Data Redaction in Smart-Contract-Enabled Permissioned Blockchains

Gennaro Avitabile[1,†], Vincenzo Botta[2,*,†], Daniele Friolo[2,†] and Ivan Visconti[3,†]

[1]*IMDEA Software Institute, Campus de Montegancedo s/n, 28223 Pozuelo de Alarcón, Madrid, Spain*
[2]*Sapienza University of Rome, Piazzale Aldo Moro 5, 00185 Rome, Italy*
[3]*University of Salerno, Via Giovanni Paolo II, 132, 84084 Fisciano, Italy*

Abstract

Balancing immutability and compliance with regulations stands as a significant challenge in the realm of blockchain technology applications. Due to the increase in privacy regulations (e.g., the General Data Protection Regulation (GDPR) in the European Union), it is essential to address the problem of deleting data from a blockchain without compromising the security and transparency of the blockchain itself.

Several works proposed techniques to address the above problem, and the most effective ones are mainly applicable to permissioned blockchains. In particular, in their seminal work, Ateniese et al. [EuroS&P 2017] were the first to propose a redactable blockchain. Their approach focuses on permissioned blockchains and they showed how to change the content of a transaction without breaking its previous links to the blockchain by using a special cryptographic hash function (i.e., chameleon hash functions) and multi-party protocols to make the required distributed computations.

We observe that the redaction technique of Ateniese et al. does not take into account the possibility that the blockchain includes smart contracts and that a redaction of a transaction might leave inconsistencies in the logic of the contracts, making some remaining non-redacted transactions invalid. We find this decision rather limiting since decentralized and publicly verifiable computation guaranteed by smart-contract-enabled blockchains is indeed necessary for modern applications.

In order to overcome the above limitations of the applicability of the redaction techniques of Ateniese et al., we propose a redaction technique with wider applicability that leverages succinct non-interactive arguments of knowledge (SNARKs) to realize what we call a *proof-of-consistency*. A SNARK will be computed to show that the current state of a smart contract is consistent with respect to a previous state since there existed transactions, now redacted, that produced the current state from a former publicly verifiable state. Therefore, a successful verification of such a SNARK allows to maintain the consistency of the non-redacted transactions according to the state of the smart contract, and the resulting blockchain maintains public verifiability and consistency in the presence of redacted transactions.

Keywords

Blockchain, Redaction, Permissioned, Smart Contract, SNARK

1. Introduction

Motivated by the widespread adoption of decentralized platforms, and the requirements imposed by the GDPR and in general by laws, many techniques have been proposed to erase data from blockchains. As shown by Matzett *et al.* [1], even a blockchain like Bitcoin, designed only for financial use, stores illegal data such as child pornography and dark web services. Because of that, some countries could have banned blockchain usage. A seminal work from Ateniese *et al.* [2] proposed a technique that allows to relax the immutability of a blockchain while still maintaining the consistency of its structure (i.e., the validity of previously computed hash pointers) through chameleon hash functions. In a nutshell, a chameleon hash function is a hash function that has a public hash key and a private trapdoor. Unless the trapdoor is known, this function achieves collision resistance as a standard hash function. In theory, knowledge of prior collisions could be exploited to find more collisions for an arbitrary new message,

*Corresponding author. The majority of the work was done while affiliated with the University of Warsaw.
†These authors contributed equally.
gennaro.avitabile@imdea.org (G. Avitabile); botta.vin@gmail.com (V. Botta); friolo@di.uniroma1.it (D. Friolo); visconti@unisa.it (I. Visconti)

even without knowledge of the trapdoor and this must therefore be considered leading to a special definition of collision resistance. Two things must be noticed here: (1) their solution requires changing the specifications of the blockchain to use this hash function; (2) the trapdoor must be kept secret by someone who is trusted, introducing an additional trust assumption in the blockchain. These two points make the solution of Ateniese *et al.* suitable for permissioned blockchains in which the governance maintaining the blockchain is already trusted (the majority of the parties cooperating within the governance must be honest) and already controls the blockchain's consensus. To perform the redaction, a multi-party computation protocol is executed among all the parties of the consortium. Following the seminal work of [2] several other works proposed redaction techniques for the permissioned setting [3, 4, 5, 6, 7, 8].

1.1. Blockchain Structure

A blockchain is a series of ordered blocks whose design lies in deploying a chain in which each block is linked to the previous one. This linking operation is obtained using hash functions. The hash function is used to make any block dependent on the content of the previous block so that everyone can publicly verify that the current state of the blockchain was not tampered with. Each block contains data that can be divided into transactions. Each transaction can contain a coin transfer from some users to other users or other data that can be stored on the blockchain to make them publicly available and verifiable. As said before, the hash is used to link the blocks, so a block B_i in the chain contains a value that is the hash of the transactions of the previous block B_{i-1} together with some additional values that depend on the consensus mechanism. Even if it seems that the blockchain should be immutable, there are scenarios in which it is mandatory to delete some data (e.g., in case of compliance with laws and regulations). Therefore, a technique that allows the editing of the blockchain is highly desirable.

1.2. Previous Solution

Let us give a high-level description of the chameleon-hash-function-based solution of [2]. Each member of the blockchain consortium, or governance, has a share of a trapdoor of a chameleon hash function. Whenever a block B must be modified, the consortium members agree on a new B'. Then, to replace B with B' without altering its hash value, the consortium computes a collision in the hash function with a multi-party protocol executed from the consortium members' trapdoor shares. The collision-resistance property of the chameleon hash function and the honest majority of the members of the governance ensure that when the trapdoor is kept private, the blockchain is consistent and resilient to any chain tampering attempts.

