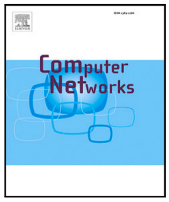
Computer Networks 197 (2021) 108346

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet



Achieve space-efficient key management in lightning network ☆

Guiyi Wei [a], Xiaohang Mao [a], Rongxing Lu [b], Jun Shao [a,*], Yunguo Guan [b], Genhua Lu [a]

[a] Zhejiang Gongshang University, Hangzhou 310018, China
[b] The Faculty of Computer Science, University of New Brunswick, Fredericton E3B 5A3, Canada

ARTICLE INFO

Keywords:
Lightning network
Bitcoin
Space efficiency
Hash function
Trapdoor one-way function

ABSTRACT

The low transaction throughput, high transaction latency, and unfriendly micropayment are the main obstacles hindering Bitcoin use in time-sensitive environments. To mitigate these problems, various solutions have been proposed. The lightning network (LN) is considered one of the most promising ones, and it has been widely deployed in different versions. However, the LN itself is subject to a scalability problem due to the used channel state revocation technique. It especially requires $\mathcal{O}(\bar{n})$ storage cost to store the private keys, where $\bar{n}$ is the number of transactions that happened in the channel and expected to be infinite. Though there are some techniques to resolve this problem, none of them is compatible with the current Bitcoin system. Aiming at solving this dilemma, in this paper, we propose two space-efficient Bitcoin-compatible key management schemes for the LN, based on the hash function and trapdoor one-way function, respectively. Both schemes reduce the storage complexity from $\mathcal{O}(\bar{n})$ to $\mathcal{O}(1)$. The detailed security analysis shows that our schemes hold the security level of the original LN or its variants. The extensive experimental results demonstrate that our proposed schemes are efficient and feasible, with a significant reduction in storage overhead.

1. Introduction

Bitcoin, proposed in 2008 [1], has become the most popular cryptocurrency in recent years. According to the report from the tokeninsight [2], the annual spot exchanges trading volume of cryptocurrency in 2019 reached USD 13.8 trillion, of which the total Bitcoin trading volume accounts for 48.29%. Furthermore, Bitcoin has been accepted as a legal currency by several countries and unrestricted in 128 of 257 countries/regions [3]. However, the popularity of Bitcoin does not make any difference for its imperfections, namely low transaction throughput, high transaction latency, and unfriendly micropayment.

- Low transaction throughput: The Bitcoin network can only handle seven transactions per second (tps), which is far from the peak volume of 47,000 tps of Visa [4].
- High transaction latency: According to the rules in the Bitcoin network, blocks are generated with the speed of one block in 10 min, and each transaction can be confirmed only after six blocks, which is unacceptable for time-sensitive tasks.
- Unfriendly micropayment: Transactions are not recorded into the blockchain for free, and the transaction fee is only related to the transaction size but not the number of bitcoins sent. Hence, the transaction fee might be much higher than the actual value of bitcoins sent.

In the presence of the above challenges, the payment channel technique [5] has become the most prominent scaling solution, where almost all the transactions happen without interacting with the blockchain. The main idea behind the payment channel is to take the channel as a local ledger maintained only by the two involving parties instead of all the parties in the Bitcoin system. Among all the transactions in the channel, only the funding transaction opening the channel and the settlement transaction closing the channel are required to be recorded on the blockchain. Other intermediate transactions updating the local ledger state are processed off-chain and maintained among these two parties only. These intermediate transactions are also named off-chain transactions. As we know, once the transactions are only maintained between two parties, we can have high transaction throughput with low latency freely. Hence, the three imperfections of Bitcoin are solved.

The payment channel can be easily extended to the payment channel network concept by combining the channels into paths, which is

☆ This work was supported by the National Key Research and Development Program of China [2019YFB1804500], the Natural Science Foundation of Zhejiang Province [grant number LZ18F020003], the National Natural Science Foundation of China [grant number U1709217], NSERC Discovery Grants (04009), and LMCRF-S-2020-03.

* Corresponding author.
E-mail addresses: weigy@zjgsu.edu.cn (G. Wei), mxiaohangg@gmail.com (X. Mao), rlu1@unb.ca (R. Lu), chn.junshao@gmail.com (J. Shao), yguan4@unb.ca (Y. Guan), genhualu22@gmail.com (G. Lu).

https://doi.org/10.1016/j.comnet.2021.108346
Received 15 February 2021; Received in revised form 2 July 2021; Accepted 19 July 2021
Available online 26 July 2021

usually realized by the hash timelock contract [6,7]. Among various payment channel networks, the lightning network (LN) [7] is the most widely deployed one on top of Bitcoin and has been implemented in different versions [8–10]. By the end of 2019, there are more than 10,000 nodes, 35,000 channels, and 1000 bitcoins in the LN. Many lightning payment wallets, such as Tippin.me [11] and Wallet of Satoshi [12], have been issued, and more and more well-known cryptocurrency payment gateways have also started to integrate the LN in operation [13]. Furthermore, applications including marketplace [14], games [15], and other fields [16] based on the LN have also been implemented.

Although the LN can alleviate the Bitcoin network's scalability problem, the LN itself is subject to a scalability problem due to the used state revocation technique. Specifically, to revoke a previous off-chain transaction, both parties should exchange their private keys corresponding to the previous off-chain transactions. With these private keys, the honest party can obtain all the bitcoins in the channel if his/her counterparty uploads one of the previous off-chain transactions to the blockchain. It is easy to see that the complexity for storing the keying materials is $\mathcal{O}(\bar{n})$, where $\bar{n}$ is the number of previous off-chain transactions. Under an ideal condition, the value of $\bar{n}$ can be infinite as expected since each channel could be part of many payment paths. In this case, even if the size of a single private key is relatively small, the storage cost would be relatively high when $\bar{n}$ becomes extremely large. Although there exist several techniques that can optimize the storage overhead of the LN and achieve $\mathcal{O}(1)$ storage cost [17–21], none of them is Bitcoin-compatible.

In this paper, aiming at solving the space efficiency issue of the LN, we would like to propose two space-efficient key management schemes, which achieve $\mathcal{O}(1)$ storage cost for both parties without any modification on the core of the current Bitcoin system or the LN. Hence, our proposed schemes are compatible with the variants of the LN [22,23]. These two schemes are based on the hash function [24] and trapdoor one-way function [25], respectively. The key idea in these two schemes stems from the following observation: To achieve $\mathcal{O}(1)$ storage cost, we should guarantee that the current private key sent to the counterparty can derive previous private keys by anyone but cannot generate future private keys. On the other hand, the owner can use the private key to generate future private keys. This observation leads to the demand of the one-wayness and trapdoor; hence the hash function and trapdoor one-way function are employed in this paper. Our contributions in this paper can be summarized as follows.

- To the best of our knowledge, our paper is the first one focusing on solving the space efficiency issue of the LN. We reduce the storage overhead for keying materials in the LN from $\mathcal{O}(\bar{n})$ to $\mathcal{O}(1)$. In other words, we propose two key management mechanisms for the LN.
- Compared to the previous related techniques, our proposal is Bitcoin-compatible. Our proposed schemes especially do not require a new consensus rule for Bitcoin script, avoid forks in the Bitcoin blockchain, and facilitate the deployment of the existing LN.
- The detailed security analysis shows that our proposed schemes do not reduce the security strength of the original/revised LN. We also implement a prototype to show the feasibility and efficiency of our proposal.

The remainder of this paper is organized as follows. In Section 2, we formalize the system model and security model, and identify our design goals. Then, we give some preliminaries, including the definitions of hash function and trapdoor one-way function in Section 3. In Section 4, we present the detailed descriptions of our proposed schemes and provide the security analysis, followed by the performance evaluation in Section 5. Section 6 reviews the related works. In the end, Section 7 gives the conclusions of our paper.

