SEPoW: Secure and Efficient Proof of Work Sidechains

Taotao Li[1,2](✉), Mingsheng Wang[1,2], Zhihong Deng[1], and Dongdong Liu[1,2]

[1] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China
{litaotao,wangmingsheng,dengzhihong,liudongdong}@iie.ac.cn
[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100049, China

Abstract. Since the advent of sidechains in 2014, they have been acknowledged as the key enabler of blockchain interoperability and upgradability. However, sidechains suffer from significant challenges such as centralization, inefficiency and insecurity, meaning that they are rarely used in practice. In this paper, we present SEPoW, a secure and efficient sidechains construction that is suitable for proof of work (PoW) sidechain systems. The drawbacks for the centralized exchange of cross-chain assets in the participating blockchains are overcome by our decentralized SEPoW. To reduce the size of a cross-chain proof, we introduce merged mining into our SEPoW such that the proof consists of two Merkle tree paths regardless of the size of the current blockchain. We prove that the proposed SEPoW achieves the desirable security properties that a secure sidechains construction should have. As an exemplary concrete instantiation we propose SEPoW for a PoW blockchain system consistent with Bitcoin. We evaluate the size of SEPoW proof and compare it with the state-of-the-art PoW sidechains protocols. Results demonstrate that SEPoW achieves a proof size of 416 bytes which is roughly 123×, 510× and 62000× smaller than zkRelay proof, PoW sidechains proof and BTCRelay proof, respectively.

Keywords: Sidechains · Merged mining · Decentralized construction · Succinct proof · Proof of work

1 Introduction

Since blockchain technology was introduced by Satoshi [1] in 2008, various blockchains with different characteristics and their applications have been gaining increasing attention and adoption by communities. However, blockchains suffer from several fundamental open questions [2], as organized below: (i) Interoperability: How can assets or other data be interoperated and transferred among different blockchains? (ii) Upgradability: How can a new functionality and an implementation problem, e.g., smart contract and transaction malleability [3], be supported and corrected in a deployed blockchain?

Y. Lai et al. (Eds.): ICA3PP 2021, LNCS 13157, pp. 376–396, 2022.
https://doi.org/10.1007/978-3-030-95391-1_24

Fortunately, a major approach to overcoming the above-raised questions is to use *sidechains* [4], which are a technology that can allow different blockchains to communicate with each other and react to events taking place in the other as desired. At present, sidechains have two forms. The former is parallel chains. This means that any two chains, for example, Bitcoin [1] and Ethereum [5], are treated as equals; any of them can be the sidechain of the other. The latter is parent-child chains. In this case, a sidechain can be a "child" of a parent chain; the child chain is somehow bootstrapped from the parent chain.

There are, however, concerns regarding the deficiency of security and efficiency in existing constructions for proof of work sidechains [2,4,6–13], which are explained as below:

- Some existing constructions for sidechains are highly centralized making them resistant to change, vulnerable to attacks and failures [14–16]. For example, trusted intermediaries on the involved blockchains are responsible for maintaining and managing the exchange of cross-chain assets, which will inevitably lead to a single-point-failure. Moreover, centralized constructions cast a major contradiction to the decentralized nature of blockchain.
- Cross-chain proof used for testifying the validity of cross-chain assets, consisting of a list of block headers that increases linearly with the size of the entire blockchain as well as a cryptographic proof, is fairly large, which reduces the efficiency of the constructions and limits potential scalability development.
- About security properties, many constructions for sidechains lack formal definition and rigorous proof. The security of cross-chain assets transfer can not be guaranteed. Even worse, when a sidechain is corrupted, another sidechain is unable to avoid the damage caused by that compromised sidechain.

Those challenges motivate our work.

1.1 Our Contributions

We present SEPoW, a secure and efficient construction that is suitable for proof of work sidechain systems, to complement deficiencies in the security and efficiency in existing constructions for proof of work sidechains.

In this work, we show that our SEPoW is decentralized and allows bidirectional communication between proof of work blockchains without trusted intermediaries. This means that the exchange of all cross-chain assets is managed not by a centralized party but by all honest nodes in participating blockchains.

To reduce the size of a cross-chain proof, we introduce the merged mining mechanism into our SEPoW such that a public chain of block headers is shared by the participating blockchains. Exploiting this, the proof consists of two Merkle tree paths: a transaction Merkle tree path and a merged Merkle tree path; their sizes depend on the number of transactions included in a block and the number of participating blockchains, respectively, regardless of the size of the current blockchain.

Next, we prove the security of SEPoW using a secure cross-chain proof protocol and a collision resistant hash function. Our SEPoW captures: (i) cross-chain assets can be transferred securely when the security assumptions of the participating blockchains are held, namely that an honest majority of computational power exists in the participating blockchains, and (ii) the firewall property maintained by SEPoW guarantees that any catastrophic blockchain corruption, such as a violation of the blockchain security assumptions, does not impact other blockchains.

Further, we present a concrete exemplary construction for proof of work sidechains. For conciseness our SEPoW is outlined with regard to a generic proof of work (PoW) blockchain consistent with the Bitcoin protocol [1] that underlies the Bitcoin blockchain, which is the most popular PoW blockchain so far.

We also evaluate the size of SEPoW proof and compare it with the state-of-the-art PoW sidechains protocols. Results demonstrate that SEPoW achieves a proof size of 416 bytes which is roughly 123x, 510x and 62000x smaller than zkRelay proof [12], PoW sidechains proof [7,17] and BTCRelay proof [6] for the blockchain length of 300000 and a 1 MBytes block.

1.2 Organization

The rest of this paper is organized as follows. In Sect. 2, preliminaries of SEPoW are introduced. We provide an exemplary concrete construction for proof of work sidechains in Sect. 3. The security of SEPoW and the discussion of merged mining are proved and presented in Sect. 4 and Sect. 5, respectively. In Sect. 6, we evaluate the size of SEPoW proof and compare it with the state-of-the-art work. Related works are proposed in Sect. 7. We conclude this paper in the last section.

2 Preliminaries

2.1 Cross-Chain Proof Protocol

We introduce a cross-chain proof protocol [17] that attests to the validity of cross-chain transactions into SEPoW. In this protocol the prover is a miner or full node on a chain denoted by $\mathcal{C}_1$; the verifier does not have access to $\mathcal{C}_1$, but holds a list of block headers on the chain denoted by $\mathcal{C}_2$ he/she participates in. The prover wants to convince the verifier that the predicate (i.e., the transaction tx is confirmed in $\mathcal{C}_1$) is true by sending a valid proof.

The definition of a cross-chain proof protocol is given below.

Definition 1 (Cross-chain proof protocol). *A cross-chain proof protocol for a predicate* $\textsc{q}$ *(i.e., an event occurred on blockchain) is a pair of probabilistic polynomial time (PPT) algorithms (P, V) such that:*

- ***P.*** *The algorithm takes as input a full chain* $\mathcal{C}$*, and outputs proof* π *about the predicate* $\textsc{q}$*. We note this as* $\pi \leftarrow P\ (\mathcal{C})$*.*
- ***V.*** *The algorithm takes as input a proof* π *produced by an honest node or a malicious node, and outputs a decision* $d \in \{T, F\}$*. We note this as* $d \leftarrow V\ (\pi)$*. The predicate* $\textsc{q}$ *is true if* $V(\pi) = T$ *and false otherwise.*

A desired security property of a cross-chain proof protocol is shown as follows.

Definition 2 (Security). *A cross-chain proof protocol for a predicate* $\textsc{q}$ *is secure if for all environments and for all PPT adversaries* $\mathcal{A}$ *and for all rounds* $r \geq \eta k$*, if* V *receives a set of proofs* Π *at the beginning of round* r*, at least one of which has been generated by the honest prover* $\mathcal{P}$*, then the output of* V *at the end of round* r *has the following constraints:*

– *If the output of* V *is false, then the evaluation of* $\textsc{q}$ *for all honest nodes must be false at the end of round* $r - \eta k$*.*
– *If the output of* V *is true, then the evaluation of* $\textsc{q}$ *for all honest nodes must be true at the end of round* $r + \eta k$*.*

Note that the parameter η represents the rate at which new blocks are produced and k is the number of subsequent blocks. See more details in [17].

