Moderated Redactable Blockchains: A Definitional Framework with an Efficient Construct

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

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

Abstract. Blockchain is a multiparty protocol to reach agreement on the order of events, and to record them consistently and immutably without centralized trust. In some cases, however, the blockchain can benefit from some *controlled* mutability. Examples include removing private information or unlawful content, and correcting protocol vulnerabilities which would otherwise require a *hard fork*. Two approaches to control the mutability are: *moderation*, where one or more designated administrators can use their private keys to approve a redaction, and *voting*, where miners can vote to endorse a suggested redaction. In this paper, we first present several attacks against existing redactable blockchain solutions. Next, we provide a definitional framework for moderated redactable blockchains. Finally, we propose a provable and efficient construct, which applies a single digital signature per redaction, achieving a much simpler and secure result compared to the prior art in the moderated setting.

Keywords: Blockchain · Bitcoin · Moderated Redactable Blockchain · Formal Threat Model · Signature Scheme

1 Introduction

The concept of blockchain was pioneered by Bitcoin [16]. It is a distributed protocol that allows all honest parties to keep a ledger of event logs in a consistent manner and without any trust assumption. There are various incarnations of blockchains, which may relax or strengthen some of the conditions. The original blockchain is *permissionless*, meaning any party can participate in the protocol. *Permissioned* blockchains operate in an authenticated environment, where joining the network is subject to an administrative decision. A *private* blockchain is a specific type of permissioned blockchain, where every participant can view the ledger, but only an authorized set of entities can append. For further discussion, see [13].

One of the most important properties of blockchain is the *immutability* of the ledger. After all, cryptocurrencies require that once a transaction is recorded, it cannot be undone. However, this desirable property has its downsides. Criminals have occasionally appended arbitrary contents to the ledger that is forbidden by national or international laws—such as child abuse [11,12] and malware [18]. Another use case is where some information about a user is stored in the ledger, and later the user requests them to be removed [15], exercising the "right to be forgotten" under privacy laws such as the General Data Protection Regulation (GDPR) [5]. A third case is when a massive fraud has been made possible due to a flaw in the blockchain protocol. In immutable blockchains, the only way to invalidate such fraudulent transactions is by updating the protocol and the software—a process known as a *hard fork*. The DAO Attack [4] is an example, which resulted in a hard fork in Ethereum [20] back in 2016. For further discussion, see [1].

To overcome the limitations associated with immutability, several researchers proposed solutions for *controlled* mutability. The literature has two approaches for controlling the mutability: *Moderated* [1,6,10], where redactions can only be applied by a designated set of users (known as the administrators), and *unmoderated* (or voting-based) [19,8] where suggested redactions are voted on, and applied only if they receive a quorum of votes within a specific period. Notice that the terms permissioned and moderated are orthogonal: In permissioned blockchains, users need administrative permission to join the network. In moderated blockchains, administrators must approve redactions (changes to the blocks in the ledger). Even in a blockchain that is both moderated and permissioned, the administrators in charge of admitting users can be different from the administrators in charge of approving redactions.

In this paper, four novel attacks are presented against existing redactable blockchains: Two attacks against moderated constructs, and two against the unmoderated ones. Learning from the attacks, we suggest the goals for a definitional framework for redactable blockchains, and put forward an adversarial model and a security definition satisfying those goals. Finally, two constructs of redactable blockchains are presented: The former serves as an instrumental example, and is proven incorrect and insecure. The latter resolves the issues, and we prove it both correct and secure in our definitional framework.

2 Previous Work

Moderated Redactable Blockchains. In their seminal work, Ateniese et al. [1] constructed the first redactable blockchain. They proposed a special primitive called an *enhanced chameleon hash function.* A chameleon hash function is a collision-resistant hash function, such that finding collisions is easy given a private (trapdoor) key. The *enhanced* version satisfies the additional property that finding collisions (without the private key) is hard, even if the adversary can get collisions for inputs of her choice from an oracle. The primitive is rather complex and involved: In the standard model, it requires a witness whose size is 18 group elements under the SXDH assumption, or 39 group elements under the DLIN assumption [1]. Derler et al. [7] extended the above idea above to attribute-based chameleon hashes. Instead of applying redactions freely at the block level, the administrators are bound by a fine-grained policy on what attributes they can change. They employ *ciphertext-policy attribute-based encryptions* and *chameleon hashes with ephemeral trapdoors*. Recently, Grigoriev and Shpilrain [10] proposed a simple construct based on textbook RSA. However, Section 4 shows that it is insecure.

Interestingly, none of the work listed above provides a security model/definition tailored specifically for redactable blockchains, and therefore their constructs have no security proofs: While [1,7] focus on proving the security of the underlying cryptographic primitives (e.g., the enhanced chameleon hash function), [10] has no rigorous proof of security. We also show that all constructs succumb to reversion attacks.

Unmoderated (Voting-based) Redactable Blockchains. Puddu et al. [19] defined an idea called μchain for enabling mutability for proof-of-work blockchains. The mutability is controlled by fiat, imposed by consensus, and is publicly verifiable. It can be used in both moderated and unmoderated settings: In the moderated setting, the sender can create multiple mutations of a transaction, and encrypt all but one (the active transaction). The decryption key is distributed between miners using a secret-sharing scheme. The sender also proposes a policy as to how other mutations can be activated, and by whom. If a mutation request is approved by this policy, miners decrypt the intended mutation by a multi-party decryption protocol. In the unmoderated setting, the mutation to be activated is voted on. Deuber et al. [8] discuss various issues with μchain. They also propose a distributed consensus protocol for redaction. Their protocol does not require heavy cryptographic operations or trusting a set of administrators. It starts when a participant proposes a redaction. If the proposed block satisfies the verification algorithm, it enters a voting phase. If enough miners vote for it within a certain period of time, the change is applied to the ledger.

In Section 4, we show that care must be taken when dealing with votes. In particular, if not properly designed and implemented, it is possible to redact a block containing a vote for some previous block, which may render the corresponding redactions invalid. Furthermore, we explore possible ways where a minority group can prevent a policy to be applied, or even go against the policy.