Since it is needed that the governance is known and fixed, this result is mainly suitable for permissioned blockchains, even though, as the authors of [2] outline, their solution could potentially find applications also to permissionless blockchains by distributing the trapdoor among the full nodes or a chosen subset of participants. This is however problematic since the number of full nodes can drastically change over time, and there is no obligation for the nodes with a share of the trapdoor to remain active forever (losing the possibility of reconstructing the trapdoor).

We observe that even though one limits the usage of the techniques in [2] exclusively to permissioned blockchains, there is a major problem: the transactions originally in B and modified in B' may be somehow related to subsequent transactions. This becomes more evident when considering a smart-contract-enabled blockchain. Roughly, a smart contract is a piece of code that can be fed with inputs coming from valid transactions. A smart contract is associated with a state, and each time an input is given to the smart contract, the state changes. The current state of the smart contract can be inferred from the blockchain.

In the presence of a blockchain redaction it is possible that even if the chaining among blocks and within transactions is consistent, the logic inferred from transactions stored in the blockchain may be broken. This is evident when considering the state of the smart contract they refer to.

Simple solutions and their drawbacks. A direct approach to fix the problem of maintaining the consistency of the state of a smart contract in the presence of redacted transactions, still remaining within the techniques envisioned by Ateniese *et al.*, consists of possibly redacting also subsequent correlated transactions in such a way that the state of the blockchain remains consistent. We will now discuss why such a repeated redaction process can be very harmful for those use cases that relied on data available in smart contracts.

Let us consider as a toy example a smart contract that keeps a counter initialized to 0. Suppose a transaction tx_1 with the aim of increasing the counter is submitted. The state of the counter is now increased to 1, assuming tx_1 gets accepted by the blockchain. If subsequent transactions $tx_2, \ldots, tx_n$ also get accepted, this would lead to a final state having n as the value of the counter. It is clear that if tx_1 is removed from the blockchain, then any transaction that crucially relied on the value of the counter being n would not be consistent anymore. Indeed, the remaining transactions add up to $n-1$ and not n. In turn, consistency can be enforced by also redacting those transactions that relied on the value of the counter being n, and this in turn might require further redactions severely damaging those applications that relied on those data that would be removed.

In light of the above discussion, we have the following natural open problem.

Open Problem: *Is there a redaction technique for updating a transaction in permissioned smart-contract-enabled blockchains without affecting other transactions and still maintaining the smart contract in a consistent state?*

We obviously require that all redaction operations do not affect the public verifiability feature of the blockchain in any way since otherwise, we would end up in the classical setting where clients have to trust servers.

1.3. Our Solution

At a high level, a smart contract ϕ consists of a set of functions $f_1, \ldots, f_n$. Transactions point to the address of a smart contract and contain an input for a smart contract function f_i. Each function of ϕ takes as input the current state of the smart contract (that can be inferred by the blockchain content) and the input contained inside the transaction that points to ϕ. Given these two values as inputs, f_i produces a new state for ϕ. Note that, by following Ethereum Smart Contracts documentation[1], we do not allow smart contract functions to read transaction data inside the blockchain; only the smart contract states and the input.

Inspired by the approach of [9], we use the following technique to manage the redaction of a transaction that changes the state of a smart contract. Let us suppose that a transaction tx must be redacted. Before modifying tx, the governance checks if there are smart contracts with states that depend on the execution of tx (this can be done efficiently by maintaining this information in some ad-hoc data structure) and that would have inconsistent states in case tx is redacted. Let tx be a transaction that was pointing to some smart contract function in ϕ. The governance will look for the subsequent transactions of ϕ depending on the state `st` (i.e., it will look for the transactions directly affected by tx). Let $tx_1, \ldots, tx_m$ be the set of transactions that have to be redacted. Let us assume that $tx_1, \ldots, tx_m$ directly affect transactions $tx'_1, \ldots, tx'_n$. The governance will generate, for each transaction tx'_i, $i \in [n]$, which prior to the redaction was updating the state of a smart contract to a specific value, a proof π_i showing that there were transactions tx_j, $j \in [m]$, such that the transaction tx'_i leads to that given value of the smart contract. After $\pi_1, \ldots, \pi_n$ are generated, the governance can replace through the techniques of [2] the old transactions with the redacted ones. The proofs are auxiliary redaction information that can be either added to the redacted blocks or made available independently.

Going back to our example of updating a counter, if the governance redacts transaction tx_1, then the directly affected transaction is tx_2 (other transactions updating the counter and other transactions

[1] https://ethereum.org/en/developers/docs/smart-contracts/

using the next values of the counter are indirectly affected) since it was setting two as value of the counter starting from the value one set by tx_1. In order to avoid an impact on other transactions, we need that after executing tx_2, despite the redaction of tx_1 the value of the counter is still two, and this should be publicly verifiable as a correct value. With our solution, before deleting tx_1, the governance will prepare a proof π that certifies that tx_2 is consistent with the state of the smart contract having two as value of the counter since there existed a transaction tx_1 setting already the value to one. We call the type of proof described above as *proof-of-consistency*.

Given this high-level overview of our technique relying on proofs-of-consistency, it is clear that, in our redaction technique, the state of the blockchain before and after a redaction remains unchanged, and state consistency can be publicly verified. We remark that our technique can be efficiently implemented depending on the specific SNARK to be computed, and this can depend on several factors (e.g., the format of transactions, the computations that they perform, the complexity of the state of the smart contract). We also remark that the technique is not applicable to all possible updates of transactions. For instance, in the above case of the counter, the technique is applicable when tx_1 must be removed, but it is not applicable when tx_1 must be changed to a transaction aiming at setting the value of the counter to a value different than one (e.g., through two increments of the counter), since this would conflict with the fact that after executing tx_2 the value must be two.