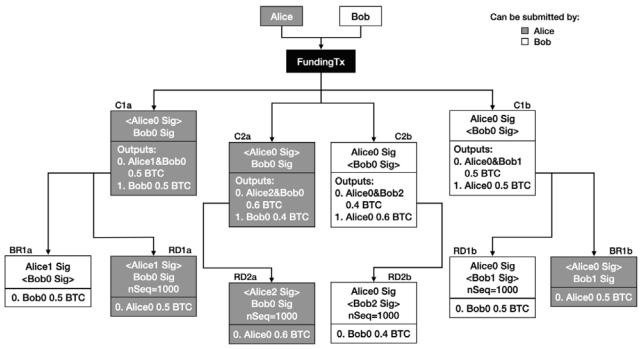


Fig. 1. The Revocable Sequence Maturity Contract (RSMC) used in the lightning network.

2. System model and design goals

In this section, we formalize our system model, especially the Revocable Sequence Maturity Contract, and security model, and identify our design goals.

2.1. System model—RSMC

The LN [7] is a second layer payment protocol that runs on top of blockchain-based cryptocurrencies such as Bitcoin to solve the scalability problem. It consists of two kinds of transaction contracts: Revocable Sequence Maturity Contract (RSMC) and Hashed Timelock Contract (HTLC). The RSMC is used to realize a lightning payment channel, and the HTLC aims to chain multiple lightning channels to construct secure transfers through multiple hops between two nodes. The space efficiency issue we aim to solve in this paper is due to the channel state revocation mechanism used in the RSMC. Hence we only review the RSMC in this section but omitting the HTLC. Interested readers please refer to the lightning network white paper [7].

There are four kinds of transactions in the RSMC: one funding transaction ($\mathsf{tx}_{\mathsf{fund}}$), multiple commitment transactions ($\mathsf{tx}_{\mathsf{com}}$), multiple revocable delivery transactions ($\mathsf{tx}_{\mathsf{rd}}$), and multiple breach remedy transactions ($\mathsf{tx}_{\mathsf{br}}$). Only the funding transaction, one commitment transaction, and one revocable delivery (breach remedy) transaction need to be submitted to the blockchain. For easy description, we name the two involving parties in the RSMC as Alice and Bob, respectively.

$\mathsf{tx}_{\mathsf{fund}}$ has two inputs from Alice and Bob and one input representing a UTXO (unspent transaction output) with a 2-of-2 multi-signature address associated with Alice and Bob. Alice or Bob cannot submit $\mathsf{tx}_{\mathsf{fund}}$ right after its generation. It is because Alice/Bob can lock the coins of the counterparty in $\mathsf{tx}_{\mathsf{fund}}$ by refusing to provide the corresponding signature. To dispense with this threat, Alice and Bob should prepare a $\mathsf{tx}_{\mathsf{com}}$ for each other to allocate balance in the UTXO.

Let us take Alice as an example for simplicity. Alice generates her $\mathsf{tx}_{\mathsf{com}}$ with the UTXO as the input and two outputs (see C1b in Fig. 1). One output (the output 1 in C1b in Fig. 1) sends coins directly to Alice, the other output (the output 0 in C1b in Fig. 1) sends coins to another 2-of-2 multi-signature address shared between Alice and Bob. The latter output accompanies a $\mathsf{tx}_{\mathsf{rd}}$ with Alice's signature and a time-locked output to Bob (see RD1b in Fig. 1). With $\mathsf{tx}_{\mathsf{com}}$ and $\mathsf{tx}_{\mathsf{rd}}$, the balance in the UTXO of $\mathsf{tx}_{\mathsf{fund}}$ is allocated to two parts. One goes directly to Alice; the other goes indirectly to Bob with a time delay. Alice sends $(\mathsf{tx}_{\mathsf{com}}, \mathsf{tx}_{\mathsf{rd}})$ to Bob, and she can submit her part of $\mathsf{tx}_{\mathsf{fund}}$ to the Bitcoin blockchain after receiving $(\mathsf{tx}'_{\mathsf{com}}, \mathsf{tx}'_{\mathsf{rd}})$ (see C1a and RD1a in Fig. 1, respectively) from Bob.

When the balance state in the UTXO changes, Alice has to prepare a new transaction pair $(\mathsf{tx}_{\mathsf{com}}, \mathsf{tx}_{\mathsf{rd}})$ (see C2b and RD2b in Fig. 1, respectively) accordingly and send them to Bob. To revoke the previous commitment transaction, Alice is required to send Bob her private key

corresponding to the 2-of-2 multi-signature address used in the previous $\mathsf{tx}_{\mathsf{com}}$.

When Alice chooses to close the payment channel, she can submit the newest $(\mathsf{tx}'_{\mathsf{com}}, \mathsf{tx}'_{\mathsf{rd}})$ (C2a and RD2a in Fig. 1 for example) with her signatures to the Bitcoin blockchain. Bob can receive the coins as soon as the transaction is recorded in the Bitcoin blockchain, while Alice receives the coins after the time delay specified in $\mathsf{tx}'_{\mathsf{rd}}$. Due to the state revocation mechanism used in the RSMC, Alice would not submit the previous $(\mathsf{tx}'_{\mathsf{com}}, \mathsf{tx}'_{\mathsf{rd}})$. If so, Bob can use his private key and the private key from Alice to generate $\mathsf{tx}'_{\mathsf{br}}$ (see BR1a in Fig. 1 if the submitted $\mathsf{tx}'_{\mathsf{com}}$ is C1a in Fig. 1) to get the coins immediately.

The state revocation mechanism used in the RSMC can guarantees that any party would not submit the previous commitment and revocable delivery transaction pair. However, at the same time, Alice and Bob should store all the private keys received from the counterparty, which causes the space efficiency issue. In this paper, we aim to solve this problem without changing any main working flow in the RSMC.

2.2. Security model

In this paper, we assume that the underlying LN is already secure. Although there are many attacks on the current LN system [22,23], most of them do not target RSMC we try to improve but other parts of the LN, which makes our security assumption on the LN reasonable. We also assume that the involving parties, Alice and Bob, could be malicious. They would try their best to obtain the coins in the funding transaction. For example, Alice would reveal the private keys for future commitment transactions from the private key sent from Bob. Alice also wants to submit the previous commitment transaction without losing the coins stored in the revocable delivery transaction.

2.3. Design goal

Based on the above system model and security model, our design goal is to propose a space-efficient key management mechanism for the RSMC of LN to reduce the storage overhead without losing the security of the original LN. Specifically, the following properties our proposal should satisfy.

- Inheritance: Our proposal inherits all the good properties of the original LN, such as instant payments and Bitcoin-compatibility.
- Low Storage: This paper's primary goal is to achieve space-efficient RSMC for the LN. The involving party in the RSMC only needs $\mathcal{O}(1)$ storage to generate his/her future private keys and derive the previous private keys of the counterparty.
- Security: Our proposal will not affect the security of the underlying LN. In particular, (1) Alice/Bob cannot derive the private keys for future commitment transactions of Bob/Alice from the private key sent from Bob/Alice. (2) Alice/Bob cannot send an invalid private key to Bob/Alice as the previous commitment transaction's key.

3. Preliminaries

Before delving into the design of our proposed space-efficient key management in the LN, we would like to review some basic tools, including the hash function [24] and trapdoor one-way function [26], which are used in our proposal.

3.1. Hash function

The hash function is a fundamental building block in many cryptographic systems. Our first solution for the space efficiency issue is also based on the hash function.

In general, the hash function converts a message of arbitrary length to a message digest of predetermined length. The main security requirement of the underlying hash function H in our proposal is one-wayness.