3 Preliminaries

Assignment Notation. Assignments are denoted as $x \leftarrow 2$. To say something holds by definition, we use $x \overset{\text{def}}{=} y$. The symbol $x = y$ is used for checking or asserting equality.

List Manipulation. Let $\mathcal{L} \overset{\text{def}}{=} [B_0, \dots, B_\ell]$ be a list. The elements of the list can be addressed by their index: $B_i \overset{\text{def}}{=} \mathcal{L}[i]$ for $0 \le i \le \ell$. We use the following notation to address sublists: For integers i, j with $0 \leq i \leq j \leq \mathsf{len}(\mathcal{L})$, define $\mathcal{L}[i:j] \overset{\text{def}}{=} [B_i, \ldots, B_j]$. If $j < i$, the sublist is empty. If $\mathcal{L}_1$ and $\mathcal{L}_2$ are two list, their concatenation is denoted by $\mathcal{L}_1 + \mathcal{L}_2$.

Blocks. A block B is denoted by a tuple, such as (P, C, V, W), containing various components. Each component can be set to a default value, such as the empty string ε. Blockchains may add other or remove components of their choice to the block structure. Here is the description of the most common components: P is the prefix of the block. It is often a function of previous blocks in the ledger. C is the content of the block (in cryptocurrency nomenclature, it is the set of transactions). V is the version of the block. W is the witness of the block. It is used in redactions. We assume the existence of efficient algorithms $\mathsf{Prefix}(B)$, $\mathsf{Content}(B)$, $\mathsf{Version}(B)$, and $\mathsf{Witness}(W)$, which efficiently extract the relevant component from block B. If we are interested in a block except one of its components, we denote it by striking through that component: $B^{\cancel{W}}$ is block B except its W component.

4 Novel Attacks on Previous Constructs

In this section, we explain several attacks against certain previous constructs, which carry over their desired security properties from immutable blockchain models [9,17], to the redactable setting. We stress that most attacks can be easily prevented by small modifications in the corresponding construct. However, the mere existence of the attacks in the face of security proofs shows that one should consider an adversarial model tailored for the redactable blockchains. Due to a lack of space, we only provide an overview of the attacks.

Moderator Circumvention Attack: The attack is specific to the GS Construct [10], whose block relationship is depicted in Fig. 1. The attacker can craft two blocks B and B', append B to the ledger, and at any point in time replace it with B'. It works without administrator involvement, since the witness verification simply holds for both blocks. It works as follows:

1. Pick Z from $\mathbb{Z}_n$ uniformly at random. Retry this step if Z has order 2.
2. Let $e \leftarrow f(P, C)$ and $e' \leftarrow f(P, C')$.
3. Let $W \leftarrow Z^{e'} \pmod{n}$ and $W' \leftarrow Z^{e} \pmod{n}$.
4. Output $B \leftarrow (P, C, W)$ and $B' \leftarrow (P, C', W')$.

It can be verified that $P_{\text{next}} = W^e = W'^{e'} = Z^{e \cdot e'} \pmod{n}$. Thus, replacing B with B' does not affect the prefix of the next block.

Reversion Attack: The attack can be applied to both the GS [10] and the AMVA [1] constructs, both of which are in the moderated settings. Consider a block B, which was later redacted to B' with the help of the administrators. An adversary can simply revert a redacted block B' to its previous state B: Since no versioning scheme is in place, all versions of a block are valid.

Vote Erasure Attack: The attack is applicable to the DMTS Construct [8], which is in the voting setting. Here, a redaction is approved if a quorum of

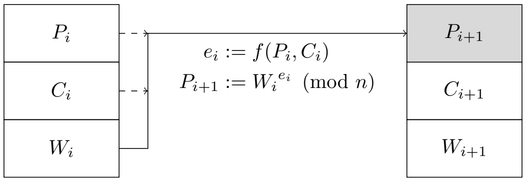

Fig. 1. The relationship between two consecutive blocks in the GS Construct. C_i is the content. W_i is the witness, which is picked uniformly from $\mathbb{Z}_n$ such that it does not to have order 2. The prefix P_{i+1} depends on all parts of block B_i via the relation $P_{i+1} \leftarrow W_i^{f(P_i,C_i)} \pmod{n}$, where n is an RSA modulus and f is an efficient integer-valued function.

miners endorse it by including their approval in the blocks they mine. Votes are recorded as ordinary transactions in the blocks. An attacker can redact blocks containing votes, essentially reducing the total number of votes for a particular redaction, putting the ledger in an inconsistent state: An already approved block is no longer verified.

Miner Corruption Attack: The attack is applicable to the DMTS Construct [8]. Let the approval quorum be $\rho \overset{\text{def}}{=} \frac{3}{4}$, as suggested by the paper: When a redaction is proposed, at least three out of the next four mined blocks should carry a vote approving the redaction. Consider an adversary who controls 49% of the miners, all of whom *abstain* from endorsing any redactions. A simple combinatorial analysis shows that even if all honest miners vote in favor of all redactions, only $\binom{4}{3}(0.51)^3(0.49) + (0.51)^4 \approx 33\%$ of them are approved. Furthermore, for an adversarially suggested redaction, even if all honest miners refrain from voting, there is a $\binom{4}{3}(0.49)^3(0.51) + (0.49)^4 \approx 30\%$ chance of approval. Increasing ρ decreases the chance of honest redactions, while decreasing it increases the chance of adversarial redactions.

5 Defining Moderated Redactable Blockchain

5.1 Design Goals

Section 4 demonstrates that adapting existing models and definitions of immutable blockchains to the redactable setting is challenging, as mutability opens a variety of ways for an adversary to attack the blockchain. We propose decoupling the two notions: A challenger is introduced, who enforces most of the restrictions imposed by an immutable blockchain. On the other hand, we allow the adversary to control the participants in the network, receive an arbitrary number of redactions, and install an arbitrary number of blocks in the ledger. In designing our definitional framework, we pursued the following goals:

- **Bitcoin independence:** The framework should *not* impose Bitcoin protocol or data structures. For instance, the blockchain designer might opt not to include the hash of the previous block in the current block.

