Tri-op redactable blockchains with block modification, removal, and insertion

Mohammad Sadeq Dousti[1] and Alptekin Küpçü[2]

[1] Johannes Gutenberg University of Mainz, Mainz, Germany
modousti@uni-mainz.de
[2] Koç University, İstanbul, Turkey
akupcu@ku.edu.tr

Abstract. In distributed computations and cryptography, it is desirable to record events on a public ledger, such that later alterations are computationally infeasible. An implementation of this idea is called *blockchain*, which is a distributed protocol that allows the creation of an immutable ledger. While such an idea is very appealing, the ledger may be contaminated with incorrect, illegal, or even dangerous data, and everyone running the blockchain protocol has no option but to store and propagate the unwanted data. The ledger is bloated over time, and it is not possible to remove redundant information. Finally, missing data cannot be inserted later. *Redactable* blockchains were invented to allow the ledger to be mutated in a controlled manner. To date, redactable blockchains support at most two types of redactions: block modification and removal. The next logical step is to support block insertions. However, we show that this seemingly innocuous enhancement renders all previous constructs insecure. We put forward a model for blockchains supporting all three redaction operations, and construct a blockchain that is provably secure under this formal definition.

Keywords: Bitcoin, blockchain, redactable blockchain, block change, block insertion, block removal

1 Introduction

A traditional problem in distributed computing and cryptography is joint agreement: A set of parties want to reach at an agreement (consensus), while the communication is noisy, and some parties may cooperate and deliberately act maliciously [1]. Nakomoto [15] proposed a solution in the context of cryptocurrencies, called *Bitcoin*. The goal was a decentralized digital currency, where the peers jointly agree on performing financial transactions and register them in a distributed ledger. Since the transactions were grouped together to form blocks, and blocks were chained together in such a way that changing them is computationally infeasible, the protocol is referred to as *blockchain*. Blockchain provides a mechanism to append any type of data to the ledger, and it can be seen as a protocol to jointly agree upon and irrevocably register events on a distributed ledger.

In general, a blockchain can be permissionless or permissioned [12]: Bitcoin is permissionless, as the parties do not need authentication to join the protocol. Permissioned blockchains, on the other hand, require a party to authenticate first before being able to take part in the protocol. The latter type is quite common in exclusive environments, such as corporate applications (e.g., [4]). Due to the ability to identify parties and the possibility to remove the malicious ones, permissioned blockchains can make simplifying assumptions to achieve higher performance. However, they are not useful in environments without central trust.

Several issues with blockchains were brought into attention: Since the ledgers are immutable, unwanted data can be recorded in the ledger. Blockchain participants, who save the ledger locally and propagate the changes, have therefore no option but to save and propagate the unwanted data, including child pornography [10,11] and malware [17]. In some jurisdictions, the propagator of such illegal content is considered a complicit [14]. However, with an immutable ledger, the participants have no option to remove the illegal content, or stop propagating it.

Another possible issue with immutability is lack of privacy. Once private information is recorded in an immutable ledger, it cannot be removed. Therefore, immutable blockchains cannot observe the "right to be forgotten", endowed to users by privacy laws such as the EU General Data Protection Regulation (GDPR) or California Consumer Privacy Act (CCPA).

Protocol or implementation flaws pose another threat to immutability. In one famous case, the Ethereum cryptocurrency had an implementation flaw that made the DAO Attack [5] possible. As a result, malicious transactions were recorded in the immutable ledger of Ethereum. The only possible solution was for the majority of participants to upgrade their software. The new software resolved the flaw, effectively invalidating the malicious transactions. This is called a *hard fork*, and it is the last line of defense against such influential attacks on immutable blockchains.

Immutability has other drawbacks as well: The size of the ledger increases over time, and storing and processing it becomes increasingly resource intensive. There is no possibility to reduce the size of ledger by removing redundant information from it. Furthermore, there is no option to insert a block within the ledger, even if the participants later note that a block is mistakenly not registered (due to protocol issues, implementation mistakes, or oversight).

For these reasons, the concept of ***redactable*** blockchains was introduced. A redactable blockchain allows the ledger to be controllably mutated, once the need arises. In [8], two types of redactable blockchains were identified: (1) **Moderated:** In protocols of this type [2,6,9,8], a centralized authority (one or more administrators) have a secret redaction key. Once a redaction request is approved, it performs the required changes to the ledger using the secret key. The changes can be verified by all participants using the associated public key. (2) **Unmoderated:** In protocols of this type [18,7], the parties vote on redaction requests. Once a certain quorum of votes is attained, the redaction is applied. The votes are added to the ledger as evidence.

An important remark is to distinguish these concepts with permissioned or permissionless blockchains, as moderation has nothing to do with user authentication. Moderation authority exists only for controlling the redaction operations, but not permitting users to join the normal system. A blockchain can be both permissioned and moderated, but even in that case it is possible to have two sets of authorities: one for authorizing users to join the blockchain, and one for authorizing the redactions.

In this work, we opted for the *moderated* setting, since there are some serious attacks [7,8] against the existing unmoderated protocols.

1.1 Related work in the moderated setting

The concept of redactable blockchains was pioneered by Ateniese et al. [2]. Their main idea was to construct a protocol on top of Bitcoin, and therefore had to cope with the limitations posed by Bitcoin. The resulting construct was therefore complicated: They introduced a primitive called *enhanced chameleon hash function*, in which collisions are hard to find except for an entity owning a secret key. The suggested construction of such hash functions required non-standard assumptions and a large redaction witness (e.g., under the DLIN assumption, it required 39 group elements). Derler et al. [6] show how this idea can be further fine tuned to make the primitive policy based: Any entity that has enough privileges to satisfy a policy can find collisions. The idea is implemented by incorporating another primitive called a *ciphertext-policy attribute-based encryption* (CP-ABE).

Grigoriev and Shpilrain [9] suggested a simple and efficient solution based on RSA, but their construct is insecure [8]. Dousti and Küpçü [8] provided another simple solution based on any strongly unforgeable digital signature. The efficiency of their protocol is a result of completely departing from Bitcoin, and not being restricted to its data structures and verification algorithms.

Except for [2], all previous work [6,9,8,18,7] (both moderated and unmoderated) support only one type of redaction: Changing blocks. In [2], the authors show how block removals can be supported as well.

1.2 Contributions

We propose the first model and security definition for redactable blockchains that support three redaction operations: block change (`chg`), block insertion (`ins`) and block removal (`rem`). All previous redactable blockchains supported block change. Removal is previously used in one work [2] to shrink the ledger. To the best of our knowledge, we initiate the support for block insertion. While frequent insertions cause the ledger to bloat, discreet use of insertion might be useful in some scenarios. For instance, the participants might decide that a block B must have been recorded at index i, but the ledger did not record it due to a bug in the blockchain protocol/software, an oversight, or an attack. Instead of a hard fork, the ledger can be redacted by inserting B at index i.

Interestingly, we show that naively incorporating the seemingly innocuous insert operation renders the previous constructs insecure. We demonstrate this by what we call a "twin-block attack" on [2,6,9], and a "change-of-operation attack" on [8].