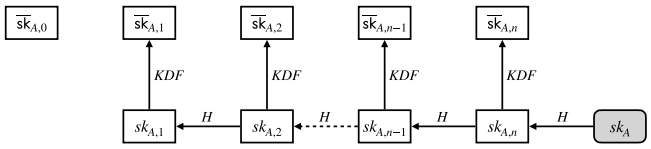


Fig. 2. The relationship among the private keys in our first scheme.

In particular, it is always easy to compute $h = H(m)$ with a message m; however, it is computationally impossible to compute m satisfying $h = H(m)$ given the value of h.

3.2. Trapdoor one-way function

The trapdoor one-way function is a special kind of one-way function. Generally speaking, it is hard to compute x satisfying $y = f(x)$ given the one-way function f and one output y of $f(\cdot)$. While it would become feasible if the one has the knowledge of the underlying trapdoor for the trapdoor one-way function f. This special property is quite useful in our proposal.

RSA cryptosystem [27] is one of the famous trapdoor one-way functions. In this paper, we make use of RSA algorithm to implement our second proposed scheme. Especially, $y = f(x)$ becomes $y = x^e \pmod N$ and $x = f^{-1}(y)$ becomes $x = y^d \pmod N$, where N is the RSA modulus, (N, e) is the public key of RSA, d is the private key of RSA, $ed \equiv 1 \pmod{\phi(N)}$, and $\phi(\cdot)$ is the Euler's totient function.

4. Our space-efficient key management for the RSMC

This section gives two schemes for improving the key management of state revocation mechanism in the RSMC. The main idea in these two schemes is to divide key generation into two directions: one is deriving the old keys, the other is generating the new keys. Everyone can do the former one, while only the one with some extra information can do both. Furthermore, these schemes do not modify the LN backbone protocol, and their implementations do not need smart contracts or any new script code. For saving space, we omit the steps for checking the validity of transactions in this section.

4.1. Hash-based scheme

4.1.1. Description of our hash-based scheme

The system parameter in our first scheme includes two security parameters ℓ, ℓ', the upper limit n of commitment transactions happening in the RSMC, a hash function $H : \{0,1\} \to \{0,1\}^{\ell'}$, and a key derivation function KDF that generates the private key of the underlying signature scheme used in the blockchain.

Open the payment channel. Alice and Bob do the following steps symmetrically. We take Alice as an example for simplicity.

1. Choose two random elements $\overline{\mathsf{sk}}_{\mathsf{A},0}$ and sk_A from the range of private key of the underlying signature and $\{0,1\}^{\ell}$, respectively.
2. Compute $sk_{A,1} = H(sk_{A,2})$ and $\overline{\mathsf{sk}}_{\mathsf{A},1} = \mathsf{KDF}(sk_{A,1})$, where $sk_{A,i} = H(sk_{A,i+1})$ for $i \in \{1, n-1\}$ and $sk_{A,n} = H(sk_A)$ as shown in Fig. 2.
3. Compute two public keys $\overline{\mathsf{pk}}_{\mathsf{A},0}$ and $\overline{\mathsf{pk}}_{\mathsf{A},1}$ corresponding to $\overline{\mathsf{sk}}_{\mathsf{A},0}$ and $\overline{\mathsf{sk}}_{\mathsf{A},1}$, respectively. Furthermore, send $(\overline{\mathsf{pk}}_{\mathsf{A},0}, \overline{\mathsf{pk}}_{\mathsf{A},1})$ to Bob via an authenticated channel.
4. By using $\overline{\mathsf{pk}}_{\mathsf{B},0}$ from Bob and $\overline{\mathsf{pk}}_{\mathsf{A},0}$, Alice generates her part of the funding transaction. Note that the input of the funding transaction is one of Alice's UTXOs, and the 2-of-2 multi-signature address in the output corresponds to $\overline{\mathsf{pk}}_{\mathsf{A},0}$ and $\overline{\mathsf{pk}}_{\mathsf{B},0}$.

5. With $\overline{\mathsf{pk}}_{\mathsf{B},1}$ from Bob, Alice generates one commitment transaction and one revocable delivery transaction. In this step, Alice needs to use $\overline{\mathsf{sk}}_{\mathsf{A},0}$ to sign the commitment transaction and revocable delivery transaction, and the 2-of-2 multi-signature address in the commitment transaction corresponds to $(\overline{\mathsf{pk}}_{\mathsf{A},0}, \overline{\mathsf{pk}}_{\mathsf{B},1})$.
6. Send the resulting funding transaction, commitment transaction, and revocable delivery transaction to Bob via an authenticated channel.
7. Store $(sk_A, \overline{\mathsf{sk}}_{\mathsf{A},0}, sk_{A,1}, \overline{\mathsf{pk}}_{\mathsf{B},1})$ locally. Alice also needs to store the received commitment transaction and revocable delivery transaction from Bob.

By combining Alice and Bob's parts of the funding transaction, we can have the complete funding transaction submitted to the Bitcoin blockchain. Note that except for steps 1, 2, and 7, the rest of the steps above are the same as those in the original LN.

Update the channel state. When the balance state of the funding transaction changes, Alice and Bob do the following steps symmetrically. We take Alice as an example for simplicity. Assume this is the ith $(1 < i \leq n)$ time to change the balance state. In the following, we use the value of i as the subscript of the commitment.

1. Compute $sk_{A,i}$ from sk_A as shown in Fig. 2, and $\overline{\mathsf{sk}}_{\mathsf{A},i}$ from $\overline{\mathsf{sk}}_{\mathsf{A},i} = \mathsf{KDF}(sk_{A,i})$.
2. Compute the public key $\overline{\mathsf{pk}}_{\mathsf{A},i}$ of $\overline{\mathsf{sk}}_{\mathsf{A},i}$. Furthermore, send $\overline{\mathsf{pk}}_{\mathsf{A},i}$ to Bob via an authenticated channel.
3. After receiving $\overline{\mathsf{pk}}_{\mathsf{B},i}$ from Bob, Alice generates a new commitment transaction and revocable delivery transaction with $(\overline{\mathsf{sk}}_{\mathsf{A},0}, \overline{\mathsf{pk}}_{\mathsf{A},0}, \overline{\mathsf{pk}}_{\mathsf{B},i})$ and $(\overline{\mathsf{sk}}_{\mathsf{A},0}, \overline{\mathsf{pk}}_{\mathsf{B},i})$, respectively, where $(\overline{\mathsf{pk}}_{\mathsf{A},0}, \overline{\mathsf{pk}}_{\mathsf{B},i})$ is used to generate the 2-of-2 multi-signature address.
4. Send the resulting commitment transaction and revocable delivery transaction to Bob along with $sk_{A,i-1}$ via an authenticated channel.
5. Upon receiving one new commitment transaction, one new revocable delivery transaction, as well as $sk_{\mathsf{B},i-1}$ from Bob, Alice checks whether $sk_{B,i-2} = H(sk_{B,i-1})$ holds[1] and whether $(\overline{\mathsf{sk}}_{\mathsf{B},i-1}, \overline{\mathsf{pk}}_{\mathsf{B},i-1})$ is a valid key pair, where $\overline{\mathsf{sk}}_{\mathsf{B},i-1} = \mathsf{KDF}(sk_{B,i-1})$. If one of them is invalid, Alice should ask Bob to resend the corresponding value. Otherwise, she stores the received transactions to replace the old commitment transaction and revocable delivery transaction.
6. Store $\overline{\mathsf{sk}}_{\mathsf{A},i}$, $sk_{B,i-1}$, and $\overline{\mathsf{pk}}_{\mathsf{B},i}$ locally, and remove $\overline{\mathsf{sk}}_{\mathsf{A},i-1}$, $sk_{B,i-2}$, and $\overline{\mathsf{pk}}_{\mathsf{B},i-1}$ from her storage. Besides the commitment transaction and revocable delivery transaction, we have five values $(sk_A, \overline{\mathsf{sk}}_{\mathsf{A},0}, sk_{A,i}, sk_{B,i-1}, \overline{\mathsf{pk}}_{\mathsf{B},i})$ in Alice's storage.