- **Consensus independence:** The framework should *not* impose a specific consensus mechanism, such as the proof of work (PoW) or the proof of stake (PoS). Rather, it should depend on an abstraction that provides consensus.
- **General content:** The framework should *not* assume that the content of each block includes a set of transactions. Rather, the content must be treated as an arbitrary bit string.
- **Simplicity:** The framework should be as simple as possible. With this aim, we abstract out the distributed nature of the network by a centralizing challenger.
- **Moderation:** The framework should support the moderated setting. This is by choice rather than merit, meaning a framework for the unmoderated setting is equally important, but is left as future work.
- **Operation segregation:** The framework should *not* combine operations which are semantically different. For instance, consider redaction and installation: When an administrator is asked for a redaction, he should merely return a redacted block, rather than installing the block in the ledger. The installation must be performed separately.
- **Allowing adversarial transformation:** The framework should allow the adversary to append any valid block at the end of the ledger. Also, she must be able to receive the redaction of as many blocks as she wants. Finally, she must be able to install any valid redaction.
- **Ledger consistency:** The ledger must remain consistent at all times. That is, there should not be a valid transformation that invalidates one or more blocks already installed in the ledger (cf. Section 4).

5.2 Informal Model

Fig. 2 illustrates our definitional framework informally. Notice that it resembles a game between a challenger and a single adversary. It is as if she has total control over the participants in the blockchain: As long as she plays by the rules, she can append any valid block to the ledger, request any block content to be redacted to an arbitrary yet valid value, and install any valid redacted block. Furthermore, no modification is made to the chain without the adversary saying so. In fact, the challenger is an abstraction of an ideal consensus protocol. The goal of the adversary is to create a redacted block which is not provided by the administrators controlled by the challenger, and install it in the ledger.

Observe the similarity with the way signature schemes are modeled: Obtaining redactions for arbitrary content are akin to acquiring a signature on arbitrary messages (the adaptive chosen message attack). Furthermore, the security definition is similar: Any new redaction constitutes an attack, which is akin to existential forgery in signature schemes. In fact, as shown in Section 6, a strongly unforgeable signature scheme can be used to construct a secure redactable blockchain in our model.

In what follows, we abstract out a redactable blockchain as a tuple of efficient algorithms. The abstraction pertains to a centralized setting, where there is a challenger with a private key, playing against an adversary with the public key

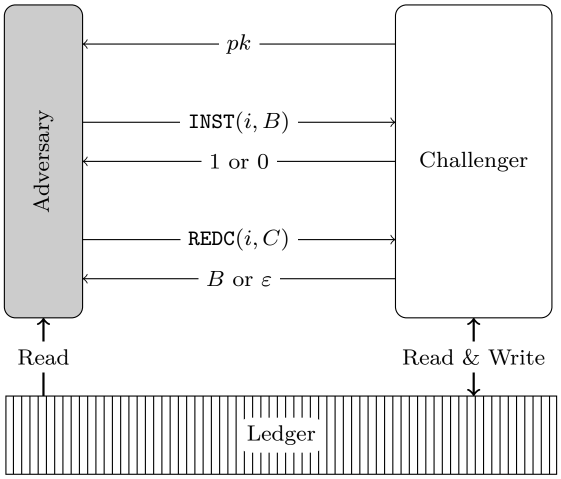

Fig. 2. The proposed adversarial model. The challenger creates a key pair and the ledger. It gives the public key *pk* to the adversary, and provides her with read-only access to the ledger. All write operations (installations) should go through the challenger's `INST` interface, by specifying the location i pointing to a valid block index in the ledger, and the block B to be installed. The challenger returns 1 if the installation is successful, and 0 otherwise. The adversary can also request redactions via the challenger's `REDC` interface. She provides the redaction location i, as well as the new block's content C. If the operation is successful, the challenger returns a redacted block B, which can then be installed using its `INST` interface. Otherwise, the challenger returns an empty block ε. The adversary is deemed successful if she installs a redacted block which is not obtained via the `REDC` interface.

and read-only access to the ledger. The adversary can install blocks by asking the challenger, who accepts the request as long as the adversary abides by the rules. The verification algorithm distinguishes valid blocks from invalid ones. Contrary to previous work such as [1,8], which explicitly use the proof-of-work verification in their model, we let each construct decide on its own verification algorithm. For instance, a construct may use separate verification algorithms for normal and redacted blocks. This simplifies and generalizes the scheme. The adversary can also ask the challenger to redact block contents, in hope that she learns how to redact a block without the challenger's help. The adversary is deemed successful if she can generate a new redaction.

We realize that block versioning is useful, and therefore incorporate it into our formalization below. If a solution does not employ versioning, those parts in the definition may be ignored.

5.3 Definition

The blockchain storage (the ledger) is modeled as a list of blocks $\mathcal{L} \overset{\text{def}}{=} [B_0, B_1, \ldots, B_\ell]$. The list starts at index 0, and the block at $\mathcal{L}[0]$ is called the *genesis* block. This block is generated initially, and it helps in simplifying the presentation. We assume that the variable ℓ always keeps the number of real (non-genesis) blocks: $\ell \overset{\text{def}}{=} \mathsf{len}(\mathcal{L}) - 1$. Initially, $\ell \leftarrow 0$, as there is only one block in the ledger (the genesis block) Upon appending each new block, ℓ is incremented. The value ℓ is *not* an upper bound: $\mathcal{L}$ can grow to include any polynomial number of blocks. The ledger is published as a *read-only* list. The only way an adversary can modify $\mathcal{L}$ is via a call to the challenger's `INST` interface, as depicted by Fig. 2.