We then create a secure construct, and prove its security under our improved model. The construct alleviates all previous issues by introducing the concept of *triple versioning*, where each block holds three types of "versions": A globally unique version number, the type of operation based on which the block is created, and the intended index of the block. Our construct uses a single digital signature per redaction, and is thus very efficient. Furthermore, it does not require on any non-standard assumptions.

It is noteworthy that while devising our security model, we faced a choice. One choice resulted in a simpler model, but the security definition was be stricter. The other choice resulted in a more complex model with a laxer security definition. We opted for the former one. Surprisingly, it resulted in a simpler construct and a simpler security proof than the alternative choice, the security guarantee is stronger (due to the stricter security definition).

1.3 Organization

The rest of this paper is organized as follows: Section 2 provides the necessary background for the rest of the paper. In Section 3, we illustrate the four types of block operations in the ledger, and introduce the utility functions Ind, Prev, Next and Pivot used throughout the paper. In Section 4, we propose our model and security definition for blockchains supporting three types of redaction. Section 5 shows why previous constructs are insecure in this model. In Section 6 we incorporate an idea called *triple versioning* to build an improved construct, and show that it is secure under the proposed model. Section 7 suggests some possible enhancements, and concludes the paper.

2 Background

Notation. Deterministic assignments are denoted by $z := 9$, while probabilistic assignments are denoted by an arrow: $z \leftarrow B(n)$ for the output of a probabilistic algorithm B, or $z \leftarrow S$ for uniformly random selection from a finite set S. The symbol $x = y$ is used for checking/asserting equality.

Lists. Let $\mathcal{L} = [B_0, \dots, B_\ell]$ be a list. Note that we use 0 and ℓ as the indices of the first and the last element of the list, respectively. The list may grow arbitrarily, but ℓ is always updated accordingly to be the index of the last element. The elements of the list can be addressed by their index: $\mathcal{L}[i] = B_i$ for $0 \le i \le \ell$. For integers i, j with $0 \le i \le j \le \ell$, define $\mathcal{L}[i:j] \overset{\text{def}}{=} [B_i, \dots, B_j]$. If $j < i$, then $\mathcal{L}[i:j] = [\,]$ by definition. If $\mathcal{L}_1$ and $\mathcal{L}_2$ are two lists, their concatenation is denoted by $\mathcal{L}_1 + \mathcal{L}_2$.

Negligible function. A function $\mathsf{negl}\colon \mathbb{N} \to [0,1]$ is called negligible if it decays faster than the inverse of any positive polynomial. Formally, for any $c \in \mathbb{N}$, there exists $n_0 \in \mathbb{N}$, such that for all integers $n > n_0$ we have $\mathsf{negl}(n) < n^{-c}$.

Definition 1 (Strongly unforgeable signature scheme). *Consider a signature scheme* $(\mathsf{GenSig}, \mathsf{Sign}, \mathsf{VerifySig})$ *where* GenSig *generates a pair of public and secret keys,* Sign *generates a signature given the secret key and a message, and* $\mathsf{VerifySig}$ *verifies a signature on some message using the public key. The signature scheme is called* strongly *unforgeable under chosen-message attack (sUF-CMA), if for any efficient adversary taking part in the following game, there exists a negligible function* negl*, such that the probability of the adversary winning is at most* $\mathsf{negl}(\lambda)$*, where* λ *is the security parameter.*

- $\mathsf{GenSig}(1^\lambda)$ *generates a pair of public and secret keys* (pk, sk).
- *The adversary is given* pk*, as well as access to the signing oracle* $\mathsf{Sign}_{sk}(\cdot)$*. The signing oracle receives a message, and returns a valid signature on it. The adversary may interact with the oracle polynomially many times.*
- *The adversary outputs a pair* (m^*, σ^*)*. She is said to win the game if* $\mathsf{VerifySig}(pk, m^*, \sigma^*) = 1$*, and* σ^* *is never returned by the signing oracle on input* m^*.

Our construction assumes the existence of sUF-CMA signature schemes. The assumption is equivalent to the existence of ordinary (i.e., not necessarily strong) UF-CMA signature schemes [13], which can be constructed from one-way functions [19]. However, very efficient constructs of sUF-CMA signature schemes exists [3, p. 230].

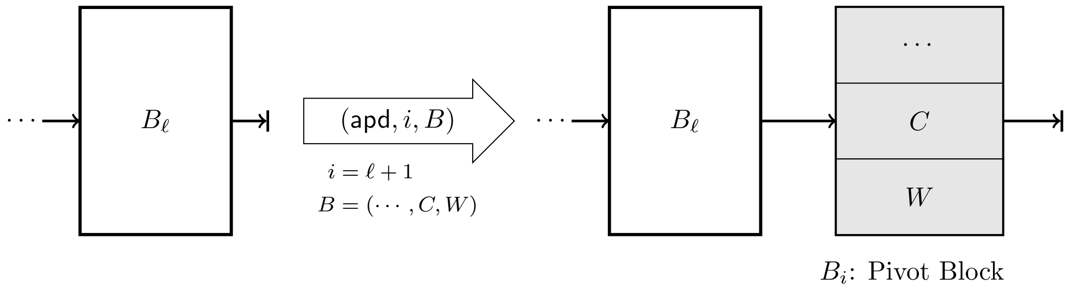
3 Blockchain operations

This section illustrates the ways a ledger can be modified using diagrams in Figure 1. To this end, let us first fix the block structure. The most important part of a block is the block content C. Blocks must also contain a witness W supporting the validity of redactions. Other information, such as block prefix, version number, randomness and so on are immaterial to the discussion of this section. We simply use an ellipsis $(\cdots)$ to show the omission of such extra information.

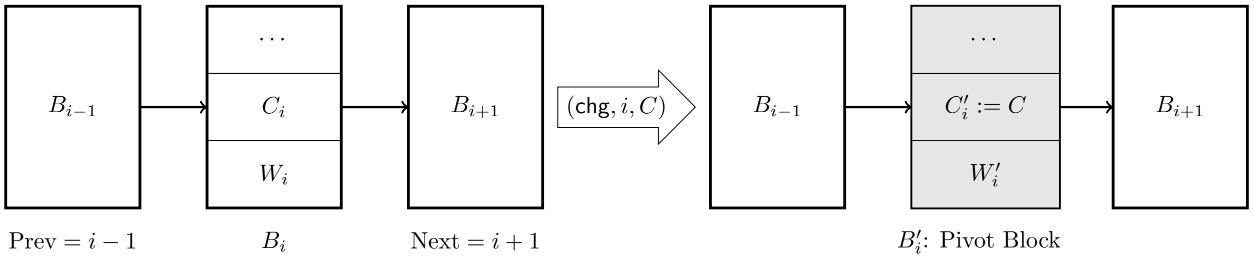
The first type of blockchain operation is *appending* a block. It does not require access to the secret key. Here, we define the *pivot block* as the block being appended. All other operations that can be performed on the ledger are redactions. They require access to the secret key. For each redaction operation, we define one block as the *pivot block*: It is the block for which a new witness is computed (using the secret key). The specifics for each operation is explained next:

Append operation $\texttt{apd}$ *(Figure 1a):* Consider the appending request $(\texttt{apd}, i, B)$, asking to append the block B at location i. When the last block of the ledger is at location ℓ, the operation is only successful for $i = \ell + 1$. Here, the *pivot block* is the block being appended (B).

