{t_{i-1},\alpha_{i-1}} - M - o_{i-1}^{\tilde{t}_{i-1,i},remain(U_{i-1},U_i)}, -o_i^{\tilde{t}_{i-1,i},remain(U_i,U_{i-1})} - o_i^{t_{i-1},\alpha_{i-1}})$

Figure 7-1: Extensive form of game Γ_{HTLC}

7.3.3.1 Payoff Model

If U_{i-1} chooses not to forward, then either party receives a payoff 0 since no off-chain contract got established, i.e., $u_{U_{i-1}}(\theta_b, NF, s_b) = u_{U_i}(\theta_b, NF, s_b) = 0, \theta_b \in \Theta_{U_i}$ and $s_b \in S_{U_i}$.

We analyze the payoff of U_{i-1} when it has chosen F:

A. **N** had chosen an *uncorrupt* U_i. We analyze the payoff of each case:

I. Instantaneous Response, i.e., $t \to 0$: Payoff of both the parties, is discussed.

a. U_i *accepts the payment*: U_i claims the payment and U_{i-1} gets processing fee $f(\alpha_{i-1})$ from its preceding neighbour U_{i-2}. The payoffs are defined as $u_{U_{i-1}}(uco, F, Ac) = f(\alpha_{i-1})$ and, $u_{U_i}(uco, F, Ac) = \alpha_{i-1}$ respectively.

b. U_i *rejects the payment*: None of them gains anything, and the channel balance is restored.
$u_{U_{i-1}}(uco, F, Rt) = u_{U_i}(uco, F, Rt) = 0$.

II. Delayed Response, i.e., $0 < t < t_{i-1}$: Payoff of both the parties is discussed.

a. U_i *waits and then accepts the payment*:

- U_{i-1} can earn $f(\alpha_{i-1})$ only after U_i resolves the payment. However, if U_i delays for t units in acquiring α_{i-1} coins, then U_{i-1} suffers a loss due to coins remaining locked in channel $id_{i-1,i}$. The opportunity cost is defined as $O(r_{U_{i-1}}, t, \alpha_{i-1})$. It is also denoted as $o_{i-1}^{t,\alpha_{i-1}}$. This factor is subtracted from the payoff U_{i-1} obtains if U_i had accepted the payment instantly. $u_{U_{i-1}}(uco, F, W \,\&\, Ac) = f(\alpha_{i-1}) - o_{i-1}^{t,\alpha_{i-1}}$.
- U_i loses the opportunity to earn profit by utilizing α_{i-1} coins for the next t unit of time. The expected profit that U_i could have made using α_{i-1} within the next t units is $O(r_{U_i}, t, \alpha_{i-1})$, also denoted as $o_i^{t,\alpha_{i-1}}$. This shows that delaying is costly. Thus, $u_{U_i}(uco, F, W \,\&\, Ac) = \alpha_{i-1} - o_i^{t,\alpha_{i-1}}$.

b. U_i *waits and then rejects the payment*: The payoff of U_{i-1}, denoted as $u_{U_{i-1}}(uco, F, W \,\&\, Rt)$, is $-o_{i-1}^{t,\alpha_{i-1}}$. Payoff of U_i, denoted as $u_{U_i}(uco, F, W \,\&\, Rt)$, is $-o_i^{t,\alpha_{i-1}}$, since it loses the opportunity to utilize the coins.

III. U_i *griefs*: If U_i fails to respond within time t_i, U_{i-1} will close the channel by going on-chain.

- U_{i-1} cannot utilize α_{i-1} coins locked in the off-chain contract. The opportunity cost is $O(r_{U_{i-1}}, t_{i-1}, \alpha_{i-1})$, also denoted as $o_{i-1}^{t_{i-1},\alpha_{i-1}}$. Additionally, due to closure of channel, U_{i-1} fails to utilize the residual capacity $remain(U_{i-1}, U_i)$ for the next $T - (t_{i-1} +$

$t_{contract\_initiate} - t_{i-1,i})$ unit of time, where $t_{i-1,i}$ is the timestamp at which channel $id_{i-1,i}$ was opened and $t_{contract\_initiate}$ is the current timestamp at which the off-chain contract got initiated in the channel. We use a shorter notation $\tilde{t}_{i-1,i}$ to denote $T - (t_{i-1} + t_{contract\_initiate} - t_{i-1,i})$. The opportunity cost of the remaining balance is $O(r_{U_{i-1}}, T - (t_{i-1} + t_{contract\_initiate} - t_{i-1,i}), remain(U_{i-1}, U_i))$ or $o_{i-1}^{\tilde{t}_{i-1,i}, remain(U_{i-1},U_i)}$. Along with that U_{i-1} has to pay the transaction fee M for settling on-chain. Hence, payoff $u_{U_{i-1}}(uco, F, Gr) = -o_{i-1}^{t_{i-1},\alpha_{i-1}} - o_{n-1}^{\tilde{t}_{i-1,i}, remain(U_{i-1},U_i)} - M$.

- If U_{i-1} had previously transferred coins to U_i then $remain(U_i, U_{i-1}) > 0$. In that case, U_i incurs a loss $O(r_{U_i}, T - (t_{i-1} + t_{contract\_initiate} - t_{i-1,i}), remain(U_i, U_i - 1))$, also denoted as $o_i^{\tilde{t}_{i-1,i}, remain(U_i,U_{i-1})}$, due to closure of channel after timeout period t_{i-1}. Additionally, it loses $o_i^{t_{i-1},\alpha_{i-1}}$, as it fails to earn and utilize the coins for other purpose. Hence, payoff $u_{U_i}(uco, F, Gr) = -o_i^{\tilde{t}_{i-1,i}, remain(U_i,U_{i-1})} - o_i^{t_{i-1},\alpha_{i-1}}$. Since U_i doesn't respond, we do not subtract M from the payoff, as it has not gone on-chain for settling the transaction.

B. **N** has chosen a corrupt node. The latter executes a self-payment of amount α via a path of length n via $n-1$ intermediaries. This implies that $U_0 = U_n$. The amount forwarded is $\alpha + \sum_{i=1}^{n-1} f(\alpha_i)$, where, $f(\alpha_i)$ is the fee charged by an intermediate node $U_i, i \in [1, n-1]$. Since the cost incurred per payment is C and U_n has to keep α coins locked for time D, the bribe offered must compensate for all these costs. The amount of bribe offered by the attacker is L where $L = \alpha + C + I_{D,\alpha}$, where $I_{D,\alpha} \approx 2o_n^{D,\alpha}$. Since the purpose of U_n is to mount attack, it would not be interested in performing payments like other participants. This implies that U_n has not accepted any payment from U_{n-1} and $remain(U_n, U_{n-1}) = 0$. We analyze each case as follows:

I. Instantaneous Response, i.e., $t \to 0$:

a. *U_i or U_n accepts the payment*: U_n ends up losing approximatey $\sum_{i=1}^{n-1} f(\alpha_i)$, as it needs to pay $(n-1)$ intermediaries. It had already incurred a cost C. U_{n-1} has successfully forwarded the amount. Thus, payoffs are $u_{U_{n-1}}(co, F, Ac) = f(\alpha_{n-1})$ and $u_{U_n}(co, F, Ac) = -C - \sum_{i=1}^{n-1} f(\alpha_i)$.

b. *U_i or U_n rejects the payment*: $u_{U_{n-1}}(co, F, Rt) = 0$ and $u_{U_n}(co, F, Rt) = -C$.

II. Delayed Response, i.e., $0 < t < t_{n-1}$ where $t_{n-1} = D$:

a. *U_i or U_n waits and then accepts the payment*:

- Payoff of U_{n-1} is same as the payoff it had obtained when U_n is not corrupt and chooses to wait and accept the payment. Thus $u_{U_{n-1}}(co, F, W \mathbin{\&} Ac) = f(\alpha_{n-1}) - o_{n-1}^{t,\alpha_{n-1}}$.
- $\eta_n^{t,\alpha_{n-1}}$ defines the net profit received by U_n for keeping α_{n-1} coins unutilized till time t, where:

$$\eta_n^{t,\alpha_{n-1}} = \begin{cases} -C - O(r_{U_n}, t, \alpha_{n-1}), & 0 < t < D - \delta \\ L - C - O(r_{U_n}, t, \alpha_{n-1}), & t \geq D - \delta \end{cases} \tag{7.5}$$

Delaying till time $t < D - \delta$, will not result in any profit, U_n loses the setup cost and the revenue had it utilized α_{n-1} for t units of time. If it delays for at least $D - \delta$, it gets paid for the work done, i.e. $\eta_n^{D-\delta,\alpha_{n-1}}$. Since $\delta \to 0$, $\eta_n^{D-\delta,\alpha_{n-1}} \approx \eta_n^{D,\alpha_{n-1}} = L - C - O(r_{U_n}, D, \alpha_{n-1})$. But $I_{D,\alpha_{n-1}} \approx 2o_n^{D,\alpha_{n-1}}$, which implies $L - C - o_n^{D,\alpha_{n-1}} = \alpha + C + I_{D,\alpha_{n-1}} - C - o_n^{D,\alpha_{n-1}} \approx \alpha + o_n^{D,\alpha_{n-1}}$. Upon accepting a self-payment, it ends up paying a processing fee to $n-1$ intermediaries. Thus, the payoff of U_n, $u_{U_n}(co, F, W \mathbin{\&} Ac) = -\sum_{i=1}^{n-1} f(\alpha_i) + \eta_n^{t,\alpha_{n-1}}$.

b. *U_i or U_n waits and then rejects the payment*: Payoff of U_{n-1} and U_n are $u_{U_{n-1}}(co, F, W \mathbin{\&} Rt) = -o_{n-1}^{t,\alpha_{n-1}}$ and $u_{U_n}(co, F, W \mathbin{\&} Rt) = \eta_n^{t,\alpha_{n-1}}$ respectively.

III. *U_i or U_n griefs*: U_n successfully mounts the attack, it gets an incentive L from attacker. It loses C, which is the cost for mounting the attack and the opportunity cost $o_n^{D,\alpha_{n-1}}$. Since the channel is unilaterally funded by U_{n-1}, $remain(U_n, U_{n-1}) = 0$. Thus there is no loss associated due to closure of channel. The payoff of U_{n-1} is the same as the payoff it had obtained when U_n is uncorrupt and chooses to grief. Thus, $u_{U_{n-1}}(co, F, Gr) = -o_{n-1}^{D,\alpha_{n-1}} - o_{n-1}^{\tilde{t}_{n-1,n}, remain(U_{n-1},U_n)} - M$ and $u_{U_n}(co, F, Gr) = L - C - o_n^{D,\alpha_{n-1}} = \eta_n^{D,\alpha_{n-1}}$.

7.3.4 Game Analysis

We infer from the payoff model that the corrupt node can select either of the strategies for mounting the attack:

- Reject the payment just before lock time D elapses: $U_n i$ (or U_n) rejects the conditional payment forwarded by U_{i-1} just before the contract's lock time elapses. With high probability, U_{i-1} will choose to co-operate and settle the transaction off-chain. The benefit is that U_i can go on executing several instances of griefing attacks without the need of opening a fresh channel each time.

- Do not respond: This is as per the conventional definition of griefing. U_i (or U_n) chooses not to respond, and U_{i-1} closes the channel unilaterally after the contract's lock time expires.

We assume that the corrupt node applies a mixed strategy over these two strategies, selecting the first strategy with probability q and the second strategy with probability $1-q, q \in [0,1]$.

Given $U'_{i-1}s$ belief that **N** selects a corrupt U_i with probability θ_i and an uncorrupt U_i with probability $1-\theta_i$, the expected payoff of U_{i-1} is calculated by applying backward induction on the game tree Γ_{HTLC} shown in Fig. 7-1. If U_{i-1} plays *NF*, then U_i cannot take any action, hence both gets a payoff of 0. If U_{i-1} plays *F*; an *uncorrupt* U_i chooses *Ac* as its best response since $u_{U_i}(uco, F, Ac) \geq u_{U_i}(uco, F, s'), \forall s' \in S_{U_i}$; a corrupt U_i (also U_n) can choose either to *grief* or *Wait & Reject* at $D-\delta$ as its best response since $u_{U_i}(co, F, Gr) = u_{U_i}(co, F, W \text{ \& } Rt$ at time $D-\delta) \geq u_{U_i}(co, F, s''), \forall s'' \in S_{U_n}$. A corrupt U_i (or U_n) applies mixed strategy, either choosing to *Grief* with probability $1-q$ or it can *Wait and Reject* at time $D-\delta$ with probability q. The expected payoff of U_{i-1} for selecting *forward*, denoted as $\mathbb{E}_{U_{i-1}}(F)$, and expected payoff for selecting *not forward*, denoted as $\mathbb{E}_{U_{i-1}}(NF)$ are:

$$\begin{gathered}\mathbb{E}_{U_{i-1}}(F) = \theta_i\Big(- q o_{n-1}^{D-\delta,\alpha_{n-1}} + (1-q)(-o_{n-1}^{D,\alpha_{n-1}} - o_{n-1}^{\tilde{t}_{n-1,n},remain(U_{n-1},U_n)} - M)\Big) \\ +(1-\theta_i)f(\alpha_{i-1}) \\ \mathbb{E}_{U_i}(NF) = 0\end{gathered} \tag{7.6}$$

Since $\delta \to 0$, we consider $o_{n-1}^{D-\delta,\alpha_{n-1}} \approx o_{n-1}^{D,\alpha_{n-1}}$. Substituting in Eq. 7.6, if $\mathbb{E}_{U_{i-1}}(F) > \mathbb{E}_{U_{i-1}}(NF)$ then $U'_{i-1}s$ best response is *F* and we derive $\theta_i < \frac{f(\alpha_{i-1})}{o_{n-1}^{D,\alpha_{n-1}}+f(\alpha_{i-1})+(1-q)(o_{n-1}^{\tilde{t}_{n-1,n},remain(U_{n-1},U_n)}+M)}$. Hence, U_{n-1} chooses *Forward* if $\theta_i < \frac{f(\alpha_{i-1})}{o_{n-1}^{D,\alpha_{n-1}}+f(\alpha_{i-1})+(1-q)(o_{n-1}^{\tilde{t}_{n-1,n},remain(U_{n-1},U_n)}+M)}$, else it chooses *Not Forward*; *corrupt* U_n can either choose *Grief* or *Wait & Reject* at time $D-\delta$; *uncorrupt* U_n chooses *Accept*; is a perfect Bayesian equilibrium.

Comparative Static Analysis. We take the derivative of $\mathbb{E}_{U_{i-1}}(F)$ with respect to θ_i, to observe how the expected payoff of U_{i-1} changes with small change in θ_i, keeping other variables constant.

$$\frac{\mathrm{d}\mathbb{E}_{U_{i-1}}(F)}{\mathrm{d}\theta_i} = -f(\alpha_{i-1}) - o_{n-1}^{D,\alpha_{n-1}} - (1-q)(o_{n-1}^{\tilde{t}_{n-1,n},remain(U_{n-1},U_n)} + M) \tag{7.7}$$

$\frac{\mathrm{d}\mathbb{E}_{U_{i-1}}(F)}{\mathrm{d}\theta_i} < 0$ implies that $\mathbb{E}_{U_{i-1}}(F)$ is a decreasing function upon varying θ_i. The higher the probability of selecting a corrupt node, the expected payoff of U_{i-1} will go down because of the probability of facing an attack increases. The loss incurred is mainly due to delay in resolution of payment and closure of channel (in some cases), as fee $f(\alpha_{i-1})$ is negligible compared to the other two factors. In other words, if θ_i decreases, the higher the chance an *uncorrupt* recipient gets selected. $U'_{i-1}s$ expected payoff will increase.

