Detect and Isolate an Adversary in Fakey and Griefing-R Attack on Lightning Network

Abstract—**The off-chain payment solutions called layer-2 solutions, address scalability issues in the blockchain. It helps to process transactions faster and reduces the computational cost of blockchain. One of the off-chain-based solutions is the Lightning Network. In the Lightning Network, multi-hop payment is performed using HTLCs. Two prominent attacks against HTLCs are a Fakey attack, where an adversary manipulates the payment key, and a Griefing-R attack, where an adversary denies giving preimage to the payment hash. Both these attacks exhaust the channel capacities of the routing nodes and lead to reduced throughput. Therefore, we proposed a scheme using a cryptographic commitment and signature scheme that detects an adversary performing any of these attacks and punishes the adversary by isolating it from the network. We show the correctness of the scheme in security analysis.**

Index Terms—**Payment channel network; Hash time-locked contract; Commitments; Fakey and Griefing Attack**

1. Introduction

Blockchain is one of the most fascinating technologies that offers decentralization and pseudonymity [1]. Although Blockchain has been widely adopted in the payment space, it suffers from scalability issues. Layer-2 protocols enable users to perform off-chain transactions, which massively cuts down the transaction's processing cost and helps to make a scalable blockchain [2]. One such example is a Lightning Network, a Payment Channel Network (PCN) where multiple lightning nodes are connected through payment channels. Payment channels require parties to lock funds a prior. The payer must find a path if the two parties do not have a direct channel and route the payment through intermediaries using Hash Time-Locked Contracts (HTLCs).

In spite of its success and widespread adoption, we see major cyber attacks that target the underlying HTLCs [3], [4]. Examples of such attacks are the Fake Hashed Key Attack (Fakey) and the Griefing Attack [5], [6]. The major goal of these attacks is to throttle the throughput by exhausting the channel capacity of the network such that the locked funds can not be used to make further payments until their timelock expires. If an adversary can lock the funds in multiple paths simultaneously, most of the payment channel network will be stalled. Thus, the adversary can launch a Denial of Service attack.

The authors who proposed the Fakey attack have also suggested a countermeasure against Fakey attack using Symmetric Encryption [6]. Their solution does not detect an adversary and requires an extra round of communication from the receiver to the sender. Another solution countering a Griefing attack was proposed by the authors in [7]. They proposed two contracts, i.e., a payment and penalty contract. The receiver must lock the funds in a penalty contract beforehand. In case a receiver refuses to provide a preimage, all other nodes across the path will get compensation from the penalty contract. Their solution neither detects an adversary nor prevents a Fakey attack. It is indeed required to identify such adversaries who launch these attacks and discourage them from performing such adversarial actions. In addition, we need a solution where the sender cannot manipulate the hashed key in HTLC. To address such critical challenges, we propose a solution using cryptographic commitments and digital signatures for PCN.

In the proposed strategy, first, the receiver commits to a payment hash and shares a signed version of the commitment with the sender. The sender must sign and share his signed version of the commitment with the receiver before forming an onion routing packet. The HTLC contract uses commitment instead of payment hash, unlike traditional HTLC. The malicious sender can not change the commitment value as it does not have the trapdoor. However, if it tries to send different commitment with the routing nodes, then an honest receiver can identify the cheating attempt, discloses the identity of the malicious node to the PCN, and broadcast the sender's channel ID and IP/TOR address to the network. The receiver requests the network nodes to close the channels with the malicious sender. The same punishment will be given to the malicious receiver in case the receiver denies sharing the pre-image and trapdoor used in the commitment. The proposed solution helps in detecting adversaries in a Fakey and Griefing by a receiver (Griefing-R) attack. Further, every node in the network will close the channel associated with these adversarial nodes. Thus, this proposed solution isolates the malicious node in the payment path. In the proposed scheme, we disclose the identity of the sender or receiver if he cheats. If no one cheats, then intermediaries will not know who the sender or receiver is. Thus, the proposed scheme does not violate the anonymity property of PCN.

1.1. Motivation and Problem Definition

Two prominent attacks, Fakey and Griefing-R attack that can be launched against HTLC are shown in Figure 1 and

2, respectively. The example diagrams consider four users U_0, U_1, U_2, and U_3. U_0 wishes to send a payment to U_3 via U_1 and U_2.