2.2 Merged Mining

To generate a succinct cross-chain proof for a predicate, we introduce a merged mining mechanism [18,19] that allows a miner to produce multiple blocks at different chains with a single PoW solution. Exploiting this, a public chain of block headers, produced and maintained by the miners running the merged mining, is shared by the participating sidechains. Thus, the proof is only two Merkle tree paths: a transaction Merkle tree path and a merged Merkle tree path. As an exemplary concrete instantiation, in Sect. 3.1.4 we describe how a merged mining mechanism is useful for generating a succinct proof for a predicate.

Miners are allowed to perform merged mining for all involved sidechains based on their mining power. The process of the merged mining is as follows.

- Step 1 (Build merged-block-header). Miners will verify transactions for all involved c sidechains (an efficient way that verifies all transactions without monitoring all sidechains is given in Sect. 5) and collect c transaction Merkle tree root r_i, i.e., part **C** in Fig. 1a. Then, the miners calculate the root mr (part **B** in Fig. 1a) of the merged Merkle tree over the leaves encoded from the following TxRoots:
$$(r_1, r_2, ..., r_c). \tag{1}$$
- Step 2 (Search for nonce). Without merged mining, similar to Bitcoin, miners are required to search for n nonce ne that each satisfies
$$\mathrm{H}(pub, r, ne) \leq \mathrm{T}\,. \tag{2}$$
Where pub denotes public block parameters described in part **A** in Fig. 1a. With merged mining, miners only need to search for a nonce ne that satisfies
$$\mathrm{H}(pub, mr, ne) \leq \mathrm{T}\,. \tag{3}$$
Here we assume the difficulty target T is the same for all involved sidechains.

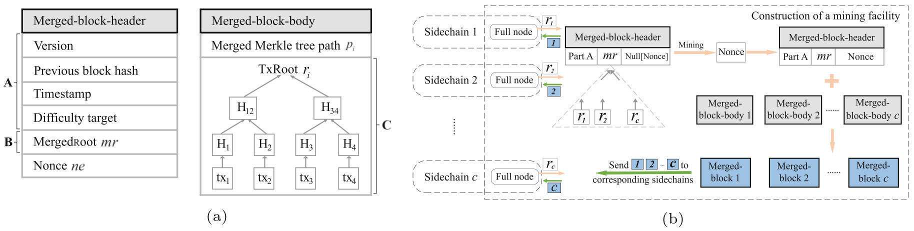


Fig. 1. The construction of a merged-block (a) and a mining facility (b).

- Step 3 (Diffuse block). Upon finding a valid ne, the miner will compose sidechain-specific merged-blocks and send them to corresponding sidechain networks, as described in Fig. 1b. It is worth pointing out that a merged-block-header is shared by these merged-blocks. As shown in Fig. 1b, for each sidechain, a merged-block includes a merged-block-header and a merged-block-body. A merged Merkle tree path p connecting r to mr is used to verify whether r is tied in the corresponding merged-block-header.
- Step 4 (Extend blockchain). In each sidechain, miners and full nodes verify the merged-block according to the specification of a block. Then the sidechain-specific merged-block will be appended in the corresponding sidechain if it is considered valid.

2.3 Security Definition of Sidechains

The first security definition of sidechains was introduced by Gaži et al. [2]. In a secure sidechains system, two fundamental properties, persistence and liveness, are necessary because a robust public transaction ledger must satisfy persistence and liveness [20]. Especially, a critical security feature that a secure sidechains system should have is the firewall property in which any catastrophic chain corruption, such as a violation of the chain security assumptions, does not impact other chains.

A formal security definition of sidechains is presented as follows.

Definition 3 (Sidechains security). *A system-of-sidechain ledgers protocol Π for $\{\mathbf{L}_i\}_{i\in[c]}$ is pegging-secure with liveness parameter $u \in \mathbb{N}$ with respect to:*

- *a set of assumptions $\mathbb{A}_i$ for ledgers $\{\mathbf{L}_i\}_{i\in[c]}$,*
- *a merge mapping* merge $(\cdot)$,
- *validity languages $\mathbb{V}_{\mathsf{A}}$ for each $\mathsf{A} \in \bigcup_{i\in[c]} \mathsf{Assets}\ (\mathbf{L}_i)$,*

if for all PPT adversaries, all rounds r and for $\mathcal{S}_r \triangleq \{i : \mathbb{A}_i[r]\ \text{holds}\ \}$ we have that except with negligible probability in the security parameter:

- ***Ledger persistence:** For each $i \in \mathcal{S}_r, \mathbf{L}_i$ satisfies the persistence property.*
- ***Ledger liveness:** For each $i \in \mathcal{S}_r, \mathbf{L}_i$ satisfies the liveness property parametrized by u.*

- ***Firewall:*** *For all* $\mathsf{A} \in \bigcup_{i\in\mathcal{S}_r} \mathsf{Assets}\left(\mathbf{L}_i\right)$

$$\pi_{\mathsf{A}}\left(\mathrm{merge}\left(\{\mathbf{L}_i^{\cup}[r] : i \in \mathcal{S}_r\}\right)\right) \in \pi_{\mathcal{S}_r}\left(\mathbb{V}_{\mathsf{A}}\right).$$

Where $\mathbb{A}_i$ denotes the security assumption of a ledger $\mathbf{L}_i$. For instance, a majority of authority (computational power, stakeholding, or node) is never controlled by the adversary. merge$(\cdot)$ is a function that combines a set of ledger states $\mathsf{ST} = \{\mathbf{L}_1, \mathbf{L}_2, ..., \mathbf{L}_c\}$ into a single ledger state denoted by merge(ST). A concrete instantiation of merge$(\cdot)$ we will give later. For each asset denoted by A, the validity language $\mathbb{V}_{\mathsf{A}}$ can capture specific rules of behavior for A, e.g., an asset A is transferred from a chain to the other chain. Note that $\mathsf{Assets}\left(\mathbf{L}\right)$ is the set of all assets that belong to the ledger $\mathbf{L}$. π is a ledger state projection. Specifically speaking, $\pi_{\mathsf{A}}\left(\mathbf{L}\right)$ denotes the projection of the transactions of $\mathbf{L}$ with respect to the asset A.

2.4 An Example of an Asset A

In this part we describe an example of a fungible asset A, and present the validity language $\mathbb{V}_{\mathsf{A}}$ with respect to the asset A. This example is a modification version based on an asset example in [2].

Instantiating $\mathbb{V}_{\mathsf{A}}$. For validity language $\mathbb{V}_{\mathsf{A}}$ we consider two ledger: the main-chain ledger $\mathbf{L}_1 \triangleq \mathbf{MC}$ and the sidechain ledger $\mathbf{L}_2 \triangleq \mathbf{SC}$. For the asset A, each transaction tx has the form tx = ((oAddr, UTXO, σ), (dAddr, a, π)), here:

- oAddr is an origin address on the origin ledger $\mathbf{L}_{\mathsf{ori}}$. Note that a bitcoin address is derived from its public key.
- UTXO is an unspent transaction output. A UTXO represents the unspent amount, and is locked by the private key held by the sender.
- σ is a signature generated by the sender on the metadata ((oAddr, UTXO), (dAddr, a, π)), which is used to unlock a UTXO.
- dAddr is a destination address on the destination ledger $\mathbf{L}_{\mathsf{des}}$. We say either $\mathbf{L}_{\mathsf{ori}} = \mathbf{L}_{\mathsf{des}}$, meaning that tx is a *local transaction*, or $\mathbf{L}_{\mathsf{ori}} \neq \mathbf{L}_{\mathsf{des}}$, meaning that tx is a *cross-chain transfer transaction.*
- a is the transfer amount.
- π is the succinct proof data that validates the validity of a *cross-chain transfer transaction*. Note that π is empty iif tx is a *local transaction* or an *origin transaction.*

Instantiating Merge$(\cdot)$. A merge$(\cdot)$ function takes as input a pair of ledger states ($\mathbf{MC}$, $\mathbf{SC}$) outlined above, and outputs a single ledger state.

3 Implementing Sidechain Ledger

We provide SEPoW for sidechains that are based on Bitcoin. Our protocol will execute a system of sidechains with sidechain security with respect to Definition 3 under an assumption on honest hashing-power majority. Here our SEPoW adopts the form of parent-child chains.