7.4 Countermeasure for the griefing attack

Several countermeasures have been proposed to counter griefing attacks. Upfront payments for incentivizing rational nodes for faster resolution of payments have been discussed in [16]. However, this scheme introduces a huge amount of economic barriers for a payer. Also, it will never demotivate an attacker from mounting the attack as it will never lose any coins. Proof-of-Closure of channels was proposed in [8] where two timeouts were set for ab *HTLC*, a hard timeout, and a soft timeout period. The countermeasure does not work because the malicious node does not lose anything in the process. In the next section, we discuss another payment protocol, Hashed Timelock Contract with Griefing Penalty or *HTLC-GP*, that claims to address the above shortcomings and disincentivize griefing attacks.

7.4.1 Hashed Timelock Contract with Griefing Penalty or HTLC-GP

The protocol *HTLC-GP* [97], [96] has been developed after modifying *HTLC* to counter griefing attack. The underlying idea is that if the party griefs, it gets penalized, and the amount locked is distributed as compensation amongst the affected parties. The total penalty charged is proportional to the summation of the collateral locked in each channel, forming the path routing payment. *Collateral locked* in a channel by an off-chain contract is the product of coins locked in the off-chain contract and timeout period of the contract. We provide a high-level overview of the protocol HTLC-GP in the next section.

7.4.1.1 Overview

If the party wants to cancel the conditional payment in *HTLC*, it may simply request the counterparty forwarding the contract to cancel it mutually. The counterparty complies with the request, as it has no intention of keeping the coins unutilized. To incorporate penalty in *HTLC*, both parties must lock coins. The condition is that a party that has formed the contract for claiming a conditional payment needs to resolve it within a given time. Failure to respond will result in the loss of coins. There is no way for a party to unilaterally cancel the conditional payment by going on-chain in *HTLC*. The counterparty may take advantage of this situation and refuse to cancel the contract mutually. After the lock time elapses, the counterparty goes on-chain and claims the penalty. The problem is termed *reverse griefing*. Thus, a new protocol termed *HTLC-GP* was proposed. It is a two-round protocol where the first round involves locking of penalty, termed as *Cancellation round*. The round is initiated by the payee and proceeds in the reverse direction. The penalty is locked by the party as a guarantee against a payment to be forwarded by the counterparty in the next round. The second round is termed as *Payment round*. It involves locking the payment value in off-chain contracts from payer to payee.

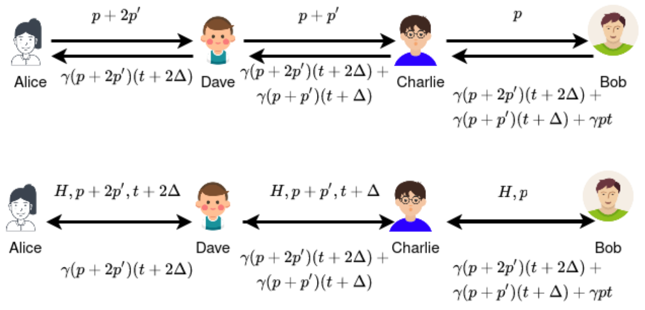

$p + 2p'$
$p + p'$
p
Alice
$\gamma(p + 2p')(t + 2\Delta)$
Dave
$\gamma(p + 2p')(t + 2\Delta) +$
$\gamma(p + p')(t + \Delta)$
Charlie
Bob
$\gamma(p + 2p')(t + 2\Delta) +$
$\gamma(p + p')(t + \Delta) + \gamma pt$
$H, p + 2p', t + 2\Delta$
$H, p + p', t + \Delta$
H, p
Alice
Dave
Charlie
Bob
$\gamma(p + 2p')(t + 2\Delta)$
$\gamma(p + 2p')(t + 2\Delta) +$
$\gamma(p + p')(t + \Delta)$
$\gamma(p + 2p')(t + 2\Delta) +$
$\gamma(p + p')(t + \Delta) + \gamma pt$

Figure 7-2: Formation of contract in *HTLC-CP*

We explain the protocol with the help of an example shown in Fig. 7-2. *Alice* wants to transfer b units to *Bob*. Each party that forwards a payment must be guaranteed by its counterparty to receive compensation if there is an incidence of a griefing attack. The amount of compensation charged must be proportional to the collateral locked in the path. We define this proportionality constant as the *rate of griefing-penalty per unit time*, denoted as γ.

The first round termed as *Cancellation round* proceeds in the following way: *Bob* has to lock $\gamma b D_1 + \gamma b D_2 + \gamma b D$ coins for D unit of time. This amount is the cumulative penalty to be distributed among *Alice, Dave* and *Charlie*, if Bob griefs. After *Charlie* receives the cancellation contract, he locks $\gamma b D_1 + \gamma b D_2$ in the contract formed with *Dave* for D_2 units. The latter locks $\gamma b D_1$ coins in the contract formed with *Alice* for D_1 units of time. The second round termed as *Payment round* proceeds in a similar way like in HTLC. Payment value b is forwarded from *Alice* to *Bob* via the intermediaries.

Calculation of cumulative penalty In the example, *Bob* is required to lock $\gamma b D_1 + \gamma b D_2 + \gamma b D$ coins as a guarantee against the payment amount to be forwarded by *Charlie*. Since the lock time of the contract is D, one might question why *Bob* must take into account the lock time of the other contracts while locking penalty. We do so to prevent other intermediate parties from becoming a victim of reverse griefing. Consider the situation where *Bob* locks $3\gamma bD$ coins as penalty, *Charlie* locks $2\gamma bD_2$ as penalty and *Dave* locks γbD_1. If *Bob* griefs, *Charlie* keeps the compensation γbD and forwards $2\gamma bD$ to *Dave*. *Dave* is greedy and refuses to cancel the off-chain contract with *Charlie*. After elapse of D_2, it goes on-chain and claims $2\gamma D_2$. Since $D_2 > D$, *Charlie* incurs a loss of $2\gamma b(D_2 - D)$. Thus, we account for the lock time of each contract while calculating compensation to prevent the loss of coins of uncorrupt parties.

Countering Griefing Attack Suppose *Bob* griefs. He will pay a compensation of $\gamma bD_1+\gamma bD_2+\gamma bD$ units to *Charlie*, as per the terms of the contract. After D expires, *Charlie* goes on-chain. He closes the channel, unlocks b coins, and claims compensation. He requests *Dave* to cancel the off-chain contract, offering a compensation of $\gamma bD_1 + \gamma bD_2$. *Dave* cancels the contract off-chain, unlocks b units from the contract, and claims the compensation from *Charlie*. If *Charlie* decides to grief, *Dave* can claim the compensation by going on-chain and closing the channel. *Dave* requests *Alice* to cancel the contract by offering a compensation of γbD_1. Except for *Bob*, none of the parties lose coins.

In the next section, we formulate a game model for griefing attacks in HTLC-GP and study its effectiveness.

7.5 Analysis of Griefing Attack in HTLC-GP

7.5.1 System Model

For payment of α from U_0 to U_n via $(n-1)$ intermediaries, we denote the cumulative compensation to be locked by U_i if U_{i-1} forwards α_{i-1} as $Z_{\alpha_{i-1},i}, i \in [1, n]$ where $Z_{\alpha_{i-1},i} = Z_{\alpha_{i-2},i-1} + c_{\alpha_{i-1},i-1}$; $c_{\alpha_{i-1},i-1}$ is the compensation charged by node U_{i-1} and $Z_{\alpha_{i-2},i-1}$ is used for compensating other nodes $U_j, j < i$. Note that $Z_{val,0} = 0, \forall val \in \mathbb{R}^+, \alpha_{n-1} = \alpha$. $c_{\alpha_{i-1},i-1}$ charged by a node U_{i-1}, must be proportional to the collateral it has locked in the off-chain contract formed with node U_i for timeout period $t_{i-1}, i \in [1, n]$. $t_{i-1} = t_i + \Delta$, $t_{n-1} = D$ and $c_{\alpha_{i-1},i-1} = \gamma\alpha_{i-1}t_{i-1}$, γ being the rate of griefing-penalty per unit time. An off-chain contract between node U_{i-1} and U_i requires U_{i-1} locking an α_{i-1} coins and U_i must lock $Z_{\alpha_{i-1},i}$ coins. Here $Z_{\alpha_{n-1},n}$ denoted as $Z_\alpha = \sum\limits_{i=0}^{n-1} c_{\alpha_i,i}$ is locked by U_n as guaranteed compensation against the amount locked by party U_{n-1}. Again,

U_{n-1} has locked $Z_{\alpha_{n-2},n-1}$ with the contract formed with U_{n-2} for time t_{n-1}.

7.5.1.1 Change in System Assumption

In HTLC-GP, we consider a payment channel to be dual-funded. This implies that even in channel $id_{i-1,i}, i \in [1,n]$ both the parties U_{i-1} and U_i lock coins i.e., $locked(U_{i-1}, U_i) > 0$ and $locked(U_i, U_{i-1}) > 0$.

7.5.2 Attacker Model for HTLC-GP

The cost of the attack increases as U_n has to lock extra coins as guarantee. However, the attacker does not increase the incentive offered per attack. Thus, U_n will not lock more than α coins. The former will distribute this amount between the cumulative penalty locked in the contract formed with U_{n-1} and the amount to be forwarded for payment. This implies α is the summation of transaction value $v + \sum\limits_{i=1}^{n-1} f(v_i)$ and the cumulative penalty imposed $Z_v = \sum\limits_{i=0}^{n-1} c_{v_i,i} = \gamma \sum\limits_{j=0}^{n-1} v_j t_j$ where $v_j = v_0 - \sum\limits_{k=1}^{j}, j \in [0, n-1]$. Since $\sum\limits_{i=1}^{n-1} f(v_i) << v$, we consider $v_0 + Z_v \approx v + Z_v = \alpha$ or, $v = \frac{\alpha}{1+\gamma \sum\limits_{j=0}^{n-1} t_j}$.

U_n executes a self-payment of v coins, choosing a path of length n. Each intermediate party and the recipient U_n have to lock coins as a guarantee against the payment amount forwarded.

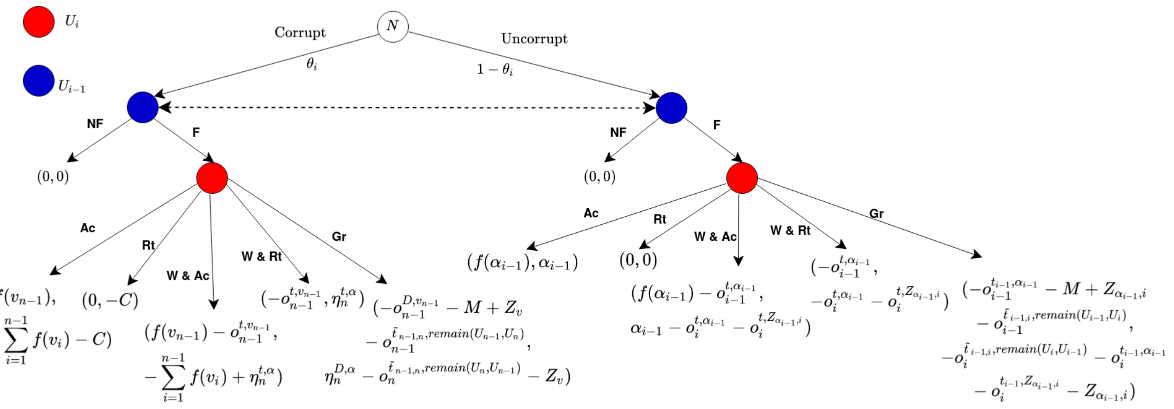
7.5.3 Game Model

It is feasible adapt the game model Γ_{HTLC} used for *HTLC* in *HTLC-GP*. We study the interaction between U_{i-1} and U_i as a *sequential Bayesian game* $\Gamma_{HTLC-GP}$, shown in Fig. 7-3.

7.5.3.1 Payoff Model

When U_{i-1} chooses not to forward, both U_{i-1} and U_i receives a payoff 0 since no off-chain contract got established, i.e., $u_{U_{i-1}}(\theta_b, NF, s_b) = u_{U_i}(\theta_b, NF, s_b) = 0, \theta_b \in \Theta_{U_i}$ and $s_b \in S_{U_i}$. We analyze the payoff of each case when U_{i-1} chooses to forward:

A. **N** had chosen an *uncorrupt* U_i. We analyze the payoff of each case:



U_i
U_{i-1}
N
Corrupt
θ_i
Uncorrupt
$1-\theta_i$
NF
F
$(0,0)$
Ac
Rt
W & Ac
W & Rt
Gr
$(f(v_{n-1}), -\sum_{i=1}^{n-1} f(v_i) - C)$
$(0, -C)$
$(f(v_{n-1}) - o_{n-1}^{t,v_{n-1}}, -\sum_{i=1}^{n-1} f(v_i) + \eta_n^{t,\alpha})$
$(-o_{n-1}^{t,v_{n-1}}, \eta_n^{t,\alpha})$
$(-o_{n-1}^{D,v_{n-1}} - M + Z_v - o_{n-1}^{\tilde{t}_{n-1,n},remain(U_{n-1},U_n)}, \eta_n^{D,\alpha} - o_n^{\tilde{t}_{n-1,n},remain(U_n,U_{n-1})} - Z_v)$
$(f(\alpha_{i-1}), \alpha_{i-1})$
$(0,0)$
$(f(\alpha_{i-1}) - o_{i-1}^{t,\alpha_{i-1}}, \alpha_{i-1} - o_i^{t,\alpha_{i-1}} - o_i^{t,Z_{\alpha_{i-1},i}})$
$(-o_{i-1}^{t,\alpha_{i-1}}, -o_i^{t,\alpha_{i-1}} - o_i^{t,Z_{\alpha_{i-1},i}})$
$(-o_{i-1}^{t_{i-1},\alpha_{i-1}} - M + Z_{\alpha_{i-1},i} - o_{i-1}^{\tilde{t}_{i-1,i},remain(U_{i-1},U_i)}, -o_i^{\tilde{t}_{i-1,i},remain(U_i,U_{i-1})} - o_i^{t_{i-1},\alpha_{i-1}} - o_i^{t_{i-1},Z_{\alpha_{i-1},i}} - Z_{\alpha_{i-1},i})$

Figure 7-3: Extensive form of the game $\Gamma_{HTLC-GP}$

I. Instantaneous Response, i.e., $t \to 0$: Upon instant acceptance or rejection of payment, the payoffs are the same as that observed in Γ_{HTLC}.

II. Delayed Response, i.e., $0 < t < t_i$:

a. *U_i waits and then accepts the payment*:

- U_{i-1} incurs loss due to rise in opportunity cost of α_{i-1} coins locked in off-chain contract with U_i. Thus $u_{U_{i-1}}(uco, F, W \,\&\, Ac) = f(\alpha_{i-1}) - o_{i-1}^{t,\alpha_{i-1}}$.
- U_i has to keep $Z_{\alpha_{i-1},i}$ coins locked in the contract established in channel $id_{i-1,i}$. Thus, it faces loss not only due to delay in claiming α_{i-1} coins from U_{i-1} but also due to unutilization of $Z_{\alpha_{i-1},i}$. The expected profit that could have been made using α_{i-1} and $Z_{\alpha_{i-1},i}$ within the next t unit of time is $O(r_{U_i}, t, \alpha_{i-1})$, also denoted as $o_i^{t,\alpha_{i-1}}$, and $O(r_{U_i}, t, Z_{\alpha_{i-1},i})$, denoted as $o_i^{t,Z_{\alpha_{i-1},i}}$. Thus, $u_{U_i}(uco, F, W \,\&\, Ac) = \alpha_{i-1} - o_i^{t,\alpha_{i-1}} - o_i^{t,Z_{\alpha_{i-1},i}}$.

b. *U_i waits and then rejects the payment*: The payoff of U_{i-1} is $u_{U_{i-1}}(uco, F, W \,\&\, Rt) = -o_{i-1}^{t,\alpha_{i-1}}$ and payoff of U_i is $u_{U_i}(uco, F, W \,\&\, Rt) = -o_i^{t,\alpha_{i-1}} - o_i^{t,Z_{\alpha_{i-1},i}}$.