Fake Hashed Key attack: In a Fake Hashed Key attack as shown in Figure 1, receiver U_3 generates SHA-256 hash of pre-image x, i.e. P_H, and sends P_H to U_0. However,

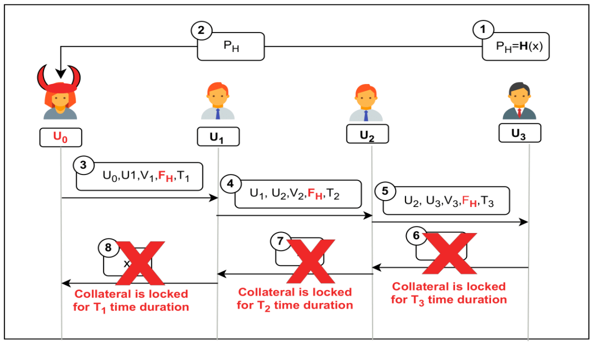

Figure 1. Fake Hashed Key Attack by the malicious sender. The senderU_0 sends F_H instead of P_H, which gets detected when it reaches the receiver U_3. The users U_3, U_2, U_1 can't open the preimage and wait for their timelocks to expire to get their locked collateral.

malicious sender U_0 manipulates the received P_H and shares fake payment hash, i.e., F_H, with other users in the path. Users U_1 and U_2 unknowingly accept F_H. U_3 has the pre-image x, but its hash will not match the fake hash F_H provided by U_2. Thus, U_3 will not finalize the payment, and users in the path need to wait for their timelocks to expire in order to redeem the locked amounts [6].

Griefing-R attack: In the Griefing attack by a receiver, the malicious receiver may deny the pre-image to, or share the wrong pre-image with, its preceding node. The malicious receiver may even go offline. In Figure 2, receiver U_3

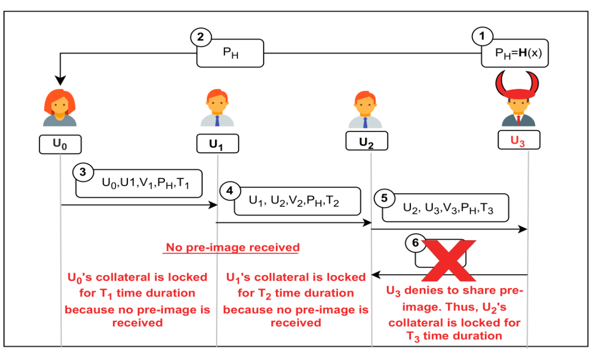

Figure 2. A Griefing-R Attack by the malicious receiver. The receiver U_3 denies to provide pre-image x, which impacts all other nodes backward in the path and keeps waiting for their timelock expiry. After timelock expiry, these nodes can roll back their funds to their wallet.

performs a Griefing attack by denying pre-image x to U_2 and payment fails. After timelock expiry, each user across the path can roll back funds to their wallet [5]. An adversary exhausts the channel capacities of the nodes in the payment channel network by launching these attacks.

The goal of our proposed scheme is to ensure that the sender sends the correct payment hash (i.e., P_H received from the U_3) across the path without any manipulation, and the receiver U_3 gives the correct pre-image x to the requester U_2.

1.2. Our Contribution

Our primary contribution to this work is as follows:

- We provide a solution that forces the sender to use the same key generated by the receiver to create the hashlock of the HTLC. The proposed solution detects an adversary performing a Fakey Attack by the sender and Griefing attack by the receiver.
- The proposed solution punishes an adversary by disclosing its identity with cheating proof, which closes all its payment channels with the other lightning nodes.

We discuss the necessary background information about Lightning Network and HTLC in Section 2. Section 3 describes the proposed solution in detail. Section 4 analyzes the security of the proposed scheme. Finally, we conclude the work in Section 5.

2. Background

2.1. Lightning Network

In the Lightning Network, two parties who wish to open a channel make the payment to a 2-of-2 multi-signature contract. Such a contract is submitted to the blockchain, creating a channel. Each channel has an associated capacity. The parties who do not have a direct channel between them, use multi-hop payment channels where payment is performed through intermediaries, which charge fees for relaying the funds. The locked funds in the contract is utilized to make further payments between the parties without submitting it to the blockchain. If both parties agree to update the channel state, they exchange their signatures on the previous commitment and create a new commitment transaction. In case a counterparty is not co-operative, then the party can unilaterally close the channel by broadcasting commitment transactions to the network. Each time, a previous commitment is invalidated using a revocation mechanism where parties share secret keys used for signing that transaction, and then a new commitment can be generated. The malicious party can not commit to an older state of the channel because of the relative timelock mechanism used in Lightning Network [8].