Definition 1 defines five efficient algorithms that constitute a moderated redactable blockchain scheme. We then express two syntactical requirements: Every block created correctly must be verifiable, and so is every block redacted correctly. Throughout, the following *transformation* is used: It expresses the effect of installing a block B at position i of ledger $\mathcal{L}$, where $1 \leq i \leq \ell + 1$:

$$\mathsf{Transform}(\mathcal{L}, i, B) \overset{\text{def}}{=} \mathcal{L}[0 : i-1] + [B] + \mathcal{L}[i+1 : \ell]\,. \tag{1}$$

Notice that $\mathsf{Transform}$ returns a new ledger, rather than changing $\mathcal{L}$. By list manipulation rules defined in Section 3, if $i+1 > \ell$, the rightmost sublist $\mathcal{L}[i+1 : \ell]$ is empty. The resulting ledger has the same length as $\mathcal{L}$ if $1 \leq i \leq \ell$, and is longer than $\mathcal{L}$ by one block if $i = \ell + 1$.

Definition 1. *A* moderated redactable blockchain scheme *is a tuple of probabilistic polynomial time algorithms* $\mathcal{RBC} \overset{\text{def}}{=} (\mathsf{Gen}, \mathsf{Create}, \mathsf{Verify}, \mathsf{Redact}, \mathsf{Install})$ *satisfying the following:*

1. *The* ***key-generation algorithm*** $\mathsf{Gen}(1^\lambda)$***:*** *Takes as input a unary security parameter* 1^λ *and outputs* $(pk, sk, \mathcal{L})$*, where* pk *is the* public key*,* sk *is the* private key*, and* $\mathcal{L}$ *is the ledger. We assume that* $|pk|, |sk|$ *are polynomial in* λ*, and* λ *can be inferred from* pk *or* sk*.*
2. *The* ***block-creator algorithm*** $\mathsf{Create}(pk, \mathcal{L}, C)$***:*** *Takes as input the public key* pk*, the ledger* $\mathcal{L}$*, and a content* C*. It generates and returns a block* B *containing* C*, to be appended at the end of* $\mathcal{L}$*.*
3. *The* ***block-verifier algorithm*** $\mathsf{Verify}(pk, \mathcal{L}, i, B)$***:*** *Takes as input the public key* pk*, the ledger* $\mathcal{L}$*, a positive integer* $i \leq \ell + 1$*, and a block* B*. It performs two verifications, denoted* Φ *and* Ψ*, which are specified as part of* Verify *description by the blockchain designer. Let:*

$$V \leftarrow \mathsf{Version}(B)\,, \tag{2}$$
$$\vec{V} \leftarrow \left[\mathsf{Version}(\mathcal{L}[0]), \ldots, \mathsf{Version}(\mathcal{L}[\ell])\right], \tag{3}$$
$$\mathcal{L}^* \leftarrow \mathsf{Transform}(\mathcal{L}, i, B)\,. \tag{4}$$

   Verify *returns* 1 *if and only if both* $\Phi(\vec{V}, i, V)$ *and* $\Psi(pk, \mathcal{L}^*)$ *return* 1*. Algorithm* Φ *prevents reversion attacks by comparing the version of* B *with (possibly all) existing block versions. Algorithm* Ψ *checks the the consistency of the ledger for* $\mathcal{L}^*$ *that results from installing* B *at position* i *of* $\mathcal{L}$*.*
4. *The* ***redaction algorithm*** $\mathsf{Redact}(sk, \mathcal{L}, i, C)$***:*** *Takes as input the private key* sk*, the ledger* $\mathcal{L}$*, a positive integer* $i \leq \ell$*, and a content* C*. It returns a block* B *containing* C*, to replace* $\mathcal{L}[i]$*.*

5. *The **block-installer algorithm** $\mathsf{Install}(pk, \mathcal{L}, i, B)$**:** Takes as input the public key pk, the ledger $\mathcal{L}$, a positive integer $i \leq \ell + 1$, and a block B. If $\mathsf{Verify}(pk, \mathcal{L}, i, B)$ is 0, it returns 0. Otherwise, it installs B at index i of $\mathcal{L}$ (replacing an existing block in case $i \leq \ell$), and returns 1. Formally, a successful installation of B at index i is denoted by $\mathcal{L} \leftarrow \mathsf{Transform}(\mathcal{L}, i, B)$, as defined by Equation (1).*

For any moderated redactable blockchain scheme $\mathcal{RBC}$, the following correctness requirements must be satisfied.

Definition 2 (Correctness). *It is required that for every λ, every $(pk, sk, \mathcal{L})$ output by $\mathsf{Gen}(1^\lambda)$, and any valid content C:*

*(a) **Anyone can create a valid block to be appended to the ledger:** Let $B \leftarrow \mathsf{Create}(pk, \mathcal{L}, C)$. Then*

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

*(b) **The administrator can change any block of the ledger to contain any valid content:** For any positive integer $i < \ell$, let $B \leftarrow \mathsf{Redact}(sk, \mathcal{L}, i, C)$. Then*

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

Let $\mathcal{RBC}$ be a moderated redactable blockchain scheme per Definition 1, and consider Experiment 1 for an adversary $\mathcal{A}$ and security parameter λ.

1. $\mathsf{Gen}(1^\lambda)$ is run to obtain $(pk, sk, \mathcal{L})$. The set $\mathsf{Hist} \leftarrow \emptyset$ is set to empty.
2. Adversary $\mathcal{A}$ is given pk, a read-only view of $\mathcal{L}$, and access to oracles $\mathtt{REDC}_{sk,\mathcal{L}}(\cdot,\cdot)$ and $\mathtt{INST}_{pk,\mathcal{L}}(\cdot,\cdot)$.
   - The REDC oracle responds to queries of the form (i, C) by returning a redacted block $B \leftarrow \mathsf{Redact}(sk, \mathcal{L}, i, C)$. It also adds (i, B) to the set Hist, i.e., $\mathsf{Hist} \leftarrow \mathsf{Hist} \cup \{(i, B)\}$.
   - The INST oracle responds to queries of the form (i, B) by returning a bit $b \leftarrow \mathsf{Install}(pk, \mathcal{L}, i, B)$.
3. Finally, $\mathcal{A}$ outputs (i^*, B^*). She succeeds, and the experiment returns 1, if and only if all of the following conditions hold:

$$(a)\ 0 < i^* < \ell, \quad (b)\ \mathsf{Verify}(pk, \mathcal{L}, i^*, B^*) = 1, \quad (c)\ (i^*, B^*) \notin \mathsf{Hist}\,.$$

Experiment 1. The redaction experiment $\mathsf{Redact}_{\mathcal{A},\mathcal{RBC}}(\lambda)$. The success conditions can be explained as: (a) The index i points to an *internal* block of the ledger (as otherwise it is not an attack), (b) The block B^* is valid for position i^*, and (c) The pair (i^*, B^*) is new, meaning that B^* is not received from the redaction oracle in response to a query for index i^*. A particular observation is that the adversary wins if B^* is received from REDC, but for another location $i' \neq i^*$.