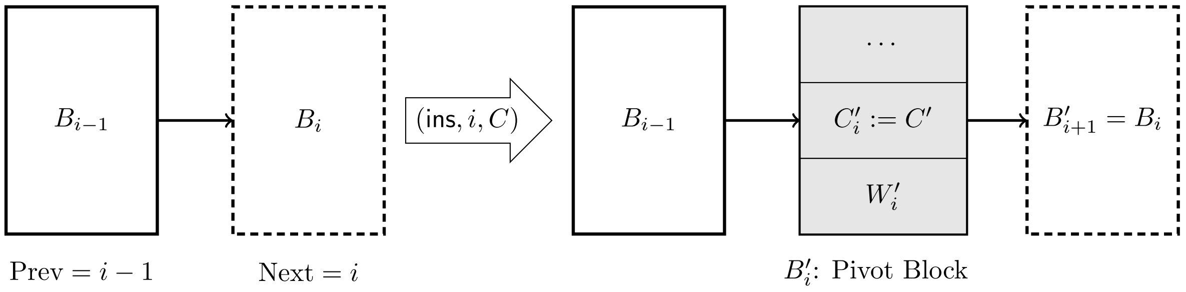
Change operation $\texttt{chg}$ *(Figure 1b):* Consider the redaction request $(\texttt{chg}, i, C')$, asking to change the content of the i^{th} block to C'. Here, the *pivot block* is the block being changed



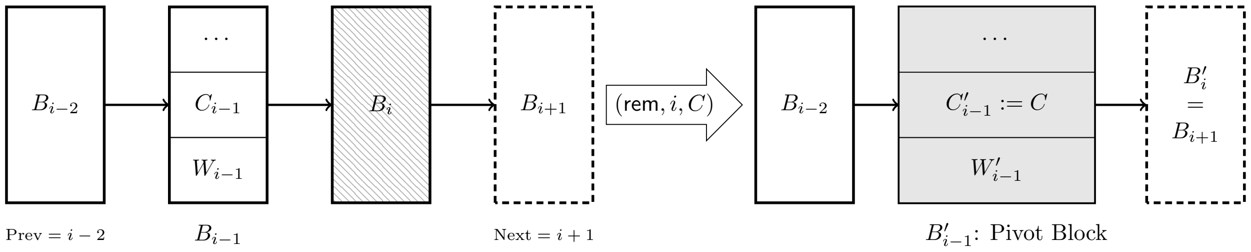
(a) Append operation $(\texttt{apd}, i, B)$. The pivot block is the one being appended.



(b) Redaction operation $(\texttt{chg}, i, C)$. The pivot block is the one being changed.



(c) Redaction operation $(\texttt{ins}, i, C)$. The pivot block is the one being inserted.



(d) Redaction operation $(\texttt{rem}, i)$. The pivot block is the one preceding the block being removed.

Fig. 1. Illustration of various block redaction operations. The grayed fields of the pivot block are affected by the corresponding operation.

Table 1. Various operations for a redactable blockchain. $\mathsf{Ind}_\ell(\mathsf{op})$ denotes the set of valid indices i for operation op on a ledger whose last index is ℓ. For each operation, previous and next indices are listed. Note that they refer to block indices in the ledger *prior* to performing the operation.

Operation	**Name**	**Redaction?**	$\mathsf{Ind}_\ell(\mathsf{op})$	$\mathsf{Prev}(\mathsf{op}, i)$	$\mathsf{Next}(\mathsf{op}, i)$	$\mathsf{Pivot}(\mathsf{op}, i)$
`apd`	Append	No	$\{\ell+1\}$	$i-1$	$i+1$	i
`chg`	Change	Yes	$\{1,\dots,\ell\}$	$i-1$	$i+1$	i
`ins`	Insert	Yes	$\{1,\dots,\ell\}$	$i-1$	i	i
`rem`	Remove	Yes	$\{1,\dots,\ell\}$	$i-2$	$i+1$	$i-1$

(the i^{th} block). The administrator uses the secret key sk to find a proper witness W'_i so that the redaction is valid.

Insertion operation `ins` *(Figure 1c):* Consider the redaction request $(\mathtt{ins}, i, C')$, asking to insert a block with content C' at index i. Here, the *pivot block* is the block being inserted (the i^{th} block). The administrator uses the secret key sk to find a proper witness W'_i.

Removal operation `rem` *(Figure 1d):* Consider the redaction request $(\mathtt{rem}, i)$, asking to remove the block at index i. Here, the *pivot block* is the block preceding the block being removed (the block with index $i-1$). The administrator uses the secret key sk to find a proper witness W'_{i-1}.

Table 1 summarizes the operations, as well as utility functions Ind, Prev, Next, and Pivot. These utility functions are used in later sections of the paper to simplify definitions. Ind gives the set of valid indices for an operation. Pivot returns the index of the pivot block for each operation, while Prev and Next return the index of the previous and next blocks. Notice that all indices refer to the locations in the ledger prior to performing the operation.

4 The tri-op model

Our model augments that of [8] with two types of new redactions: Block removal, and block insertion. The model is described as a game between a challenger and an adversary. The challenger generates a pair of keys, and initializes the ledger. The public key is handed over to the adversary. She also receives a read-only access to the ledger, as well as oracle access to the challenger. The adversary can make two types of queries to the challenger:

1. **Installation query:** The adversary asks the challenger to install a block at a specific location in the ledger, while performing one of the four types of operations (append, change, remove, or insert). The challenger verifies the request, and performs it in case it is valid.
2. **Redaction query:** The adversary asks the challenger to redact a block using one of three redaction operations (change, remove, or insert). The challenger verifies the request. If the operation is valid, the challenger uses the secret key to perform the operation, and returns a block that can later be installed by the adversary in the ledger using an installation query.

This model abstracts out the distributed nature of a blockchain. As a result, it does not capture all real-world attacks. For instance, the adversary does not get the option to change multiple blocks at once. Therefore, as shown in the next section, the constructs need not worry about chaining the blocks together. In some sense, it is an ideal-world model of redactions. When used with real-world constructs, it should somehow be composed with a blockchain model that captures attacks against ordinary blockchains. Our model is simple enough, and is suitable for proof-reading constructs against non-distributed attacks. In Section 7, we discuss possible extensions to to capture a distributed model.

We next provide a definition of tri-op redactable blockchains, which as the name suggests, support three types of redaction operations. The definition uses the following useful function:

$$\mathsf{Transform}(\mathcal{L},(\mathtt{op},i,B)) \overset{\text{def}}{=} \mathcal{L}\big[0:\mathsf{Prev}(\mathtt{op},i)\big]+[B]+\mathcal{L}\big[\,\mathsf{Next}(\mathtt{op},i):\ell\big] \tag{1}$$

It defines the transformation performed on ledger $\mathcal{L}$ when performing an installation of block B at location i using operation $\mathtt{op} \in \{\mathtt{apd},\mathtt{chg},\mathtt{rem},\mathtt{ins}\}$. It is assumed that $i \in \mathsf{Ind}_\ell(\mathtt{op})$, as otherwise it will be rejected.