2.2. Hash Time-locked Contract

In PCN, multi-hop payment is performed using HTLC. A sender (U_0) wants to send some funds to receiver (U_n) through the network of payment channels across the path P = (U_0 $\rightarrow$ U_1 $\rightarrow$ U_2 $\rightarrow$ ....U_n). The receiver U_n selects a

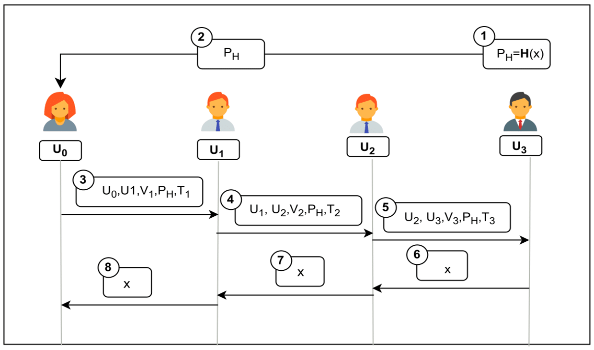

Figure 3. Hash Time-locked Contract (HTLC). In diagram, HTLC contract between any two user is represented as $(U_i, U_{i+1}, P_H, v_i, T_i)$ where i = 0 to 3. and T_i's are timestamps such that $T_1 > T_2 > T_3$. The payment values are represented by v_i notation, and P_H denotes the payment hash.

pre-image 'x' and generates payment hash $P_H = H(x)$. U_n shares P_H with U_0.

Further, U_0 shares P_H among all the nodes in the path. The HTLC contract between any two nodes U_i and U_{i+1} is defined as HTLC$(U_i, U_{i+1}, v_i, P_H, T)$. This contract implies that user U_i is locking v_i funds for user U_{i+1} which U_{i+1} can unlock if he provides pre-image 'x' for P_H within time T_i. In case, U_{i+1} doesn't provide pre-image x within T_i, then U_i can roll back his fund v_i to its wallet after timestamp T_i expiry. If users U_{i+1} wish to terminate the HTLC with U_i, then U_{i+1} can create another commitment transaction with an updated balance of U_i and U_{i+1} by invalidating earlier commitment. The payment transfer will become successful after pre-image x reaches user U_i [9]. We show an HTLC example considering four users in Figure 3.

2.3. Commitment Scheme:

A commitment scheme consists of the commit phase and reveal phase. In the commit phase, the prover chooses a message m to commit to the verifier without revealing it using the commitment algorithm $(decom, com) \leftarrow commit(1^\lambda, m)$. Here, λ represents a security parameter. This property is called the hiding property. In the reveal phase, the prover convinces the verifier that the message value m was indeed committed, by showing decom, using a verification algorithm $\{0, 1\} \leftarrow Verify(com, decom, m)$. This property is called binding property [10].

3. Proposed Solution

This section discusses the proposed scheme and script required for implementing the solution.

3.1. Protocol Details

The proposed scheme consists of two phases, pre-HTLC and HTLC. We consider the scenario of multi-hop payment where user U_0 wishes to send payment of v_3 units to user U_3. The user U_0 does not have a direct channel to U_3; therefore U_0 finds a path and decides to route the payment using onion routing through existing channels via user U_1 and U_2 as shown in Figure 4. We are following the same format of the onion packet as described in [9]. Also, sender (U_0) and receiver (U_3) use bitcoin private-public key pair for signing the commitments [2].

a) Pre-HTLC phase:
During this Pre-HTLC phase, the sender U_0 creates and sends a payment request as $< U_0, U_3, v_3 >$ to U_3 (Step 0 in Figure 4). After receiving the payment request, the receiver node (U_3) selects a pre-image (x) and generates its hash (P_H) using the SHA-256 algorithm (Step 1) . Further, U_3 commits P_H using a trapdoor s and generates commitment c. We use cryptographic commitment due to its binding and hiding properties [10]. U_3 signs the commitment using its secret key from Bitcoin key pair [2]. Anyone with access to the U_3 public key can verify the signature on c. U_3 shares c and signed c with U_0 (Step 2). Later, U_0 must sign the received commitment c and send signed c back to user U_3 in Steps 3 and 4.