The main challenge in SEPoW is how to ensure secure cross-chain transfers. To achieve this, we introduce a cross-chain proof protocol described above into SEPoW. Consider two sidechains $\mathcal{C}_1$ and $\mathcal{C}_2$ (the notations $\mathcal{C}_1$ and $\mathcal{C}_2$ will be used throughout the rest of this paper), as well as a predicate (e.g., a cross-chain transaction took place in the sidechain $\mathcal{C}_1$ (resp. $\mathcal{C}_2$)). A cross-chain proof protocol for the predicate means that, the prover of $\mathcal{C}_1$ (resp. $\mathcal{C}_2$) can convince the verifier of $\mathcal{C}_2$ (resp. $\mathcal{C}_1$) that the predicate is true by generating a valid proof.

It is easy to establish *valid but not succinct* cross-chain proof for any computable predicate: the prover provides the entire linearly-growing chain of block headers as proof and the verifier simply selects the longest chain. To address this problem, we also introduce a merged mining mechanism described above into SEPoW, which allows the miners of sidechains to generate multiple blocks at different sidechains with a single PoW solution. Exploiting this, a public chain of block headers is shared by the participating sidechains. In this case, our proof is composed of two Merkle tree paths and thus is succinct.

3.1 The Sidechain Construction

We now present an elaborate module design of SEPoW based on the fundamental building block described above: cross-chain proof protocol and merged mining. These interacting modules are initialization, maintenance, cross-chain transfer and generating cross-chain proof. A graphical depiction about SEPoW is shown in Fig. 2.

3.1.1 The Sidechain Initialization

The initialization of a new **SC** can be launched by any miners of **MC** deploying the configuration transaction that configures **SC** described below. This action only requires the miners to follow the rule in the configuration to support it (i.e., following merged mining). A sidechain that is successfully created will obtain a unique identifier $\mathtt{id}_{\mathbf{SC}}$.

Consider two rounds d_η, s_η on **MC**, as described in Fig. 2. d_η means that the first configuration transaction tx_0 has been included in a block on **MC**. s_η denotes the start time of the sidechain if it is successfully activated. The configuration transaction contains a set of predefined rules that describe how to activate the sidechain and determine s_η successfully. A typical example is as follows: a sidechain starts in the **MC**-round s_η that meets: (i) $\mathrm{s}_\eta - \mathrm{d}_\eta > v_1$, here v_1 denotes the number of round, which is used to determine the round s_η; (ii) at least v_2-majority mining power on **MC** is controlled by honest miners that have supported **SC**.

The process of the activation is as follows (see more details in Algorithm 1).

First, the miners of **MC** that support the sidechain mine a new block including the configuration transaction tx_0 on mainchain. If the sidechain is successfully activated, then during the round s_η the miners of **MC** that support the

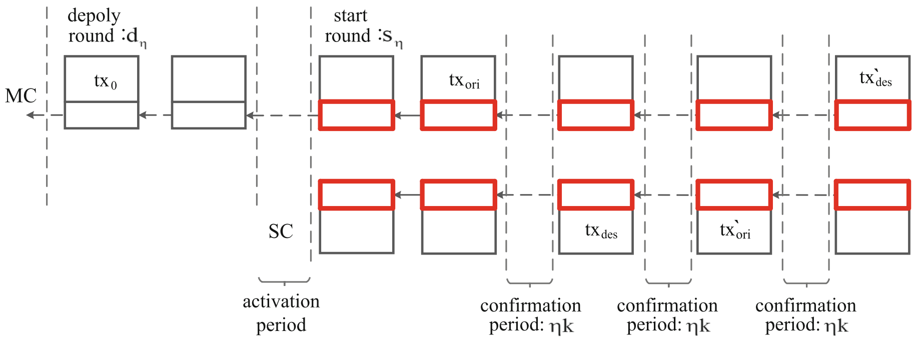


Fig. 2. An overview of SEPoW. **MC** is at the top, while **SC** is at the bottom. d_η is the round of deploying sidechain. s_η is the round of starting sidechain. Solid arrows is used to connect adjacent blocks. Dashed arrows denote some blocks are omitted. Red rectangles denote some merged-block-headers. Transactions tx of interest: tx_0. A configuration transaction; $\mathsf{tx}_{\mathsf{ori}}$. An origin transaction carrying withdraw operation from **MC** to **SC**; $\mathsf{tx}_{\mathsf{des}}$. A corresponding deposit transaction from **MC** to **SC**; $\mathsf{tx}'_{\mathsf{ori}}$. An origin transaction carrying withdraw operation from **SC** to **MC**; $\mathsf{tx}'_{\mathsf{des}}$. A corresponding deposit transaction from **SC** to **MC**.

sidechain create a genesis block GB = (merged-block-header $\triangleq$ (pub, mr, ne, $\mathtt{id}_{\mathbf{SC}}$), merged-block-body $\triangleq$ (p, r)) for **SC**. Here there are the variables pub, ne, p, r, as described in Fig. 1a. Note that GB can be created as soon as ne is found. In addition, if the activation of the sidechain fails, the initialization of the sidechain is aborted.

To demonstrate that **SC** has been successfully created, the miners of **MC** that adopted **SC** broadcast a special transaction `success_sidechain(id`$_{\mathbf{SC}}$`)` into **MC**. Otherwise, the miners of **MC** that supported **SC** broadcast another special transaction `failure_sidechain(id`$_{\mathbf{SC}}$`)` into **MC**. In this case, we will deduce whether the sidechain is valid, according to the transactions of **MC** only.

3.1.2 The Sidechain Maintenance

Once **SC** is successfully created, both **MC** and **SC** will be maintained by miners and their respective full nodes.

For mainchain, its maintenance procedure is executed by **MC**-workers who are acted by the miners running merged mining and the full nodes of **MC** running the proof of work protocol, cf. Algorithm 2. For concreteness, the miners attempt to mine new blocks on top of both the longest $\mathcal{C}_{\mathbf{MC}}$ and $\mathcal{C}_{\mathbf{SC}}$ by running the merged mining mechanism described above, and then broadcast them as soon as their nonce ne is found. The full nodes of **MC** only maintain the longest $\mathcal{C}_{\mathbf{MC}}$.

MC-workers, on every new round after the round s_η, receive all the possible **MC**-chains $\mathcal{C}_{\text{MC-}col}$ from the other peers via `Diffuse`, and then check them to find the "best" chain denoted by $\bar{\mathcal{C}}$. In this case they choose the chain $\bar{\mathcal{C}}$. Adopting the chain $\bar{\mathcal{C}}$ is done for chain validity function (using `Check-Chain` given in Line 4 of

Algorithm 1. SC initialization.

```
1: upon DeploySidechain($\mathrm{id}_{\mathbf{SC}}$) do
2:     state_sidechain[$\mathrm{id}_{\mathbf{SC}}$] $\leftarrow$ initializing
3:     block $\leftarrow$ pack configuration_sidechain
4:     mine block on MC
5: end upon
6: upon MC.NewRound() do
7:     s $\leftarrow$ MC.RoundIndex()
8:     $f_1 \triangleq$ (state_sidechain[$\mathrm{id}_{\mathbf{SC}}$]=initialized)
9:     if $f_1$ = true then
10:        if FailureActivation then
11:            state_sidechain $\leftarrow$ failed
12:            block $\leftarrow$ pack failure_sidechain
13:            mine block on MC
14:        else if SuccessActivation then
15:            state_sidechain $\leftarrow$ initialized
16:            block $\leftarrow$ pack success_sidechain
17:            $\mathrm{s}_\eta$ $\leftarrow$ ActivationRound()
18:            mine block on MC
19:        end if
20:    end if
21:    $f_2 \triangleq (\mathrm{s} = \mathrm{s}_\eta)$
22:    if SuccessActivation $\wedge$ ($f_2$ = true) then
23:        $ne \leftarrow$ search(nonce)
24:        header $\leftarrow$ ($pub$, $ne$, $mr$, $\mathrm{id}_{\mathbf{SC}}$)
25:        body $\leftarrow$ $(p, r)$
26:        GM $\leftarrow$ (header, body)
27:    end if
28: end upon
```