Definition 2. *A quintuple of efficient algorithms* $3\mathcal{RBC} = (\mathsf{Gen},\mathsf{Create},\mathsf{Redact},\mathsf{Verify},\mathsf{Install})$ *is called a* tri-op redactable blockchain scheme *if:*

1. *Algorithm* Gen *takes the security parameter* 1^λ *in unary and generates a pair of public and private keys* (pk,sk)*, along with the initialized ledger* $\mathcal{L}$*.*
2. *Algorithm* Create *creates a new block* B*, when executed on input the public key* pk*, the ledger* $\mathcal{L}$ *and some block content* C*.*
3. *Algorithm* Redact *generates a redacted block using the secret key. More specifically,* $\mathsf{Redact}(sk,\mathcal{L},(\mathtt{op},i,C))$ *outputs a redacted block* B *for operation* $\mathtt{op} \in \{\mathtt{chg},\mathtt{ins},\mathtt{rem}\}$ *at index* i *of ledger* $\mathcal{L}$*, where* sk *is the secret key and* C *is some block content.*

4. *Algorithm* Verify *verifies the validity of an operation on the ledger. More specifically, let* $(\mathtt{op}, i, B)$ *denote putting block* B *at index* i *of the ledger by performing operation* $\mathtt{op} \in \{\mathtt{apd}, \mathtt{chg}, \mathtt{ins}, \mathtt{rem}\}$. *Then,* $\mathsf{Verify}(pk, \mathcal{L}, (\mathtt{op}, i, B))$ *returns* 1 *if and only if this operation is valid. We observed that splitting the verification algorithm into two separate checks makes it much more flexible, as one part can be independent of the history of ledger transforms: The first part (algorithm* Φ*) verifies whether the operation is valid on the* current ledger. *The second part (algorithm* Ψ*) verifies whether the (resulting) ledger is valid, regardless of the operation. For the first part, the blockchain designer should decide what part of each block is involved in the verification. We call that part the "version of the (pivot) block":*
$$V := \mathsf{Version}(\mathcal{L}[\mathsf{Pivot}(\mathsf{op}, i)]). \tag{2}$$
We also extract the version of all existing blocks in the ledger prior to the operation:
$$\vec{V} := [\mathsf{Version}(\mathcal{L}[0]), \ldots, \mathsf{Version}(\mathcal{L}[\ell])]. \tag{3}$$
Algorithm Φ *then compares the new version information against the existing ones, when* op *is the operation and* i *is the index:* $\Phi(\vec{V}, \mathsf{op}, i, V)$. *It returns* 1 *if and only if the operation is deemed valid.*
The second algorithm $\Psi(pk, \mathcal{L}^*)$ *checks the consistency of the whole ledger assuming the transformation is done:*
$$\mathcal{L}^* := \mathsf{Transform}(\mathcal{L}, (\mathsf{op}, i, B)). \tag{4}$$
In one sense, Ψ *performs a static check on the ledger, while* Φ *performs a dynamic check on the operation. The verification algorithm returns 1 if and only if both* $\Phi(\vec{V}, \mathsf{op}, i, V)$ *and* $\Psi(pk, \mathcal{L}^*)$ *return 1.*
5. *Algorithm* Install *modifies the ledger according to the requested operation. More specifically, when executed as* $\mathsf{Install}(pk, \mathcal{L}, (\mathsf{op}, i, B))$*, the algorithm first performs* $\mathsf{Verify}(pk, \mathcal{L}, i, B)$*, and returns 0 if the verification fails. Otherwise, it performs* $\mathcal{L} = \mathsf{Transform}(\mathcal{L}, (op, i, B))$*, and returns* 1.

Any tri-op redactable blockchain $3\mathcal{RBC}$ should be correct, meaning that when the algorithms Create and Redact are executed on valid inputs, their output must satisfy the verification algorithm:

Definition 3. *Let* λ *be any security parameter, and* $3\mathcal{RBC} = (\mathsf{Gen}, \mathsf{Create}, \mathsf{Redact}, \mathsf{Verify}, \mathsf{Install})$ *be as in Definition 2. Let* $(pk, sk, \mathcal{L}) \leftarrow \mathsf{Gen}(1^\lambda)$*, and assume that* $\mathcal{L}$ *is transformed any number of times by applying the* Install *algorithm. The following correctness conditions should hold:*

1. ***Algorithm* Create *is correct:*** *Let B be generated by* $\mathsf{Create}(pk,\mathcal{L},C)$. *Then*

$$\mathsf{Content}(B) = C \quad \wedge \quad \mathsf{Verify}(pk,\mathcal{L},(\mathsf{apd},\ell+1,B)) = 1\,. \tag{5}$$

2. ***Algorithm* Redact *is correct:*** *For any* $\mathtt{op} \in \{\mathtt{chg},\mathtt{ins},\mathtt{rem}\}$ *and* $i \in \mathsf{Ind}_\ell(\mathtt{op})$, *let B be generated by* $\mathsf{Redact}(sk,\mathcal{L},(\mathtt{op},i,C))$. *Then*

$$\mathsf{Content}(B) = C \quad \wedge \quad \mathsf{Verify}(pk,\mathcal{L},(\mathtt{op},i,B)) = 1\,. \tag{6}$$

We can now give the security definition for a tri-op redactable blockchain, which is based on Experiment 1. The experiment uses a set Hist to keep track of queries made to the redaction oracle: When a valid query (op,i,C) is made to this oracle and a block B is returned as response, the element (op,i,B) is added to Hist. The adversary wins if she performs a valid install query for a redacted block not in Hist.

It is noteworthy that once a valid installation query is processed, the experiment empties the set Hist. This provides the adversary with a great opportunity: She can win even if she can get two redacted blocks from the administrator, and install both of them. In other words, for a secure $3\mathcal{RBC}$, once a redacted block is installed, all previously obtained redacted blocks must be rendered invalid.

The above definition is very strict. Alternatively, we could have kept all previously obtained blocks in Hist to relax the definition. However, since the insertion and removal of blocks relocates all succeeding blocks, the index i in corresponding elements of Hist must be readjusted: A block insertion should add 1 to the index of triplets in Hist, whose index is after the installation location. Similarly, a block removal should subtract 1 from those indices. While the relaxation in security definition is perceivable, we did not incorporate it in Experiment 1, as otherwise it would unnecessarily complicate the experiment. Surprisingly, this stricter model simplified our construction and security proof (next section) compared to the alternative.

Definition 4. *A tri-op redactable blockchain scheme $3\mathcal{RBC}$ is* secure) *if for all efficient adversaries $\mathcal{A}$ who takes part in Experiment 1, there exists a negligible function* negl *such that* $\Pr\left[\,\mathsf{Redact}_{\mathcal{A},3\mathcal{RBC}}(\lambda) = 1\,\right] \le \mathsf{negl}(\lambda)$.