User U_0 is now ready to form an onion packet. U_0 finds a path to U_3 and generates a hop payload for each user across the path. Further, U_0 creates an onion packet that is identical to that used in HTLC, except for the addition of a new field containing the "payment secret.". We use commitment c instead of payment secret in the hop payload, unlike traditional HTLC. c includes a payment secret also, but the secret can be verified only after c is decommitted. U_0 forwards the packet to the next hop user U_1. The detailed description of onion packet formation and onion routing is given in [9]. Each user gets a session key from the received onion packet and needs to derive the shared secret using his own secret key. This shared secret is used for the decryption of the onion packet. U_1 decrypts the onion packet and gets his payload containing c. U_1 forwards the onion packet further to U_2 in path. Similarly, U_2 decrypts the onion packet, unlocks c, and forwards the onion packet further to user U_3.

b) HTLC phase:
In the HTLC phase, each user creates an HTLC contract with the next user in the path. The instructions for creating the contract are given in the hop payload of each user. U_0 creates a contract i.e., $< U_0, U_1, v_1, c, T_1 >$ with user U_1; we call it HTLC1 (Step 5). The HTLC contract includes commitment c, in place of P_H in traditional HTLC. The contract says that *if U_1 provides trapdoor s to reveal the commitment and pre-image for committed hash c before time T_1, then U_1 can redeem the locked payment of value v_1.* Similarly, U_1 creates an HTLC contract (i.e., HTLC2 in Step 6) with U_2 for payment value v_2 and timelock of T_2. U_2 creates an HTLC contract with U_3 (i.e., HTLC3 in Step 7).

Since U_3 has the trapdoor s and pre-image x, it reveals both values to U_2 (Step 8). U_2 can decommit c using trapdoor s and obtain the payment hash P_H. With this, U_3 can redeem payment value v_3 from U_2. Subsequently, U_2 can share the trapdoor s and pre-image x with U_1 and claim the payment v_2 (Step 9). It continues backward until

U_0 receives the trapdoor s and pre-image x (Step 10). We show this workflow in Figure 4 and describe the locking and unlocking script for the proposed solution in the following subsection.

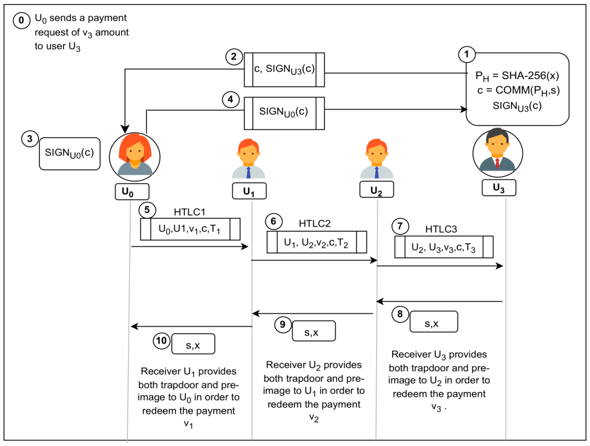

Figure 4. Cryptographic Commitment and Signature-based HTLC

3.2. Script for the proposed scheme

We show only the required modifications in the HTLC script for the proposed scheme. The revocation and redemption after timelock expiry script will remain the same as described in [9]. $OP\_DECOMMIT$ is the newly added opcode; therefore, it is highlighted in a different color.

Locking Script (ScriptPubKey)

```
OP_IF
    # To local node via HTLC-success transaction.
    ⟨c⟩ OP_DECOMMIT OP_SWAP OP_SHA256
    OP_EQUALVERIFY 2 OP_SWAP ⟨local_htlcpub
    key⟩ 2 OP_CHECKMULTISIG
OP_ENDIF
```

Unlocking Script (ScriptSig)

```
# The local node can redeem the HTLC with the witness:
    ⟨local_htlcsig⟩⟨ payment_preimage⟩⟨ trapdoor⟩
```

---

4. Security Analysis

In this Section, we show the correctness of the proposed scheme.

Assumptions. We consider a system where the sender or receiver can be corrupt and intermediate nodes are honest.

Setup. Consider, U_0 is connected to U_3 via users U_1 and U_2. Let the path be $P =< U_0, U_1, U_2, U_3 >$. U_0 makes a payment request of v_3 units to U_3 which is denoted as $< U_0, U_3, v_3 >$. We make the following claims:

Claim (a): The honest receiver U_3 will detect and isolate a corrupt sender U_0 mounting a Fakey attack.

Claim (b): The honest sender U_0 will detect and isolate a corrupt receiver U_3 launching a Griefing attack.