Algorithm 2. MC maintenance.

```
1: upon MC.NewRound do
2:     $\mathcal{C}_{\text{MC-}col}$ $\leftarrow$ chains collected via Diffuse
3:     $\triangleright$ chain validity
4:     $\mathcal{C}_{valchain}$ $\leftarrow$ Check-Chain($\mathcal{C}_{\text{MC-}col}$)
5:     $\triangleright$ chain comparison
6:     $\bar{\mathcal{C}}$ $\leftarrow$ Check-Comparison($\mathcal{C}_{valchain}$)
7:     $\mathsf{tx}_\mathsf{s}$ $\leftarrow$ transactions collected via Diffuse
8:     $\triangleright$ transaction validity
9:     $\mathsf{tx}_\mathsf{valtx}$ $\leftarrow$ Check-Tx($\mathsf{tx}_\mathsf{s}$)
10:    $\triangleright$ extend chain by invoking merged mining
11:    $(B_{\text{MC}}, B_{\text{SC}})$ $\leftarrow$ Merged-PoW($\mathsf{tx}_\mathsf{valtx}$, $\bar{\mathcal{C}}$)
12:    $\bar{\mathcal{C}} \leftarrow \bar{\mathcal{C}} \,||\, B_{\text{MC}}$
13:    $\tilde{\mathcal{C}}$ $\leftarrow$ receive the best chain w.r.t. SC via Diffuse
14:    $\tilde{\mathcal{C}} \leftarrow \tilde{\mathcal{C}} \,||\, B_{\text{SC}}$
15:    respective networks $\leftarrow$ Diffuse($\bar{\mathcal{C}}$, $\tilde{\mathcal{C}}$)
16: end upon
```

Algorithm 2) and chain comparison function (using `Chain-Comparison` given in Line 6 of Algorithm 2), as well as transaction validity function (using `Check-Tx` given in Line 9 of Algorithm 2). Then, the miners try to extend $\tilde{\mathcal{C}}$ by running the merged mining (using `Merged-PoW` given in Line 11 of Algorithm 2) described below.

Most importantly, the miners can simultaneously extend $\mathcal{C}_{\mathbf{MC}}$ and $\mathcal{C}_{\mathbf{SC}}$ by invoking `Merged-PoW`. In this case, a public chain of block headers is formed naturally in $\mathcal{C}_{\mathbf{MC}}$ and $\mathcal{C}_{\mathbf{SC}}$. Let us recall the work of `Merged-PoW`. First, the miners collect two valid TxRoot r from $\mathcal{C}_{\mathbf{MC}}$ and $\mathcal{C}_{\mathbf{SC}}$, and then calculate the root mr of the merged Merkle tree over the two TxRoot r. Next, the miners search for a nonce ne that satisfies $\mathrm{H}(pub, mr, ne) \leq \mathrm{T}$ (see more details in Eq. 3). Once ne is found, the miner composes specific-chain merged-blocks, consisting of a public merged-block-header and a specific-chain merged-block-body (as described in Fig. 1a), and sends them (B_{MC}, B_{SC}) to corresponding blockchain networks. Finally, workers of **MC** (resp. **SC**) verify B_{MC} (resp. B_{SC}) and append it to $\mathcal{C}_{\mathbf{MC}}$ (resp. $\mathcal{C}_{\mathbf{SC}}$) if it is considered valid. As a result, a public chain of merged-block-headers is formed naturally in $\mathcal{C}_{\mathbf{MC}}$ and $\mathcal{C}_{\mathbf{SC}}$.

Regarding the sidechain, its maintenance procedure is similar to the mainchain. Hence we only present their differences.

The main difference is that in Algorithm 2, **MC**-workers, all the possible **MC**-chains $\mathcal{C}_{\text{MC-}col}$, the two new blocks B_{MC} and B_{SC}, as well as all occurrences of the best chain $\tilde{\mathcal{C}}$ with respect to **SC**, are respectively replaced by **SC**-workers, **SC**-chains $\mathcal{C}_{\text{SC-}col}$, B_{SC} and B_{MC}, as well as the best chain $\mathcal{C}$ with respect to **MC**.

Algorithm 3. Transferring funds from **MC** into **SC**.

```
1: ▷ withdraw operation
2: function Withdraw(oAddr, dAddr, UTXO)
3:     $\sigma \leftarrow Sign_{sk}$ ((oAddr, UTXO), (dAddr, $a$))
4:     $\mathsf{tx}_{\mathsf{ori}} \leftarrow$ ((oAddr, UTXO, $\sigma$), (dAddr, $a$))
5:     mine block carrying $\mathsf{tx}_{\mathsf{ori}}$ on MC
6: end function
7: ▷ deposit operation
8: function Deposit(oAddr, dAddr, $a$)
9:     ▷ $\mathsf{tx}_{\mathsf{ori}}$ is included in the stable MC
10:    wait until block buried under $k$ blocks
11:    $\pi \leftarrow$ generate a cross-chain proof about $\mathsf{tx}_{\mathsf{ori}}$
12:    $\sigma \leftarrow Sign_{sk}$ ((oAddr, UTXO), (dAddr, $a$, $\pi$))
13:    $\mathsf{tx}_{\mathsf{des}} \leftarrow$ ((oAddr, UTXO, $\sigma$), (dAddr, $a$, $\pi$))
14:    mine block carrying $\mathsf{tx}_{\mathsf{des}}$ on SC
15: end function
```

3.1.3 Cross-Chain Transfer

Now nodes (or clients) can move funds from **MC** to **SC** by a cross-chain transfer transaction, which consists of a transaction $\mathsf{tx}_{\mathsf{ori}}$ carrying the withdraw operation, and a transaction $\mathsf{tx}_{\mathsf{des}}$ carrying the deposit operation. Two of them have the same fields, except for that metadata π for $\mathsf{tx}_{\mathsf{ori}}$ is empty and metadata π for $\mathsf{tx}_{\mathsf{des}}$ includes a cross-chain proof. The *origin transaction* $\mathsf{tx}_{\mathsf{ori}}$ that only involves the state in **MC** is handled by **MC**-workers, while the *destination transaction* $\mathsf{tx}_{\mathsf{des}}$ that only involves the state in **SC** is handled by **SC**-workers.

Moving funds from the mainchain **MC** into the sidechain **SC** works as follows. First, a client on **MC** diffuses $\mathsf{tx}_{\mathsf{ori}}$ with the desired UTXO and the valid receiving address on **SC**. If $\mathsf{tx}_{\mathsf{ori}}$ is considered valid, the corresponding block B carrying $\mathsf{tx}_{\mathsf{ori}}$ will be generated by the miner and only be appended to **MC**; the withdraw operation will be executed.

When B has been buried under k blocks within **MC**, **MC**-workers create a cross-chain proof π about the predicate, claiming that $\mathsf{tx}_{\mathsf{ori}}$ has been included in the stable mainchain **MC**. Here, π contains a transaction Merkle tree path for $\mathsf{tx}_{\mathsf{ori}}$ as well as a merged Merkle tree path p. The construction of π is described below. We now suppose that π has been produced, and then received by the client that will diffuse it.

After that, the corresponding $\mathsf{tx}_{\mathsf{des}}$ is composed by the client in **MC** and forwarded to the sidechain **SC**. If $\mathsf{tx}_{\mathsf{des}}$ is considered valid, it contains: (i) a valid cross-chain proof π; (ii) a valid signature σ; (iii) sufficient UTXO that is not less than the transfer amount a. If included, the deposit operation will be executed, concluding the completion of transferring from **MC** to **SC**. The core of transferring from **MC** to **SC** is shown in Algorithm 3.

Withdrawing to MC. Clients can then transfer their funds from **SC** back into **MC**. They follow the reverse procedure of Algorithm 3.

3.1.4 Generating Cross-Chain Proof

In the part we present a concrete construction of a cross-chain proof π. Let us consider cross-chain transactions consisting of the origin transaction $\mathsf{tx}_{\mathsf{ori}}$ occurring in **MC** and the destination transaction $\mathsf{tx}_{\mathsf{des}}$ occurring in **SC**, as well as the predicate Q which claims that $\mathsf{tx}_{\mathsf{ori}}$ has been included in block B in the stable **MC**. To maintain the transfer of cross-chain assets for the **SC** verifiers (i.e., full nodes in **SC**) that cannot evaluate Q, a cross-chain proof that deduces Q is essential.