2. Related Work and Comparison

Ateniese *et al.* [2] proposed a mechanism to update block contents in blockchains. Their solution is very efficient and is mainly applicable to the permissioned setting. Their deletion approach can potentially go unnoticed by users not actively participating in the redaction process. They rely on the concept of chameleon hash functions, consisting of equipping hash functions with trapdoors facilitating the discovery of many preimages for a given digest. In our approach, we depart from their mechanisms and address the important issue of dealing with the consistency of the permissioned blockchains in which smart contracts are deployed.

Puddu *et al.* [10] proposed a protocol where users can define alternative versions, referred to as "mutations", for their transactions. These mutations can be activated later through an expensive secure multi-party computation (MPC) protocol. Approval for a modification request is contingent upon a voting procedure utilizing proofs of work. In their solution, only the transaction's creator can allow its modifications, thereby preventing the deletion of content inserted by malicious entities. Their solution is thought to work in the permissionless setting, but this approach has the significant drawback of giving the miners the power to forbid any form of modification to the mined blocks, effectively bypassing the whole mutation mechanism.

Deuber *et al.* [11] introduced a redactable blockchain protocol for the permissionless setting. In their protocol, users can suggest modifications by inscribing proposals directly onto the blockchain. The approval of redaction proposals is determined through a voting process that combines consensus and computational power. This protocol requires an online voting procedure on the blockchain since they operate on a permissionless blockchain and they want to modify the view of all nodes. The drawbacks of online voting technique are discussed in [12, 9].

Thyagarajan *et al.* [13] introduced Reparo, which is an enhancement of Deuber *et al.*'s solution allowing to redact or modify blocks that are incorporated into the blockchain before the implementation of the software update containing the redaction protocol. Similar to Deuber *et al.*, Reparo is thought to be used on permissionless blockchains and relies on resource-intensive and interactive consensus protocols that necessitate several days for their execution.

Grigoriev *et al.* [14] proposed a data redaction mechanism based on the RSA cryptosystem, with a primary focus on permissioned blockchains. The idea is similar to the idea expressed by [2], and the same issues of [2] about smart contracts (i.e., it is not clear how to modify the transactions subsequent to a redacted transaction to keep the state consistent) applies also to their work.

Botta *et al.* [9] designed and implemented a mechanism for securely deleting some data specifically

from the Bitcoin blockchain. Their technique relies on transparent non-interactive succinct arguments of knowledge, allowing any node to locally delete some data from Bitcoin transactions, still preserving the public verifiability of the blockchain. Their technique requires that a node that wants to delete locally some data can do it directly creating a succinct proof that the modified block is the result of a redaction of a prior block that had the same content except for some *neutral* data and that produced a Merkle root that is consistent with the chaining of the blocks. The redaction technique proposed does not require any coordination among nodes. They apply their technique only to those data of Bitcoin (i.e., the free texts allowed by coinbase transaction and the one that can be added after an OP_RETURN instruction) that are irrelevant for the state of the blockchain (i.e., the UTXOs). We will depart from their technique and generalize it to preserve the consistency of the state of a smart contract in a permissioned blockchain.

Zhang *et al.* [5] proposed an enhanced version of [2] in which the redaction proposal can be done only by authorized users that have a specific secret. This is obtained using decentralized attribute-based encryption.

Zhou *et al.* [4] presented a solution for permissioned blockchains, similarly to [2] they use a chameleon hash function and allow two types of modifications: the transaction modification that can be performed only by the owner of the transaction and the transaction deletion that can be done by a regulator in case the transaction must be deleted. To achieve this, each transaction is hashed with a particular chameleon hash function that depends on the identity of the user that performs the hash. The regulator possesses a secret key that can be used to derive the secret keys of all other users for the hash function. Every modification or deletion must be accepted by the consensus mechanism. Moreover, the regulator has the power to modify all the transactions related to the modified or deleted transaction recovering the secret key of the owner of the impacted transaction.

Peng *et al.* [6] also proposed a system based on a voting procedure that is more efficient compared to [11]. Indeed, the voting procedure is done using an aggregate signature among the parties in the consortium on a block proposed by a user off-chain, publishing on-chain a way to verify the result of the ballot.

Wang *et al.* [15] proposed a different solution for permissioned blockchains based on verifiable delay functions. In their approach, when a redaction proposal is sent to a supervisor, the supervisor quickly constructs a chain fork to redact the proposed blocks. This fork is proposed to the leader node and the accounting nodes. If the produced fork keeps consistency it is spread as the new blockchain. Even if this solution can be efficient, it requires that the supervisor updates the entire blockchain.

Dai *et al.* [16] proposed a solution that exploits both the vote solution and the chameleon hash function, indeed their solution requires a vote to decide if a given block should be substituted, and once it is decided, it is replaced with the help of the chameleon hash function to keep the consistency of the links among the blocks.

The limitations of [2] apply essentially to all above works since they do not address consistency of states of smart contracts and the availability of legitimate transactions, once a prior and correlated transaction has been redacted.

3. Preliminaries

We will denote by $[k]$ the set $\{1, \ldots, k\}$.

3.1. Succinct Non-Interactive Argument of Knowledge

In this paper, we use the notion of succinct non-interactive argument of knowledge (SNARK). We defer to [17, 18, 19, 20] for a more precise description. Let R be a binary relation computable in polynomial time. We identify with the pair (x, w) the elements in the relation, where we call x the statement and w the witness. A SNARK for a relation R consists of two probabilistic polynomial-time (PPT) algorithms Prove and Verify that both have access to a common reference string (CRS), and potentially to a random oracle (RO) O:

- Prove(CRS, x, w): this is a PPT algorithm that takes as input the CRS, a statement x, and a witness w s.t. $(x, w) \in R$, and has oracle access to O. It outputs a proof π.
- Verify(CRS, x, π): this is a deterministic polynomial-time algorithm that takes as input the CRS, a statement x, a proof π, and has oracle access to O. It outputs 1 if the proof is accepted, and 0 otherwise.