Proof- (a) Suppose the U_0 sends a payment request to U_3 where U_0 is a corrupt user and U_3 is an honest user. U_3 selects a pre-image x and generates it's hash P_H using SHA-256 algorithm. Further, U_3 generates a trapdoor s and commits P_H using s. We denote the commitment by c. User U_3 shares c and $SIGN^c_{U_3}$ with U_0. User U_0 also shares $SIGN^c_{U_0}$ with U_3. Further, U_0 selects $\bar{x}$ and generates a fake hash F_H. U_0 commits to F_H using different trapdoor, say $\bar{s}$ which is known only to U_0. U_0 shares the commitment $\bar{c}$ with all nodes in the path P. The honest receiver U_3 uses $< decom, \bar{c}, s >$ to reveal $\bar{c}$, but decommit function returns the result "false." Since $\bar{c} \neq c$, s can not be used for decommitting P_H. U_3 gets to know that U_0 has cheated and changed the commitment. To punish U_0, receiver U_3 takes the original c, $SIGN^c_{U_0}$, and retrieves U_0 IP/TOR address using U_0's public key and channel ID. U_3 broadcasts a proof of cheating as $< channelID_{U_0}, IPaddr_{U_0}, SIGN^c_{U_0}, c >$ to the PCN. After receiving the cheating proof, all nodes are supposed to close the channels with user U_0. Thus, U_0 will be isolated from the PCN.

Proof- (b) In the pre-HTLC phase as described in Section 3, U_3 sends commitment c and $SIGN^c_{U_3}$ to U_0. The corrupt U_3 may deny providing trapdoor s and pre-image x to U_2. Hence, U_2 will wait for timelock T_3 to expire in order to redeem v_3 units. Similarly, $U0$ and U_1 will also not receive s, x. For redemption, U_0 and U_1 will wait for their timelocks to expire. U_0 knows that only U_3 can provide s and x. Since U_0 has U_3's signature on c, U_0 has evidence that U_3 has not supplied the correct s and x to U_2. Thus, an honest sender U_0 can broadcast $< SIGN^c_{U_3}, c \in contract >$ as proof of cheating to the PCN. This results in all other nodes in the PCN closing their channels to U_3.

Thus, in case an adversary performs Fakey or Griefing-R attack, the adversary will be punished by getting isolated from the PCN.

5. Conclusion

In this paper, we have proposed a scheme that detects a FAKEY or a Griefing-R attack and the adversary (sender or receiver) who launches it. We utilized basic but powerful cryptographic schemes, i.e., cryptographic commitments and signatures. The proposed approach guarantees that the sender sends the same payment key received from the receiver and forces the receiver to provide the correct pre-image and trapdoor. This solution reveals the identity of the node in case it cheats and provides cheating proof to the network to isolate the corrupt node from PCN. In the future, we will experimentally evaluate the proposed scheme using a real-world Lightening Network snapshot and check its effectiveness.

References

[1] Zhao, Wenbing et al. *"Blockchain-enabled cyber–physical systems: A review."* IEEE Internet of Things Journal 8.6 (2020): 4023-4034.

[2] Antonopoulos, Andreas M., and David A. Harding. "Mastering bitcoin. " O'Reilly Media, Inc.", 2023.

[3] Mizrahi, Ayelet, and Aviv Zohar. *"Congestion attacks in payment channel networks."* International Conference on Financial Cryptography and Data Security. Berlin, Heidelberg: Springer, 2021.

[4] Tochner S, Schmid S, Zohar A. *"Hijacking routes in payment channel networks: A predictability tradeoff."* arXiv preprint arXiv:1909.06890. 2019 Sep 15.

[5] Robinson and Daniel. *"Htlcs considered harmful."* Stanford Blockchain Conference. 2019.

[6] Khalil, Alvi Ataur, Mohammad Ashiqur Rahman, and Hisham A. Kholidy. *"FAKEY: Fake Hashed Key Attack on Payment Channel Networks."* 2023 IEEE Conference on Communications and Network Security (CNS). IEEE, 2023.

[7] Mazumdar, Subhra, Prabal Banerjee, and Sushmita Ruj. "Griefing-penalty: Countermeasure for griefing attack in lightning network." arXiv preprint arXiv:2005.09327 (2020).

[8] Poon, J., and Dryja, T. (2015). *"The bitcoin lightning network."* Scalable o-chain instant payments, 20-46.

[9] Antonopoulos, Andreas M., Olaoluwa Osuntokun, and René Pickhardt. "Mastering the Lightning Network. " O'Reilly Media, Inc.", 2021.

[10] Fischlin, Marc. *"Trapdoor commitment schemes and their applications."* Diss. Frankfurt (Main), Univ., Diss., 2001.