It is easy to see that steps 2 and 3 above are the same as those in the original LN.

Close the payment channel. The following steps are for closing the payment channel in our proposal.

- Like that in the original LN, if Alice or Bob in our proposal wants to close the payment channel, she/he needs to submit the newest commitment transaction and the successive revocable delivery transaction. However, different from the original LN, the one needs to compute the private key $sk_{I,i}$ from $\overline{\mathsf{sk}}_{I,i} = \mathsf{KDF}(sk_{I,i})$, where $I \in \{A, B\}$ and i is the subscript of the current commitment transaction.
- When one of Alice and Bob is a malicious player who submits a previous commitment transaction to the Bitcoin blockchain, the counterparty needs to generate the corresponding breach remedy transaction like that in the original LN. However, different from the original LN, the counterparty needs to compute the malicious player's private key $\overline{\mathsf{sk}}_{I,j}$ from $\overline{\mathsf{sk}}_{I,j} = \mathsf{KDF}(sk_{I,j})$, where $I \in \{A, B\}$, $sk_{I,j}$ is computed from $sk_{I,i-1}$ with $i-1-j$ hash computations $(sk_{I,k-1} = H(sk_{I,k}))$, $sk_{I,i-1}$ is the private key received from the malicious player, and i and j are the subscripts of the current and submitted commitment transactions, respectively.

[1] This check is only for the case $i > 2$.

4.1.2. Analysis of our hash-based scheme

Compared to the original LN, we make changes in the private keys' management mechanism for transactions. In the original LN, the private keys are generated independently while generated with a hash chain method in our scheme. Nothing changes in the transactions submitted to the Bitcoin blockchain. Hence, our hash-based scheme inherits the following properties of the original LN naturally: instant payments and Bitcoin-compatibility. The following will show that the hash chain method does not affect the private keys' security.

- Confidentiality of the private key: Alice or Bob cannot deduce the counterparty's future private key. According to the description, the private key $\overline{\mathsf{sk}}_{I,i}$ is computed from $\overline{\mathsf{sk}}_{I,i} = \mathsf{KDF}(sk_{I,i})$ and $sk_{I,i-1} = H(sk_{I,i})$. On the other hand, Alice or Bob only has the knowledge of $sk_{I,j}$'s $(1 \leq j < i)$ of the counterparty. With the one-wayness of the hash function, no one can obtain $sk_{I,i}$ from $sk_{I,j}$ $(1 \leq j < i)$. Hence, we have this property.
- Unforgeability of the private key: Alice or Bob cannot send an invalid private key to the counterparty as her/his private key corresponding to the revoking commitment transaction. Recall the checking process. The one should check whether $sk_{I,i-2} = H(sk_{I,i-1})$ holds and whether $(\overline{\mathsf{sk}}_{I,i-1}, \overline{\mathsf{pk}}_{I,i-1})$ is a valid key pair of the signature scheme, where $\overline{\mathsf{sk}}_{I,i-1} = \mathsf{KDF}(sk_{I,i-1})$. The former condition guarantees that the current $sk_{B,i-1}$ can always generate the counterparty's previous private keys, and the latter one ensures that the current $sk_{I,i-1}$ can always yield the valid $\overline{\mathsf{sk}}_{I,i-1}$. Hence, we obtain this property.

4.2. Trapdoor one-way function based scheme

4.2.1. Description of our trapdoor one-way function based scheme

In our trapdoor one-way function based scheme, all the entities know a key derivation function KDF that generates the private keys of the underlying signature scheme used in the blockchain. The whole trapdoor one-way function based scheme proceeds as follows.

Open the payment channel. Alice and Bob do the following steps symmetrically. We take Alice as an example for simplicity as in the hash-based scheme.

1. Choose a trapdoor one-way function f_{A} with a random trapdoor td_{A}. After that, send f_{A} to Bob via an authenticated channel while keeping td_{A} secret. We assume that the domain and range of f_{A} are the same for simplicity.
2. Choose two random elements $\overline{\mathsf{sk}}_{\mathsf{A},0}$ and $sk_{A,1}$ from the range of private key of the underlying signature scheme and the domain of f_{A}, respectively. Furthermore, compute $\overline{\mathsf{sk}}_{\mathsf{A},1} = \mathsf{KDF}(sk_{A,1})$.
3. Compute the public keys $\overline{\mathsf{pk}}_{\mathsf{A},0}$ and $\overline{\mathsf{pk}}_{\mathsf{A},1}$ corresponding to $\overline{\mathsf{sk}}_{\mathsf{A},0}$ and $\overline{\mathsf{sk}}_{\mathsf{A},1}$, respectively. Furthermore, send $(\overline{\mathsf{pk}}_{\mathsf{A},0}, \overline{\mathsf{pk}}_{\mathsf{A},1})$ to Bob via an authenticated channel.
4. After receiving $\overline{\mathsf{pk}}_{\mathsf{B},0}$ from Bob, Alice generates her part of the funding transaction with her input and the 2-of-2 multi-signature address corresponding to the two public keys $\overline{\mathsf{pk}}_{\mathsf{A},0}$ and $\overline{\mathsf{pk}}_{\mathsf{B},0}$.
5. After receiving $\overline{\mathsf{pk}}_{\mathsf{B},1}$ from Bob, Alice generates one commitment transaction and one revocable delivery transaction with $(\overline{\mathsf{sk}}_{\mathsf{A},0}, \overline{\mathsf{pk}}_{\mathsf{A},0}, \overline{\mathsf{pk}}_{\mathsf{B},1})$ and $(\overline{\mathsf{sk}}_{\mathsf{A},0}, \overline{\mathsf{pk}}_{\mathsf{B},1})$, respectively, where $(\overline{\mathsf{pk}}_{\mathsf{A},0}, \overline{\mathsf{pk}}_{\mathsf{B},1})$ is used to generate the 2-of-2 multi-signature address.

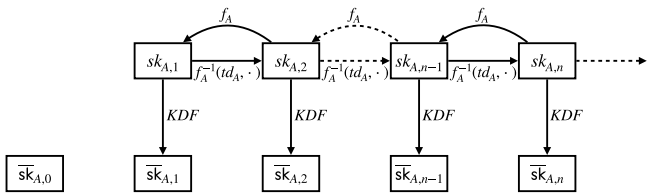


Fig. 3. The relationship among the private keys in our second scheme.

6. Send the resulting funding transaction, commitment transaction, and revocable delivery transaction to Bob via an authenticated channel.
7. Store $(\overline{\mathsf{sk}}_{\mathsf{A},0}, sk_{A,1}, \overline{\mathsf{pk}}_{\mathsf{B},1}, f_{\mathsf{B}}, \mathsf{td}_{\mathsf{A}})$ locally, where f_{B} is Bob's trapdoor one-way function. Alice also needs to store the received commitment transaction and revocable delivery transaction from Bob.

By combining Alice and Bob's parts of the funding transaction, we can have the complete funding transaction submitted to the Bitcoin blockchain. As our hash-based scheme, steps 3–6 above are the same as those in the original LN.

Update the channel state. As the hash-based scheme, when the balance state of the funding transaction changes, Alice and Bob do the following steps symmetrically. We also take Alice as an example for simplicity and use the subscript of the current commitment as the time of the balance update.