1. Let $(pk, sk, \mathcal{L}) \leftarrow \mathsf{Gen}(1^\lambda)$. The set of query-responses Hist is initialized to the empty set $\emptyset$.
2. Adversary $\mathcal{A}$ is given pk and a read-only access to the ledger $\mathcal{L}$. She can query two oracles $\mathtt{REDC}_{sk,\mathcal{L}}(\cdot,\cdot,\cdot)$ and $\mathtt{INST}_{pk,\mathcal{L}}(\cdot,\cdot,\cdot)$, which operate as explained below.
3. Upon receiving (op, i, C) where $\mathsf{op} \in \{\mathtt{chg}, \mathtt{ins}, \mathtt{rem}\}$, the redaction oracle REDC generates B by running $\mathsf{Redact}(sk, \mathcal{L}, (\mathsf{op}, i, C))$. If $B \neq \perp$, it adds (op, i, B) to Hist. Finally B is returned.
4. Upon receiving (op, i, B), the installation oracle INST executes $\mathsf{Install}(pk, \mathcal{L}, (\mathsf{op}, i, B))$. Let b be the output of the installation algorithm. If $b = 0$, the experiment ends by returning 0. Otherwise,
   - `If:` $\mathsf{op} \neq \mathsf{apd}$ and $(\mathsf{op}, i, B) \notin$ Hist: The adversary $\mathcal{A}$ succeeds.
     The experiment ends by returning 1.
   - `Else:` In case $(\mathsf{op}, i, B) \in$ Hist, the installation oracle sets Hist $:= \emptyset$.
     The adversary receives 1, and the experiment continues.

Experiment 1. The redaction experiment $\mathsf{Redact}_{\mathcal{A},3\mathcal{RBC}}(\lambda)$.

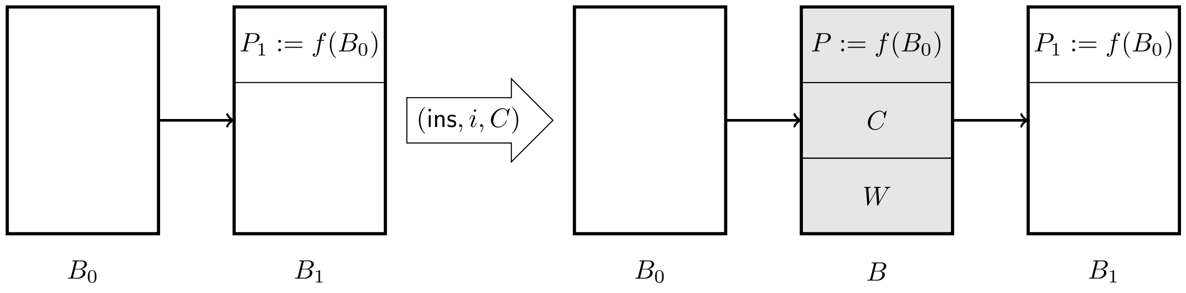


Fig. 2. Illustration of how twin blocks are created.

5 Attacks on previous work

In this section, we show how adding support for block insertion renders previous moderated blockchains insecure. Most previous constructs succumb to what we call a "twin-block attack" (Section 5.1), while one construct is susceptible to a "change-of-operation attack" (Section 5.2).

5.1 Twin-block attack

Most of the previous redactable blockchains in the moderated setting [2,6,9] follow the following structure: Blocks contain a prefix P, some content C, and a witness W. The witness is computed by the administrator when redacting a block. The prefix of a block is computed as a function of the previous block: $P_{i+1} := f(B_i)$, where f is some deterministic function.

Once insertion is recognized as an operation, the deterministic nature of f becomes an issue: When a block is inserted between two blocks, its prefix equals to that of the next block: see Figure 2, and read on for a formal proof. We refer to two consecutive blocks with the same prefixes as *twin blocks*. Next, it is shown how the adversary can create and exploit such blocks to make some arbitrary modifications to the ledger, without consulting the administrator first.

Consider a "toy" ledger $\mathcal{L} = [B_0, B_1]$. By definition, P_1 is a deterministic function of the previous block:

$$P_1 := f(B_0). \tag{7}$$

Two-step setup: The adversary $\mathcal{A}$ performs the following:

1. $\mathcal{A}$ queries Redact on $(\texttt{ins}, 1, C)$, and receives block $B = (P, C, W)$. In particular, W is set in such a way that the prefix of the next block B_1 remains valid: In the context of enhanced chameleon hash functions introduced in Section 1.1, the administrator uses its secret key to find W such that B and B_0 form a collision for f.
2. $\mathcal{A}$ queries Install on $(\texttt{ins}, 1, B)$. The resulting ledger is $\mathcal{L} = [B_0, B, B_1]$.

The following identities hold in the ledger:

$$P = f(B_0), \tag{8}$$
$$P_1 = f(B). \tag{9}$$

Equation (8) holds since B is a valid next block of B_0, and Equation (9) holds because B_1 is a valid next block of B. Together with Equation (7) we get:

$$P = P_1 = f(B_0) = f(B). \tag{10}$$

Attack 1: Without any other Redact queries, adversary can ask Install on $(\texttt{ins}, 1, B)$, resulting in the valid ledger $\mathcal{L} = [B_0, B, B, B_1]$. The ledger is valid because B is a valid next block of B_0 (as was the case before), and B is a valid next block of itself: $P = f(B)$ as per Equation (10). Therefore, the adversary can insert B in the ledger indefinitely by querying Install on $(\texttt{ins}, 1, B)$ an arbitrary number of times.

Attack 2: Starting with $\mathcal{L} = [B_0, B, B, B_1]$ obtained in the previous attack, $\mathcal{A}$ can ask the redaction oracle to change the block B at index 1, but apply it to its "twin block" B at index 2:

1. $\mathcal{A}$ queries Redact on $(\mathtt{chg}, 1, \widetilde{C})$, and receives block $\widetilde{B} = (\widetilde{P}, \widetilde{C}, \widetilde{W})$, satisfying the following identities:
$$\widetilde{P} = f(B_0), \qquad P = f(\widetilde{B}). \tag{11}$$
2. $\mathcal{A}$ queries $\mathsf{Install}$ on $(\mathtt{chg}, 2, \widetilde{B})$, resulting in the valid ledger $\mathcal{L} = [B_0, B, \widetilde{B}, B_1]$. This is because combining Equations (10) and (11), we get:
$$\widetilde{P} = f(B), \qquad P_1 = f(\widetilde{B}). \tag{12}$$
Therefore $\widetilde{B}$ is a valid next block of B, and B_1 is a valid next block of $\widetilde{B}$.

5.2 Change-of-operation attack

The construct of [8] has the block structure $B = (C, V, W)$, where C is the block content, V is the block version number, and W is a witness to be provided whenever a block is redacted. Notice that blocks do not have a prefix, as the model is abstract, and chaining blocks occurs at a lower level.

- The versioning scheme is global, meaning that each block will have a unique version number. In other words, with each installation (be it an append or a redaction), a new version number is introduced.
- The witness for a non-redacted block is the empty string ε. Once a block is redacted, the witness becomes a digital signature on a value specified below. The signing secret key is only known to the administrator.