When the block B carrying $\mathsf{tx}_{\mathsf{ori}}$ has been buried under k blocks within **MC**, the cross-chain proof π about Q is produced by the provers on **MC**, and contains:

– **The transaction Merkle tree path.** The path for $\mathsf{tx}_{\mathsf{ori}}$ is produced from the transaction Merkle tree located in B' body (see more details in Fig. 1a). It testifies that $\mathsf{tx}_{\mathsf{ori}}$ is aggregated in the transaction Merkle tree root r.
– **The merged Merkle tree path $\boldsymbol{p}$.** The path is generated from the merged Merkle tree over the pair of (r_1, r_2) of the involved mainchain and sidechain. $\boldsymbol{p}$ connecting the transaction Merkle tree root r to the merged Merkle tree root mr is used to attest that r is tied in B' header.

The above two paths allow the provers to convince the verifiers that Q is true. Concrete space requirements about the proof are discussed in Sect. 6.1.

4 Proofs of Security

In this section, we first prove that SEPoW from Sect. 3 satisfies persistence and liveness, then prove that SEPoW achieves the firewall property, similar to the proof method of Gaži et al. [2].

4.1 Persistence and Liveness

Lemma 1 (Persistence and Liveness). *Consider SEPoW from Sect. 3 with respect to the assumptions $\mathbb{A}_{\mathbf{MC}}$ and $\mathbb{A}_{\mathbf{SC}}$. For all rounds r, if $\mathbb{A}_{\mathbf{MC}}[r]$ (resp. $\mathbb{A}_{\mathbf{SC}}[r]$) holds, then* **MC** *(resp.* **SC***) achieves persistence and liveness up to round r with overwhelming probability in k.*

Proof (*sketch*). We directly borrow previous work [20,21] to prove that both persistence and liveness hold. In [20] it was shown that the Bitcoin protocol with the honest majority of computational power provides three security properties: common prefix, chain quality and chain growth. Further, According to the work [21–23], persistence and liveness needed by a ledger can be derived from the above three properties. Therefore, SEPoW satisfies persistence and liveness, completing the proof.

4.2 Firewall Property

Lemma 2. *For all PPT adversaries $\mathcal{A}$, SEPoW from Sect. 3 using a secure cross-chain proof protocol and a collision resistant hash function achieves the firewall property with the assumptions $\mathbb{A}_{\mathbf{MC}}$ and $\mathbb{A}_{\mathbf{SC}}$ with respect to overwhelming probability in k.*

Proof (*sketch*). To illustrate that the firewall property holds, we employ the idea of computational reduction in our proof. The line of the proof is as follows: suppose the firewall property is broken, an insecure cross-chain proof protocol or hash function is used by our protocol; then we show the probability of using the insecure cross-chain proof protocol or hash function is negligible.
We denote by $\mathcal{A}$ an arbitrary PPT adversary attacking the firewall property, and denote by $\mathcal{Z}$ an arbitrary environment supporting the execution of $\mathcal{A}$. We will consider two PPT adversaries:

- $\mathcal{A}_1$ is an adversary attacking the cross-chain proof protocol.
- $\mathcal{A}_2$ is an adversary attacking the hash function.

Next, we start by describing the behavior of these adversaries.

The Adversary $\mathcal{A}_1$bf . First, it models the execution of $\mathcal{A}$. That is, $\mathcal{A}$ requests that a cross-chain proof about the predicate Q that an origin transaction is included in $\mathbf{MC}$ is produced (without loss of generality, in this proof we consider only the cross-chain transfers from $\mathbf{MC}$ to $\mathbf{SC}$ due to the symmetry of SEPoW), $\mathcal{A}_1$ calls its proof generation algorithm $\mathbf{P}$ (as described in Sect. 2.1) to get the corresponding proof to provide to $\mathcal{A}$.

$\mathcal{A}_1$ monitors the ledgers, $\mathbf{L_{MC}}$ and $\mathbf{L_{SC}}$, adopted by honest $\mathbf{MC}$-workers and $\mathbf{SC}$-workers, and for every round r checks the state of all honest $\mathbf{MC}$-workers and $\mathbf{SC}$-workers. To evaluate whether $\mathcal{A}$ has succeeded, the adversary inspects whether $\mathbf{L} = \text{merge}\,(\mathbf{L_{MC}}, \mathbf{L_{SC}}) \notin \mathbb{V}_{\mathsf{A}}$. If $\mathcal{A}_1$ can not find such a round r and entities $\mathbf{MC}$-workers, $\mathbf{SC}$-workers, it returns FAILURE.

Otherwise it exists a round r such that $\mathbf{L} = \text{merge}\,(\mathbf{L_{MC}}, \mathbf{L_{SC}}) \notin \mathbb{V}_{\mathsf{A}}$. Suppose that $\mathbf{L}'$ is the prefix of $\mathbf{L}$ that satisfies $\mathbf{L}' \notin \mathbb{V}_{\mathsf{A}}$ and $\mathsf{tx} = \mathbf{L}'[-1]$. If tx has $\mathsf{oAddr}(\mathsf{tx}) \notin \mathbf{MC}$ or $\mathsf{dAddr}(\mathsf{tx}) \notin \mathbf{SC}$, then $\mathcal{A}_1$ returns FAILURE. Otherwise $\mathsf{oAddr}(\mathsf{tx}) \in \mathbf{MC}$ and $\mathsf{dAddr}(\mathsf{tx}) \in \mathbf{SC}$ (and so $\mathsf{tx} \in \mathbf{L_{SC}}$). Therefore, there must exist a predicate $\textsc{q}$ that the origin transaction (tx') corresponding to tx is committed in $\mathbf{L_{MC}}$ is true.

Let $\textsc{q}^*$ be the set of $\mathbf{L_{MC}}$ including all predicates up to and containing $\textsc{q}$. We will show that $\textsc{q}^*$ must contain a predicate attested by a forgery cross-chain proof. $\mathcal{A}_1$ inspects every predicate $\textsc{q}_r \in \textsc{q}^*$. $\textsc{q}_r$ involves a proof π_r for an origin transaction. $\mathcal{A}_1$ produces a proof π_r' for $\textsc{q}_r$ based on the view of $\mathbf{SC}$-workers, and examines whether the following *predicate violation* condition holds:

$$(\textsc{q}_r \in \mathsf{true}) \wedge (\neg\, \mathrm{V}(\pi_r) \vee (\pi_r \neq \pi_r')). \tag{4}$$

Suppose it exists a round r^* that satisfies the condition (4) and then outputs the tuple $(\textsc{q}_{r^*}, \pi_{r^*})$. Otherwise $\mathcal{A}_1$ returns FAILURE.

The Adversary $\mathcal{A}_2$. Similar to $\mathcal{A}_1$, $\mathcal{A}_2$ models the execution of $\mathcal{A}$. When $\mathcal{A}$ requests that a proof of a predicate $\textsc{q}$ is created, $\mathcal{A}_2$ invokes its algorithm $\mathbf{P}$ to get the corresponding proof to provide to $\mathcal{A}$.

$\mathcal{A}_2$ monitors the ledgers, $\mathbf{L_{MC}}$ and $\mathbf{L_{SC}}$, adopted by honest $\mathbf{MC}$-workers and $\mathbf{SC}$-workers, and for every round r checks the state of all honest $\mathbf{MC}$-workers and $\mathbf{SC}$-workers. $\mathcal{A}_2$ checks whether $\mathbf{L} = \text{merge}\,(\mathbf{MC}, \mathbf{SC}) \notin \mathbb{V}_{\mathsf{A}}$, to evaluate whether $\mathcal{A}$ has succeeded. If the adversary can not find such a round r and entities $\mathbf{MC}$-workers, $\mathbf{SC}$-workers, it returns FAILURE. Suppose tx, tx' are as described in $\mathcal{A}_1$. If tx has $\mathsf{oAddr}(\mathsf{tx}) \notin \mathbf{MC}$ or $\mathsf{dAddr}(\mathsf{tx}) \notin \mathbf{SC}$, then $\mathcal{A}_2$ returns FAILURE. Then the predicate $\textsc{q}$ that tx' is committed in $\mathbf{L}_1$ is true, and the corresponding cross-chain proof π for tx' was created. If $\textsc{q}$ is false, then $\mathcal{A}_2$ returns FAILURE. When $\textsc{q}$ for $\mathbf{SC}$-workers is true, there exists a cross-chain proof π' that attests to tx' in the view of $\mathbf{SC}$-workers. Based on the above results, $\mathcal{A}_2$ finds a collision for hash function and returns it.