1. Compute $sk_{A,i}$ and $\overline{\mathsf{sk}}_{\mathsf{A},i}$ by $sk_{A,i} = f_{\mathsf{A}}^{-1}(\mathsf{td}_{\mathsf{A}}, sk_{A,i-1})$ and $\overline{\mathsf{sk}}_{\mathsf{A},i} = \mathsf{KDF}(sk_{A,i})$, respectively. Clearly, we have the relationship among the private keys as shown in Fig. 3.
2. Compute the public key $\overline{\mathsf{pk}}_{\mathsf{A},i}$ of $\overline{\mathsf{sk}}_{\mathsf{A},i}$. Furthermore, send $\overline{\mathsf{pk}}_{\mathsf{A},i}$ to Bob via an authenticated channel.
3. After receiving $\overline{\mathsf{pk}}_{\mathsf{B},i}$ from Bob, Alice generates a new commitment transaction and revocable delivery transaction with $(\overline{\mathsf{sk}}_{\mathsf{A},0}, \overline{\mathsf{pk}}_{\mathsf{A},0}, \overline{\mathsf{pk}}_{\mathsf{B},i})$ and $(\overline{\mathsf{sk}}_{\mathsf{A},0}, \overline{\mathsf{pk}}_{\mathsf{B},i})$, respectively, where $(\overline{\mathsf{pk}}_{\mathsf{A},0}, \overline{\mathsf{pk}}_{\mathsf{B},i})$ is used to generate the 2-of-2 multi-signature address.
4. Send the resulting commitment transaction and revocable delivery transaction to Bob along with $sk_{A,i-1}$ via an authenticated channel.
5. Upon receiving a new commitment transaction and revocable delivery transaction, as well as $sk_{B,i-1}$ from Bob, Alice checks whether $sk_{B,i-2} = f_{\mathsf{B}}(sk_{B,i-1})$ holds[2] and whether $(\overline{\mathsf{sk}}_{\mathsf{B},i-1}, \overline{\mathsf{pk}}_{\mathsf{B},i-1})$ is a valid key pair, where $\overline{\mathsf{sk}}_{\mathsf{B},i-1} = \mathsf{KDF}(sk_{B,i-1})$. If one of them is invalid, Alice should ask Bob to resend the corresponding value. Otherwise, she stores the received transactions to replace the old commitment transaction and revocable delivery transaction.
6. Store $sk_{A,i}$, $sk_{B,i-1}$, and $\overline{\mathsf{pk}}_{\mathsf{B},i}$ locally, and remove $\overline{\mathsf{sk}}_{\mathsf{A},i-1}$, $sk_{B,i-2}$, and $\overline{\mathsf{pk}}_{\mathsf{B},i-1}$ from her storage. Besides the commitment transaction and revocable delivery transaction, we have six elements $(\overline{\mathsf{sk}}_{\mathsf{A},0}, sk_{A,i}, sk_{B,i-1}, \overline{\mathsf{pk}}_{\mathsf{B},i}, f_{\mathsf{B}}, \mathsf{td}_{\mathsf{A}})$ in Alice's storage.

Like the hash-based scheme, steps 2 and 3 above are the same as those in the original LN.

Close the payment channel. The following steps are for closing the payment channel in our trapdoor one-way function based scheme.

- If Alice or Bob in our proposal wants to close the payment channel, she/he needs to submit the newest commitment transaction and the successive revocable delivery transaction as that in our hash-based scheme.
- If Alice or Bob is a malicious player who submits a previous commitment transaction to the blockchain, the counterparty does almost the same steps as that in our hash-based scheme, except for the method to compute $sk_{I,j}$ of the malicious player. In particular, in our trapdoor one-way function based scheme, $sk_{I,j}$ is computed from $sk_{I,i-1}$ with $i-1-j$ times trapdoor one-way function computation, where $I \in \{A, B\}$ and i and j are the subscripts of the current and submitted commitment transactions, respectively.

4.2.2. Analysis of our trapdoor one-way function based scheme

As we can see from the description, our second scheme is almost the same as our first scheme. The only difference is how to generate future keys and derive old keys from the current key. Hence, the second scheme still inherits instant payments and Bitcoin-compatibility properties from the original LN. In the following, we will show that the trapdoor one-way function method does not affect the private keys' security, which is quite similar to that of the hash-based scheme.

- Confidentiality of the private key: Alice or Bob cannot reveal the counterparty's future private key. According to the description, the private key $\overline{\mathsf{sk}}_{I,i}$ is computed from $\overline{\mathsf{sk}}_{I,i} = \mathsf{KDF}(sk_{I,i})$ and $sk_{I,i-1} = f_{\mathsf{A}}(sk_{I,i})$. On the other hand, Alice or Bob only has the knowledge of $sk_{I,j}$'s $(1 \le j < i)$ of the counterparty. With the one-wayness of the trapdoor one-way hash function, no one can obtain $sk_{I,i}$ from $sk_{I,j}$ $(1 \le j < i)$ without knowing the underlying trapdoor td_I. Hence, we have this property.
- Unforgeability of the private key: Alice or Bob cannot send an invalid private key to the counterparty as the private key corresponding to the commitment transaction. Recall the checking process. The one should check whether $sk_{I,i-2} = f_I(sk_{I,i-1})$ holds and whether $(\overline{\mathsf{sk}}_{I,i-1}, \overline{\mathsf{pk}}_{I,i-1})$ is a valid key pair of the signature scheme, where $\overline{\mathsf{sk}}_{I,i-1} = \mathsf{KDF}(sk_{I,i-1})$ and $I \in \{A, B\}$. The former condition guarantees that the current $sk_{B,i-1}$ can always generate the counterparty's previous private keys, and the latter one ensures that the current $sk_{I,i-1}$ can always yield the valid $\overline{\mathsf{sk}}_{I,i-1}$. Hence, we obtain this property.

5. Performance evaluation

This section will evaluate the performance of our proposed space-efficient key management for the LN in terms of storage overhead and computational cost.

In specific experiments, we use the SHA256 algorithm as the hash function used in the hash-based scheme, and $(\mathrm{mod}\ q)$ as the underlying key derivation function, where q is the prime order of the group used in ECDSA. In the trapdoor one-way function based scheme, we use RSA algorithm with a 2048-bit modulus as the candidate trapdoor one-way function and SHA256 algorithm plus $(\mathrm{mod}\ q)$ as the underlying key derivation function. Furthermore, we implement our proposed schemes using Java. Then, we perform experiments on Ubuntu 20.04 with Intel(R) Xeon(R) Gold 6140 CPU @ 2.30 GHz and 8 GB RAM.

5.1. Storage overhead

In the original LN, the involving party (Alice or Bob) needs to store two private keys $(\overline{\mathsf{sk}}_{I,0}, \overline{\mathsf{sk}}_{I,i})$ for signing commitment transactions and revocable delivery transactions, respectively. To generate the breach remedy transactions, he/she still needs to store $\bar{n}$ private keys $\overline{\mathsf{sk}}_{I,j}$'s of his/her counterparty, where $\bar{n}$ is the number of commitment transactions that happened in the payment channel. Furthermore, the party needs to store one public key $\overline{\mathsf{pk}}_{I,i}$ of his/her counterparty to check the validity of the received private key from the counterparty. Finally, the party needs to store the newest commitment transaction and revocable delivery transaction from the counterparty. In summary, the party needs $(2+\bar{n})|\overline{\mathsf{sk}}|+|\overline{\mathsf{pk}}|+|\mathsf{tx}_{\mathsf{com}}|+|\mathsf{tx}_{\mathsf{rd}}|$ storage overhead, where $|\cdot|$ denotes the bit length of x.

[2] This check is only for the case $i > 2$.