III. *U_i griefs*:

- Even in this case, due to closure of channel U_{i-1} fails to utilize the residual capacity $remain(U_{i-1}, U_i)$ for the next $T - (t_{i-1} + t_{contract\_initiate} - t_{i-1,i})$ unit of time. U_{i-1} incurs a loss of $o_{i-1}^{t_{i-1},\alpha_{i-1}} + o_{i-1}^{\tilde{t}_{i-1,i},remain(U_{i-1},U_i)} + M$. However, it can claim a compensation of $Z_{\alpha_{i-1},i}$ by going on-chain and closing the channel. Payoff $u_{U_{i-1}}(uco, F, Gr) = -(o_{i-1}^{t_{i-1},\alpha_{i-1}} + o_{i-1}^{\tilde{t}_{i-1,i},remain(U_{i-1},U_i)} + M) + Z_{\alpha_{i-1},i}$.
- U_i incurs loss of $Z_{\alpha_{i-1},i}$ coins to compensate U_{i-1}. It fails to earn revenue due to non-utilization of $Z_{\alpha_{i-1},i}$ coins in channel $id_{i-1,i}$ for period t_{i-1}. The additional loses suffered

are due to inability to claim α_{i-1} coins and using it within t_{i-1} unit of time and failure in utilizing the residual capacity $remain(U_i, U_{i-1})$ for the next $T-(t_{i-1}+t_{contract\_initiate}-t_{i-1,i})$ unit of time. Thus, payoff of U_i is $u_{U_i}(uco, F, Gr) = -o_i^{\tilde{t}_{i-1,i},remain(U_i,U_{i-1})} - o_i^{t_{i-1},\alpha_{i-1}} - o_i^{t_{i-1},Z_{\alpha_{i-1},i}} - Z_{\alpha_{i-1},i}$.

B. **N** has chosen a corrupt node. The latter executes a self payment of amount v, as stated in Attacker Model in Section 7.5.2. The payoffs for each case has been defined as follows:

I. Instantaneous Response, i.e., $t \to 0$: Upon instant acceptance or rejection of payment, the payoffs are the same as that observed in Γ_{HTLC}.

II. Delayed Response, i.e., $0 < t < D$:

a. *U_i or U_n waits and then accepts the payment*:

- Payoff of U_{n-1} is $u_{U_{n-1}}(co, F, W \ \& \ Ac) = f(v_{n-1}) - o_{n-1}^{t,v_{n-1}}$, due to delay in claiming of v_{n-1} coins as U_n delays in resolving payment.
- U_n keeps $v + Z_v \approx \alpha$ coins locked for mounting the attack and receives a bribe L. The value $\eta_n^{t,\alpha}$ is the same as defined in Eq.7.5. Upon accepting a self-payment of amount v, the corrupt node ends up paying a processing fee to $n-1$ intermediaries. Thus, the payoff of U_n is $u_{U_n}(co, F, W \ \& \ Ac) = -\sum_{i=1}^{n-1} f(v_i) + \eta_n^{t,\alpha}$.

b. *U_i or U_n waits and then rejects the payment*: Payoff of U_{n-1}, $u_{U_{n-1}}(co, F, W \ \& \ Rt) = -o_{n-1}^{t,v_{n-1}}$ and $u_{U_n}(co, F, W \ \& \ Rt) = \eta_n^{t,\alpha}$. When $t \approx D - \delta$ where $\delta \to 0$, $\eta_n^{t,\alpha}$ attains the maximum value.

III. *U_i or U_n griefs*: U_n successfully mounts the attack, it gets an incentive L from the attacker, but at the same time loses Z_v in order to compensate the affected parties. U_{n-1} loses channel and the expected revenue due to coins remaining locked but gets the compensation Z_v. Thus, the payoffs of U_{n-1} and U_n are $u_{U_{n-1}}(co, F, Gr) = -o_{n-1}^{D,v_{n-1}} - o_{n-1}^{\tilde{t}_{n-1,n},remain(U_{n-1},U_n)} - M + Z_v$ and $u_{U_n}(co, F, Gr) = L - C - o_n^{D,\alpha} - Z_v - o_{n-1}^{\tilde{t}_{n-1,n},remain(U_n,U_{n-1})} = \eta_n^{D,\alpha} - Z_v - o_n^{\tilde{t}_{n-1,n},remain(U_n,U_{n-1})}$ respectively.

7.5.4 Game Analysis

The expected payoff of U_{i-1} is calculated by applying backward induction to the game tree $\Gamma_{HTLC-Penalty}$ shown in Fig. 7-3. If U_{i-1} plays *NF*, then U_i cannot take any action, hence both gets a payoff of 0. If U_{i-1} plays *F*; an *uncorrupt* U_i chooses *Ac* as its best response but a corrupt U_i will choose *Wait & Reject* at $D-\delta$ as its best response since $u_{U_i}(co, W \ \& \ Rt$ at time $D-\delta) \geq u_{U_i}(co, F, s''), \forall s'' \in$

S_{U_i}. The expected payoff of U_{i-1} for selecting *forward*, denoted as $\mathbb{E}_{U_{i-1}}(F)$, and expected payoff for selecting *not forward*, denoted as $\mathbb{E}_{U_{i-1}}(NF)$ are:

$$\begin{aligned} \mathbb{E}_{U_{i-1}}(F) &= \theta_i(-o_{n-1}^{D-\delta,v_{n-1}}) + (1-\theta_i)f(\alpha_{i-1}) \\ \mathbb{E}_{U_i}(NF) &= 0 \end{aligned} \tag{7.8}$$

Since $\delta \to 0$, we consider $o_{n-1}^{D-\delta,v_{n-1}} \approx o_{n-1}^{D,v_{n-1}}$. Substituting in Eq. 7.8, if $\mathbb{E}_{U_{i-1}}(F) > \mathbb{E}_{U_i}(NF)$, then U_{i-1} chooses F and we derive $\theta_i < \frac{f(\alpha_{i-1})}{f(\alpha_{i-1})+o_{n-1}^{D,v_{n-1}}}$. Hence, U_{i-1} chooses *Forward* if $\theta_i < \frac{f(\alpha_{i-1})}{f(\alpha_{i-1})+o_{n-1}^{D,v_{n-1}}}$, else it chooses *Not Forward*; *corrupt* U_i chooses *Wait & Reject* at time $D-\delta$; *uncorrupt* U_i chooses *Accept*; is a perfect Bayesian equilibrium.

Comparative Static Analysis. We take the partial derivative of $\mathbb{E}_{U_{i-1}}(F)$ with respect to θ_i, to observe how the expected payoff of U_{i-1} changes with small change in θ_i, keeping other variables constant.

$$\frac{\mathrm{d}\mathbb{E}_{U_{i-1}}(F)}{\mathrm{d}\theta_i} = -f(\alpha_{i-1}) - o_{n-1}^{D,v_{n-1}} \tag{7.9}$$

$\frac{\mathrm{d}\mathbb{E}_{U_{i-1}}(F)}{\mathrm{d}\theta_i} < 0$ implies that $\mathbb{E}_{U_{i-1}}(F)$ is a decreasing function upon varying θ_i. It implies a higher probability of selecting a corrupt node, the expected payoff of U_{i-1} will go down as the probability of facing an attack increases. However, U_{i-1} doesn't lose the channel, unlike in the game Γ_{HTLC}, where U_{i-1} had to settle on-chain if the corrupt node did not respond.

Comparing θ_i for which U_{i-1} chooses F *in Γ_{HTLC} and $\Gamma_{HTLC-GP}$:* Since $f(\alpha_{i-1}) + o_{n-1}^{D,v_{n-1}} < o_{n-1}^{D,\alpha_{n-1}} + f(\alpha_{i-1}) + (1-q)(o_{n-1}^{\tilde{t}_{n-1,n},remain(U_{n-1},U_n)} + M)$, the cut-off of θ_i for which U_{i-1} will choose to forward a payment is higher in $\Gamma_{HTLC-GP}$ than in Γ_{HTLC} even if $q \to 0$. The corrupt player has to invest some amount as penalty and as a consequence, the payment amount reduces from α_{n-1} to v_{n-1}. Additionally, the corrupt player now fixes its strategy to cancel the payment just before elapse of locktime. Hence, the player U_{i-1} is not bound to go on-chain and unlock the coins from the off-chain contract.

7.5.5 Gap in Security Analysis of HTLC-GP

The security goal of *HTLC-GP* [96] states that the protocol punishes the attacker and compensates the victims. A detailed security analysis has been provided in the paper but it works under the assumption that the attacker will not respond and the victim will go on-chain to claim the penalty. To avoid paying the griefing penalty, a rational corrupt node will cancel the payment just before the contract lock time elapses i.e., at time $D - \delta$, where $\delta \to 0$ [8]. Such a node doesn't lose its channel, providing the advantage of mounting the attack repeatedly.

The uncorrupt parties cannot be protected from griefing attacks if we do not penalize the corrupt node for each time interval. Since Bitcoin scripting language is not Turing-complete, we cannot have a single off-chain contract where we can define penalty as a function of time. There is no way to execute a transaction like this: *If t' time units have elapsed, pay amount p. If t'+1 time units have elapsed, pay amount* $p + \delta$. CheckSequenceVerify [39] imposed on the first condition of elapse of t' time unit makes the transaction eligible for broadcasting on-chain after elapse of time t'+1. This might lead to a race condition, and the victim might not receive proper compensation. Thus it is not possible to design a protocol to compensate the victims affected by other forms of griefing attacks with the current scripting language.

Until now, our focus was on whether a corrupt party is losing coins if it attacks the network. We did not analyze the impact of the penalty on the attacker's behavior. The attacker will continue to invest in the network if the return on investment is good enough. If the return on investment diminishes, the attacker will refrain from mounting the attack and instead prefer to invest in another activity. Based on this idea, we analyze the effectiveness of *HTLC-GP* in the next section.

7.5.6 Effectiveness of HTLC-GP

The attacker has an objective of maximizing the damage by locking as much network liquidity possible for the given budget $\mathcal{B}_{EX}$. Penalty involves locking extra coins, and this increases the cost of the attack. Once the cost of the attack increases, the attacker will be able to corrupt fewer nodes. We define a metric that measures the coins remaining unutilized in the network.

Measuring success rate of attack *Capacity locked* in a path routing payment is the summation of the coins locked in the off-chain contract instantiated on the channel forming the path. It is the metric used for measuring the success of the attack from the attacker's point of view [89], [90].

Ignoring the processing fee (negligible quantity), assuming all the payments executed are of value α and the bribe offered per instance is L, the attacker can corrupt $\frac{\mathcal{B}_{EX}}{L}$ nodes in the networks. We also assume that for any node $U \in V$, $\mathbb{E}_{\mathbb{U}}(\text{F}) > \mathbb{E}_{\mathbb{U}}(\text{NF})$. So each self-payment gets routed and reaches the payee.

Claim 7.0.1. Given the total budget of the attack is $\mathcal{B}_{EX}$, incentive per attack being L, transaction value per payment being α, HTLC timeout period is D, time taken to settle a transaction on-chain being Δ, n is the maximum allowed path length and a corrupt recipient rejects the payment at time $t' = D - \delta$, where $\delta \to 0$, the capacity locked upon using HTLC-GP is less than the capacity locked in HTLC, the loss percent being $\frac{\gamma n(\frac{D}{2}+\frac{\Delta(n-2)}{6})}{1+\gamma n(D+\frac{(n-1)\Delta}{2})}$

Proof: In *HTLC*, a given instance of attack locks $(n-1)\alpha$ coins in the path routing payment. The capacity locked in $\frac{\mathcal{B}_{EX}}{L}(n-1)\alpha$.

In *HTLC-GP*, U_n executes self-payment of amount v. It cancels the contract at time $t' = D-\delta$. Capacity locked is $((n-1)v + \sum\limits_{i=1}^{n-1} Z_{v_{i-1},i})\frac{\mathcal{B}_{EX}}{L}$. In both the cases, we exclude the coins locked by the corrupt node while computing the unusable capacity and fee charged by the intermediate parties. Thus we substitute $Z_{v_{i-1},i}$ with $Z_{v,i}$. We measure the difference in capacity locked to judge the effectiveness.

$$\begin{aligned}
&\tfrac{\mathcal{B}_{EX}}{L}(n-1)\alpha - \big(\tfrac{\mathcal{B}_{EX}}{L}(n-1)v + \tfrac{\mathcal{B}_{EX}}{L}\sum_{i=1}^{n-1} Z_{v,i}\big)\\
&\quad = \tfrac{\mathcal{B}_{EX}}{L}((n-1)\alpha - (n-1)v - \sum_{i=1}^{n-1} Z_{v,i})\\
&\quad = \tfrac{\mathcal{B}_{EX}}{L}((n-1)\alpha - (n-1)v - \gamma v\sum_{i=0}^{n-1}\sum_{j=0}^{i} t_j)
\end{aligned} \tag{7.10}$$

Substituting $v = \frac{\alpha}{1+\gamma\sum\limits_{j=0}^{n-1} t_j}$,

$$= \gamma\frac{\alpha}{1+\gamma(nD+\frac{n(n-1)\Delta}{2})}n(n-1)\frac{\mathcal{B}_{EX}}{L}(\frac{D}{2}+\frac{\Delta(n-2)}{6}) \tag{7.11}$$

The loss percent is the ratio of the difference between capacity locked in *HTLC* and *HTLC-GP* and capacity locked in *HTLC*.

$$\begin{aligned}
&\frac{\gamma\frac{\alpha}{1+\gamma(nD+\frac{n(n-1)\Delta}{2})}n(n-1)\frac{\mathcal{B}_{EX}}{L}(\frac{D}{2}+\frac{\Delta(n-2)}{6})}{\frac{\mathcal{B}_{EX}}{L}(n-1)\alpha}\\
&\qquad = \frac{\gamma n(\frac{D}{2}+\frac{\Delta(n-2)}{6})}{1+\gamma n(D+\frac{(n-1)\Delta}{2})}
\end{aligned} \tag{7.12}$$

The loss percent of capacity locked by the attacker is $\frac{\gamma n(\frac{D}{2}+\frac{\Delta(n-2)}{6})}{1+\gamma n(D+\frac{(n-1)\Delta}{2})}$. The value is greater than 0 for $n \geq 2$. This proves that attacker ends up locking less capacity when HTLC-GP is used as a payment protocol. ■

Inference We observe that the loss percent $\frac{\gamma n(\frac{D}{2}+\frac{\Delta(n-2)}{6})}{1+\gamma n(D+\frac{(n-1)\Delta}{2})}$ is dependent on γ. If γ is too low, the loss percent is not substantial, then the attacker can still go ahead with corrupting the nodes in the network. Hence the payment protocol *HTLC-GP* is *weakly effective* in disincentivizing the attack.