Definition 3. *A redactable blockchain scheme $\mathcal{RBC}$ is* existentially unredactable under chosen-redaction attacks*, or just* secure*, if for all probabilistic polynomial-time adversaries $\mathcal{A}$ taking part in Experiment 1, there is a negligible function* negl *such that* $\Pr\left[\mathsf{Redact}_{\mathcal{A},\mathcal{RBC}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$.

6 Constructs Based on Signature Schemes

In this section, we first provide a simple construction of redactable blockchains in the moderated setting (Construct 1). We show that the construct is insecure, but it serves an illustrative purpose. We then present a variation (Construct 2) that is proven secure under Definition 3. Both constructs completely delegate the blockchain functionality to the challenger of Fig. 2: Any write operation must go through the challenger, and therefore we are not worried about keeping an immutable total ordering of the blocks. It is similar in nature to the ideal functionality in a hybrid multi-party setting, except that our model is game-based rather than simulation-based.

The main primitive used in both constructs is a signature schemes *strongly* unforgeable under adaptive chosen-message attack (sUF-CMA). Due to a lack of space, we define sUF-CMA signatures informally; the interested reader can consult [3, p. 531]: A signature scheme is sUF-CMA secure if, given public key and access to the signing oracle, the adversary cannot generate a valid message-signature pair (m, σ), such that the pair is new. Here, *new* means σ is not returned by the signing oracle in response to query m. This gives the adversary more freedom than the (weak) UF-CMA signatures, where the adversary wins if m in the output (m, σ) is never queried to the signing oracle.

Boneh et al. [2, p. 230] provide a list of many constructions of efficient sUF-CMA signatures in the literature, both in the standard and the random oracle models. There are also efficient transformations that convert any UF-CMA secure signature to an sUF-CMA secure one [14].

6.1 An Incorrect and Insecure Construct

Construct 1 is incorrect and insecure, but helps in understanding the way our definitional framework works. It uses the following ideas: The adversary can ask the challenger to append any valid block at the end of the blockchain. Normal blocks do not include any information about each other (such as the hash of the previous block). Such information, necessary for the secure operation of an ordinary blockchain, is abstracted via the ideal functionality in the model: The adversary is not allowed to make any direct writes to the ledger, and therefore the challenger can keep the ledger blocks in their total order. The redactability is achieved with a signature schemes *strongly* unforgeable under adaptive chosen-message attack (sUF-CMA), denoted $(\mathsf{GenSig}, \mathsf{Sign}, \mathsf{VerifySig})$: The challenger installs a redacted block only if its witness holds the signature of itself and the next block. The reversion attack (Section 4) is prevented by introducing version numbers in the block structure: Initially, each block carries version 1. Upon each redaction, the version number is incremented. The verification function of the blockchain checks whether the version of a redacted block is strictly greater than the version of the block being replaced. This way, the adversary cannot reinstall a previously valid block again.

Construction 1 (Insecure and Incorrect). *The redactable blockchain* $\mathcal{RBC}_{\text{bad}}$ *is defined as follows. The block structure is* $B \overset{\text{def}}{=} (C, V, W)$*, where each block contains content* C*, version* V*, and witness* W*.*

- $\mathsf{Gen}(1^\lambda)$ *simply calls the generator for the underlying signature scheme to obtain the public and private keys:* $(pk, sk) \leftarrow \mathsf{GenSig}(1^\lambda)$*. It sets* $\mathcal{L} \leftarrow [B_0]$*, where* $B_0 \leftarrow (\varepsilon, 1, \varepsilon)$*.*
- $\mathsf{Create}(pk, \mathcal{L}, C)$ *returns* $B \leftarrow (C, 1, \varepsilon)$*.*
- $\mathsf{Verify}(pk, \mathcal{L}, i, B)$ *returns* 1 *if and only if all of the following conditions hold:*
  - B *has correct structure, and* $i \leq \ell + 1$ *is a positive integer.*
  - $\Phi(\vec{V}, i, V)$ *returns* 1*: This happens if and only if* $(i = \ell + 1) \wedge (V = 1)$ *(the block is being appended and has version* 1*), or* $(i \leq \ell) \wedge (\vec{V}[i] < V)$ *(an existing block is being redacted, and the new version is greater than the existing one to foil reversion attacks).*
  - $\Psi(pk, \mathcal{L}^*)$ *returns* 1*: This happens if and only if for every pair* (B, B') *of subsequent blocks in* $\mathcal{L}^*$*, if* $\mathsf{Version}(B) > 1$ *(i.e., if* B *is redacted), then*

    $$\mathsf{VerifySig}(pk, B^{\cancel{W}} \mid\mid B', W) = 1\,, \tag{5}$$

    where $W \overset{\text{def}}{=} \mathsf{Witness}(B)$*, and* $B^{\cancel{W}} \overset{\text{def}}{=} (C, V)$ *(i.e., block* B *except* W*). Put simply, this means that* W *is a valid signature on* $C \mid\mid V \mid\mid C' \mid\mid V' \mid\mid W'$*.*
- $\mathsf{Redact}(sk, \mathcal{L}, i, C)$*: Creates* $B \leftarrow (C, V, W)$ *using content* C*, where* $V \leftarrow \mathsf{Version}(\mathcal{L}[i]) + 1$ *and* W *is a signature on the current block except* W *itself (denoted* $B^{\cancel{W}}$*), as well as the next block* $\mathcal{L}[i+1]$*:*

  $$W \leftarrow \mathsf{Sign}(sk, B^{\cancel{W}} \mid\mid \mathcal{L}[i+1])\,.$$

  Notice that incrementing the version number, as well as the computation of witness by signing the current and next blocks, are consistent with the requirements of Verify*.*
- $\mathsf{Install}(pk, \mathcal{L}, i, B)$*: Works exactly as specified in Definition 1.*