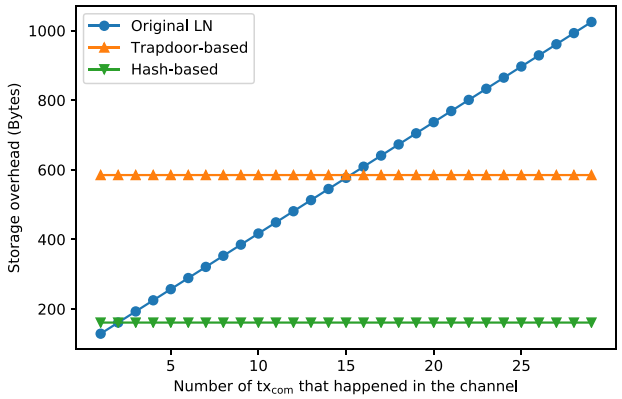


Fig. 4. The comparison among the original LN and our proposed schemes in terms of storage cost.

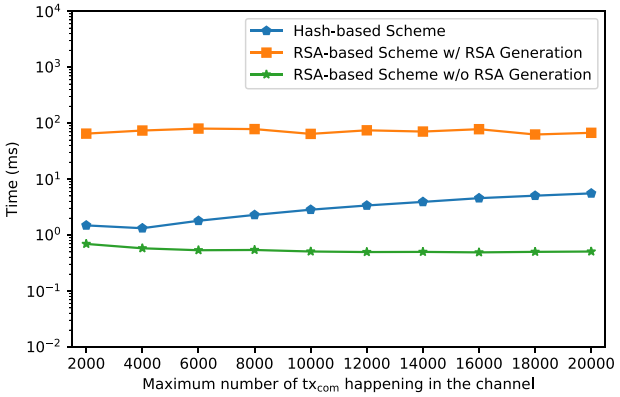


Fig. 5. Computational cost for opening a channel in our schemes.

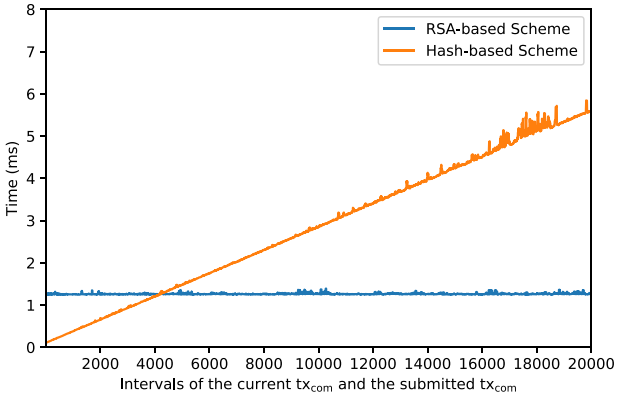


Fig. 6. Computational cost for closing a channel in our schemes.

According to the description of our hash-based scheme, the involving party needs to store one private key $\overline{\mathsf{sk}}_{I,0}$ to sign commitment transactions, and one secret $sk_{I,i}$ to generate the private key of the current revocable delivery transactions.[3] Furthermore, the party stores one secret sk_I for generating all other secrets $sk_{I,i}$'s. Unlike the original LN, the party only needs to store one private key $sk_{I,i-1}$ of the counterparty to generate breach remedy transactions. Finally, the party needs to store one public key $\overline{\mathsf{pk}}_{I,i}$ of his/her counterparty, the newest commitment transaction, and revocable delivery transaction from the counterparty, like that in the original LN. In summary, the party needs $|\overline{\mathsf{sk}}| + 3|sk| + |\overline{\mathsf{pk}}| + |\mathsf{tx}_{\mathsf{com}}| + |\mathsf{tx}_{\mathsf{rd}}|$ storage overhead. It is easy to see that our hash-based scheme's storage cost has nothing to do with the number of transactions in the payment channel.

Similar to our hash-based scheme, the involving party in our second scheme needs to store one private key $\overline{\mathsf{sk}}_{I,0}$ for signing commitment transactions, one secret $sk_{A,i}$ for generating the private key of the current revocable transaction, and the trapdoor td_I. The party further stores store one private key $sk_{I,i-1}$ and the trapdoor one-way function f_I of the counterparty to generate the breach remedy transactions. Like that in the original LN, the party needs to store one public key $\overline{\mathsf{pk}}_{I,i}$ of his/her counterparty and the newest commitment transaction and revocable delivery transaction from the counterparty. In summary, the party needs $|\overline{\mathsf{sk}}| + |sk| + |\mathsf{td}| + |f| + |\overline{\mathsf{pk}}| + |\mathsf{tx}_{\mathsf{com}}| + |\mathsf{tx}_{\mathsf{rd}}|$ storage overhead. Like our hash-based scheme, our second scheme's storage cost has nothing to do with the number of transactions that happened in the payment channel.

In Fig. 4, we give a comparison among the original LN and our proposed schemes. Note that we ignore the storage cost due to the same components in the three schemes, including $\overline{\mathsf{sk}}_{I,0}$, $\overline{\mathsf{pk}}_{I,i}$, $\mathsf{tx}_{\mathsf{com}}$, and $\mathsf{tx}_{\mathsf{rd}}$. From this figure, we can easily see that the complexity of storage in our proposed schemes is $\mathcal{O}(1)$. The number of updates happening in a channel varies with its type and its owner's activities. According to the statistics provided by 1ML,[4] the updated channels in 24 h and the total number of channels are respectively 45,181 and 51,599, and the average channel age is 325.1 days, which indicates a non-negligible average number of transactions happening in an active channel. Hence, the storage cost of our proposed schemes outperforms that of the original LN.

[3] This secret $sk_{I,i}$ could be removed since it can be computed from the secret sk_I that can generate all other secrets. However, we remain this value to save computational cost.

[4] https://1ml.com/statistics (Accessed: Jun 25, 2021).

5.2. Computational cost

In our proposed schemes, many operations also happen in the original LN, such as the commitment transaction generation during opening the payment channel process. To clearly show the computational cost due to the key management used in our proposed schemes, we ignore the cost of these common operations in this section. Hence, we have the following performance analysis.

- Hash-based scheme: The party (Alice or Bob) executes n hash computations, one KDF computation to open the payment channel, where n is the maximum number of transactions that can happen in the channel. To update the payment channel, the party executes $n-i+1$ hash computations, and two KDF computations, where i is the subscript of the current commitment transaction. Finally, the party needs to execute one KDF computation and $i-1-j$ hash computations at most to close the channel, where j is the subscript of the previous commitment transaction submitted to the Bitcoin blockchain.
- Trapdoor one-way function based scheme: The party executes one KDF computation to open the payment channel, one computation of the inversion of the underlying trapdoor one-way function f_I, two KDF computations, and one computation of f_I to update the channel state, and $i-1-j$ computations of the inversion of f_I and one KDF computation at most to close the channel.

We give the experimental results for opening/closing a payment channel in Figs. 5–6. Note that we implement f_I with RSA algorithm, where one can reduce $i-1-j$ computations of the inversion of f_I to one computation by

$$sk_{I,j} = (sk_{I,i-1})^{d^{i-1-j} \pmod{\phi(N)}} \pmod{N},$$

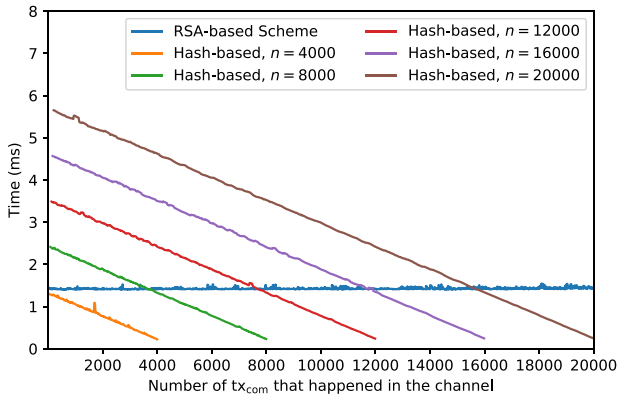


Fig. 7. Computational cost for each update to the channel state vs the number of tx_{com} that happened in the channel.