7.6 Guaranteed Minimum Compensation for further disincentivizing Griefing Attack

If the incidence of griefing attack increases, then γ can be increased. However, the disadvantage of increasing the rate of griefing penalty means uncorrupt nodes have to put a higher amount at stake for routing small-valued payments. The success rate of payments decreases due to a lack of liquidity in the channels. Our objective is to increase the cost of the attack without forcing uncorrupt parties to lock high penalties. The objective can be achieved if the maximum path length allowed for routing payments is decreased accordingly. *Mizrahi and Zohar* [100] suggested that decreasing the maximum path length for routing payments increases the cost of the attack. However, an abrupt reduction in the maximum allowed path length may lead to higher failure in executing transactions. Thus we must design a mechanism by which one can adjust the path length based on the rate of incidence of griefing attacks in the network.

The major source of earning for a node is the processing fee by routing transactions. If there is a griefing attack, then the affected parties fail to earn due to locked collateral. We introduce a new parameter ζ, termed as *Guaranteed Minimum Compensation*. If payment forwarded by U_{i-1} is α, U_i must lock a minimum cumulative penalty $i\zeta\alpha, i \in [1, n]$, where each party $U_j, j \in [0, i)$ is entitled to receive a compensation $\zeta\alpha$. Based on the data provided [37], the fee earned by each node on a single day is quite low compared to the amount routed. Thus, we set ζ in the range $[0, 1)$ to avoid over-compensation. For a given rate of penalty, the maximum cumulative penalty, $Z_{\alpha,max}$, that U_n is required to lock, is $\gamma\alpha \sum_{i=1}^{n}(D + (n - i)\Delta)$ [6]. If $\gamma \sum_{i=1}^{n}(D + (n - i)\Delta)$ is k, where $k \in \mathbb{R}^+$ then $Z_{\alpha,max} = k\alpha$.

[6]For ease of analysis, we ignore the processing fee charged by each intermediate party

7.6.1 Adjusting Maximum Path Length

We estimate the maximum allowed path length, $\tilde{n}$, for routing payments when a minimum threshold on compensation ζ is imposed.

Proposition 7.1. Given the maximum cumulative griefing-penalty for a payment α is $k\alpha$, and the guaranteed minimum compensation ζ, the maximum path length for routing transaction is $\frac{k}{\zeta}$.

Proof: In *HTLC-GP*, $Z_{\alpha,max} = k\alpha$. Upon introduction of *guaranteed minimum compensation*, any node routing a payment α is entitled to a minimum compensation $\zeta\alpha$ upon being affected by the griefing attack. If the compensation falls below this amount, the node will refuse to forward the contract. Thus, the recipient must bear a cumulative penalty of at least $\zeta\tilde{n}\alpha$ for a path of length $\tilde{n}$. Thus, the following criteria must hold:

$$\begin{aligned} &\tilde{n}\zeta\alpha \leq Z_{\alpha,max} \\ &or, \tilde{n}\zeta\alpha \leq k\alpha \qquad (7.13) \\ &or, \tilde{n} \leq \frac{k}{\zeta} \end{aligned}$$

Thus the maximum path length in HTLC-GP$^{\zeta}$ is $\frac{k}{\zeta}$. ■

If the incidence of griefing attack increases, then the maximum allowed path length can be decreased by increasing ζ.

7.6.2 Estimating Rate of Griefing-Penalty $\gamma^{\zeta,k}$

Once the maximum path length is adjusted, the rate of griefing-penalty γ is no more a free variable that can be set too high or too low. Given that $k\alpha$ is the maximum cumulative penalty for routing a payment of α, and the path length decreases from n to $\tilde{n}$ then γ must be increased. We call this new rate of griefing-penalty $\gamma^{\zeta,k}$ and calculate the rate of griefing-penalty based on these factors.

Proposition 7.2. Given ζ as the guaranteed minimum compensation, ratio of maximum cumulative griefing penalty and the transaction amount is k, and the least timeout period of the off-chain contract being D, the rate of griefing-penalty $\gamma^{\zeta,k}$ is $\frac{2\zeta^2}{2\zeta D+\Delta(k-\zeta)}$.

Proof: Using the value of $Z_{\alpha,max} = k\alpha$ and maximum path length for routing is $\tilde{n}$, we estimate

$\gamma^{\zeta,k}$:

$$\begin{aligned} k\alpha = \gamma^{\zeta,k} \textstyle\sum_{i=1}^{\tilde{n}}(D + (\tilde{n} - i)\Delta)\alpha \\ or, \qquad \gamma^{\zeta,k} = \frac{k}{\Delta\tilde{n}(\frac{D}{\Delta} + \frac{\tilde{n}-1}{2})} \end{aligned} \tag{7.14}$$

Replacing $\tilde{n}$ by the expression given in Proposition 7.1, we have $\gamma^{\zeta,k} = \frac{2\zeta^2}{2\zeta D + \Delta(k-\zeta)}$. ■

7.7 Modifying HTLC-GP to HTLC-GP$^\zeta$

In this section, we discuss the modified payment protocol HTLC-GP$^\zeta$. The corrupt node can still mount the attack without paying any penalty. However, our objective is to disincentivize the attacker by abruptly decreasing the capacity locked.

7.7.1 Protocol Description

For relaying funds from U_0 to $U_{\tilde{n}}$, both of them decides on k for a given ζ and sets $\tilde{n}$. The rate of griefing-penalty $\gamma^{\zeta,k}$ is calculated accordingly for a given HTLC timeout period D. The total amount that the payer needs to transfer is $\tilde{\alpha} = \alpha + \sum\limits_{i=1}^{\tilde{n}-1} f(\alpha_i)$. We denote each $\alpha_i = \tilde{\alpha} - \sum\limits_{k=1}^{i} f(\alpha_k), i \in [0, \tilde{n} - 1], \alpha_0 = \tilde{\alpha}$. Each node U_i samples a pair of secret key and public key (sk_i, pk_i), the public key of each node is used to encrypt the information of establishing contract with the neighboring node. An off-chain contract between U_i and U_{i+1} has a timeout period of t_i. $t_{\tilde{n}-1}$ is set to D. We discuss the various phases of the modified version of payment protocol, the description being similar to *HTLC-GP* [96].

<u>`Pre-processing Phase`</u>

U_0 must not use the exact path for routing payment. If the exact path length is used, U_1 locks a penalty $\gamma^{\zeta,k}\alpha_0 t_0$ with U_0, it can easily figure out the identity of the sender, violating privacy. Thus, it randomizes the exact path length using a random function ϕ, and shares $\phi(\tilde{n})$ with $U_{\tilde{n}}$. The latter calculates the cumulative penalty $\gamma^{\zeta,k}\phi(\tilde{n})\alpha D$ used for establishing the *cancellation contract*.

Calculating $\phi(\tilde{n})$ A routing attempt cost ψ is added such that $\gamma^{\zeta,k}\phi(\tilde{n})\alpha D \approx \gamma^{\zeta,k}((\psi+\alpha_0)t_0 + \Sigma_{j=1}^{\tilde{n}-1}\alpha_j t_j)$. This acts like a blinding factor and U_1 cannot infer the identity of the sender from the penalty it locks in the off-chain contract formed with U_0.

The following steps of the protocols are executed:

- $U_{\tilde{n}}$ samples two random numbers x and r where $x \neq r$. It constructs the payment hash $H = \mathcal{H}(x)$ and the cancellation hash $Y = \mathcal{H}(r)$.

- The payee shares both the hashes H and Y with the payer U_0.

- The cumulative griefing-penalty to be locked by U_1 is $\text{cgp}_0 = \gamma^{\zeta,k}(\psi + \tilde{\alpha})t_0$. The cumulative griefing-penalty to be locked by any other node $U_{i+1}, i \in [1, \tilde{n}-1]$ is $\text{cgp}_i = \gamma^{\zeta,k}.(\Sigma_{j=1}^{i}(\alpha_j t_j) + (\alpha_0 + \psi)t_0)$.

- The payer uses standard onion routing [64] for propagating the information needed by each node $U_i, i \in [1, \tilde{n}]$, across the path P. U_0 sends $M_0 = E(\ldots E(E(E(\phi, Z_{\tilde{n}}, pk_{\tilde{n}}), Z_{\tilde{n}-1}, pk_{\tilde{n}-1}), Z_{\tilde{n}-2}, pk_{\tilde{n}-2}) \ldots, Z_1, pk_1)$ to U_1, where $Z_i = (H, Y, \alpha_i, t_{i-1}, \text{cgp}_{i-1}, U_{i+1}), i \in [1, \tilde{n}-1]$ and $Z_{\tilde{n}} = (H, Y, \alpha_{\tilde{n}-1}, t_{\tilde{n}-1}, \text{cgp}_{\tilde{n}-1}, null)$. Here $M_{i-1} = E(M_i, Z_i, pk_i)$ is the encryption of the message M_i and Z_i using public key pk_i, $M_{\tilde{n}} = \phi$.

- U_1 decrypts M_0, gets Z_1 and M_1. $M_1 = E(\ldots E(E(E(\phi, Z_{\tilde{n}}, pk_{\tilde{n}}), Z_{\tilde{n}-1}, pk_{\tilde{n}-1}), Z_{\tilde{n}-2}, pk_{n-2}), \ldots, Z_2, pk_2)$ is forwarded to the next destination U_2. This continues till party $U_{\tilde{n}}$ gets $E(\phi, Z_{\tilde{n}}, pk_{\tilde{n}})$.

`Two-Round Locking Phase` It involves two rounds: *establishing Cancellation Contract* and *establishing Payment Contract*.

- *Establishing Cancellation Contract*: $U_{\tilde{n}}$ initiates this round and each player $U_i, i \in [1, \tilde{n}]$ locks their respective cumulative griefing-penalty cgp_{i-1}.
  - $U_{\tilde{n}}$ decrypts and gets $Z_{\tilde{n}}$. It checks $\gamma^{\zeta,k}\phi(\tilde{n})\alpha D \overset{?}{=} cgp_{\tilde{n}-1}$ and $\alpha_{\tilde{n}-1} \overset{?}{=} \alpha$. If this holds true, the payer forms a contract with $U_{\tilde{n}-1}$, locking $cgp_{\tilde{n}-1}$.
  - For any other party $U_i, i \in [1, \tilde{n}-1]$, it first checks $cgp_i - \gamma^{\zeta,k}\alpha_i t_i \overset{?}{=} cgp_{i-1}$. This ensures that there is sufficient coins to be locked as penalty in the contract to be formed with U_{i-1}. Next, it checks $\gamma^{\zeta,k}\alpha_i t_i \geq \zeta\alpha_i$. This check ensures that U_i is guaranteed a minimum compensation upon being affected by griefing attack. If both the condition satifies, U_i forms the off-chain contract with U_{i-1}, locking cgp_{i-1}.

- The off-chain contract for locking penalty in layman terms: *'U_{i+1} can withdraw the amount $cgp_i = \gamma^{\zeta,k}.(\Sigma_{j=1}^{i}(\alpha_j t_j) + (\alpha_0 + \psi)t_0)$ from the contract contingent to the release of either $x : H = \mathcal{H}(x)$ or $r : Y = \mathcal{H}(r)$ within time t_i. If the locktime elapses and U_{i+1} does not respond, U_i claims cgp_i after the locktime elapses.'*.

The pseudocode of the first round of Locking Phase for $U_{\tilde{n}}$, any intermediate party $U_i, i \in [1, \tilde{n}-1]$ and payer U_0 is stated in Procedure 17, Procedure 18 and Procedure 19 respectively.

Procedure 17: Establishing Cancellation Contract: First Round of Locking Phase for $U_{\tilde{n}}$

```
Input: $(Z_{\tilde{n}}, \phi(\tilde{n}), \gamma^{\zeta,k}, \alpha)$
$U_{\tilde{n}}$ parses $Z_{\tilde{n}}$ and gets $H', Y', \alpha', t', \text{cgp}_{\tilde{n}-1}$.
if $t' \geq t_{now} + \Delta$ and $\alpha' \stackrel{?}{=} \alpha$ and $k\alpha \stackrel{?}{=} cgp_{\tilde{n}-1}$ and $H' \stackrel{?}{=} H$ and $Y' \stackrel{?}{=} Y$ and
$remain(U_{\tilde{n}}, U_{\tilde{n}-1}) \geq cgp_{\tilde{n}-1}$ then
    Send Cancel_Contract_Request$(H, Y, t', \text{cgp}_{\tilde{n}-1}, \gamma^{\zeta,k})$ to $U_{\tilde{n}-1}$
    if acknowledgement received from $U_{\tilde{n}-1}$ then
        $remain(U_{\tilde{n}}, U_{\tilde{n}-1}) = remain(U_{\tilde{n}}, U_{\tilde{n}-1}) - \text{cgp}_{\tilde{n}-1}$
        establish $Cancel\_Contract(H, Y, t', \text{cgp}_{\tilde{n}-1})$ with $U_{\tilde{n}-1}$
        Record $t_{\tilde{n}}^{form} = current\_clock\_time$
    end
    else
        abort
    end
end
else
    abort.
end
```

- *Establishing Payment Contract*: U_0 initiates the next rounded provided it has received the cancellation contract and $cgp_0 \geq \zeta\alpha_0$. The conditional payment is forwarded till it reaches the payer $U_{\tilde{n}}$. This proceeds as normal *HTLC*.

  - Each node $U_i, i \in [0, \tilde{n}-1]$ checks that for a belief θ_{i+1} regarding U_{i+1}'s type, the expected payoff on forwarding the contract is greater than 0. If so, it forwards the terms of the off-chain contract to U_{i+1}, locking α_i coins.

  - The off-chain contract for payment in layman terms: *'U_{i+1} can claim α_i coins contingent to the release of $x : H = \mathcal{H}(x)$ within time t_i. If U_{i+1} does not respond, U_i unlocks α_i coins from the contract either by releasing preimage $r : Y = \mathcal{H}(r)$ or after the locktime elapses.'*

Procedure 18:

```
]Establishing Cancellation Contract: First Round of Locking Phase for $U_i, i \in$ [1,n-1]
Input: $(H', Y', t', \text{cgp}_i, \gamma^{\zeta,k})$
$U_i$ parses $Z_i$ and gets $H, Y, \alpha_i, t_{i-1}, \text{cgp}_{i-1}$.
if $H' \overset{?}{=} H$ and $Y \overset{?}{=} Y'$ and $t' + \Delta \overset{?}{\leq} t_{i-1}$ and $cgp_i - \gamma^{\zeta,k}\alpha_i t' \overset{?}{=} cgp_{i-1}$ and $\gamma^{\zeta,k}\alpha_i t' \geq \zeta\alpha_i$ and $remain(U_i, U_{i+1}) \geq \alpha_i$ and $remain(U_i, U_{i-1}) \geq cgp_{i-1}$ and (current_time not close to contract expiration time) then
    Sends acknowledgment to $U_{i+1}$ and waits for the off-chain contract to be established
    Send Cancel_Contract_Request$(H, Y, t_{i-1}, \text{cgp}_{i-1}, \gamma^{\zeta,k})$ to $U_{i-1}$
    if acknowledgement received from $U_{i-1}$ then
        $remain(U_i, U_{i-1}) = remain(U_i, U_{i-1}) - \text{cgp}_{i-1}$
        establish $Cancel\_Contract(H, Y, t_{i-1}, \text{cgp}_{i-1})$ with $U_{i-1}$
    end
    else
        abort
    end
end
else
    abort.
end
```

Procedure 19: Establishing Cancellation Contract: First Round of Locking Phase for U_0

```
Input: $(H', Y', t', \text{cgp}', \gamma^{\zeta,k})$
if $t' \overset{?}{=} t_0$ and $cgp' \overset{?}{=} cgp_0 \geq \zeta\alpha_0$ and $H' \overset{?}{=} H$ and $Y' \overset{?}{=} Y$ and $remain(U_0, U_1) \geq \alpha_0$ then
    Sends acknowledgment to $U_1$
    Confirm formation of penalty contract with $U_1$
    Initiate the second round, the establishment of payment contract
end
else
    abort.
end
```

The pseudocode of the second round of Locking Phase for U_0 and any intermediate party $U_i, i \in [1, \tilde{n}-1]$ is stated in Procedure 20 and Procedure 21 respectively.