Correctness Issues. A series of valid actions can put the ledger in a state that block creation for appending is no longer possible, violating the first requirement of Definition 2. For instance, let C_1, C_1' and C_2 be any valid contents, and consider the following actions, following $(pk, sk, \mathcal{L}) \leftarrow \mathsf{Gen}(1^\lambda)$:

$$\begin{array}{ll}
B_1 \leftarrow \mathsf{Create}(pk, \mathcal{L}, C_1), & \mathsf{Install}(pk, \mathcal{L}, 1, B_1) \\
B_1' \leftarrow \mathsf{Redact}(sk, \mathcal{L}, 1, C_1'), & \mathsf{Install}(pk, \mathcal{L}, 1, B_1') \\
B_2 \leftarrow \mathsf{Create}(pk, \mathcal{L}, C_2), & \mathsf{Install}(pk, \mathcal{L}, 2, B_2)
\end{array}$$

The first line creates and appends a block, the second line redacts it, and the third line tries to append a new block. The last $\mathsf{Install}$ fails as it calls Verify, which in turn calls Ψ: Since the version of B_1' is greater than 1, Ψ requires it to hold a signature containing information about the next block, as per Equation (5), which is not the case.

The underlying reason is that, in this particular construct, it is meaningless for the last block of the ledger to be redacted, as there is no next block to sign. It is possible *not* to increase version number for redacting the last block, or disallow such redaction by requiring $i \neq \ell$ in designing Redact.

One can violate the second requirement of Definition 2 as well, by following a series of valid actions that put the ledger in a state where redaction of some blocks are impossible. Let $\mathcal{L} \leftarrow [B_0, B_1, B_2, B_3]$ be a ledger constructed by appending three blocks, and C_1' and C_2' be valid contents. Consider the following actions:

$$\begin{aligned} B_1' &\leftarrow \mathsf{Redact}(sk, \mathcal{L}, 1, C_1'), \qquad && \mathsf{Install}(pk, \mathcal{L}, 1, B_1') \\ B_2' &\leftarrow \mathsf{Redact}(sk, \mathcal{L}, 1, C_2'), \qquad && \mathsf{Install}(pk, \mathcal{L}, 1, B_2') \end{aligned}$$

Again, the last install fails: For the pair (B_1', B_2'), algorithm Ψ requires B_1' to hold a signature on $B_1'^{\cancel{W}} || B_2'$ (see Equation (5)). However, B_1' is redacted prior to B_2': As a result, B_1' holds a signature on $B_1^{\cancel{W}} || B_2$, which becomes invalid after B_2 is redacted. Consequently, the second requirement of Definition 2 is violated.

The underlying reason is the indifference in the verification algorithms as to which block is newer. The next section shows how using unique versions can resolve this issue.

Security Issues. On the surface, it seems that the adversary cannot succeed in Experiment 1. An informal (and false) argument is as follows: We use an adversary who succeeds in the game as a subroutine, to forge a valid signature on an arbitrary message. The forger simulates the challenger. It gives the public key of the signature scheme to the adversary, and answers all redaction queries by using the *signing oracle*. When the adversary outputs a successful redaction (i, B), the witness W is a valid signature on the message $m \leftarrow B^{\cancel{W}} \mathbin{||} \mathcal{L}[i+1]$. The forger outputs (m, W) as a valid message-signature pair.

The fallacy in the above argument is that the forger must output a *new* pair (m, W), as required by sUF-CMA signature forgery. However, the informal proof does not show that this pair is new. In fact, as is explained below, it is easy for an adversary to succeed in the game without forging any signature.

Adversary $\mathcal{A}$ proceeds as follows: It creates a block $B \leftarrow (\texttt{"original"}, 1, \varepsilon)$, and appends it three times by calling the `INST` interface of the challenger on queries $(1, B)$, $(2, B)$ and $(3, B)$, respectively. At this point, $\mathcal{L} = [B_0, B, B, B]$.

Next, $\mathcal{A}$ queries the `REDC` interface of the challenger on $(1, \texttt{"modified"})$, and receives $B' \leftarrow (\texttt{"modified"}, 2, W)$, where W is a signature on $m \leftarrow B'^{\cancel{W}} \mathbin{||} B$, where $B'^{\cancel{W}}$ is $\texttt{"modified"} \mathbin{||} 2$.

While the redaction was requested for position 1, the adversary uses position 2: She outputs $(2, B')$, and halts.

At this point, $\mathsf{Hist} = \{(1, B')\}$, and therefore $(2, B')$ is new. Furthermore, B' is a valid redaction for position 2, since $\mathcal{L}[3] = \mathcal{L}[2] = B$. We conclude that the adversary breaks the security by outputting a successful reduction, without forging a new signature.

The underlying reason for this attack is duplicate blocks in the ledger. In the next section, we show how to resolve this issue by incorporating unique versioning.

6.2 A Correct and Secure Construct

To resolve the issues with Construct 1, we introduce two major modifications in the construct: First, each block must have a unique version number: The j^{th} block to be installed (be it appended or redacted) should carry version j. This guarantees the uniqueness of each block in the ledger, which in turn resolves the security issues.

To address the correctness issues, the second modification is applied: The signature is verified only when the block holding it is newer than the next block. This check is easily conducted due to the unique versioning that we introduced: For any two consecutive blocks $B \overset{\text{def}}{=} (C, V, W)$ and $B' \overset{\text{def}}{=} (C', V', W')$ in the ledger, define:

$$\psi(pk, B, B') \overset{\text{def}}{=} \begin{cases} 1 & \text{if } V' > V, \\ \mathsf{VerifySig}(pk, C \,||\, V \,||\, V', W) & \text{if } V' < V. \end{cases} \tag{6}$$

As we will see, the algorithm Ψ calls ψ for each pair of blocks in the ledger, and returns the logical AND of their results.