Let $\mathcal{L} = [(C_1, 1, \varepsilon), (C_2, 2, \varepsilon), (C_3, 3, \varepsilon)]$ be the current ledger, and assume we want to append a block whose content is C_4. The version number of this block must be unique in the ledger, so we set it to 4. The witness is set to the empty string. Therefore, the forth block is $B_4 = (C_4, 4, \varepsilon)$, and the updated ledger becomes $\mathcal{L} = [(C_1, 1, \varepsilon), (C_2, 2, \varepsilon), (C_3, 3, \varepsilon), (C_4, 4, \varepsilon)]$.

Now assume the administrator wants to approve a change to the second block, so that its new content becomes C'_2. The new block must have a unique version in the ledger, so the version is set to 5. Finally, the witness of the redacted block must be set. The signature is computed on the concatenation of three items: (1) The content of the redacted block, (2) the version of the redacted block, and (3) the version of the next block. The first two components are there to ensure that the adversary does not change the block content or version after she received a signed redacted block from the administrator. The third component is there to prevent an adversary to relocate a signed redacted block (i.e., receive a redaction for location i, but later install it at location i'). In our example, the current version of the redacted block is 5, and the version of the next block in the ledger is 3. Therefore, W'_2 is the administrator's signature on the following string: $C'_2 \,||\, 5 \,||\, 3$, and the ledger after installation of this block becomes $\mathcal{L} = [(C_1, 1, \varepsilon), (C'_2, 5, W'_2), (C_3, 3, \varepsilon), (C_4, 4, \varepsilon)]$.

Let us exemplify one last modification to our example ledger. To change the content of the third block to C'_3, its version is set to 6, and its witness W'_3 is set to a signature on $C'_3 \,||\, 6 \,||\, 4$. The ledger is updated to $\mathcal{L} = [(C_1, 1, \varepsilon), (C'_2, 5, W'_2), (C'_3, 6, W'_3), (C_4, 4, \varepsilon)]$.

The issue here is that W'_2 is no longer a valid signature, since the version number on the next block is changed. The construct resolved this issue by defining a *conditional* signature checking algorithm: The signature is only checked if the version of the current block is less than that of the next block (i.e., the next block is not changed after the current block). The algorithm Ψ is defined as the logical AND: $\bigwedge_{i=1}^{\ell} \psi(pk, \mathcal{L}[i-1], \mathcal{L}[i])$, where ψ is defined as follows for any two blocks $B = (C, V, W)$ and $B' = (C', V', W')$:

$$\psi(pk, B, B') \overset{\text{def}}{=} \begin{cases} 1 & \text{if } V' > V, \\ \mathsf{VerifySig}(pk, C \,||\, V \,||\, V', W) & \text{if } V' < V. \end{cases} \tag{13}$$

While the above construct is provably secure in the single-operation model, it suffers from a simple *change-of-operation* attack. Consider a toy example ledger $\mathcal{L} = [(C_1, 1, \varepsilon), (C_2, 2, \varepsilon)]$. The adversary requests the redaction $(\texttt{chg}, 1, C'_1)$, and receives the block $B'_1 = (C'_1, 3, W'_1)$, where W'_1 is a signature on $C'_1||3||2$. However, instead of installing $(\texttt{chg}, 1, B'_1)$, the adversary requests to install $(\texttt{ins}, 2, B'_1)$. Notice that the requested operation is changed from chg to ins, and the location is changed from 1 to 2. If succeeds, the ledger will become $\mathcal{L} = [(C_1, 1, \varepsilon), (C'_1, 3, W'_1), (C_2, 2, \varepsilon)]$, while the administrator's approved redaction was $\mathcal{L} = [(C'_1, 3, W'_1), (C_2, 2, \varepsilon)]$.

The above attack succeeds because the construct checks two conditions: (1) The redacted block version is globally unique. In this case, the version is 3, and satisfies this condition. (2)

The condition in Equation (13), which also passes: W_1' is a valid signature on its own block content and version, as well as the version of the next block.

The reason why the above attack works is that the construct does not readily generalize to the tri-op settings: The adversary can always request a redaction using one operation, and install it at the proper location with another operation. As the above example shows, the result can be catastrophic: The adversary can make structural changes to the ledger that are not approved by the administrator. In the next section, we make some changes, and suggest a new construct that is provably secure under the tri-op model.

6 Construction

In this section, we construct a tri-op redactable blockchain (Section 6.1) based on the idea of triple versioning. We then prove its security (Section 6.2) according to our proposed model and security definition.

6.1 A secure construct based on triple versioning

For a secure tri-op redactable blockchain, we improve the construct described in Section 5.2. The idea is to change the block structure to carry extra information regarding the operation and location for which the block is generated. In Definition 2, the only part of the verification algorithm that has access to this information is $\Phi(\vec{V}, \mathsf{op}, i, V)$. Therefore, the appropriate place to add the operation and location of the block is its version element. Accordingly, the version of each block becomes a triplet $V = (V_{\mathsf{me}}, V_{\mathsf{op}}, V_{\mathtt{idx}})$, where:

- V_{me} is the actual version of the current block, and should be globally unique.
- $V_{\mathsf{op}} \in \{\mathtt{apd}, \mathtt{chg}, \mathtt{ins}, \mathtt{rem}\}$ is the operator for which the block is generated.
- $V_{\mathtt{idx}}$ is the location for which the block is generated.

We then define the following verification algorithms ($\vec{V}$ is as defined in Equation (3)):

$$\mathsf{MaxV}(\vec{V}) \overset{\text{def}}{=} 1 + \max_{0 \le i \le \ell} \vec{V}_{\mathsf{me}}[i], \tag{14}$$

$$\begin{aligned}\Phi(\vec{V}, \mathsf{op}, i, V) \overset{\text{def}}{=} &\left(\mathsf{op} \in \{\mathtt{apd}, \mathtt{chg}, \mathtt{ins}, \mathtt{rem}\}\right) \wedge \left(i \in \mathsf{Ind}_{|\vec{V}|}(\mathsf{op})\right) \\ &\left(V_{\mathsf{op}} = \mathsf{op}\right) \wedge \left(V_{\mathtt{idx}} = i\right) \wedge \left(V_{\mathsf{me}} = \mathsf{MaxV}(\vec{V})\right),\end{aligned} \tag{15}$$

$$\psi(pk,B) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } V_{\mathsf{op}} \neq \mathsf{apd}, \\ \mathsf{VerifySig}(pk, C \,||\, V, W) & \text{otherwise.} \end{cases}, \tag{16}$$

$$\Psi(pk,\mathcal{L}) \stackrel{\text{def}}{=} \bigwedge_{0 \le i \le \ell} \psi(pk, \mathcal{L}[i]). \tag{17}$$

Algorithm $\mathsf{MaxV}(\vec{V})$ finds the maximum V_{me} in the ledger, and adds 1 to it. Algorithm Φ ensures that the block being installed is valid: It has the proper operation, the location at which it is installed matches the operation, and its version triplet is correct. Algorithm ψ checks whether the block is installed using a redaction operation, and if so the witness W is a valid signature on $C \,||\, V$. Finally, Ψ applies ψ to all blocks in the ledger, and returns 1 if all of them return 1.