Procedure 20: Establishing Payment Contract: Second Round of Locking Phase for U_0

```
Input: $(H, Y, \alpha_0, t_0)$
if $\theta_1 < \frac{f(\alpha_0)}{f(\alpha_0)+o^{t_0,\alpha_0}}$ and ($U_1$ has agreed to form the contract) and (current_time not close to contract expiration time) then
    $remain(U_0, U_1) = remain(U_0, U_1) - \alpha_0$
    establish $Payment\_Contract(H, Y, t_0, \alpha_0)$ with $U_1$
end
else
    abort
end
```

Procedure 21:

```
]Establishing Payment Contract: Second Round of Locking Phase for $U_i, i \in$ [1,n-1]
Input: $(H, Y, \alpha_i, t_i)$
if $\theta_{i+1} < \frac{f(\alpha_i)}{f(\alpha_i)+o_i^{t_i,\alpha_i}}$ and $t_{i-1} \geq t_i + \Delta$ and $\alpha_{i-1} \overset{?}{=} \alpha_i + f(\alpha_{i-1})$ and ($U_{i+1}$ has agreed to form the contract) and (current_time not close to contract expiration time) then
    $remain(U_i, U_{i+1}) = remain(U_i, U_{i+1}) - \alpha_i$
    establish $Payment\_Contract(H, Y, t_i, \alpha_i)$ with $U_{i+1}$
end
else
    abort
end
```

`Release Phase`: $U_{\tilde{n}}$ waits for μ units of time before either accepting or canceling the payment. If the payment contract received from $U_{\tilde{n}-1}$ is correct, $U_{\tilde{n}}$ releases the preimage x for payment hash H and claims the coins from $U_{\tilde{n}-1}$. If the latter has not forwarded the conditional payment, or the payer has encountered an error (mismatch in payment or penalty value, invalid lock time) in the terms stated in the incoming off-chain contract, $U_{\tilde{n}}$ releases the cancellation preimage r. In case of dispute, the payer goes on-chain and releases one of the preimages for settling the contract. The rest of the parties $U_i, i \in [1, \tilde{n}-1]$ either claim the coins or cancel the payment based on the preimage released. If U_{i+1} griefs and refuses to release preimage to U_i, the former has to pay the cumulative griefing-penalty cgp_i for affecting the nodes $U_k, 0 \leq k \leq i$, so that all the nodes obtain their due compensation.

The pseudocode of the Release Phase for $U_{\tilde{n}}$ and any intermediate party $U_i, i \in [1, \tilde{n}-1]$ is stated in Procedure 22 and Procedure 23 respectively.

7.7.2 Effectiveness of HTLC-GP$^\zeta$

A corrupt node can still mount the attack by canceling the payment just before the off-chain contract's lock time elapses. However, we intend to study the impact of the reduced maximum allowed path length on the effective *capacity locked* by the attacker in the network. We assume that for any node $U \in V$, $\mathbb{E}_{\mathbb{U}}(\mathrm{F}) > \mathbb{E}_{\mathbb{U}}(\mathrm{NF})$ and thus each self-payment gets routed and reaches the payee.

Claim 7.2.1. Given the total budget of the attack is $\mathcal{B}_{EX}$, incentive per attack being L, transaction value per payment being α, HTLC timeout period is D, time taken to settle a transaction on-chain being Δ, n is the maximum allowed path length for HTLC, $\tilde{n}$ is the maximum allowed path length for HTLC-GP$^\zeta$ and a corrupt recipient rejects the payment at time $t' = D - \mu$, where $\mu \to 0$, the capacity locked in HTLC-GP$^\zeta$ is less than the capacity locked in HTLC, the loss percent being $\frac{n-\tilde{n}}{(n-1)\left(1+\gamma^{\zeta,k}nD+\gamma^{\zeta,k}n\Delta\frac{n-1}{2}\right)} + \frac{\gamma^{\zeta,k}\tilde{n}((n-1)(D+\frac{(\tilde{n}-1)\Delta}{2})-\frac{\tilde{n}-1}{2}(D+\frac{(2\tilde{n})\Delta}{3}))}{(n-1)\left(1+\gamma^{\zeta,k}nD+\gamma^{\zeta,k}n\Delta\frac{n-1}{2}\right)}$

Proof: In HTLC-GP$^\zeta$, the capacity locked is $((\tilde{n}-1)v + \sum_{i=1}^{\tilde{n}-1} Z_{v,i})\frac{\mathcal{B}_{EX}}{L}$. In both the cases, we exclude the coins locked by the corrupt node while computing the capacity locked. We measure the difference in capacity locked in *HTLC* and HTLC-GP$^\zeta$.

$$\begin{aligned}
&\tfrac{\mathcal{B}_{EX}}{L}(n-1)\alpha - \tfrac{\mathcal{B}_{EX}}{L}((\tilde{n}-1)v + \sum_{i=1}^{\tilde{n}-1} Z_{v,i}) \\
&= \tfrac{\mathcal{B}_{EX}}{L}\Big((n-1)\alpha - ((\tilde{n}-1)v + \sum_{i=1}^{\tilde{n}-1} Z_{v,i})\Big) \\
&= \tfrac{\mathcal{B}_{EX}}{L}\Big((n-1)v(1+\gamma^{\zeta,k}\sum_{j=1}^{\tilde{n}} t_j) - ((\tilde{n}-1)v + \sum_{i=1}^{\tilde{n}-1} Z_{v,i})\Big) \\
&= v\tfrac{\mathcal{B}_{EX}}{L}\Big((n-\tilde{n}) + \gamma^{\zeta,k}\tilde{n}((n-1)(D+\tfrac{(\tilde{n}-1)\Delta}{2}) - \tfrac{\tilde{n}-1}{2}(D+\tfrac{(2\tilde{n}-1)\Delta}{3}))\Big)
\end{aligned} \tag{7.15}$$

The loss percent is ratio of difference of capacity locked in *HTLC* and HTLC-GP$^\zeta$ and capacity locked in *HTLC*.

$$\frac{v\frac{\mathcal{B}_{EX}}{L}\left((n-\tilde{n})+\gamma^{\zeta,k}\tilde{n}((n-1)(D+\frac{(\tilde{n}-1)\Delta}{2})-\frac{\tilde{n}-1}{2}(D+\frac{(2\tilde{n}-1)\Delta}{3}))\right)}{\frac{\mathcal{B}_{EX}}{L}(n-1)\alpha} \tag{7.16}$$

Considering $t_{n-1} = D$ and $t_i = D + (n-i-1)\Delta, i \in [0, n-1]$, where $\sum_{j=0}^{n-1} t_j = nD + \frac{n(n-1)}{2}\Delta$

and substituting $\alpha = v(1 + \gamma^{\zeta,k} \sum\limits_{j=0}^{n-1} t_j)$, the loss percent is

$$\begin{aligned} &\frac{v\frac{\mathcal{B}_{EX}}{L}\left((n-\tilde{n})+\gamma^{\zeta,k}\tilde{n}((n-1)(D+\frac{(\tilde{n}-1)\Delta}{2})-\frac{\tilde{n}-1}{2}(D+\frac{(2\tilde{n}-1)\Delta}{3}))\right)}{\frac{\mathcal{B}_{EX}}{L}(n-1)v(1+\gamma^{\zeta,k}\sum\limits_{j=0}^{n-1}t_j)} \\ &= \frac{n-\tilde{n}}{(n-1)\left(1+\gamma^{\zeta,k}nD+\gamma^{\zeta,k}n\Delta\frac{n-1}{2}\right)} + \frac{\gamma^{\zeta,k}\tilde{n}((n-1)(D+\frac{(\tilde{n}-1)\Delta}{2})-\frac{\tilde{n}-1}{2}(D+\frac{(2\tilde{n})\Delta}{3}))}{(n-1)\left(1+\gamma^{\zeta,k}nD+\gamma^{\zeta,k}n\Delta\frac{n-1}{2}\right)} \end{aligned} \quad (7.17)$$

■

Inference The loss percent is dominated by the factor $\frac{n-\tilde{n}}{n-1}$. For a given k, the higher the factor ζ, lower is the maximum path length $\tilde{n}$, greater is the loss incurred.

7.8 Experimental Analysis

7.8.1 Setup

For our experiments, we use Python 3.8.2 and NetworkX, version 2.4 [69]. It is a Python package for analyzing complex networks. System configuration used is Intel Core i5-8250U CPU, Operating System: Kubuntu 20.04, Memory: 7.7 GiB of RAM. The code for our implementation is available on GitHub[7]. From the dataset mentioned in [100], twelve snapshots of Bitcoin Lightning Network taken over a year, starting from *September 2019*, have been used. Each snapshot provides information regarding the public key of the nodes and the aliases used. The network is represented in the form of channels, as a pair of public keys along with the channel capacity and the channel identifier. Since our proposed strategy for countering griefing attacks requires both the parties to fund the channel, we divide the capacity of the channel into equal halves, allocating each half as the balance of a counterparty.

7.8.2 Evaluation Methodology

Attacking Strategy: The attacker corrupts the nodes that are either pendant vertices or have just one channel in the network. It is easier and cost-effective to target such peripheral nodes rather than nodes with high centrality. A highly central node tends to earn a higher profit as transactions tend

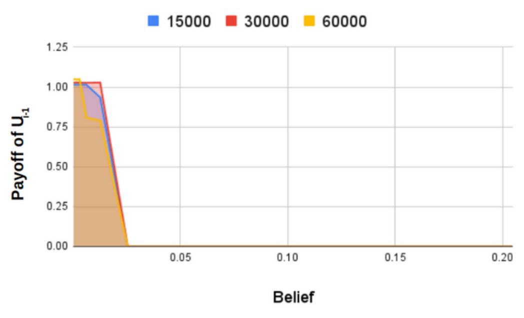
[7] https://github.com/subhramazumdar/Strategic_Analysis_Griefing

Procedure 22: Release_Phase for $U_{\tilde{n}}$

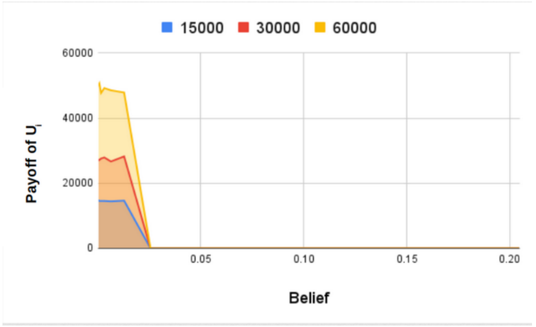
```
Input: Message $M$, time bound $\mu$
if $M \stackrel{?}{=} Payment\_Contract(H, Y, \alpha', t')$ and $current\_clock\_time - t_{\tilde{n}}^{form} \leq \mu$ then
    Parse $M$ and retrieve $(H, Y, \alpha', t')$
    if $t' \geq t_{now} + \Delta$ and $\alpha' = \alpha$ then
        $z = x$
    end
    else
        $z = r$
    end
end
else
    $z = r$
end
Release $z$ to $U_{\tilde{n}-1}$
if $current\_time < t_{\tilde{n}-1}$ then
    if $U_{\tilde{n}}$ and $U_{\tilde{n}-1}$ mutually agree to terminate Payment Contract and Cancellation Contract
    then
        if $z$=$x$ then
            $remain(U_{\tilde{n}}, U_{\tilde{n}-1}) = remain(U_{\tilde{n}}, U_{\tilde{n}-1}) + \alpha + \text{cgp}_{\tilde{n}-1}$
        end
        else
            $remain(U_{\tilde{n}}, U_{\tilde{n}-1}) = remain(U_{\tilde{n}}, U_{\tilde{n}-1}) + \text{cgp}_{\tilde{n}-1}$
            $remain(U_{\tilde{n}-1}, U_{\tilde{n}}) = remain(U_{\tilde{n}-1}, U_{\tilde{n}}) + \alpha$
        end
    end
    else
        $U_{\tilde{n}}$ goes on-chain for settlement by releasing preimage $z$.
    end
end
else
    $U_{\tilde{n}-1}$ goes on-chain for settlement, claims $(\alpha + \text{cgp}_{\tilde{n}-1})$.
    $z = null$
end
Call Release_Phase($U_{\tilde{n}-1}, z$)
```

Procedure 23:

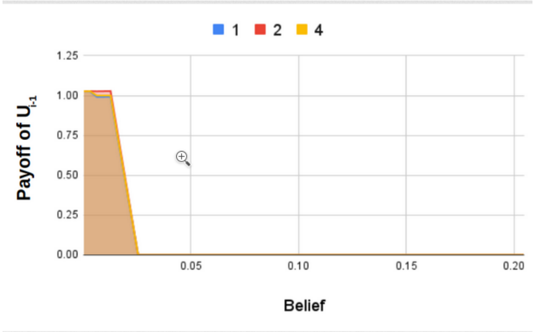
```
]Release_Phase for $U_i, i \in$ [1,$\tilde{n}$-1]
Input: $z$
Release $z$ to $U_{i-1}$
if $z \neq null$ and $current\_time < t_{i-1}$ then
    if $U_i$ and $U_{i-1}$ mutually agree to terminate Payment Contract and Cancellation Contract
    then
        if $z$=$x$ then
            $remain(U_i, U_{i-1}) = remain(U_i, U_{i-1}) + \alpha_{i-1} + \text{cgp}_{i-1}$
        end
        else
            $remain(U_i, U_{i-1}) = remain(U_i, U_{i-1}) + \text{cgp}_{i-1}$
            $remain(U_{i-1}, U_i) = remain(U_{i-1}, U_i) + \alpha_{i-1}$
        end
    end
    else
        $U_i$ goes on-chain for settlement by releasing preimage $z$.
    end
end
else
    $U_{i-1}$ goes on-chain for settlement after elapse of locktime $t_{i-1}$, claims $(\alpha_{i-1} + \text{cgp}_{i-1})$.
end
Call Release_Phase($U_{i-1}, z$)
```



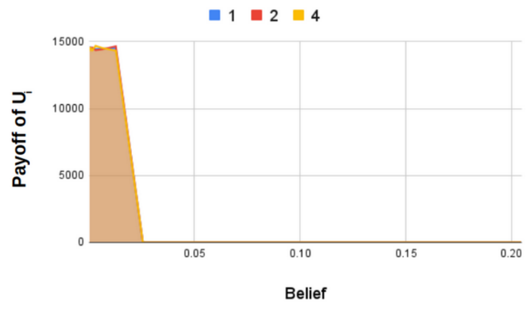
(a) Payoff of U_{i-1}, varying transaction value



(b) Payoff of U_i, varying transaction value



(c) Payoff of U_{i-1}, varying rate of arrival of transaction



(d) Payoff of U_i, varying rate of arrival of transaction

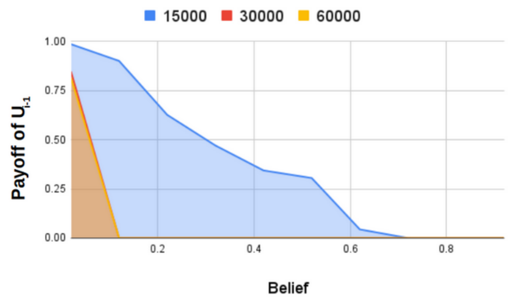
Figure 7-4: Simulation of Γ_{HTLC}

to get routed through such nodes. Also, the attacker needs to offer a higher incentive per attack, which may not be a good strategy. On the other hand, peripheral nodes can be easily incentivized to deviate, as they haven't gained much trust in the network. Such nodes do not expect to earn much by behaving altruistically.