Probability Analysis. We define the following events:

– SC-FAILURE$[r]$: $\mathcal{A}$ is successful at round r, i.e., $\pi_{\mathsf{A}}\,(\text{merge}\,(\{\mathbf{L}_i[t] : i \in \mathcal{S}_t\})) \notin \pi_{\mathcal{S}_t}\,(\mathbb{V}_{\mathsf{A}})$.
– CCPP-BREAK: $\mathcal{A}_1$ finds such a round r^* such that the condition (4) holds.
– HASH-BREAK: $\mathcal{A}_2$ finds a collision for the hash function.

Next, we will argue that the probability $\Pr[\textsc{sc-failure}[r]]$ is negligible for every time round r. We will demonstrate this probability in three successive claims.

The first claim shows that one of $\textsc{ccpp-break}$, $\textsc{hash-break}$ happens if $\textsc{sc-failure}[r]$ happens. As a result, according to a union bound, we have

$$\Pr[\textsc{sc-failure}[r]] \le \Pr[\textsc{ccpp-break}] + \Pr[\textsc{hash-break}].$$

The other two claims show that $\Pr[\textsc{ccpp-break}]$ and $\Pr[\textsc{hash-break}]$ are negligible.

Claim 1: $\textsc{sc-failure}[r] \Rightarrow \textsc{ccpp-break} \vee \textsc{hash-break}$.

By the Lemma 2, there exist two ledgers $\mathbf{L_{MC}}$ and $\mathbf{L_{SC}}$. Thus $\textsc{sc-failure}[r]$ is meant to

$$\text{merge}(\{\mathbf{L_{MC}}[r], \mathbf{L_{SC}}[r]\}) \notin \mathbb{V}_{\mathsf{A}}.$$

Without loss of generality, suppose that tx, tx' and $\textsc{q}$ are as described in $\mathcal{A}_1$. By the Lemma 2 and the Lemma 2 from [2], we deduce that tx is a *destination transaction*, and $\mathsf{oAddr}(\mathsf{tx}) \in \mathbf{MC}$ and $\mathsf{dAddr}(\mathsf{tx}) \in \mathbf{SC}$. Thus, $\textsc{q}$ is true. If $\mathcal{A}_1$ discovers such a round r^* that satisfies the condition (4), then $\textsc{ccpp-break}$ has happened and the claim holds. Therefore, for each predicate $\textsc{q}_r$ involving a proof π_r, the following condition holds:

$$(\textsc{q}_r \in \mathsf{true}) \wedge \mathrm{V}(\pi_r) \Rightarrow (\pi_r = \pi_r'). \tag{5}$$

Thus, we have a set of all predicates, each of which is true and is proved by the corresponding cross-chain proof π_r; this equation $\pi_r = \pi_r'$ holds. However, the origin transaction tx' has been committed by π_r, but not committed by π_r'. Therefore, there must exist a Merkle tree collision, meaning that a hash collision found by $\mathcal{A}_2$ occurs. As a result, $\textsc{hash-break}$ happens.

Claim 2: $\Pr[\textsc{ccpp-break}]$ is negligible.

Suppose that $\textsc{ccpp-break}$ happens. We can discover that the condition (4) holds. This case, however, is negligible according to the assumption that the cross-chain proof protocol in use is secure.

Claim 3: $\Pr[\textsc{hash-break}]$ is negligible.

It is the same as claim 2, there exists another cross-chain proof π for the origin transaction corresponding to tx is created, which contradicts with the assumption that the hash function in use is collision-resistant.

Relying on the three claims above, we conclude that for negligible probability $\mathbf{negl}$, $\Pr[\textsc{sc-failure}] \le \mathbf{negl}$. Therefore, $\pi_{\mathsf{A}}(\text{merge}(\{\mathbf{L}_i[r] : i \in \mathcal{S}_r\})) \in \pi_{\mathcal{S}_r}(\mathbb{V}_{\mathsf{A}})$ holds, completing the proof.

The above two Lemmas directly prove Theorem 1 presented below.

Theorem 1 (Sidechains security). *SEPoW from Sect. 3 using a secure cross-chain proof protocol and a collision resistant hash function is secure with respect to assumptions $\mathbb{A}_{\mathbf{MC}}$ and $\mathbb{A}_{\mathbf{SC}}$, and* merge *and $\mathbb{V}_{\mathsf{A}}$ defined in Sect. 2.4.*

5 Discussion of Merged Mining

Transaction Verification. In the merged mining mechanism described in Sect. 2.2, merged miners should validate all transactions in sidechains; otherwise invalid transactions may be included in some sidechain transaction Merkle Trees. To achieve this, the merged miners are required to monitoring all involved sidechains. However, this contradicts the sidechains agnosticism [7]: miners of **MC** do not need to be aware of **SC** at all. Only the entities interested in facilitating cross-chain events must be aware of both. To resolve this question, we introduce a cryptographic tool, the recursive composition of zk-SNARKs [24,25], which can create proofs that attest to the validity of other proofs. We leverage the tool to generate a succinct proof of sidechain transaction Merkle tree that attests to the correctness of all transactions at the base of the tree. By verifying the validity of the proof, the merged miners can efficiently evaluate the validity of all transactions in the tree without re-executing transaction verification and monitoring all involved sidechains.

Consider an example that a list of transactions tx_1, tx_2, tx_3, tx_4 in a sidechain transaction Merkle tree needs to be proved. First, a full node in the sidechain generates "base" proofs proving the validity of single transactions. Then two adjacent "base" proofs are merged and further generate a new "merge" proof. Finally, these "merge" proofs are recursively merged into a "merge" proof called root proof. As a result, the root proof attests to the validity of all the transactions.

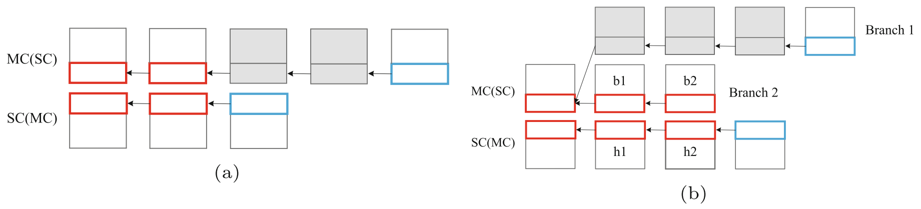


Fig. 3. Asynchronous extension of sidechains (a); Sidechain fork (b). (Color figure online)

Sidechains Extension. In SEPoW, some "regular miners" exist who do not run merged mining but only work on the chain they participate in. This will cause the computational power of each sidechain to be different. In this case, one sidechain extension will exceed the other sidechain. A typical example is shown in Fig. 3a. Where **MC** extension exceeds **SC**. A red public chain of block headers is broken due to the two gray blocks on **MC** that the "regular miners" mine. As a result, cross-chain proofs are affected.

To overcome this problem, merged miners need to rebuild a new public chain of block headers to inherit the previous public chain and backup the information testifying these gray blocks' validity into **SC**. For the former, merged miners run merged mining on the top of two chains and produce new public block headers (blue block headers in Fig. 3a). For the latter, full nodes on **MC** send gray blocks information, consisting of block headers verifying consensus and proofs attesting to the validity of transaction Merkle trees, to merged miners. Upon receiving them, merged miners check their validity by calling a hash function and the zk-SNARKs described above. Then merged miners include these hashes of valid gray block headers into subsequent merged-block-headers by merged mining (here, the hashes are seen as the leaves of a merged Merkle tree).

Fork Solution. The setting of merged mining seems to be a bit coercive in that it makes all sidechains collapse into one single blockchain. That is, the "fork" in a sidechain will affect all sidechains unstable. A typical example is shown in Fig. 3b. Where both **MC** (i.e., Branch 2) and **SC** maintain merged mining and form a red public chain of block headers. With blockchains growth, however, the length of Branch 1 exceeds Branch 2. Thus, branch 1 becomes the "main" chain according to the longest chain rule, while Branch 2 is discarded. Meanwhile, blocks b1 and b2 become "orphan" blocks. As a result, the red public chain of block headers on **MC** and **SC** is broken; cross-chain proofs are affected.

To complement this deficiency, the following efforts need to be done: 1) merged miners run merged mining on top of Branch 1 and **SC**, and create a new public chain of block headers (blue block headers in Fig. 3b); 2) the information of gray blocks on Branch 1 need to be backed up to **SC** via the way described in **Sidechains Extension**; 3) merged miners need to mark "orphan" blocks as invalid in subsequent merged-blocks. This is because adversaries may utilize the headers information of h1 and h2 blocks, including some proofs that testify to the validity of "orphan" blocks transactions, to implement the double-spend attack.