Prove is used by the prover and Verify by the verifier. Informally, these algorithms must satisfy the following properties:

- Completeness: An honest prover convinces an honest verifier with overwhelming probability.
- Knowledge Soundness: If a malicious prover Prove* is able to prove a statement x with a given probability p, there exists a PPT extractor E such that if Prove* produces with non-negligible probability an accepting proof π for a statement x, then E with access to the prover outputs a witness w for x with overwhelming probability.
- Zero knowledge: The proof computed by Prove does not reveal any additional information apart the veracity of the claim.
- Succinctness: Verify runs in polynomial time in the size of the statement x; moreover, the proof size and the running time of the verifier are sublinear in the size of the witness for x.

When in the above definition the CRS can not have a trapdoor affecting knowledge soundness and zero knowledge, a SNARK is usually referred as STARK, where the letter "T" remarks the transparency of the involved CRS.

3.2. Chameleon Hash Functions

Chameleon hash functions were first introduced in [21]. In this work, we report the definition from [2]. At a very high level, a chameleon hash function is a cryptographic hash function that contains a trapdoor. It is hard to find a collision except if the trapdoor is known, indeed knowing the trapdoor it is possible to efficiently generate collisions.

More precisely a chameleon hash function is a tuple of PPT algorithms $CH = (HGen, Hash, HVer, HCol)$ specified as follows.

- $(hk, tk) \leftarrow HGen(1^\lambda)$: The probabilistic key generation algorithm $HGen$ takes as input the security parameter $\lambda \in \mathbb{N}$, and outputs a public hash key hk and a secret trapdoor key tk.
- $(h, \nu) \leftarrow Hash(hk, m)$: The probabilistic hashing algorithm $Hash$ takes as input the hash key hk, a message $m \in M$, and outputs a pair (h, ν) that consists of the hash value h and a check string ν.
- $d = HVer(hk, m, (h, \nu))$: The deterministic verification algorithm $HVer$ takes as input a message $m \in M$, a candidate hash value h, and a check string ν, and returns a bit d that equals 1 if (h, ν) is a valid hash/check pair for the message m (otherwise d equals 0).
- $\nu' \leftarrow HCol(tk, (h, m, \nu), m')$: The probabilistic collision finding algorithm $HCol$ takes as input the trapdoor key tk, a valid tuple (h, m, ν), and a new message $m' \in M$, and returns a new check string ν' such that $HVer(hk, m, (h, \nu)) = HVer(hk, m', (h, \nu')) = 1$. If (h, ν) is not a valid hash/check pair for message m then the algorithm returns $\perp$.

A chameleon hash function $CH = (HGen, Hash, HVer, HCol)$ with message space M satisfies correctness if there exists a negligible function μ for all $m \in M$ such that $\Pr\left[HVer(hk, m, (h, \nu)) = 1 : (h, \nu) \leftarrow Hash(hk, m); (hk, tk) \leftarrow HGen(1^\lambda) \right] \geq 1 - \mu(\lambda)$.

Ateniese *et al.* introduced a stronger definition of collision resistance stating that CH is collision-resistant if the following holds: no PPT algorithm $\mathcal{A}$, given the public hash key hk, can find two pairs (m, ν) and (m', ν') that are valid under hk and such that $m \neq m'$, with all but a negligible probability and this must hold even when $\mathcal{A}$ can see an arbitrary number of collisions generated using the trapdoor key tk corresponding to hk.

3.3. Permissioned Smart-Contract-Enabled Blockchains in a Nutshell

To formalize blockchains we rely on the notation of [22]. A blockchain is a chain C of blocks, where each block has the form $B_i = (s, x, ctr)$. Intuitively, x is the set of transactions in B, s is the head of the block and ctr is a value used to make the chaining of the hash consistent. A block B is valid if a predicate $\mathsf{validblock}(B) = 1$. The function validblock is arbitrarily chosen by the parties of the consortium that maintains the blockchain.

The last published block is called the head of the chain, denoted by $Head(C)$. A chain C with a head $Head(C) = (s, x, ctr)$ can be extended to $C' = C||B'$ by attaching a (valid) block $B' = (s', x', ctr')$; the head of the new chain C' is $Head(C') = B'$. The function $len(C)$ denotes the length of a chain C (i.e., its number of blocks). For a chain C of length n and any $k \geq 0$, we denote by $C^{\lceil k}$ the chain resulting from removing the k rightmost blocks of C, and analogously we denote by $^{\rceil k}C$ the chain resulting in removing the k leftmost blocks of C. If C is a prefix of C' we write $C \prec C'$.

3.3.1. Handling Smart Contracts

Since we are considering smart-contract-enabled blockchains, we provide our formalization of a smart contract. Recall that each block $B = (s, x, ctr)$, and x is the set of transactions contained in B. To account for smart contracts, we consider the following three types of transactions: (i) standard transactions, (ii) smart contract transactions, and (iii) state transactions. A standard transaction tx is a tuple $(\mathtt{addr}_{tx}, \mathtt{data}, \mathtt{toaddr})$, where $\mathtt{addr}_{tx}$ is the address of the transaction, $\mathtt{data}$ is either raw data or smart contract's inputs as we specify later, and $\mathtt{toaddr}$ is either the address of the recipient smart contract or 0 when tx contains raw data.