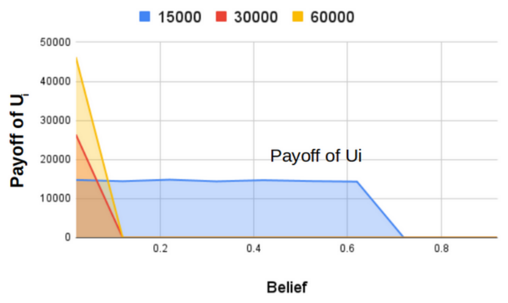
We divide the analysis into three parts, the first part simulating the sequential Bayesian Games, and the next two parts deal with the evaluation of *HTLC-GP* and HTLC-GP$^{\zeta}$.

7.8.2.1 Dataset and Parameters

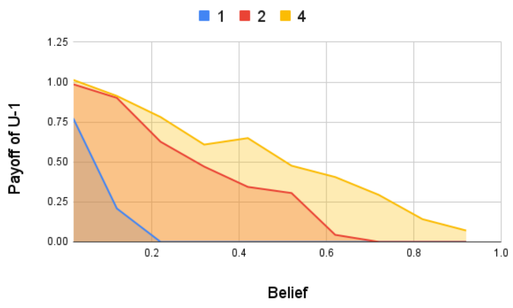
- (i) *Simulation of sequential Bayesian games*: The first part analyzes the payoff of U_{i-1} and U_i involved in the games Γ_{HTLC} and $\Gamma_{HTLC-GP}$. We simulate the games Γ_{HTLC} and, $\Gamma_{HTLC-GP}$ respectively, and estimate the payoff of U_{n-1} and U_n. We consider a *Poisson distribution* for the arrival of transaction in a given channel [82]. The rate of arrival of the transaction is varied between 1 and 4 for the next $10\ blocks$. The path length is set at 20



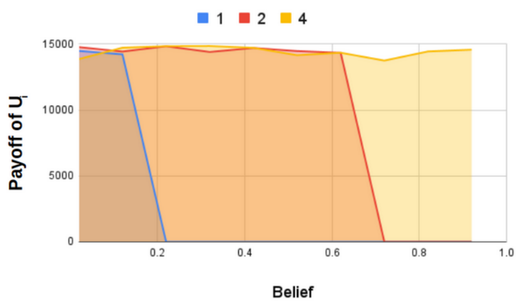
(a) Payoff of U_{i-1}, varying transaction value



(b) Payoff of U_i, varying transaction value



(c) Payoff of U_{i-1}, varying rate of arrival of transaction



(d) Payoff of U_i, varying rate of arrival of transaction

Figure 7-5: Simulation of $\Gamma_{HTLC-GP}$

and $D = 100$. The transaction amount is varied from 15000 satoshis to 60000 satoshis. The mining fee for closing a channel is $0.00000154\ BTC$[8]. q is set to 0.7. If the coins remaining unutilized are C, the party tries to estimate the fee earned in the future had it utilized the coins. We set per_tx_val to 1000 satoshis, thus a party will earn by processing $\frac{C}{1000}$ transactions.

Assumptions. Note that while analyzing the effectiveness of *HTLC-GP* and HTLC-GP$^{\zeta}$, we ignore the fact that a node may not be willing to forward the payment request based on the belief it has regarding the neighbor. It would require rigorous statistical analysis to set a belief for each node and we skip this analysis when the payment request is routed across the network. Instead, we focus on the worst-case capacity locked, assuming any node receiving a payment request forwards it to its neighbor.

[8]We have considered the data for mining fee observed for one particular channel closure https://blockstream.info/tx/c0471c9ff72a883aa45058029049ffa12b92d7379f44447bc1df52382c725c01, the mining fee can vary as observed for various closed channels in https://1ml.com/channel?order=closedchannels

- (ii) *Capacity locked by the attacker in HTLC-GP*: The second part analyses the decrease in capacity locked compared to HTLC when a penalty is introduced. We also analyze the rate of the successful transaction when γ is varied. The transaction value is varied between 10000 satoshis to 100000 satoshis. γ is varied between 10^{-3} to 10^{-7}.
  - (a) The budget of the attacker is varied between $0.05\ BTC - 6.25\ BTC$. The path length is set to 20 and D is set to 100.
  - (b) This set of experiments analyzes the rate of the successful transaction when there is no griefing attack. We vary the number of transactions between 3000-9000 and path length is varied between 5 and 20.
- (iii) *Capacity locked by the attacker in HTLC-GP$^{\zeta}$*: The third part analyses the further decrease in capacity locked compared to HTLC-GP when the concept of guaranteed minimum compensation is introduced. The budget of the attacker is varied between $0.05\ BTC - 6.25\ BTC$. The transaction value is varied between 10000 satoshis to 100000 satoshis. We vary the parameter k between 0.005 to 2. For a fixed k, ζ is varied so that path length ranges between 2 to 20. Both D and Δ are set to 100.

7.8.3 Observations

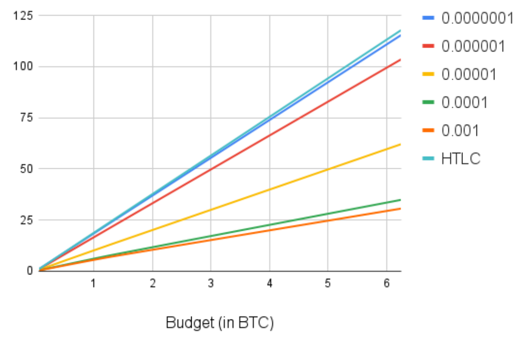
We discuss our observation in this section:

- *Simulation Result of the Games*:
  - Γ_{HTLC}: For transaction value ranging $15000 - 60000$ (in satoshis) and rate of arrival of transaction fixed to 10 within 10 blocks , the plots in Fig. 7-4(a) and 7-4(b) shows the expected payoff of U_{i-1} and expected payoff of U_i varying with the belief θ_i. U_{i-1}'s payoff decreases with increase in θ_i, confirming the theoretical result provided in *Comparative Static Analysis* in Section 7.3.4. Payoff of U_i remains more or less constant for a fixed transaction amount, but increases with increase in the transaction amount. For $\theta_i \geq 0.025$, U_{i-1} acts cautious and chooses *not forward*, as forwarding will lead to negative payoff. Both U_{i-1}'s and U_i's payoff drops to 0 from this point onwards.

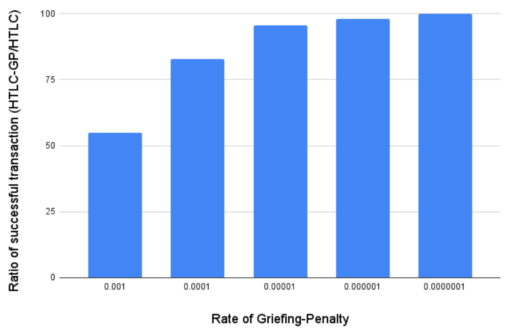
    In Fig. 7-4(c) and 7-4(d), the rate of arrival of transaction is varied between 1 and 4 within a period of 10 blocks and transaction amount is 15000 satoshi. We observe that for $\theta_i < 0.025$, payoff of U_{i-1} and U_i remains positive and invariant.

- $\Gamma_{HTLC-GP}$: The plots in Fig. 7-5(a) and 7-5(b) shows that for $\theta_i \geq 0.1$, U_{i-1} acts cautious and chooses *not forward* for transaction varying between 30000 satoshi and 60000 satoshis. For transaction amount 15000, expected payoff of U_{i-1} and U_i remains positive till $\theta_i < 0.7$.

  In Fig. 7-5(c) and 7-5(d), given the rate of arrival of transaction is 1, U_{i-1} chooses to *forward* till $\theta_i \leq 0.2$. When the rate of arrival of transaction is 2, U_{i-1} chooses to *forward* till $\theta_i \leq 0.7$ and when rate of arrival is 4, $\theta_i \leq 0.9$.



(a) Capacity locked (in BTC) vs Adversary's Budget



(b) Ratio of successful transaction (HTLC-GP/HTLC) upon varying γ

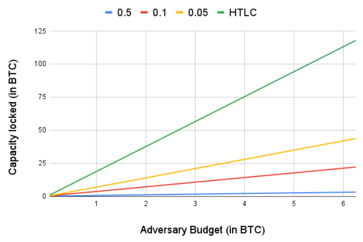
Figure 7-6: γ is varied between 10^{-3} to 10^{-7}

- *Effectiveness of HTLC-GP*: We discuss our observation for HTLC-GP:

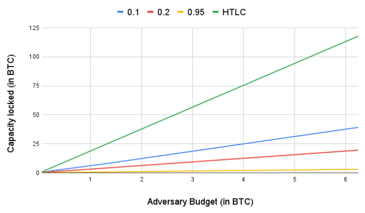
  - The capacity locked drops from 90% to 20% when γ is varied between 10^{-7} to 10^{-3} as shown in Fig.7-6(a). We see a sharp decrease in capacity locked when γ increases from 10^{-6} to 10^{-5}, with the capacity, locked dropping from 82% to 50%. When γ is 10^{-4}, the capacity locked drops to 25%.

  - In Fig.7-6(b), the ratio of successful transaction executed drops to 54% when γ is 10^{-3} and it is around 99% when γ is 10^{-7}.

- *Effectiveness of HTLC-GP*$^{\zeta}$: The observation for percentage loss in capacity is tabulated in Table 7.2 and the corresponding plot is shown in Fig.7-7(a)-(i). k is varied between 0.005 and 2, and for each k, the factor ζ is varied so that the maximum path length ranges between 2 and 20. We observe that on varying k and ζ, γ varies between 10^{-7} to 10^{-3}. The drop in capacity locked in the network ranges between 18% to 46%. The drop in collateral locked is significant for the lower value of γ and the difference reduces for $\gamma > 10^{-5}$.



(a) $k =$(b) $k =$(c) $k =$(d) $k =$(e) $k =$(f) $k =$(g) $k =$ (h) $k = 1, \zeta \in \{0.05, 0.1, 0.5\}$
$0.005, \zeta \in 0.01, \zeta \in 0.05, \zeta \in 0.1, \zeta \in 0.25, \zeta \in 0.5, \zeta \in 0.75, \zeta \in$
$\{0.00025, 0.0005, 0.0025, 0.005, 0.0125, 0.025, 0.0375, 0.075, 0.375\}$
$0.0025\}$ $0.0005,$
$0.001,$
$0.0045\}$



(i) $k = 2, \zeta \in \{0.1, 0.2, 0.95\}$

Figure 7-7: Capacity locked vs Adversary's Budget

7.8.4 Discussion of Results

- *Expected Payoff in Γ_{HTLC} and $\Gamma_{HTLC-GP}$*:
  - *Transaction amount is varied*: We see that expected payoff of U_{i-1} in $\Gamma_{HTLC-GP}$ remains positive for a higher value of θ_i compared Γ_{HTLC}. The reason is that in the first game, U_i can choose not to respond, forcing U_{i-1} to go on-chain and close the channel. Since the mining fee for closing the channel is quite high, the stakes are higher. Thus U_{n-1} tends to stop forwarding payment for θ_i as low as 0.025. In the second game, U_i will always resolve the payment just before the lock time elapses to avoid paying penalty. This prevents abrupt closure of the channel. It is observed that the cutoff value of θ_i is higher for transaction amount 150000 satoshis. This is because the capacity locked is higher when the transaction amount increases, hence the risk is higher.
  - *Rate of the arrival of the transaction is varied*: In Γ_{HTLC}, varying the rate of arrival of the transaction has no impact on the payoff of both U_{i-1} and U_i because mining fee of channel closure dominates the result. In $\Gamma_{HTLC-GP}$, the value of θ_i increases with an increase in the

k	ζ	$\gamma^{\zeta,k}$	Maximum Path Length	γ	Maximum Path Length	Ratio of capacity locked	
		$HTLC-GP^{\zeta}$	$HTLC-GP^{\zeta}$	HTLC-GP	HTLC-GP	$\frac{HTLC-GP^{\zeta}}{HTLC}$	$\frac{HTLC-GP}{HTLC}$
0.005	0.00025	2.4×10^{-7}	20	2.4×10^{-7}	20	96.89%	96.89%
	0.0005	9.1×10^{-7}	10	9.1×10^{-7}	20	46.3%	89.86%
	0.0025	1.4×10^{-5}	2	1.4×10^{-5}	20	5.21%	50.7%
0.01	0.0005	4.7×10^{-7}	20	4.7×10^{-7}	20	94%	94%
	0.001	1.8×10^{-6}	10	1.8×10^{-6}	20	44.8%	81.5%
	0.005	2.8×10^{-5}	2	2.8×10^{-5}	20	5.1%	42%
0.05	0.0025	2.4×10^{-6}	20	2.4×10^{-6}	20	78%	78%
	0.005	9.1×10^{-6}	10	9.1×10^{-6}	20	38.2%	54%
	0.025	1.6×10^{-4}	2	1.6×10^{-4}	20	4.7%%	33%
0.1	0.005	4.8×10^{-6}	20	4.8×10^{-6}	20	67.5%	67.5%
	0.01	1.8×10^{-5}	10	1.8×10^{-5}	20	33.5%	45.5%
	0.05	3.3×10^{-4}	2	3.3×10^{-4}	20	4.45%	32.5%
0.25	0.0125	1.2×10^{-5}	20	1.2×10^{-5}	20	53%	53%
	0.025	4.5×10^{-5}	10	4.5×10^{-5}	20	28%	40%
	0.1125	6.9×10^{-4}	2	6.9×10^{-4}	20	4.2%	32%
0.5	0.025	2.4×10^{-5}	20	2.4×10^{-5}	20	44%	44%
	0.05	9.1×10^{-5}	10	9.1×10^{-5}	20	22.1%	38.5%
	0.2	1.1×10^{-3}	2	1.1×10^{-3}	20	3.8%	31.5%
0.75	0.0375	3.6×10^{-5}	20	3.6×10^{-5}	20	41%	41%
	0.075	1.36×10^{-4}	10	1.36×10^{-4}	20	21.2%	35.6%
	0.3	1.7×10^{-3}	2	1.7×10^{-3}	20	3.51%	30.04%
1	0.05	4.8×10^{-5}	20	4.8×10^{-5}	20	38%	38%
	0.1	1.8×10^{-4}	10	1.8×10^{-4}	20	20%	34%
	0.5	3.3×10^{-3}	2	3.3×10^{-3}	20	3.4%	29.98%
2	0.1	10^{-4}	20	10^{-4}	20	35%	35%
	0.2	3.6×10^{-4}	10	3.6×10^{-4}	20	18%	32%
	0.95	6.1×10^{-3}	2	6.1×10^{-3}	20	3.34%	29.01%

Table 7.2: Capacity Locked when k and ζ is varied

rate of arrival of the transaction. The reason is that the distribution is positively skewed for a lower arrival rate. The Poisson distribution becomes more symmetric and less peaked as the rate increases.