6 Performance Evaluation

In this section, we evaluate the size of SEPoW proof and compare it with the state-of-the-art work.

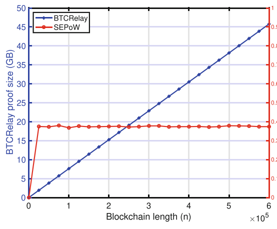
6.1 Size of Cross-Chain Proofs

We first evaluate the size of SEPoW proofs, consisting of a transaction Merkle tree path denoted by *path* and a merged Merkle tree path denoted by p. Similar to Bitcoin Core, we assume a 256-bit hash function is used to build the Merkle tree construction, and a block of 1 MBytes and a block header of 80 bytes are applied to SEPoW. More specifically, in 1 MB block including up to 2048 transactions (in fact, since January 1, 2020, a Bitcoin block includes 1869 transactions[1] on average), we have in bits: $|p| = |256 + 256| = 512$ bits, $|path| = |\log(2048) \times 256| = 2{,}816$ bits, and hence the size of SEPoW proofs is $|path| + |p| = 512 + 2816 = 416$ bytes.

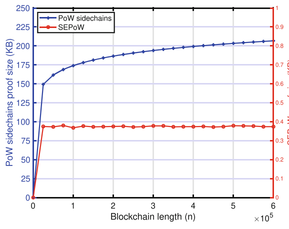
[1] See https://blockchair.com/zh/bitcoin/charts/total-transaction-count.

6.2 Comparison with Existing Work

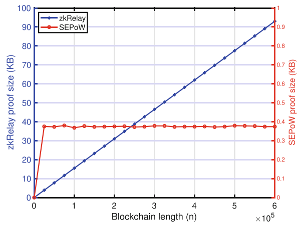
Without loss of generality, let us consider some work: BTCRelay [6], PoW sidechains [7,17] and zkRelay [12], which have been implemented or can be evaluated. Their cross-chain proofs sizes can be denoted as $O(n \cdot |BH| + \log_2 |BT| \cdot |H|)$ [6], $O(\log_{1/m}(2)\lambda \cdot ((\log_2(n)+1) \cdot |BH| + \log_2(n) \cdot \lceil \log_2(\log_2(n,2),2) \rceil \cdot |H|))$ [26] and $O(1/504 \cdot n \cdot |BH|)$ [12], respectively, according to their experimental results. Here, n is the length of a blockchain, $|BH|$ is the size of a block header, $|BT|$ is the number of transactions included in the block B, $|H|$ is the size of a hash, m and λ denote an attacker who controls a m fraction of honest chain's mining power succeeds with probability $2^{-\lambda}$.



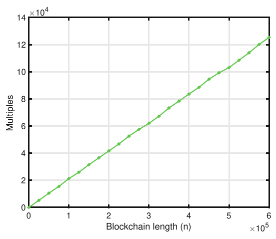
(a1) BTCRelay vs SEPoW



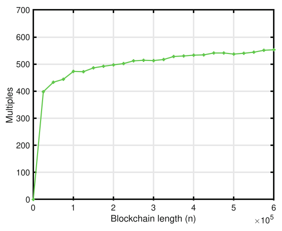
(a2) PoW sidechains vs SEPoW



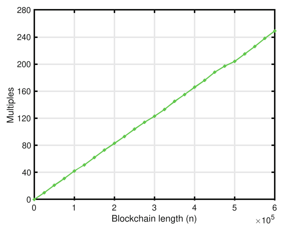
(a3) zkRelay vs SEPoW



(b1) Multiples



(b2) Multiples



(b3) Multiples

Fig. 4. Comparison of SEPoW, BTCRelay, PoW sidechains and zkRelay at different blockchain length, $|BH| = 80$ bytes, $|BT| = 2048$, $|H| = 32$ bytes, $m = 0.5$, $\lambda = 50$.

As depicted in Fig. 4, we make comparisons of SEPoW with BTCRelay, PoW sidechains and zkRelay in terms of the size of a cross-chain proof (The source code related to this comparison experiment has been released to the Github[2]). Figure 4a(1–3) show the impact of blockchain length on a cross-chain proof. Here, BTCRelay proof and zkRelay proof, located in Fig. 4a1 and 4a3, are linear and sublinear in the length of the blockchain, respectively; PoW sidechains proof, located in Fig. 4a2, is logarithmic in the length of the blockchain. More importantly, our SEPoW proof is composed of two Merkle tree paths regardless of the blockchain length, which significantly outperforms all proofs of the above, especially for longer chains.

[2] See https://github.com/01007467319/sepow.git.

Figure 4b(1–3) show the multiple relationships between SEPoW proof and the existing work at different blockchain length. Where, the multiple that SEPoW proof is less than BTCRelay proof and zkRelay proof, indicated by Fig. 4b1 and 4b3, increases linearly; the multiple that SEPoW proof is less than PoW sidechains proof, indicated by Fig. 4b2, is exponential growth. Note that for a blockchain length (n) of 300000 and block header size (|BH|) of 80 bytes, SEPoW achieves a proof size of 416 bytes which is roughly 123×, 510× and 62000× smaller than zkRelay proof, PoW sidechains proof and BTCRelay proof, respectively.

7 Related Work

Recently, some researchers have focused on federation construction for sidechains. Dilley et al. [9] designed a federation construction that allows cross-chain assets transfer between disparate blockchains. In this construction cross-chain assets are managed by a trusted committee and are transferred only when the majority of the committee sign cross-chain transactions. Similarly, Back et al. [4] proposed a federation cross-chain solution, in which cross-chain assets are transferred by a trusted group of parties. However, their work has not entirely overcome the political centralization risk as the federation constructions still rely on a trusted committee to maintain and manage cross-chain assets transfer.

To prevent centralization risk, decentralized constructions for sidechains have been proposed. Kiayias et al. [7] presented the first decentralized construction for proof of work sidechains. The construction that is built based on a cryptographic primitive, Non-Interactive Proofs of Proof-of-Work (NIPoPoWs) [17], allows the passing of any information between proof of work blockchains without a trusted third party. However, its cross-chain proof has linear in the length of the blockchain and thus is fairly large. Sztorc [10] and Lerner [11] proposed Drivechains, a decentralized construction for proof of work sidechains. In Drivechains cross-chain assets moved from Bitcoin to Drivechain are authenticated by SPV proofs consisting of all Bitcoin block headers. Yet, these SPV proofs are quite large. An implemented and decentralized construction for sidechains was given in BTCRelay [6]. It supports assets transferred from Bitcoin to Ethereum but not back. To verify the validity of the transactions that took place in Bitcoin, BTCRelay requires saving the entirety of Bitcoin block headers into Ethereum; limiting any potential scalability.

Some other studies are devoted to generating succinct cross-chain proofs about the transactions that took place in a blockchain. Garoffolo et al. [8] proposed Zendoo, a decentralized construction for blockchain systems that allows communication with different sidechains without trusted intermediaries. The construction introduces zk-SNARKs [27] to produce a constant-sized proof that attests to the validity of all cross-chain transactions during a period. Westerkamp et al. [12] presented an efficient sidechains construction, which uses zkSNARK-based chain-relays to generate a succinct proof that testifies the validity of cross-chain assets. The proof size does not grow linearly with the number of block headers but is constant for any batch size. However, these work lacked a formal security definition and proof.

In some different efforts, Gaži et al. [2] presented the first formal treatment of sidechains and proof of stake sidechains construction. To attest to the validity of cross-chain assets, they introduce a trust committee chosen among sidechain block creators to generate the sidechain certificates. The first Bitcoin sidechain in production was given in RSK [13]. It supports smart contracts functionality, compatible with the Ethereum standards. Yet, RSK proofs (from Bitcoin to RBTC) have linear complexity in the length of Bitcoin blockchain and RSK lacked a formal security definition and proof.

Furthermore, different ideas regarding cross-chain transfers, such as Polkadot [28], Cosmos [29], Tendermint [29], Blockstream's Liquid [30] and Interledger [30], have been proposed. Their construction was centralized and lacked formal security definition and proof. Other related effort also included COMIT [31,32], Plasma [33,34], NOCUST [35,36], Dogethereum [37] and XCLAIM [38].