A smart contract transaction contains a smart contract ϕ, that is composed of a tuple $(\mathtt{addr}_\phi, f_1, \ldots, f_\ell)$. We will sometimes refer to ϕ as a vector, therefore $\phi[i]$ corresponds to the i-th element of the tuple.

In ϕ, the value $\mathtt{addr}_\phi$ is the address of the smart contract, while $f_1, \ldots, f_\ell$ are functions. ϕ further initializes a state $\mathtt{st}_\phi^0$ containing the initial state of the internal variables of its functions $(f_1, \ldots, f_\ell)$. The state transaction contains the state $\mathtt{st}_\phi$ of some smart contract ϕ[2].

When a standard transaction $tx = (\mathtt{addr}_{tx}, \mathtt{data}, \mathtt{toaddr})$ is used as an input for $\phi = (\mathtt{addr}_\phi, f_1, \ldots, f_\ell)$, the field $\mathtt{toaddr}$ of tx matches with the $\mathtt{addr}_\phi$ field of ϕ and $\mathtt{data}$ will be a pair $(i, \mathtt{inp})$ where $\mathtt{inp}$ is the input to the function f_i inside ϕ, with $i \leq \ell$.

Let $MapTxSC(C, tx)$ be the function outputting the smart contract ϕ triggered by tx (i.e. ϕ inside C such that $\mathtt{toaddr} = \phi[1]$). $MapTxSC(C, tx)$ outputs $\perp$ if such smart contract is not found. Further, let $StHead(C, \phi)$ be a function outputting the index of the last state of ϕ inside C. Given ϕ, when a new transaction tx with $\mathtt{toaddr} = \mathtt{addr}_\phi$ and $\mathtt{data} = (i, \mathtt{inp})$ is accepted, the function f_i inside ϕ will create a new state $\mathtt{st}_\phi^m$ with $m = StHead(C, \phi) + 1$. This new state will be inserted in some state transaction to be published inside a new block of C.

Since we are indexing smart contracts independently from the smart contract transaction they belong to, we assume the existence of a function $MapState(C, \mathtt{st})$ mapping a smart contract state to its own state transaction together with the corresponding block, and a function $MapSC(C, \phi)$ mapping a smart contract to its own smart contract transaction together with the corresponding block.

4. The Redaction Technique of [2]

The idea of [2] is to set the hash function used to chain the different blocks in the blockchain, to be a chameleon hash function.

We borrow the same idea of [2], and stress that the idea of using a hash function to chain blocks in such a way that some form of consistency is ensured is used in most of the commonly known blockchains. Then, a consensus mechanism is built on top of this hash-based chaining mechanism. A validblock algorithm can be arbitrarily defined depending on the consensus mechanism used.

[2]The information matching $\mathtt{st}_\phi$ to ϕ can be the address $\mathtt{addr}_\phi$ of ϕ inserted into the state $\mathtt{st}_\phi$ during its creation.

In the solution of [2], only a set of trusted authorities can redact the blockchain. Here, we report the case of a single trusted authority. However, in [2] it is shown how to generalize it to a decentralized setting using a secure multi-party computation protocol.

Ateniese *et al.* [2] extend the block definition of [22] as $B = (s, x, ctr, (h, \nu))$, where the new component (h, ν) is the hash/check pair for a chameleon hash. The hash function is defined to be a chameleon hash $(HGen, Hash, HVer, HCol)$, and the validation predicate for a block is the one used for proof-of-work. Also, the extension of a blockchain is similar to the one described in Section 3.3. Given a chain C with head $Head(C) = (s, x, ctr, (h, \nu))$, the extension is obtained attaching a (valid) block $B' = (s', x', ctr', (h', \nu'))$ such that $s' = H(ctr, h)$.

Their algorithm used to redact the blockchain takes as input a set of block indices I and $|I|$ transaction sets $\{x'_i\}_{i \in I}$ that are going to replace the original transactions $\{x_i\}_{i \in I}$. For each block B_i, the algorithm generates a new collision ν'_i using the collision generation algorithm $HCol$ on input the trapdoor, the hash h_i included in the i-th block, s_i, the old collision value ν_i and the new set of transactions x'_i. Then, it generates a new block $B'_i = (s_i, x'_i, ctr_i, (h_i, \nu'_i))$ and outputs the modified chain C' with the old blocks B_i replaced with the new redacted blocks B'_i.

The formal description of the redaction algorithm[3] of [2] is reported in Algorithm 1.

Data: The chain C of length n, a set of block indices $I \subseteq [n]$, a set of values $\{x'_i\}_{i \in I}$, and the chameleon hash trapdoor key tk.
Result: The redacted blockchain C'
$C' \leftarrow C$;
Parse C' as $(B_1, \ldots, B_n)$;
for $i \in [n]$ **do**
  if $i \in I$ **then**
    Parse the i-th block of C' as $B_i = (s_i, x_i, ctr_i, (h_i, \nu_i))$;
    $\nu'_i \leftarrow HCol(tk, (h_i, s_i||x_i, \nu_i), (s_i||x'_i))$;
    $B'_i = (s_i, x'_i, ctr_i, (h_i, \nu'_i))$;
    $C' \leftarrow C'^{\lceil n-i+1}||B'_i||^{\rceil i}C'$;
  end
end

Algorithm 1: Redaction algorithm from [2].

The above approach is very efficient and can be easily put into practice. Unfortunately, when the blockchain enables computation of smart contract functions, we observe the following issue.