N is the underlying RSA modulus, and d is the private key of RSA. From Fig. 5, we can see that our hash-based function is more efficient for opening a channel. In Fig. 6, we do not give the results of the normal close case, since it only requires an additional key derivation function computation that can be ignored. From this figure, we can see that our hash-based function is still more efficient if the interval is not very huge. As we know, opening/closing a payment channel only happens once for one channel, we are more concerned with the updating case, which is given in Fig. 7. From this figure, we can see when n becomes very huge, our trapdoor one-way function based scheme is more efficient than our hash-based scheme in most cases, where n is the maximum number of commitment transactions happening in the channel.

5.3. Discussion

From Fig. 4, we can see that the hash-based scheme is better than the trapdoor one-way function based scheme in terms of storage cost, though both of them require $\mathcal{O}(1)$ storage complexity. Furthermore, from Figs. 5–7, we can see that the hash-based scheme is more efficient than the trapdoor one-way function based scheme if the maximum number of transactions in the channel is not very large. In this case, the hash-based scheme is the right choice. However, when the maximum number of transactions becomes very huge or unlimited as the RSMC is expected, the trapdoor one-way function based scheme becomes the better choice.

6. Related work

Lightning network is currently the most promising payment channel scheme and has been implemented using different programming languages [8–10]. However, existing protocols are still in early development. There is a rich body of literature that have been proposed to present improvements, such as privacy enhancement [23,28,29], routing algorithms optimization [30,31], channel rebalancing [32] and channel factories [33]. To the best of our knowledge, although some research results can be used to alleviate the space efficiency issue we studied in this paper, none of them is compatible with Bitcoin. In this paper, we utilized the hash function and trapdoor one-way function to solve the problem. Hence, in this section, we will review the works related to the space efficiency issue, hash-based solutions, and trapdoor one-way function based solutions.

6.1. Channel techniques solving space efficiency issue

Decker and Wattenhofer [34] proposed a bidirectional payment channels scheme by using the time lock as a parallel work to the lightning network. They also introduced the concept of invalidation tree to reset the channel state, and its storage cost is $\mathcal{O}(1)$. However, the channel's lifetime is limited due to the depth of the tree and the absolute lock time associated with each node. Miller et al. [17] introduced the concept of the state channel named Sprites that extended the payment channel to support the execution of arbitrary applications. Dziembowski et al. [21] proposed another kind of state channel, virtual channel Perun, to allow both parties to conduct payments without interacting with intermediaries on the path. Later on, Dziembowski et al. [18] presented the general construction of state channels through the virtual channel. In the above proposals, state channels are based on the Turing complete scripting language and achieve the state replacement through a monotonically increased state version. Thus, only $\mathcal{O}(1)$ storage cost is required. However, the Turing complete scripting language rules Bitcoin out. Combining the UTXO technique with the replacement technique in the state channel, Decker et al. [19] proposed Eltoo, introducing enforceable state numbers and floating transactions to achieve state replacement so that the channel has no expiration time and storage costs is $\mathcal{O}(1)$. However, this proposal is not Bitcoin-compatible because it introduced a new consensus rule requiring Bitcoin to support Sprites' state replacement technique. Pedrosa et al. [20] proposed lightning factories following the Eltoo channels approach. One can aggregate the last breach remedy transaction into previous old states, so the storage cost is $\mathcal{O}(1)$, but this feature requires the introduction of the same consensus rule as Eltoo. In summary, the current improvements for the space efficiency issue are achieved by introducing new state replacement techniques or updating new consensus rules for the Bitcoin script. Hence, they are incompatible with Bitcoin.

6.2. Hash chain

In our first scheme, we make use of the hash function as a hash chain. This technique was proposed by Lamport [35] and applied to one-time password authentication in an insecure environment. Subsequently, based on the same construction, Merkle [36] utilized the hash chain to construct a Merkle authentication tree and proposed a message authentication scheme. Rivest and Shamir [37] proposed PayWord, a micropayment scheme that utilizes a hash chain to coordinate the finance between two parties. Suzuki et al. [38] proposed a first-price sealed-bid auction scheme that provides privacy protection and accuracy assurance through the hash chain. Rivest et al. [39] proposed the time-release cryptography by utilizing a hash chain for time control. Subsequently, Joye and Yen [40] proposed internal release cryptography by utilizing two hash chains for key generation, supporting key escrow agencies to release users' keys within a specific time interval effectively. Even though the hash chain is very normal in many applications, it has never been used in the lightning network context.

6.3. Trapdoor one-way function chain

The key management in our second scheme can be considered as a trapdoor one-way function chain. Actually, the trapdoor one-way function chain and hash chain are inextricably linked. The hash chain provides high efficiency but has a limitation of finite length. By adopting the trapdoor one-way function, a chain of infinite length can be constructed, which can improve the hash chain's application. Bicakci and Baykal [41] proposed the first trapdoor one-way function chain, and they realized the improvement of Lamport's scheme [35] by treating the values on the chain as one-time keys. Trapdoor one-way function chain is also exploited in multicast communications. The hash

chain-based scheme proposed by Pietro et al. [42] does not support backward secrecy, has a bounded chain length, and requires the group manager to track each chain's update. To address these shortcomings, Pietro et al. [43] proposed another scheme based on trapdoor one-way function, which introduced the chameleon chain that the group manager uses the values on the chain rather than the hash chain for key nodes renewal.

7. Conclusion

The lightning network is considered as the most promising solution for the scalability issue of Bitcoin. However, the LN itself suffers from the space efficiency issue due to the state revocation mechanism used in the LN. To solve this problem, we proposed two schemes based on the hash function and trapdoor one-way function, respectively. Our security analysis shows that our schemes will not affect good properties of the original LN, and our experimental result conducts that our schemes solve the space efficiency issue efficiently. Since hash function computation is more efficient than (the inversion of) trapdoor one-way function computation, the hash-based scheme would be a better choice if the channel's maximum number of transactions is not very huge. Otherwise, the trapdoor one-way function based scheme is a wiser choice.

CRediT authorship contribution statement

Guiyi Wei: Writing – original draft. **Xiaohang Mao:** Software, Investigation – related work. **Rongxing Lu:** Security analysis, Writing – reviewing. **Jun Shao:** Propose the main idea. **Yunguo Guan:** Software, Writing – editing. **Genhua Lu:** Picture, Software.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] S. Nakamoto, Bitcoin: A peer-to-peer electronic cash system, 2008, http://bitcoin.org/bitcoin.pdf.

[2] Tokeninsight, 2019 cryptocurrency spot exchange industry annual report, 2020, https://www.tokeninsight.com/report/1030?lang=en.

[3] coin.dance, Global bitcoin legality, 2020, https://coin.dance/poli.

[4] M. Trillo, Stress test prepares visanet for the most wonderful time of the year, 2013, https://www.visa.com/blogarchives/us/2013/10/10/stress-test-prepares-visanet-for-the-most-wonderful-time-of-the-year/index.html.

[5] J. Spilman, Anti dos for tx replacement, 2013.

[6] L. Gudgeon, P. Moreno-Sanchez, S. Roos, P. McCorry, A. Gervais, SoK: Off the chain transactions, IACR Cryptol. ePrint Arch. (2019) 360.

[7] J. Poon, T. Dryja, The bitcoin lightning network: Scalable off-chain instant payments, 2016.

[8] C-lightning Daemon, 2018, https://github.com/ElementsProject/lightning/tree/master/lightningd. (Acceses in May 2018).

[9] A scala implementation of the lightning network, https://github.com/ACINQ/eclair.

[10] Lightning network daemon, https://github.com/lightningnetwork/lnd.