8 Conclusion

In this paper, we proposed SEPoW, which makes up for deficiencies in security and efficiency in existing PoW sidechains construction. In SEPoW the exchange of all cross-chain assets is managed by all honest nodes in participating blockchains. To generate a succinct cross-chain proof, we utilized the merged mining to produce a constant-size proof, regardless of the size of the current blockchain. The security of SEPoW for PoW sidechains is proved formally. We presented a detailed design of SEPoW and evaluated the size of SEPoW proof; SEPoW achieves a proof size of 416 bytes which significantly outperforms all proofs of the existing work, especially for longer chains.

Acknowledgment. The authors would like to thank the anonymous reviewers of ICA3PP 2021 for their insightful suggestions. This work is partially supported by the Shandong Provincial Key Research and Development Program under Grant Number 2019JZZY020127.

References

1. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system (2008). http://bitcoin.org/bitcoin.pdf
2. Gazi, P., Kiayias, A., Zindros, D.: Proof-of-stake sidechains. In: S&P 2019, Piscataway, pp. 139–156. IEEE (2019)
3. Decker, C., Wattenhofer, R.: Bitcoin transaction malleability and MtGox. In: Kutyłowski, M., Vaidya, J. (eds.) ESORICS 2014. LNCS, vol. 8713, pp. 313–326. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11212-1_18
4. Back, A., et al.: Enabling blockchain innovations with pegged sidechains (2014). https://www.blockstream.com/sidechains.pdf
5. Buterin, V.: Ethereum: a next-generation smart contract and decentralized application platform (2014). https://www.github.com/ethereum/wiki/wiki/White-Paper

6. BTCRelay, Community: BTCRelay reference implementation (2017). https://www.github.com/ethereum/btcrelay
7. Kiayias, A., Zindros, D.: Proof-of-work sidechains. In: Bracciali, A., Clark, J., Pintore, F., Rønne, P.B., Sala, M. (eds.) FC 2019. LNCS, vol. 11599, pp. 21–34. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-43725-1_3
8. Garoffolo, A., Kaidalov, D., Oliynykov, R.: Zendoo: a zk-SNARK verifiable cross-chain transfer protocol enabling decoupled and decentralized sidechains. In: ICDCS 2020, Piscataway, pp. 1257–1262. IEEE (2020)
9. Dilley, J., Poelstra, A., Wilkins, J., Piekarska, M., Gorlick, B., Friedenbach, M.: Strong federations: an interoperable blockchain solution to centralized third party risks. CoRR arXiv:1612.05491 (2016)
10. Sztorc, P.: Drivechain - the simple two way peg (2015). https://www.truthcoin.info/blog/drivechain/
11. Lerner, S.D.: Drivechains, sidechains and hybrid 2-way peg designs (2016). https://docs.rsk.co/Drivechains_Sidechains_and_Hybrid_2-way_peg_Designs_R9.pdf
12. Westerkamp, M., Eberhardt, J.: zkRelay: facilitating sidechains using zkSNARK-based chain-relays. In: Euro S&P Workshops, Piscataway, pp. 378–386. IEEE (2020)
13. Lerner, S.D.: Rootstock: smart contracts on bitcoin network (2015). https://blog.rsk.co/wp-content/uploads/2019/02/RSK_White_Paper-ORIGINAL.pdf
14. Singh, A., Click, K., Parizi, R.M., Zhang, Q., Dehghantanha, A., Choo, K.R.: Sidechain technologies in blockchain networks: an examination and state-of-the-art review. J. Netw. Comput. Appl. **149**, 102471 (2020)
15. Atzei, N., Bartoletti, M., Cimoli, T.: A survey of attacks on Ethereum Smart Contracts (SoK). In: Maffei, M., Ryan, M. (eds.) POST 2017. LNCS, vol. 10204, pp. 164–186. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54455-6_8
16. Moore, T., Christin, N.: Beware the middleman: empirical analysis of bitcoin-exchange risk. In: Sadeghi, A.-R. (ed.) FC 2013. LNCS, vol. 7859, pp. 25–33. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39884-1_3
17. Kiayias, A., Miller, A., Zindros, D.: Non-interactive proofs of proof-of-work. In: Bonneau, J., Heninger, N. (eds.) FC 2020. LNCS, vol. 12059, pp. 505–522. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-51280-4_27
18. Judmayer, A., Zamyatin, A., Stifter, N., Voyiatzis, A.G., Weippl, E.: Merged mining: curse or cure? In: Garcia-Alfaro, J., Navarro-Arribas, G., Hartenstein, H., Herrera-Joancomartí, J. (eds.) ESORICS/DPM/CBT -2017. LNCS, vol. 10436, pp. 316–333. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-67816-0_18
19. Wang, J., Wang, H.: Monoxide: scale out blockchains with asynchronous consensus zones. In: Lorch, J.R., Yu, M. (eds.) NSDI 2019, pp. 95–112. USENIX Association, Berkeley (2019)
20. Garay, J., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol: analysis and applications. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 281–310. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_10
21. Pass, R., Seeman, L., Shelat, A.: Analysis of the blockchain protocol in asynchronous networks. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017. LNCS, vol. 10211, pp. 643–673. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56614-6_22
22. Kiayias, A., Panagiotakos, G.: Speed-security tradeoffs in blockchain protocols. IACR Cryptol. ePrint Arch. **2015**, 1019 (2015)

23. Kiayias, A., Russell, A., David, B., Oliynykov, R.: Ouroboros: a provably secure proof-of-stake blockchain protocol. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10401, pp. 357–388. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63688-7_12
24. Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Scalable zero knowledge via cycles of elliptic curves. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014. LNCS, vol. 8617, pp. 276–294. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44381-1_16
25. Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: Recursive composition and bootstrapping for SNARKS and proof-carrying data. In: Boneh, D., Roughgarden, T., Feigenbaum, J. (eds.) STOC 2013, pp. 111–120. ACM, New York (2013)
26. Bünz, B., Kiffer, L., Luu, L., Zamani, M.: FlyClient: super-light clients for cryptocurrencies. In: S&P 2020, Piscataway, pp. 928–946. IEEE (2020)
27. Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Succinct non-interactive zero knowledge for a von Neumann architecture. In: Fu, K., Jung, J. (eds.) USENIX Security 2014, pp. 781–796. USENIX Association, Berkeley (2014)
28. Wood, G.: Polkadot: vision for a heterogeneous multi-chain framework (2016). https://www.polkadot.network
29. Buchman, E.: Tendermint: byzantine fault tolerance in the age of blockchains (2016). https://github.com/tendermint/tendermint
30. Group, T.I.P.C.: Interledger protocol v4 (2021). https://www.interledger.org
31. Hosp, J., Hoenisch, T., Kittiwongsunthorn, P.: COMIT - cryptographically-secure off-chain multi-asset instant transaction network. CoRR arXiv:1810.02174 (2018)
32. Tian, W., Pan, W., Shaobin, C., Ying, M., Anfeng, L., Mande, X.: A unified trustworthy environment establishment based on edge computing in industrial IoT. IEEE Trans. Ind. Inform. **16**(9), 6083–6091 (2020). https://doi.org/10.1109/TII.2019.2955152
33. Poon, J., Buterin, V.: Plasma: scalable autonomous smart contracts (2017). https://www.plasma.io/plasma.pdf
34. Tian, W., et al.: Propagation modeling and defending of a mobile sensor worm in wireless sensor and actuator networks. Sensors **17**(1), 139 (2017). https://doi.org/10.3390/s17010139
35. Khalil, R., Gervais, A.: NOCUST - a non-custodial 2nd-layer financial intermediary. IACR Cryptol. ePrint Arch. **2018**, 642 (2018)
36. Mingfeng, H., Anfeng, L., Tian, W., Changqin, H.: Green data gathering under delay differentiated services constraint for internet of things. Wirel. Commun. Mob. Comput. **2018**, 1–23 (2018). https://doi.org/10.1155/2018/9715428
37. Teutsch, J., Straka, M., Boneh, D.: Retrofitting a two-way peg between blockchains. CoRR arXiv:1908.03999 (2019)
38. Zamyatin, A., Harz, D., Lind, J., Panayiotou, P., Gervais, A., Knottenbelt, W.J.: XCLAIM: trustless, interoperable, cryptocurrency-backed assets. In: S&P 2019, Piscataway, pp. 193–210. IEEE (2019)