Let us assume that we have a smart contract ϕ stored in some block, and that a transaction tx appearing in a block B changes the state of ϕ from some $\mathtt{st}$ to $\mathtt{st}'$. Let $\tilde{tx}$ be a transaction inside another block B' changing the state of ϕ from $\mathtt{st}'$ to $\mathtt{st}''$. If the trusted authority redacts the block B by modifying tx, then the resulting state might be different from $\mathtt{st}'$ and in turn $\tilde{tx}$ might be invalid of anyway making updates that were not desired by the creator of that transaction, therefore leaving the smart contract in a state that is inconsistent with the logic of the transactions that were proposed and approved earlier, ending up to $\mathtt{st}''$.

To guarantee blockchain consistency (without leaving invalid transactions or transactions doing something undesired for their creators), the natural solution would require to detect all the correlated transactions appearing after the modified transaction, and then either modify or remove such transactions if invalid. As said in Section 1.2, such an approach might be extremely hard to implement and moreover might lead to a huge number of delete transactions related to ϕ.

[3] In [2] the authors further propose a solution for shrinking a blockchain that we do not report here due to its similarity with the above redaction algorithm.

5. Our Permissioned Smart-Contract-Enabled Blockchain Redaction

As we said, our solution can be used to improve the current state of the art in the case of redaction on permissioned smart-contract-enabled blockchains.

From now on, we rely on a little abuse of notation to ensure good readability. We will use the original definition of tx (i.e., $tx = (\texttt{addr}, \texttt{data}, \texttt{toaddr})$), where $\texttt{data}$ might be $(i, \texttt{inp})$, everywhere except when we feed tx as an input to a function. In such a case, we refer to the $\texttt{inp}$ field of tx.

Let us first consider a simplified version of the blockchain in which state transactions are not included in blocks, but all the states can be inferred by looking at the smart-contract transactions coupled with the standard transactions.

Let us assume that, at time t, we have a blockchain $C = (B_1, \ldots, B_t)$. Let us further assume that $B_i = \{s_i, x_i, ctr_i, (h_i, \nu_i)\}$, for some $i \in [t-2]$ contains a smart contract ϕ. Let $B_j = \{s_j, x_j, ctr_j, (h, \nu)\}$, for some $i < j < t$ be the block containing a transaction tx that updates the state of ϕ from $\texttt{st}_\phi^k$ to $\texttt{st}_\phi^{k+1}$, for some $k \in [StHead(C, \phi)]$. Let $\tilde{tx}$ be the transaction in a block B_ℓ, for some $j < \ell \leq t$ that updates the state of ϕ from $\texttt{st}_\phi^{k+1}$ to $\texttt{st}_\phi^{k+2}$.

In case of the redaction of block B_j, if the transaction tx is modified, it might affect the state $\texttt{st}_\phi^{k+1}$. In turn, it will also affect the consistency of transaction $\tilde{tx}$.

As already mentioned, our solution relies on a proof that $\tilde{tx}$ is consistent with $\texttt{st}_\phi^{k+1}$. This type of proof ensures that all the states subsequent to $\texttt{st}_\phi^{k+1}$ are consistent w.r.t. a previous version of the chain. We call this type of proof *proof-of-consistency*.

5.1. Proof of Consistency for Smart Contract States

Our proof-of-consistency can be implemented by relying on a SNARK for the Relation 1 described below.

The statement consists of the modified transaction $\tilde{tx}$, two states $\texttt{st}$ and $\texttt{st}''$, two functions f and f' (of the same smart contract ϕ), the head s of the block B to be redacted, and the chameleon hash h together with its public hash key hk. The witness, instead, contains the original transaction set x (before redaction) of the block B, the original transaction tx inside x, and a state $\texttt{st}'$. The relation shows that the function f, if triggered by the original transaction tx and the state $\texttt{st}$ contained in the statement, produces a state $\texttt{st}'$. $\texttt{st}'$ is part of the witness since it cannot be computed anymore due to the fact that the block containing $tx \in x$ will be redacted. The relation further shows that the transaction $\tilde{tx}$, if given as an input to f' together with the non-recomputable state $\texttt{st}'$, will consistently produce $\texttt{st}''$. Finally, to guarantee consistency with the block that will be redacted, the relation shows that the original x containing the transaction tx that will be later modified produces the same hash h that will appear later in the redacted block, i.e., the relation shows that the transaction set x, in the witness, together with s and the hash key hk, if given as input to $Hash$, produces a pair $(h, \cdot)$. During verification, the hash h to be placed in the statement will be taken from the redacted block. Relation 1 follows:

$$R = \left\{ ((\tilde{tx}, \texttt{st}, \texttt{st}'', f, f', s, hk, h), (x, tx, \texttt{st}')) \;\middle|\; \begin{array}{c} f(tx, \texttt{st}) = \texttt{st}' \wedge \\ f'(\tilde{tx}, \texttt{st}') = \texttt{st}'' \wedge \\ (h, \cdot) = Hash(hk, (s, x)) \end{array} \right\}. \tag{1}$$

If we pose ourselves in the settings described at the beginning of the section, then $\texttt{st} = \texttt{st}_\phi^k$, $\texttt{st}' = \texttt{st}_\phi^{k+1}$, $\texttt{st}'' = \texttt{st}_\phi^{k+2}$, and $x = x_i$. If a state transaction $\texttt{st}'$ updating $\texttt{st}$ through tx is stored inside the blockchain, we have two different cases depending on what the governance wishes to redact:

- If the governance wishes to redact both the state $\texttt{st}'$ and tx, the relation to be proven is the same as above.
- If the governance wishes to redact tx but $\texttt{st}'$ shall remain unchanged in the chain, the relation is the same as above except that $\texttt{st}'$ is part of the statement, not the witness.