- *HTLC-GP*: If γ increases, the net capacity locked by the attacker decreases. However, with increasing γ, honest participants have to lock extra collateral for a given transaction. They might have to forgo other transaction requests due to a lack of liquidity in channels. Thus the success rate of transactions drops sharply with an increase in γ.

- HTLC-GP$^{\zeta}$: When γ increases, percentage loss in capacity locked for *HTLC-GP* increases as well. But this is at the cost of a high failure rate of transactions. In this protocol, we observe that the capacity locked drops substantially even for lower values of γ when k and ζ are adjusted to reduce the maximum path length. So one need not compromise on the success rate of transactions.

  (i) D is varied: The limit on the maximum timeout period is 2016 blocks, i.e., the maximum timeout period cannot exceed this value in a given path [100]. So a corrupt node will divide 2016 blocks by $\tilde{n}$ to get the value D. $\gamma^{\zeta,k}$ will decrease if D increases. The penalty locked doesn't change much, hence the capacity locked will not differ.

(ii) ζ is varied: For a fixed value of k, ζ can be increased, reducing the maximum path length available for routing. This will increase the cost of the attack. When the majority of participants in the network adhere to non-attacking behavior, then the compensation offered can be reduced, readjusting the path length. Hence, the parameters must be chosen accordingly.

Chapter 8

Conclusion and Future Work

In this thesis, we have defined the problems faced while scaling transactions in the blockchain. Our thesis specifically deals with payment channel networks. We have done an extensive literature review, categorizing the literature into routing, payment, and griefing attacks mounted on the payment channel network. Apart from that, we have also mentioned about strategic analysis of payment in the network.

We have proposed a novel privacy-preserving routing algorithm for the payment channel network, *HushRelay* suitable for simultaneous payment across multiple paths. From the results, we inferred that our proposed routing algorithm outperforms landmark-based routing algorithms in terms of success ratio and the time taken to the route. Currently, all our implementations assume that the network is static. In the future, we would like to extend our work for handling dynamic networks as well. Our algorithms were defined for a transaction between a single-payer and payee but they can be extended to handle multiple transactions by enforcing blocking protocol or non-blocking protocol to resolve deadlocks in concurrent payments. [93].

We have proposed a novel privacy-preserving, atomic multi-path payment protocol CryptoMaze. Multiple paths routing partial payments are mapped into a set of edges. Off-chain contracts are instantiated on these edges in a breadth-first fashion, starting from the sender. The use of this technique avoids the formation of multiple off-chain contracts on channels shared across multiple paths, routing partial payments. Partial payments remain unlinkable which prevents colluding parties from censoring split payments. We execute the protocol on some instances of Lightning Network and simulated networks. We infer that our protocol has less execution time and feasible communication overhead compared to existing payment protocols from the results. As part of our future work, we intend to improve the protocol by incorporating a dynamic split of payments, similar to the work in [53]. This will reduce the computation overhead of the sender by eliminating the preprocessing step of constructing conditions for each off-chain contract. However, the main challenge is to realize such a protocol without violating unlinkability.

We have proposed a strategy for mitigating griefing attacks in Lightning Network by imposing a penalty on the adversary. This increases the total cost for launching such an attack as well as compensates for other nodes in the network affected by griefing. We have shown how our proposed strategy works in time-locked payments by proposing a new protocol *HTLC-GP*. The proposed construction not only preserves privacy but also ensures that none of the honest intermediary presents in the path gets affected due to the imposition of a penalty. As part of our future work, we would like to extend the concept of griefing-penalty to *Atomic Cross-Chain Swap*. A game-theoretic analysis for cross-chain swaps using *HTLC* in [141] states the locking collateral by both the parties results in a higher success rate of transaction. However, this protocol assumes both parties lock the same amount of collateral in a single smart contract belonging to either of the blockchains. We would like to study the impact of exchange rate volatility, and lock time of contract on the cumulative griefing penalty, with each party locking collateral in different contracts belonging to different blockchains.

Lastly, we perform a strategic analysis of griefing attacks in Lightning Network. A two-player game model has been proposed where one party chooses its strategy based on its' belief of the other player's type. We have analyzed the effectiveness of payment protocol *HTLC-GP* in the same model. It is observed that the cost of attack increases with the introduction of the penalty. However, *HTLC-GP* is found to be weakly effective in countering the attack as it is dependent on the rate of griefing penalty. To further increase the cost of the attack, we introduce the concept of guaranteed minimum compensation for the affected parties, which allows us to control the maximum path length used for routing. We discuss a modified payment protocol $HTLC-GP^{\zeta}$ and our experimental results show that the former is more effective than *HTLC-GP* in countering the griefing attack. As a part of our future work, we want to propose a fair compensation protocol considering different routing nodes have different estimates of loss suffered upon being subjected to a griefing attack. We would like to study the attack when the network has Byzantine, altruistic and rational participants [21], and analyze the effectiveness of the countermeasure in the new model.

Bibliography

[1] Bolt 3: Bitcoin transaction and script formats. https://github.com/lightningnetwork/lightning-rfc/blob/master/03-transactions.md#offered-htlc-outputs.

[2] Bolt 4: Onion routing protocol. https://github.com/lightningnetwork/lightning-rfc/blob/master/04-onion-routing.md#returning-errors.

[3] Proof of burn. https://en.bitcoin.it/wiki/Proof_of_burn.

[4] Lightning 101: Lightning network fees. https://blog.bitmex.com/the-lightning-network-part-2-routing-fee-economics/, 2019. Accessed: 2019-01-22.

[5] The lightning network (part 2) - routing fee economics. https://blog.bitmex.com/the-lightning-network-part-2-routing-fee-economics/, 2019. Accessed: 2019-03-27.

[6] Cryptomaze. https://www.dropbox.com/sh/x9pngj005dxh87b/AAAJNt-WquV0JZTspnijEXNVa?dl=0, 2020.

[7] Libgcrypt, version 1.8.4. gnupg.org/software/libgcrypt/, 26 October, 2018.

[8] Proof-of-closure as griefing attack mitigation. https://lists.linuxfoundation.org/pipermail/lightning-dev/2020-April/002608.html, April 2020.

[9] Block size limit controversy. https://en.bitcoin.it/wiki/Block_size_limit_controversy, August, 2015.

[10] Amp: Atomic multi-path payments over lightning. https://lists.linuxfoundation.org/pipermail/lightning-dev/2018-February/000993.html, February 2018.

[11] A proposal for up-front payments: Reverse bond payment. https://lists.linuxfoundation.org/pipermail/lightning-dev/2020-February/002547.html, February 2020.

[12] Libhcs : A partially homomorphic c library. https://github.com/tiehuis/libhcs, January, 2018.

[13] Raiden network. http://raiden.network/, July 2017.

[14] Snapshots : Lightning network. https://github.com/ayeletmz/Lightning-Network-Congestion-Attacks/blob/master/Attack-Simulation/lightning_congestion/snapshots/LN_2020.03.17-08.00.01.json.zip, May, 2020.

[15] Source code : Speedymurmurs: Fast and private path-based transactions. https://crysp.uwaterloo.ca/software/speedymurmurs/, Nov 25, 2017.

[16] A proposal for up-front payments. https://lists.linuxfoundation.org/pipermail/lightning-dev/2019-November/002282.html, November 2019.

[17] Source code : C based implementation of zkboo. https://github.com/Sobuno/ZKBoo/, October, 2016.

[18] The scalability trilemma in blockchain. https://medium.com/@aakash_13214/the-scalability-trilemma-in-blockchain-75fb57f646df, October 2018.

[19] Proof of elapsed time (poet) (cryptocurrency). https://www.investopedia.com/terms/p/proof-elapsed-time-cryptocurrency.asp, October, 2020.

[20] Proof of capacity (cryptocurrency). https://www.investopedia.com/terms/p/proof-of-capacity-cryptocurrency.asp, September, 2020.

[21] Amitanand S Aiyer, Lorenzo Alvisi, Allen Clement, Mike Dahlin, Jean-Philippe Martin, and Carl Porth. Bar fault tolerance for cooperative services. In *Proceedings of the twentieth ACM symposium on Operating systems principles*, pages 45–58, 2005.

[22] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002.

[23] Andreas M Antonopoulos. Mastering bitcoin. 2019.

[24] Hagit Attiya and Jennifer Welch. *Distributed computing: fundamentals, simulations, and advanced topics*, volume 19. John Wiley & Sons, 2004.

[25] Lukas Aumayr, Kasra Abbaszadeh, and Matteo Maffei. Thora: Atomic and privacy-preserving multi-channel updates. *Cryptology ePrint Archive*, 2022.

[26] Lukas Aumayr, Matteo Maffei, Oğuzhan Ersoy, Andreas Erwig, Sebastian Faust, Siavash Riahi, Kristina Hostáková, and Pedro Moreno-Sanchez. Bitcoin-compatible virtual channels. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 901–918. IEEE, 2021.

[27] Lukas Aumayr, Pedro Moreno-Sanchez, Aniket Kate, and Matteo Maffei. Blitz: Secure multi-hop payments without two-phase commits. In *30th {USENIX} Security Symposium ({USENIX} Security 21)*, 2021.

[28] Lukas Aumayr, Pedro Moreno-Sanchez, Aniket Kate, and Matteo Maffei. Donner: Utxo-based virtual channels across multiple hops. *Cryptology ePrint Archive*, 2021.

[29] Zeta Avarikioti, Mahsa Bastankhah, Mohammad Ali Maddah-Ali, Krzysztof Pietrzak, Jakub Svoboda, and Michelle Yeo. Route discovery in private payment channel networks. *Cryptology ePrint Archive*, 2021.

[30] Sarah Azouvi and Alexander Hicks. Sok: Tools for game theoretic models of security for cryptocurrencies. *arXiv preprint arXiv:1905.08595*, 2019.

[31] Vivek Bagaria, Sreeram Kannan, David Tse, Giulia Fanti, and Pramod Viswanath. Prism: Deconstructing the blockchain to approach physical limits. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 585–602, 2019.

[32] Vivek Bagaria, Joachim Neu, and David Tse. Boomerang: Redundancy improves latency and throughput in payment networks. *arXiv preprint arXiv:1910.01834*, 2019.

[33] Shehar Bano, Alberto Sonnino, Mustafa Al-Bassam, Sarah Azouvi, Patrick McCorry, Sarah Meiklejohn, and George Danezis. Consensus in the age of blockchains. *arXiv preprint arXiv:1711.03936*, 2017.

[34] Albert-László Barabási and Eric Bonabeau. Scale-free networks. *Scientific american*, 288(5):60–69, 2003.

[35] Mihir Bellare. A note on negligible functions. *Journal of Cryptology*, 15(4), 2002.

[36] Iddo Bentov and Ranjit Kumaresan. How to use bitcoin to design fair protocols. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014*, pages 421–439, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[37] Ferenc Béres, István András Seres, András Benczúr, et al. A cryptoeconomic traffic analysis of bitcoin's lightning network. *CRYPTOECONOMIC SYSTEMS*, 2020:1–24, 2020.

[38] Dan Boneh and Victor Shoup. A graduate course in applied cryptography. *Recuperado de https://crypto. stanford. edu/˜ dabo/cryptobook/BonehShoup_0_4. pdf*, 2017.

[39] BtcDrak, Mark Friedenbach, and Eric Lombrozo. Bip 112, checksequenceverify. https://github.com/bitcoin/bips/blob/master/bip-0112.mediawiki, 2015-08-10.

[40] James M Buchanan. Opportunity cost. In *The world of economics*, pages 520–525. Springer, 1991.

[41] Vitalik Buterin. Toward a 12-second block time. https://blog.ethereum.org/2014/07/11/toward-a-12-second-block-time/, July, 2014.

[42] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of CRYPTOLOGY*, 13(1):143–202, 2000.

[43] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on*, pages 136–145. IEEE, 2001.

[44] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *Journal of the ACM (JACM)*, 51(4):557–594, 2004.

[45] Yanjiao Chen, Yuyang Ran, Jingyue Zhou, Jian Zhang, and Xueluan Gong. Mpcn-rp: A routing protocol for blockchain-based multi-charge payment channel networks. *IEEE Transactions on Network and Service Management*, 2021.

[46] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, et al. On scaling decentralized blockchains. In *International Conference on Financial Cryptography and Data Security*, pages 106–125. Springer, 2016.

[47] Ivan Damgård, Mads Jurik, and Jesper Buus Nielsen. A generalization of paillier's public-key system with applications to electronic voting. *International Journal of Information Security*, 9(6):371–385, 2010.

[48] Christian Decker and Roger Wattenhofer. A fast and scalable payment network with bitcoin duplex micropayment channels. In *Symposium on Self-Stabilizing Systems*, pages 3–18. Springer, 2015.

[49] Stefan Dziembowski, Lisa Eckey, and Sebastian Faust. Fairswap: How to fairly exchange digital goods. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 967–984. ACM, 2018.

[50] Stefan Dziembowski, Lisa Eckey, Sebastian Faust, and Daniel Malinowski. Perun: Virtual payment hubs over cryptographic currencies. Technical report, IACR Cryptology ePrint Archive 2017, 2017.

[51] Stefan Dziembowski et al. Non-atomic payment splitting in channel networks. *Cryptology ePrint Archive*, 2020.

[52] Stefan Dziembowski, Sebastian Faust, and Kristina Hostáková. General state channel networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 949–966, 2018.

[53] Lisa Eckey, Sebastian Faust, Kristina Hostáková, and Stefanie Roos. Splitting payments locally while routing interdimensionally. *IACR Cryptol. ePrint Arch.*, 2020:555, 2020.

[54] Jack Edmonds and Richard M Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM (JACM)*, 19(2):248–264, 1972.

[55] Christoph Egger, Pedro Moreno-Sanchez, and Matteo Maffei. 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*, pages 801–815, 2019.

[56] Lester Randolph Ford and Delbert R Fulkerson. Maximal flow through a network. In *Classic papers in combinatorics*, pages 243–248. Springer, 2009.

[57] Ryan Fugger. Money as ious in social trust networks & a proposal for a decentralized currency network protocol. *Hypertext document. Available electronically at http://ripple. sourceforge. net*, 106, 2004.

[58] Steven D Galbraith and Pierrick Gaudry. Recent progress on the elliptic curve discrete logarithm problem. *Designs, Codes and Cryptography*, 78(1):51–72, 2016.

[59] Juan Garay, Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Rational protocol design: Cryptography against incentive-driven adversaries. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 648–657. IEEE, 2013.

[60] Befekadu G Gebraselase, Bjarne E Helvik, and Yuming Jiang. An analysis of transaction handling in bitcoin. In *2021 IEEE International Conference on Smart Data Services (SMDS)*, pages 162–172. IEEE, 2021.

[61] Befekadu G Gebraselase, Bjarne E Helvik, and Yuming Jiang. Transaction characteristics of bitcoin. In *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 544–550. IEEE, 2021.

[62] Robert S Gibbons. Dynamic games of complete information. In *Game Theory for Applied Economists*, pages 55–142. Princeton University Press, 1992.

[63] Andrew V Goldberg and Robert E Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM (JACM)*, 35(4):921–940, 1988.

[64] David Goldschlag, Michael Reed, and Paul Syverson. Onion routing. *Communications of the ACM*, 42(2):39–41, 1999.

[65] Shafi Goldwasser. Mathematical foundations of modern cryptography: computational complexity perspective. *arXiv preprint cs/0212055*, 2002.