Construction 2 (Secure). *The redactable blockchain $\mathcal{RBC}_{\text{good}}$ is defined as follows. The block structure is $B \overset{\text{def}}{=} (C, V, W)$, where each block contains content C, version V, and witness W.*

- $\mathsf{Gen}(1^\lambda)$ *simply calls the generator for the underlying signature scheme to obtain the public and private keys: $(pk, sk) \leftarrow \mathsf{GenSig}(1^\lambda)$. It sets $\mathcal{L} \leftarrow [B_0]$, where $B_0 \leftarrow (\varepsilon, 1, \varepsilon)$.*
- $\mathsf{Create}(pk, \mathcal{L}, C)$ *returns $B \leftarrow (C, V, \varepsilon)$, where V is larger than any version in the ledger (and is thus unique). Symbolically, $V \leftarrow \mathsf{MaxV}(\vec{V})$, where $\vec{V}$ is defined as in Equation (3), and*

$$\mathsf{MaxV}(\vec{V}) \overset{\text{def}}{=} 1 + \max_{0 \le i \le \ell} \vec{V}[i]\,. \tag{7}$$

- $\mathsf{Verify}(pk, \mathcal{L}, i, B)$ *returns 1 if and only if all conditions below are satisfied:*
  - *B has correct structure, and $0 < i \le \ell + 1$.*
  - *$\Phi(\vec{V}, i, V)$ returns 1: This happens if and only if $V = \mathsf{MaxV}(\vec{V})$.*
  - *$\Psi(pk, \mathcal{L}^*)$ returns 1: This happens if and only if for every pair (B, B') of subsequent blocks in $\mathcal{L}^*$, it holds that $\psi(pk, B, B') = 1$, as per Equation (6).*
- $\mathsf{Redact}(sk, \mathcal{L}, i, C)$*: If i points to an internal block (i.e., $0 < i < \ell$), it creates a block $B \leftarrow (C, V, W)$ using content C, where $V \leftarrow \mathsf{MaxV}(\vec{V})$ and*

$$W \leftarrow \mathsf{Sign}\left(sk, C \,||\, V \,||\, \mathsf{Version}(\mathcal{L}[i+1])\right).$$

- $\mathsf{Install}(pk, \mathcal{L}, i, B)$*: Works exactly as specified in Definition 1.*

Notice that for redacting the block at $i = \ell$, the private key is not required. For any C, replacing the existing block $\mathcal{L}[\ell]$ with $B \leftarrow (C, \mathsf{MaxV}(\vec{V}), \varepsilon)$ is valid. This is because there is no next block B' for which $\psi(pk, B, B') = 1$ must hold. However, the ability to redact the last block without the private key does not constitute an attack. In our model (Experiment 1), the adversary succeeds only if she redacts a block inside the ledger (i.e., $0 < i < \ell$).

Theorem 1. *$\mathcal{RBC}_{\text{good}}$ is correct per Definition 2.*

Proof. There are two conditions to check.

Condition (a): $\mathsf{Create}(pk, \mathcal{L}, C)$ returns $B \leftarrow (C, \mathsf{MaxV}(\vec{V}), \varepsilon)$. Clearly, the content of this block is C. Furthermore, if $\mathcal{L}$ is already a valid chain, so is $\mathcal{L}^* \leftarrow \mathcal{L} + [B]$. This is because the version of B is correctly computed as required by Φ. Moreover, ψ returns 1 on all pairs of blocks in $\mathcal{L}^*$ prior to the last pair (due to the validity of $\mathcal{L}$). Finally, for the last pair $(\mathcal{L}[\ell], B)$, since $\mathsf{Version}(\mathcal{L}[\ell]) < \mathsf{Version}(B)$, the return value of ψ is trivially 1. As a result, all block pairs verify, and Ψ returns 1 as well.

Condition (b): $\mathsf{Redact}(sk, \mathcal{L}, i, C)$ returns $B \leftarrow (C, \mathsf{MaxV}(\vec{V}), W)$. Clearly, the content of this block is C. Furthermore, if $\mathcal{L}$ is already a valid chain, so is $\mathcal{L}^* \leftarrow \mathsf{Transform}(\mathcal{L}, i, B)$. This is because the version of B is correctly computed as required by Φ. Moreover, ψ returns 1 on all pairs of blocks in $\mathcal{L}^*$, except perhaps the two *special* pairs involving B (the validity of other pairs is due to the validity of $\mathcal{L}$). We show that ψ also returns 1 on those special pairs, which involve B:

- The first special pair is $(\mathcal{L}[i-1], B)$. Since $\mathsf{Version}(\mathcal{L}[i-1]) < \mathsf{Version}(B)$, the return value of ψ is trivially 1.
- The second special pair is $(B, \mathcal{L}[i+1])$. Since $\mathsf{Version}(\mathcal{L}[i+1]) < \mathsf{Version}(B)$, algorithm ψ requires the block B to hold a proper witness. This holds due to the correctness of the underlying signature scheme.

As a result, all block pairs verify, and Ψ returns 1 as well. □

Theorem 2. *If the signature scheme* $(\mathsf{GenSig}, \mathsf{Sign}, \mathsf{VerifySig})$ *is* strongly *unforgeable under chosen-message attack (sUF-CMA), then $\mathcal{RBC}_{\text{good}}$ is secure per Definition 3.*

Proof. Let $\mathcal{A}$ be an adversary who, for infinitely many λ values, succeeds in the experiment $\mathsf{Redact}_{\mathcal{A},\mathcal{RBC}_{\text{good}}}(\lambda)$ with probability at least $\epsilon \overset{\text{def}}{=} \epsilon(\lambda)$. We construct a forger algorithm $\mathcal{F}$ which, for infinitely many λ values, forges a signature with probability ϵ.