In the following section, we present our extension of the redaction algorithm of Ateniese *et al.* with the integration of our proof-of-consistency.

5.2. Our New Redaction Algorithm

The new redaction algorithm is run by the governance of the blockchain, knowing which blocks with indexes in $I \subseteq [n]$ shall be redacted, together with the new sets of transactions x'_i for each $i \in I$. For each block B_i, parsed as $(s_i, x_i, ctr_i, (h_i, \nu_i))$, computes a new collision ν'_i using the trapdoor tk, h_i, x_i, the old collision ν_i and the new set of transaction x'_i related to the i-th block. Then, for each transaction tx_j in x_j to be modified with a different transaction tx'_j in x'_j, the algorithm identifies the smart contract ϕ triggered by tx_j and the state $\mathtt{st}_\phi^\ell$ that is a state update from $\mathtt{st}_\phi^{\ell-1}$ for some function $f \in \phi$. Then, finds for a transaction tx in some subsequent block triggering a function $f' \in \phi$ updating $\mathtt{st}_\phi^\ell$ to $\mathtt{st}_\phi^{\ell+1}$ and thus generates a proof-of-consistency showing that tx_j updates $\mathtt{st}_\phi^{\ell-1}$ to $\mathtt{st}_\phi^\ell$, tx updates $\mathtt{st}_\phi^\ell$ to $\mathtt{st}_\phi^{\ell+1}$, and the original transaction set x_j containing tx_j, if given as input to $Hash$, lead to the value h contained in both the original and the redacted i-th block. The redaction procedure is formally reported in Algorithm 2.

Data: The chain C of length n, a set of block indices $I \subseteq [n]$, a set of values $\{x'_i\}_{i \in I}$, and the chameleon hash trapdoor key tk.
Result: The redacted blockchain C' and auxiliary redaction information Π.
$C' \leftarrow C$;
$\Pi \leftarrow \{\}$;
Parse C' as $(B_1, \ldots, B_n)$;
for $i \in I$ **do**
  Parse B_i as $(s_i, x_i, ctr_i, (h_i, \nu_i))$;
  Parse x_i as $(tx_1, \ldots, tx_m)$;
  Parse x'_i as $(tx'_1, \ldots, tx'_m)$;
  $\nu'_i \leftarrow HCol(tk, (h_i, s_i || x_i, \nu_i), (s_i | x'_i))$;
  $B'_i = (s_i, x'_i, ctr_i, (h_i, \nu'_i))$;
  for $j \in [m]$ **do**
    if $tx_j \neq tx'_j$ **then**
      Let $\phi = MapTxSC(C', tx_j)$;
      if $\exists f \in \phi$ *s.t.* $\mathtt{st}_\phi^\ell = f(\mathtt{st}_\phi^{\ell-1}, tx_j)$ *for some* $\ell \in [StHead(C, \phi)]$ **then**
        for $k \in \{i+1, \ldots, n\}$ **do**
          if $\exists tx \in B_k$ *s.t.* $MapTxSC(C', tx) = \phi \wedge \exists f' \in \phi$ *s.t.* $\mathtt{st}_\phi^{\ell+1} = f'(\mathtt{st}_\phi^\ell, tx)$ **then**
            Generate a proof π for Relation 1 on input $((tx, \mathtt{st}_\phi^{\ell-1}, \mathtt{st}_\phi^{\ell+1}, f, f', s, hk, h_i), (x_i, tx_j, \mathtt{st}_\phi^\ell))$;
            $\Pi \leftarrow \Pi \cup \{\pi\}$;
            break;
          end
        end
      end
    end
    $C' \leftarrow C'^{\lceil n-i+1 || B'_i || \rceil^i} C'$;
  end
end

Algorithm 2: Redaction algorithm for a permissioned smart-contract-enabled blockchain.

When dealing with redacted blockchains, we introduce four informal properties that we believe are

the most significant in our context.

Chain Growth: Any chain (either redacted or not) can always be extended with a new (valid) block issued to the network.

Chain Consistency: Smart contract states must be consistent w.r.t. original data (i.e., existing data before redaction).

Privacy of Original Data: The auxiliary redaction information should not revel any significant information about original data that can not be inferred directly looking at the redacted blockchain.

Minimal Impact Redaction: The redaction procedure only modifies/deletes the minimal amount of information that is explicitly required to be redacted.

We argue below why the above properties are satisfied by our redaction technique:

Chain Growth: Due to the chameleon property of the hash functions, the hash value of the blockchain is never modified when transactions are redacted by the governance. Hence, any version of the chain, either redacted or not, will have the same hash value. Hence, the underlying consensus mechanism can be used as it is to extend any chain roaming around the network.

Chain Consistency: Chain consistency is guaranteed by the consistency of states of the smart contracts w.r.t their input transactions and the consistency between blockchain hash and data.

Regarding state consistency, when the chain is not redacted, nodes can re-run a smart contract from the standard transactions that are triggering such a smart contract and check whether all the intermediate states are consistent, and then check that the blockchain data matches the hash stored in the chain.

On the other hand, if the chain is redacted, nodes can verify that state consistency was guaranteed in the chain containing the original transactions thanks to the completeness of the proofs-of-consistency published by the governance as auxiliary redaction information.

The collision resistance property of the chameleon hash function and the knowledge soundness of the proofs-of-consistency guarantee that an adversary can forge blockchain data without knowing the trapdoor key (that is held by the trusted authority) only with negligible probability.

Privacy of Original Data: Thanks to the zero-knowledge property of SNARKs, the auxiliary redaction information does not disclose any additional information about the original data that could not already be inferred from the other still existing transactions.

Minimal-Impact Redaction: Our technique only deletes the interested information fixing the directly affected transactions, without causing changes to indirectly affected transactions within the blockchain.