[11] https://tippin.me/.

[12] https://www.walletofsatoshi.com/.

[13] D. Hamilton, The top 5 lightning network payment processors, 2018, https://www.bitcoinlightning.com/the-top-5-lightning-network-payment-processors/.

[14] https://www.bitrefill.com/.

[15] https://satoshis.games/.

[16] https://lnsms.world/.

[17] A. Miller, I. Bentov, R. Kumaresan, P. McCorry, Sprites: Payment channels that go faster than lightning, 2017, CoRR abs/1702.05812.

[18] S. Dziembowski, S. Faust, K. Hostáková, General state channel networks, in: ACM SIGSAC, CCS 2018, pp. 949–966.

[19] C. Decker, R. Russell, O. Osuntokun, Eltoo: A simple layer2 protocol for bitcoin, 2018, https://blockstream.com/eltoo.pdf.

[20] A.R. Pedrosa, M. Potop-Butucaru, S. Tucci Piergiovanni, Scalable lightning factories for Bitcoin, in: ACM/SIGAPP, SAC 2019, 2019, pp. 302–309.

[21] S. Dziembowski, L. Eckey, S. Faust, D. Malinowski, Perun: Virtual payment hubs over cryptocurrencies, in: IEEE, SP 2019, pp. 106–123.

[22] G. Malavolta, P. Moreno-Sanchez, A. Kate, M. Maffei, S. Ravi, Concurrency and privacy with payment-channel networks, in: ACM SIGSAC CCS 2017, pp. 455–471.

[23] G. Malavolta, P. Moreno-Sanchez, C. Schneidewind, A. Kate, M. Maffei, Anonymous multi-hop locks for blockchain scalability and interoperability, in: NDSS 2019.

[24] N.I. of Standards, Technology, Secure Hash Standard, Federal Information Processing Standards Publication 180-2, 2002, URL http://www.itl.nist.gov/fipspubs.

[25] W. Diffie, M.E. Hellman, New directions in cryptography, IEEE Trans. Inf. Theory 22 (6) (1976) 644–654.

[26] J.L. Massey, An introduction to contemporary cryptology, Proc. IEEE 76 (5) (1988) 533–549.

[27] R.L. Rivest, A. Shamir, L.M. Adleman, A method for obtaining digital signatures and public-key cryptosystems, Commun. ACM 21 (2) (1978) 120–126, http://dx.doi.org/10.1145/359340.359342, URL http://doi.acm.org/10.1145/359340.359342.

[28] M. Green, I. Miers, Bolt: Anonymous payment channels for decentralized currencies, in: ACM SIGSAC, CCS 2017, pp. 473–489.

[29] E. Heilman, L. Alshenibr, F. Baldimtsi, A. Scafuro, S. Goldberg, TumbleBit: An untrusted bitcoin-compatible anonymous payment hub, in: NDSS 2017.

[30] G. Malavolta, P. Moreno-Sanchez, A. Kate, M. Maffei, SilentWhispers: Enforcing security and privacy in decentralized credit networks, in: NDSS 2017.

[31] S. Roos, P. Moreno-Sanchez, A. Kate, I. Goldberg, Settling payments fast and private: Efficient decentralized routing for path-based transactions, in: NDSS 2018.

[32] R. Khalil, A. Gervais, Revive: Rebalancing off-blockchain payment networks, in: ACM SIGSAC, CCS 2017, pp. 439–453.

[33] C. Burchert, C. Decker, R. Wattenhofer, Scalable funding of bitcoin micropayment channel networks - regular submission, in: SSS 2017, Vol. 10616, 2017, pp. 361–377.

[34] C. Decker, R. Wattenhofer, A fast and scalable payment network with bitcoin duplex micropayment channels, in: SSS 2015, Vol. 9212, pp. 3–18.

[35] L. Lamport, Password authentication with insecure communication, Commun. ACM 24 (11) (1981) 770–772.

[36] R.C. Merkle, A digital signature based on a conventional encryption function, in: CRYPTO, Vol. 293, 1987, pp. 369–378.

[37] R.L. Rivest, A. Shamir, Payword and micromint: Two simple micropayment schemes, in: Security Protocols, International Workshop, Vol. 1189, 1996, pp. 69–87.

[38] K. Suzuki, K. Kobayashi, H. Morita, Efficient sealed-bid auction using hash chain, in: ICISC 2000, Vol. 2015, pp. 183–191.

[39] R.L. Rivest, A. Shamir, D.A. Wagner, Time-Lock Puzzles and Timed-Release Crypto, Tech. rep., Massachusetts Institute of Technology, USA, 1996, URL https://people.csail.mit.edu/rivest/pubs/RSW96.pdf.

[40] M. Joye, S. Yen, One-way cross-trees and their applications, in: PKC, Vol. 2274, 2002, pp. 346–356.

[41] K. Bicakci, N. Baykal, Infinite length hash chains and their applications, in: WETICE 2002, pp. 57–61.

[42] R.D. Pietro, A. Durante, L.V. Mancini, A reliable key authentication schema for secure multicast communications, in: SRDS 2003, pp. 231–240.

[43] R.D. Pietro, L.V. Mancini, A. Durante, V. Patil, Addressing the shortcomings of one-way chains, in: ACM, ASIACCS 2006, pp. 289–296.

Guiyi Wei is a professor of the School of Information and Electronic Engineering at Zhejiang Gongshang University. He obtained his Ph.D. in Dec 2006 from Zhejiang University, where he was advised by Cheung Kong chair professor Yao Zheng. His research interests include wireless networks, mobile computing, cloud computing, social networks and network security.

Xiaohang Mao obtained his master degree from the School of Computer and Information Engineering at Zhejiang Gongshang University in Jan. 2021. His research interests include applied cryptography and blockchain.

Rongxing Lu is an associate professor at the Faculty of Computer Science (FCS), University of New Brunswick (UNB), Canada. Before that, he worked as an assistant professor at the School of Electrical and Electronic Engineering, Nanyang Technological University (NTU), Singapore from April 2013 to August 2016. Rongxing Lu worked as a Postdoctoral Fellow at the University of Waterloo from May 2012 to April 2013. He was awarded the most prestigious "Governorgeneral's Gold Medal", when he received his Ph.D. degree from the Department of Electrical & Computer Engineering, University of Waterloo, Canada, in 2012; and won the 8th IEEE Communications Society (ComSoc) Asia Pacific (AP) Outstanding Young Researcher Award, in 2013. Also, Dr. Lu received his first Ph.D. degree at Shanghai Jiao Tong University, China, in 2006. Dr. Lu is an IEEE Fellow. His research interests include applied cryptography, privacy enhancing technologies, and IoT-Big Data security and privacy. He has published extensively in his areas of expertise (with H-index 72 from Google Scholar as of November 2020), and was the recipient of 9 best (student) paper awards from some reputable journals and conferences. Currently, Dr. Lu serves as the Vice-Chair (Conferences) of IEEE ComSoc CIS-TC (Communications and Information Security Technical Committee). Dr. Lu is the Winner of 2016–17 Excellence in Teaching Award, FCS, UNB.

Jun Shao received the Ph.D. degree from the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China, in 2008. He was a Post-Doctoral Fellow with the School of Information Sciences and Technology, Pennsylvania State University, Pennsylvania, PA, USA, from 2008 to 2010. He is currently a Professor with the School of Computer and Information Engineering, Zhejiang Gongshang University, Hangzhou, China. His current research interests include network security and applied cryptography.

Yunguo Guan is a Ph.D. student of the Faculty of Computer Science, University of New Brunswick, Canada. His research interests include applied cryptography and game theory.

Genhua Lu is a master student of the School of Computer and Information Engineering at Zhejiang Gongshang University. Her research interests include applied cryptography and blockchain.