[66] Grayblock. Blockchain scaling: Why pow networks can't scale. https://medium.com/coinmonks/blockchain-scaling-30c9e1b7db1b, September, 2018.

[67] Matthew Green and Ian Miers. Bolt: Anonymous payment channels for decentralized currencies. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 473–489. ACM, 2017.

[68] Paolo Guasoni, Gur Huberman, and Clara Shikhelman. Lightning network economics: Channels. *Available at SSRN 3840374*, 2021.

[69] Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[70] Runchao Han, Haoyu Lin, and Jiangshan Yu. On the optionality and fairness of atomic swaps. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, pages 62–75, 2019.

[71] Ethan Heilman, Leen Alshenibr, Foteini Baldimtsi, Alessandra Scafuro, and Sharon Goldberg. Tumblebit: An untrusted bitcoin-compatible anonymous payment hub. In *Network and Distributed System Security Symposium*, 2017.

[72] Ethan Heilman, Sebastien Lipmann, and Sharon Goldberg. The arwen trading protocols. In *International Conference on Financial Cryptography and Data Security*, pages 156–173. Springer, 2020.

[73] Philipp Hoenisch and Ingo Weber. Aodv–based routing for payment channel networks. In *International Conference on Blockchain*, pages 107–124. Springer, 2018.

[74] iryna pavlenko. Block size limit controversy. https://applicature.com/blog/blockchain-technology/blockchain-scalability, November, 2018.

[75] Marco Alberto Javarone and Craig Steven Wright. From bitcoin to bitcoin cash: a network analysis. In *Proceedings of the 1st Workshop on Cryptocurrencies and Blockchains for Distributed Systems*, pages 77–81. ACM, 2018.

[76] Jestopher. Good griefing: A lingering vulnerability on lightning network that still needs fixing. https://bitcoinmagazine.com/technical/good-griefing-a-lingering-vulnerability-on-lightning-network-that-still-needs-fixing, 2021.

[77] JA John and Norman R Draper. An alternative family of transformations. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 29(2):190–197, 1980.

[78] Don Johnson, Alfred Menezes, and Scott Vanstone. The elliptic curve digital signature algorithm (ecdsa). *International journal of information security*, 1(1):36–63, 2001.

[79] Maxim Jourenko, Mario Larangeira, and Keisuke Tanaka. Lightweight virtual payment channels. In *International Conference on Cryptology and Network Security*, pages 365–384. Springer, 2020.

[80] Maxim Jourenko, Mario Larangeira, and Keisuke Tanaka. Payment trees: Low collateral payments for payment channel networks. In *International Conference on Financial Cryptography and Data Security*, pages 189–208. Springer, 2021.

[81] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. CRC press, 2020.

[82] Yoshiaki Kawase and Shoji Kasahara. Transaction-confirmation time for bitcoin: A queueing analytical approach to blockchain mechanism. In *International Conference on Queueing Theory and Network Applications*, pages 75–88. Springer, 2017.

[83] Aggelos Kiayias and Orfeas Stefanos Thyfronitis Litos. Elmo: Recursive virtual payment channels for bitcoin. *Cryptology ePrint Archive*, 2021.

[84] Sunny King and Scott Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. *self-published paper, August*, 19, 2012.

[85] Wenting Li, Sébastien Andreina, Jens-Matthias Bohli, and Ghassan Karame. Securing proof-of-stake blockchain protocols. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pages 297–315. Springer, 2017.

[86] Changting Lin, Ning Ma, Xun Wang, and Jianhai Chen. Rapido: Scaling blockchain with multi-path payment channels. *Neurocomputing*, 406:322–332, 2020.

[87] Zilin Liu, Anjia Yang, Jian Weng, Tao Li, Huang Zeng, and Xiaojian Liang. Gmhl: Generalized multi-hop locks for privacy-preserving payment channel networks. *Cryptology ePrint Archive*, 2022.

[88] Ayelet Lotem, Sarah Azouvi, Patrick McCorry, and Aviv Zohar. Sliding window challenge process for congestion detection. *arXiv preprint arXiv:2201.09009*, 2022.

[89] Zhichun Lu, Runchao Han, and Jiangshan Yu. Bank run payment channel networks. *Cryptology ePrint Archive: Report 2020/456*, 2020.

[90] Zhichun Lu, Runchao Han, and Jiangshan Yu. General congestion attack on htlc-based payment channel networks. In *3rd International Conference on Blockchain Economics, Security and Protocols (Tokenomics 2021)*, 2021.

[91] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. A secure sharding protocol for open blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 17–30. ACM, 2016.

[92] Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, and Matteo Maffei. Silentwhispers: Enforcing security and privacy in decentralized credit networks. In *Network and Distributed System Security Symposium*, 2017.

[93] Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, Matteo Maffei, and Srivatsan Ravi. Concurrency and privacy with payment-channel networks. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 455–471. ACM, 2017.

[94] Giulio Malavolta, Pedro Moreno-Sanchez, Clara Schneidewind, Aniket Kate, and Matteo Maffei. Anonymous multi-hop locks for blockchain scalability and interoperability. In *NDSS*, 2019.

[95] Michael Maschler, Eilon Solan, and Shmuel Zamir. Game theory (translated from the hebrew by ziv hellman and edited by mike borns). *Cambridge University Press, Cambridge, pp. xxvi*, 979:4, 2013.

[96] S. Mazumdar, P. Banerjee, and S. Ruj. Time is money: Countering griefing attack in lightning network. In *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 1036–1043, 2020.

[97] Subhra Mazumdar, Prabal Banerjee, and Sushmita Ruj. Griefing-penalty: Countermeasure for griefing attack in lightning network, 2020.

[98] Subhra Mazumdar, Sushmita Ruj, Ram Govind Singh, and Arindam Pal. Hushrelay: A privacy-preserving, efficient, and scalable routing algorithm for off-chain payments. *arXiv preprint arXiv:2002.05071*, 2020.

[99] Andrew Miller, Iddo Bentov, Ranjit Kumaresan, and Patrick McCorry. Sprites: Payment channels that go faster than lightning. In *Twenty-Third International Conference on Financial Cryptography and Data Security 2019*, 2019.

[100] Ayelet Mizrahi and Aviv Zohar. Congestion attacks in payment channel networks. In *Financial Cryptography and Data Security*, pages 170–188. Springer Berlin Heidelberg, 2021.

[101] Susil Kumar Mohanty and Somanath Tripathy. n-htlc: Neo hashed time-lock commitment to defend against wormhole attack in payment channel networks. *Computers & Security*, 106:102291, 2021.

[102] Pedro Moreno-Sanchez, Aniket Kate, Matteo Maffei, and Kim Pecina. Privacy preserving payments in credit networks. In *Network and Distributed Security Symposium*, 2015.

[103] Satoshi Nakamoto. Bitcoin whitepaper. *URL: https://bitcoin. org/bitcoin. pdf-(17.07. 2019)*, 2008.

[104] Yadati Narahari. *Game theory and mechanism design*, volume 4. World Scientific, 2014.

[105] Utz Nisslmueller, Klaus-Tycho Foerster, Stefan Schmid, and Christian Decker. Toward active and passive confidentiality attacks on cryptocurrency off-chain networks. *arXiv preprint arXiv:2003.00003*, 2020.

[106] Karl J O'Dwyer and David Malone. Bitcoin mining and its energy footprint. 2014.

[107] Toju Ometoruwa. An introductory guide to hashed timelock contracts. https://btcmanager.com/an-introductory-guide-to-hashed-timelock-contracts/, April, 2019.

[108] Olaoluwa Osuntokun, Conner Fromknecht, Wilmer Paulino, Oliver Gugger, and Johan Halseth. Lightning pool: A non-custodial channel lease marketplace. 2020.

[109] Cristina Pérez-Sola, Alejandro Ranchal-Pedrosa, Jordi Herrera-Joancomartí, Guillermo Navarro-Arribas, and Joaquin Garcia-Alfaro. Lockdown: Balance availability attack against lightning network channels. In *International Conference on Financial Cryptography and Data Security*, pages 245–263. Springer, 2020.

[110] Thuy Lien Pham, Ivan Lavallee, Marc Bui, and Si Hoang Do. A distributed algorithm for the maximum flow problem. In *The 4th International Symposium on Parallel and Distributed Computing (ISPDC'05)*, pages 131–138. IEEE, 2005.

[111] Dmytro Piatkivskyi and Mariusz Nowostawski. Split payments in payment networks. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pages 67–75. Springer, 2018.

[112] Krzysztof Pietrzak, Iosif Salem, Stefan Schmid, and Michelle Yeo. Lightpir: Privacy-preserving route discovery for payment channel networks. In *2021 IFIP Networking Conference (IFIP Networking)*, pages 1–9. IEEE, 2021.

[113] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments, 2016.

[114] Pavel Prihodko, Slava Zhigulin, Mykola Sahno, Aleksei Ostrovskiy, and Olaoluwa Osuntokun. 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.

[115] Kaihua Qin and Arthur Gervais. An overview of blockchain scalability, interoperability and sustainability. *Hochschule Luzern Imperial College London Liquidity Network*, 2018.

[116] Sonbol Rahimpour and Majid Khabbazian. Spear: fast multi-path payment with redundancy. In *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies*, pages 183–191, 2021.

[117] Sophie Rain, Zeta Avarikioti, Laura Kovács, and Matteo Maffei. Towards a game-theoretic security analysis of off-chain protocols. *arXiv preprint arXiv:2109.07429*, 2021.

[118] Antoine Riard and Gleb Naumenko. Time-dilation attacks on the lightning network. *arXiv preprint arXiv:2006.01418*, 2020.

[119] Daniel Robinson. Htlcs considered harmful. In *Stanford Blockchain Conference*, 2019.

[120] Elias Rohrer, Jann-Frederik Laß, and Florian Tschorsch. Towards a concurrent and distributed route selection for payment channel networks. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pages 411–419. Springer, 2017.

[121] Elias Rohrer, Julian Malliaris, and Florian Tschorsch. 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)*, pages 347–356. IEEE, 2019.

[122] Stefanie Roos, Pedro Moreno-Sanchez, Aniket Kate, and Ian Goldberg. Settling payments fast and private: Efficient decentralized routing for path-based transactions. In *Network and Distributed System Security Symposium*, 2018.

[123] István András Seres, László Gulyás, Dániel A Nagy, and Péter Burcsi. Topological analysis of bitcoin's lightning network. *arXiv preprint arXiv:1901.04972*, 2019.

[124] Amritraj Singh, Reza M Parizi, Meng Han, Ali Dehghantanha, Hadis Karimipour, and Kim-Kwang Raymond Choo. Public blockchains scalability: An examination of sharding and segregated witness. In *Blockchain Cybersecurity, Trust and Privacy*, pages 203–232. Springer, 2020.

[125] Vibhaalakshmi Sivaraman, Shaileshh Bojja Venkatakrishnan, Kathleen Ruan, Parimarjan Negi, Lei Yang, Radhika Mittal, Giulia Fanti, and Mohammad Alizadeh. High throughput cryptocurrency routing in payment channel networks. In *17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20)*, pages 777–796, 2020.

[126] CryptoRekt Sunerok and Buk-Lee. Verge: An anonymity-centric crypto-currency. `https://cryptoactu.com/wp-content/uploads/2017/08/Verge-Anonymity-Centric-CryptoCurrency.pdf`, 2017.

[127] The OpenSSL Project. OpenSSL: The open source toolkit for SSL/TLS. `www.openssl.org`, 10 September, 2019.

[128] Saar Tochner and Stefan Schmid. On search friction of route discovery in offchain networks. In *2020 IEEE International Conference on Blockchain (Blockchain)*, pages 257–264. IEEE, 2020.

[129] Saar Tochner, Stefan Schmid, and Aviv Zohar. Hijacking routes in payment channel networks: A predictability tradeoff. *arXiv preprint arXiv:1909.06890*, 2019.

[130] Saar Tochner, Aviv Zohar, and Stefan Schmid. Route hijacking and dos in off-chain networks. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, pages 228–240, 2020.

[131] Somanath Tripathy and Susil Kumar Mohanty. Mappcn: Multi-hop anonymous and privacy-preserving payment channel network. In *International Conference on Financial Cryptography and Data Security*, pages 481–495. Springer, 2020.

[132] Itay Tsabary, Matan Yechieli, Alex Manuskin, and Ittay Eyal. Mad-htlc: because htlc is crazy-cheap to attack. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1230–1248. IEEE, 2021.

[133] Nicolas Van Saberhagen. Cryptonote v 2.0. 2013.

[134] Aaron van Wirdum. Understanding the lightning network, part 2: Creating the network. https://bitcoinmagazine.com/articles/understanding-the-lightning-network-part-creating-the-network-1465326903, June, 2016.

[135] Aaron van Wirdum. Understanding the lightning network, part 1: Building a bidirectional bitcoin payment channel. https://bitcoinmagazine.com/articles/understanding-the-lightning-network-part-building-a-bidirectional-payment-channel-1464710791, May, 2016.

[136] Bimal Viswanath, Mainack Mondal, Krishna P Gummadi, Alan Mislove, and Ansley Post. Canal: Scaling social network-based sybil tolerance schemes. In *Proceedings of the 7th ACM european conference on Computer Systems*, pages 309–322. ACM, 2012.

[137] Ryan Vlastelica. Why bitcoin woń displace visa or mastercard soon. https://www.marketwatch.com/story/why-bitcoin-wont-displace-visa-or-mastercard-soon-2017-12-15, December 2017.

[138] Peng Wang, Hong Xu, Xin Jin, and Tao Wang. Flash: efficient dynamic routing for offchain networks. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, pages 370–381, 2019.

[139] Ben Weintraub, Cristina Nita-Rotaru, and Stefanie Roos. Structural attacks on local routing in payment channel networks. In *2021 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 367–379. IEEE, 2021.

[140] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151:1–32, 2014.

[141] Jiahua Xu, Damien Ackerer, and Alevtina Dubovitskaya. A game-theoretic analysis of cross-chain atomic swaps with htlcs. In *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, pages 584–594. IEEE, 2021.

[142] Lei Yang, Vivek Bagaria, Gerui Wang, Mohammad Alizadeh, David Tse, Giulia Fanti, and Pramod Viswanath. Prism: Scaling bitcoin by 10,000 x. *arXiv preprint arXiv:1909.11261*, 2019.

[143] Ningchen Ying and Tsz Wai Wu. xlumi: Payment channel protocol and off-chain payment in blockchain contract systems. *arXiv preprint arXiv:2101.10621*, 2021.

[144] Ruozhou Yu, Guoliang Xue, Vishnu Teja Kilari, Dejun Yang, and Jian Tang. Coinexpress: A fast payment routing mechanism in blockchain-based payment channel networks. In *2018 27th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–9. IEEE, 2018.

[145] Paolo Zappalà, Marianna Belotti, Maria Potop-Butucaru, and Stefano Secci. Game theoretical framework for analyzing blockchains robustness. In *Proceedings of the 4th International Symposium on Distributed Computing, Leibniz International Proceedings in Informatics (LIPIcs), Freiburg (virtual conference), Germany*, pages 49:1–49:3, 2020.

[146] Yuhui Zhang and Dejun Yang. Robustpay+: Robust payment routing with approximation guarantee in blockchain-based payment channel networks. *IEEE/ACM Transactions on Networking*, 29(4):1676–1686, 2021.

[147] Yuhui Zhang, Dejun Yang, and Guoliang Xue. Cheapay: An optimal algorithm for fee minimization in blockchain-based payment channel networks. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2019.