The forger $\mathcal{F}$ receives as input the public key pk of the signature scheme, as well as oracle access to the signing oracle $\mathsf{Sign}_{sk}(\cdot)$. It sets $\mathsf{Hist} \leftarrow \emptyset$, generates $\mathcal{L} \leftarrow [B_0]$ as in Construct 2, runs $\mathcal{A}(pk, \mathcal{L})$, and answers its queries as follows:

- **Installation queries** $\mathtt{INST}(i, B)$**:** The forger $\mathcal{F}$ simply calls $b \leftarrow \mathsf{Install}(pk, \mathcal{L}, i, B)$, and returns b.

- **Redaction queries** $\texttt{REDC}(i, C)$**:** If $i \leq 0$ or $i \geq \ell$, the forger $\mathcal{F}$ returns ε. Otherwise, $\mathcal{F}$ creates block $B \leftarrow (C, V, W)$, where $V \leftarrow \mathsf{MaxV}(\vec{V})$, and W is computed by querying the signature oracle on $\big(C \,||\, V \,||\, \mathsf{Version}(\mathcal{L}[i+1])\big)$. It then adds (i, B) to Hist, and returns B.

If the adversary stops but does not succeed in outputting (i, B) as required in Experiment 1, the forger $\mathcal{F}$ outputs $\perp$ and halts. Otherwise, parse $B \overset{\text{def}}{=} (C, V, W)$. Since B is verified, W is a valid signature on $m \leftarrow (C \,||\, V \,||\, V_{i+1})$, where $V_{i+1} \overset{\text{def}}{=} \mathsf{Version}(\mathcal{L}[i+1])$. Subsequently, $\mathcal{F}$ outputs (m, W) as a forgery.

To show that the forgery is new, we must prove that W was never returned by the signing oracle in response to query m. Since $(i, B) \notin \mathsf{Hist}$, we consider the two remaining possibilities:

- $(i', B) \in \mathsf{Hist}$ for some $i' \neq i$: Impossible because $\mathsf{Version}(\mathcal{L}[i+1])$, which constitutes a part of m, is unique due to the uniqueness of version numbers in our solution. Therefore, no other position i' may correspond to the same m.
- $(i, B') \in \mathsf{Hist}$ for some $B' \neq B$, where B can be efficiently computed from $B' \overset{\text{def}}{=} (C', V', W')$, and W' is valid on m: For this to happen, it must be the case that B and B' are identical except in their witnesses. Then, both W and W' are valid signatures on m. This constitutes a strong forgery on the signature scheme, and $\mathcal{F}$ can output (m, W) as a valid forgery.

We conclude that the success probability of $\mathcal{F}$ in producing a valid forgery is the same as the success probability of $\mathcal{A}$ in producing a valid redaction. □

7 Conclusion

In this paper, we discussed two settings for redactable blockchains: The moderated setting, where redactions are handled by administrators, and the unmoderated setting, where redactions are voted on. Four novel attacks were discussed against previous constructs in both settings. We argued the attacks are the result of the lack of a definitional framework for redactable blockchains. We suggested the first attempt at such a framework, and explained our design decisions. Two simple constructs, both based on signature schemes, were proposed. The first one was demonstrated to be insecure, while the latter alleviated the security issues and was provably secure in our definitional framework.

References

1. Ateniese, G., Magri, B., Venturi, D., Andrade, E.: Redactable Blockchain–or–Rewriting History in Bitcoin and Friends. In: EuroS&P. IEEE (2017)
2. Boneh, D., Shen, E., Waters, B.: Strongly Unforgeable Signatures Based on Computational Diffie-Hellman. In: PKC. Springer (2006)
3. Boneh, D., Shoup, V.: A Graduate Course in Applied Cryptography. Draft (2020)

4. CoinDesk: Understanding The DAO Attack (2016), https://tinyurl.com/dao-attack
5. Council of European Union: Regulation (EU) 2016/679: General Data Protection Regulation (GDPR) (2016), https://gdpr-info.eu
6. Derler, D., Ramacher, S., Slamanig, D., Striecks, C.: I Want to Forget: Fine-Grained Encryption with Full Forward Secrecy in the Distributed Setting. IACR Cryptology ePrint Archive (2019)
7. Derler, D., Samelin, K., Slamanig, D., Striecks, C.: Fine-Grained and Controlled Rewriting in Blockchains: Chameleon-Hashing Gone Attribute-Based. IACR Cryptology ePrint Archive (2019)
8. Deuber, D., Magri, B., Thyagarajan, S.A.K.: Redactable Blockchain in the Permissionless Setting. In: Symposium on Security and Privacy. IEEE (2019)
9. Garay, J., Kiayias, A., Leonardos, N.: The Bitcoin Backbone Protocol: Analysis and Applications. In: EUROCRYPT. Springer (2015)
10. Grigoriev, D., Shpilrain, V.: RSA and Redactable Blockchains (2020), arXiv report 2001.10783
11. Hargreaves, S., Cowley, S.: How Porn Links and Ben Bernanke Snuck Into Bitcoin's Code (2013), https://tinyurl.com/bitcoin-snuck
12. Hopkins, C.: If You Own Bitcoin, You Also Own Links to Child Porn (2020), https://tinyurl.com/bitcoin-child
13. 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)
14. Liu, J.K., Au, M.H., Susilo, W., Zhou, J.: Short Generic Transformation to Strongly Unforgeable Signature in the Standard Model. In: ESORICS. Springer (2010)
15. Lumb, R.: Downside of Bitcoin: A Ledger That Can't Be Corrected (2016), https://tinyurl.com/btc-immutable
16. Nakamoto, S.: Bitcoin: A Peer-to-Peer Electronic Cash System (2009), available from http://www.bitcoin.org/bitcoin.pdf
17. Pass, R., Shi, E.: FruitChains: A Fair Blockchain. In: Symposium on Principles of Distributed Computing (2017)
18. Pearson, J.: The Bitcoin Blockchain Could Be Used to Spread Malware, INTERPOL Says (2015), https://tinyurl.com/bitcoin-malware
19. Puddu, I., Dmitrienko, A., Capkun, S.: μchain: How to Forget Without Hard Forks. IACR Cryptology ePrint Archive (2017)
20. Wood, G.: Ethereum: A Secure Decentralised Generalised Transaction Ledger (2014), Ethereum Project yellow paper