Construct 1 *Let $3\mathcal{RBC}_{\mathsf{tv}}$ be a tri-op redactable blockchain, defined as follows. Each block is structured as a triple $B = (C, V, W)$, where C is the block content, V is the block version and W is a redaction witness. The block version is a triplet $V = (V_{\mathsf{me}}, V_{\mathsf{op}}, V_{\mathtt{idx}})$.*

- $\mathsf{Gen}(1^\lambda)$ *calls the generator of some signature scheme to generate a key pair:* $(pk, sk) \leftarrow \mathsf{GenSig}(1^\lambda)$. *It then initializes the ledger as list containing a single element* $B_0 := \big(\varepsilon, (1, \mathsf{apd}, 0), \varepsilon\big)$.
- $\mathsf{Create}(pk, \mathcal{L}, C)$ *generates* $B := \big(C, (V_{\mathsf{me}}, \mathsf{apd}, \ell+1), \varepsilon\big)$, *where* $V_{\mathsf{me}} = \mathsf{MaxV}(\vec{V})$ *as per Equation (14).*
- $\mathsf{Redact}(sk, \mathcal{L}, (\mathsf{op}, i, C))$*: If* $\mathsf{op} \notin \{\mathtt{chg}, \mathtt{ins}, \mathtt{rem}\}$ *or* $i \notin \mathsf{Ind}_\ell(\mathsf{op})$*, it returns* $\perp$*. Otherwise, it creates a block* $B := (C, V, W)$ *using content* C*, where* $V = (V_{\mathsf{me}}, V_{\mathsf{op}}, V_{\mathtt{idx}}) = (\mathsf{MaxV}(\vec{V}), \mathsf{op}, i)$ *and:*

$$W \leftarrow \mathsf{Sign}_{sk}\big(C \,||\, V\big). \tag{18}$$

- $\mathsf{Verify}(pk, \mathcal{L}, (\mathsf{op}, i, B))$*: Let algorithms* Φ *and* Ψ *be as in Equation (15) and Equation (17), respectively. The verification algorithm sets* $\mathcal{L}^* := \mathsf{Transform}(\mathcal{L}, (\mathsf{op}, i, B))$*, executes* $\Phi(\vec{V}, \mathsf{op}, i, V)$ *and* $\Psi(pk, \mathcal{L}^*)$*, and returns 1 if and only if both return 1.*
- $\mathsf{Install}(pk, \mathcal{L}, (\mathsf{op}, i, B))$*: Operates in the same manner defined in Definition 2.*

6.2 Security proof

We now prove that Construct 1 satisfies the correctness and security requirements of Section 4.

Theorem 1 (Correctness). *The tri-op redactable blockchain $3\mathcal{RBC}_{\mathsf{tv}}$ is* ***correct*** *per Definition 3.*

Proof. We prove the both correctness conditions hold.

Condition (1): Algorithm $\mathsf{Create}(pk,\mathcal{L},C)$ generates a block $B := \big(C,(\mathsf{MaxV}(\vec{V}),\mathsf{apd},\ell+1),\varepsilon\big)$. The block content of B is C. Furthermore, the version of B is correctly computed as required by Φ: The version triplet includes a globally unique version number, the operation is valid, and index is in $\mathsf{Ind}_\ell(\mathsf{apd})$. Finally, if $\mathcal{L}$ is already a valid chain, so is $\mathcal{L}^* := \mathcal{L}+[B]$. This is because ψ returns 1 on all blocks in $\mathcal{L}^*$ prior to the last block (due to the validity of $\mathcal{L}$). For the last block B, since $\mathsf{Version}_{\mathsf{op}}(B) = \mathsf{apd}$, the return value of $\psi(pk,B)$ is trivially 1. As a result, all blocks verify, and Ψ returns 1 as well.

Condition (2): Algorithm $\mathsf{Redact}(sk,\mathcal{L},(\mathsf{op},i,C))$ generates $\perp$ if $\mathsf{op} \notin \{\mathtt{chg},\mathtt{rem},\mathtt{ins}\}$ or $\mathsf{op} \notin \mathsf{Ind}_\ell(\mathsf{op})$. Otherwise, it returns $B := \big(C,(\mathsf{MaxV}(\vec{V}),\mathsf{op},i),W\big)$. The block content of B is C. Furthermore, the version of B is correctly computed as required by Φ: The version triplet includes a globally unique version number, the operation is a valid redaction, and index is in $\mathsf{Ind}_\ell(\mathsf{apd})$. Finally, if $\mathcal{L}$ is already a valid chain, so is $\mathcal{L}^* := \mathsf{Transform}(\mathcal{L},(\mathsf{op},i,B))$. This is because ψ returns 1 on all blocks in $\mathcal{L}^*$ prior to the last block (due to the validity of $\mathcal{L}$). For the last block B, due to Equation (18), W is a valid signature on $C\,||\,V$. Hence, the return value of $\psi(pk,B)$ is 1, as per Equation (17).

We conclude that $3\mathcal{RBC}_{\mathsf{tv}}$ is correct.

Theorem 2 (Security). *$3\mathcal{RBC}_{\mathsf{tv}}$ is* ***secure*** *per Definition 4, assuming* $(\mathsf{GenSig},\mathsf{Sign},\mathsf{VerifySig})$ *is an sUF-CMA signature scheme as defined in Definition 1.*

Proof. For any (efficient) adversary $\mathcal{A}$, who wins in Experiment 1 with probability ϵ, we construct an efficient forger $\mathcal{F}$, who forges a signature with the same probability.

The forger $\mathcal{F}$ is given as input the public key pk of the signature scheme, and can query the signing oracle $\mathsf{Sign}_{sk}(\cdot)$ polynomially many times. It initializes the set Hist to empty, and the ledger $\mathcal{L}$ as specified by Construct 1. It then executes the adversary $\mathcal{A}(pk,\mathcal{L})$ as a subroutine. Whenever the adversary makes an oracle request, the forger responds as specified next:

- **Queries to the redaction oracle $\mathtt{REDC}(\mathtt{op},i,C)$:** If either $\mathtt{op} \notin \{\mathtt{chg},\mathtt{ins},\mathtt{rem}\}$ or $i \notin \mathsf{Ind}_\ell(\mathsf{op})$, the forger $\mathcal{F}$ returns 0 and halts. Otherwise, $\mathcal{F}$ creates a new block $B := \big(C,V,W\big)$ with content C, version triplet $V := (\mathsf{MaxV}(\vec{V}),\mathsf{op},i)$ and witness W. To compute the witness, $\mathcal{F}$ sends the query $(C \,||\, V)$ to the signing oracle. It then sets $\mathsf{Hist} := \mathsf{Hist} \cup \{(\mathsf{op}, i, B)\}$, and finally returns B to the adversary $\mathcal{A}$ for further processing.

- **Queries to the installation oracle $\texttt{INST}(\mathsf{op}, \boldsymbol{i}, \boldsymbol{B})$:** The forger $\mathcal{F}$ acts exactly as in Step 4 of Experiment 1. That is, it calls the underlying $\mathsf{Install}$ operation, and returns 0 and halts if installation fails. If installation succeeds on an append or a non-fresh redaction, Hist is cleared and the game continues. Otherwise, the adversary is successful in the game. In this case, the forger $\mathcal{F}$ parses B as the triple (C, V, W). Since the installation algorithm returns 1, we know that B is a valid block. In particular, W is a valid signature on the message $m \overset{\text{def}}{=} (C \,||\, V)$. In this case, $\mathcal{F}$ returns (m, W) as a valid forgery.

It remains to show that the returned pair is fresh, meaning that the signing oracle has never returned the signature W when queried on input m. The forger has already checked that (op, i, B) is not in the set Hist (as in Step 4 of Experiment 1). Therefore, we consider all the remaining cases:

1. ***B* was returned at some point by the $\texttt{REDC}$ oracle, but was removed from Hist by a subsequent call to $\texttt{INST}$:** This is impossible because once $\texttt{INST}$ is called, the maximum version in the ledger is increased. Therefore, no previously obtained B remains valid.
2. **$(\mathsf{op}', \boldsymbol{i}', \boldsymbol{B}) \in \mathsf{Hist}$, where $(\mathsf{op}', \boldsymbol{i}') \neq (\mathsf{op}, \boldsymbol{i})$:** This is impossible, as B includes as part of its version both the operation and the location. Changing any of causes Φ to return 0.
3. **$(\mathsf{op}, \boldsymbol{i}, \boldsymbol{B}') \in \mathsf{Hist}$ for some block $\boldsymbol{B}' = (\boldsymbol{C}, \boldsymbol{V}, \boldsymbol{W}')$:** In this case, both blocks B and B' have the same content and version, but differ in their witnesses. As a result, both W and W' verify as signatures on the message $m \overset{\text{def}}{=} (C \,||\, V)$. This constitutes a *strong signature forgery*, and therefore (m, W) is a valid strong forgery on the underlying signature scheme.
4. **None of the above:** In this case, $m \overset{\text{def}}{=} (C \,||\, V)$ is new, and W is a valid signature on it. This constitutes a *normal signature forgery*, and $\mathcal{F}$ can output (m, W) as a valid forgery.

As explained above, cases 1 and 2 are impossible: the adversary never wins the game in these cases, because her output is simply invalid. In the plausible cases 3 and 4, the forger generates a valid signature whenever the adversary succeeds. Subsequently, $\mathcal{F}$ generates a valid signature forgery with the same probability that $\mathcal{A}$ succeeds in Experiment 1, and the theorem follows.

7 Summary and future work

In this paper, we defined a model for redactable blockchains that, for the first time, incorporated all three types of redactions: Block change, block removal, and block insertion. We showed that enabling block insertions renders previous constructs susceptible to either "twin-block attack" or "change-of-operation attack". We then proposed a new construct that is proved secure in this tri-op model.

While our model is simple enough to capture many types of attacks, it is still an abstraction of the real-world. In particular, it does not capture the distributed nature of blockchains, since the model is designed as a game between an adversary and a challenger. It is a good first step toward understanding the diversity of attacks and proof-reading existing constructs. However, a distributed model would better capture all types of real-world attacks. Therefore, the next logical step is to cast our model in a multiparty setting similar to that of [16].

References

1. Akkoyunlu, E.A., Ekanadham, K., Huber, R.V.: Some Constraints and Tradeoffs in the Design of Network Communications. In: Proceedings of the 5th ACM Symposium on Operating Systems Principles. pp. 67–74 (1975)
2. Ateniese, G., Magri, B., Venturi, D., Andrade, E.: Redactable Blockchain–or–Rewriting History in Bitcoin and Friends. In: EuroS&P. pp. 111–126. IEEE (2017)
3. Boneh, D., Shen, E., Waters, B.: Strongly Unforgeable Signatures Based on Computational Diffie-Hellman. In: PKC. pp. 229–240. Springer (2006)
4. Cachin, C.: Architecture of the Hyperledger Blockchain Fabric. In: Workshop on Distributed Cryptocurrencies and Consensus Ledgers (2016)
5. CoinDesk: Understanding The DAO Attack (2016), https://tinyurl.com/dao-attack
6. Derler, D., Samelin, K., Slamanig, D., Striecks, C.: Fine-Grained and Controlled Rewriting in Blockchains: Chameleon-Hashing Gone Attribute-Based. In: NDSS. pp. 1–15. The Internet Society (2019)
7. Deuber, D., Magri, B., Thyagarajan, S.A.K.: Redactable Blockchain in the Permissionless Setting. In: Symposium on Security and Privacy. pp. 124–138. IEEE (2019)
8. Dousti, M.S., Küpçü, A.: Moderated Redactable Blockchains: A Definitional Framework with an Efficient Construct. In: Data Privacy Management, Cryptocurrencies and Blockchain Technology, pp. 355–373. Springer (2020)
9. Grigoriev, D., Shpilrain, V.: RSA and Redactable Blockchains (2020), arXiv report 2001.10783
10. Hargreaves, S., Cowley, S.: How Porn Links and Ben Bernanke Snuck Into Bitcoin's Code (2013), https://tinyurl.com/bitcoin-snuck
11. Hopkins, C.: If You Own Bitcoin, You Also Own Links to Child Porn (2020), https://tinyurl.com/bitcoin-child
12. Kolb, J., AbdelBaky, M., Katz, R.H., Culler, D.E.: Core Concepts, Challenges, and Future Directions in Blockchain: A Centralized Tutorial. ACM Computing Surveys **53**(1), 1–39 (2020)
13. Liu, J.K., Au, M.H., Susilo, W., Zhou, J.: Short Generic Transformation to Strongly Unforgeable Signature in the Standard Model. In: ESORICS. pp. 168–181. Springer (2010)
14. Matzutt, R., Hiller, J., Henze, M., Ziegeldorf, J.H., Müllmann, D., Hohlfeld, O., Wehrle, K.: A Quantitative Analysis of the Impact of Arbitrary Blockchain Content on Bitcoin. In: International Conference on Financial Cryptography and Data Security. pp. 420–438. Springer (2018)

15. Nakamoto, S.: Bitcoin: A Peer-to-Peer Electronic Cash System (2009), available from http://www.bitcoin.org/bitcoin.pdf
16. Pass, R., Shi, E.: FruitChains: A Fair Blockchain. In: Proceedings of the ACM Symposium on Principles of Distributed Computing. pp. 315–324 (2017)
17. Pearson, J.: The Bitcoin Blockchain Could Be Used to Spread Malware, INTERPOL Says (2015), https://tinyurl.com/bitcoin-malware
18. Puddu, I., Dmitrienko, A., Capkun, S.: μchain: How to Forget Without Hard Forks. Cryptology ePrint Archive, Report 2017/106 (2017)
19. Rompel, J.: One-Way Functions are Necessary and Sufficient for Secure Signatures. In: Proceedings of the 22nd annual ACM Symposium on Theory of Computing. pp. 387–394 (1990)