6. Conclusions

In this paper, we have shown that the immutability of permissioned smart-contract-enabled blockchains can be bypassed. First, we showed how previous techniques are insufficient when applied to a stateful function deployed in a smart contract. Then, we showed how to use SNARKs in several interesting cases (e.g., when a redaction consists of removing a transaction) to perform the redaction of a transaction while maintaining consistency of other correlated transactions.

Further directions of great interest are: (i) extending the applicability of permissioned blockchain redaction to cases that are not resolved by our work, (ii) extending our approach to permissionless blockchains.

Acknowledgements. We thank Vincenzo Iovino for the insightful discussions we had about redaction techniques in blockchains. The first author received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program under project PICOCRYPT (grant agreement No. 101001283), and from the Spanish Government under projects PRODIGY (TED2021-132464B-I00) and ESPADA (PID2022-142290OB-I00). The last two projects are co-funded by European Union EIE, and NextGenerationEU/PRTR funds. The second author received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program under project PROCONTRA[4] (grant agreement No. 885666) and by the project PARTHENON (B53D23013000006), under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU. The third author received funding from project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU. The last author is member of the Gruppo Nazionale Calcolo Scientifico-Istituto Nazionale di Alta Matematica (GNCS-INdAM) and his research contribution on this work is financially supported under the National Recovery and Resilience Plan (NRRP), Mission 4, Component 2, Investment 1.1, Call for tender No. 104 published on 2.2.2022 by the Italian Ministry of University and Research (MUR), funded by the European Union – NextGenerationEU– Project Title "PARTHENON" – CUP D53D23008610006 - Grant Assignment Decree No. 959 adopted on June 30, 2023 by the Italian Ministry of Ministry of University and Research (MUR).

References

[1] R. Matzutt, J. Hiller, M. Henze, J. H. Ziegeldorf, D. Müllmann, O. Hohlfeld, K. Wehrle, A Quantitative Analysis of the Impact of Arbitrary Blockchain Content on Bitcoin, in: FC 2018, volume 10957 of *LNCS*, Springer, 2018, pp. 420–438.

[2] G. Ateniese, B. Magri, D. Venturi, E. R. Andrade, Redactable Blockchain - or - Rewriting History in Bitcoin and Friends, in: EuroS&P, IEEE, 2017, pp. 111–126.

[3] J. Ma, S. Xu, J. Ning, X. Huang, R. H. Deng, Redactable blockchain in decentralized setting, IEEE Transactions on Information Forensics and Security 17 (2022) 1227–1242.

[4] G. Zhou, X. Ding, H. Han, A. Zhu, Fine-grained redactable blockchain using trapdoor-hash, IEEE Internet of Things Journal 10 (2023) 21203–21216.

[5] D. Zhang, J. Le, X. Lei, T. Xiang, X. Liao, Secure redactable blockchain with dynamic support, IEEE Transactions on Dependable and Secure Computing (2023) 1–14.

[6] C. Peng, H. Xu, H. Liao, J. Tang, T. Tang, Redactable blockchain in the permissioned setting, in: Science of Cyber Security, Springer, 2023, pp. 460–477.

[7] X. Wu, X. Du, Q. Yang, N. Wang, W. Wang, Redactable consortium blockchain based on verifiable distributed chameleon hash functions, Journal of Parallel and Distributed Computing 183 (2024) 104777.

[8] Y. Dong, Y. Li, Y. Cheng, D. Yu, Redactable consortium blockchain with access control: Leveraging chameleon hash and multi-authority attribute-based encryption, High-Confidence Computing 4 (2024) 100168.

[9] V. Botta, V. Iovino, I. Visconti, Towards data redaction in bitcoin, IEEE Trans. Netw. Serv. Manag. 19 (2022) 3872–3883.

[10] I. Puddu, A. Dmitrienko, S. Capkun, μchain: How to Forget without Hard Forks, IACR ePrint (2017) 106.

[11] D. Deuber, B. Magri, S. A. K. Thyagarajan, Redactable blockchain in the permissionless setting, in: S&P 2019, IEEE, 2019, pp. 124–138.

[12] M. S. Dousti, A. Küpçü, Moderated Redactable Blockchains: A Definitional Framework with an Efficient Construct, in: ESORICS International Workshops, volume 12484 of *LNCS*, Springer, 2020, pp. 355–373.

[4] While working at the University of Warsaw.

[13] S. A. K. Thyagarajan, A. Bhat, B. Magri, D. Tschudi, A. Kate, Reparo: Publicly Verifiable Layer to Repair Blockchains, in: FC 2021, volume 12675 of *LNCS*, Springer, 2021, pp. 37–56.
[14] D. Grigoriev, V. Shpilrain, RSA and redactable blockchains, Int. J. Comput. Math. Comput. Syst. Theory 6 (2021) 1–6.
[15] W. Wang, J. Duan, L. Wang, X. Hu, H. Peng, Strongly synchronized redactable blockchain based on verifiable delay functions, IEEE Internet of Things Journal 10 (2023) 16778–16792.
[16] W. Dai, J. Liu, Y. Zhou, K.-K. R. Choo, X. Xie, D. Zou, H. Jin, Prbfpt: A practical redactable blockchain framework with a public trapdoor, IEEE Transactions on Information Forensics and Security 19 (2024) 2425–2437.
[17] R. Gennaro, C. Gentry, B. Parno, M. Raykova, Quadratic Span Programs and Succinct NIZKs without PCPs, in: EUROCRYPT 2013, volume 7881 of *LNCS*, Springer, 2013, pp. 626–645.
[18] G. Danezis, C. Fournet, J. Groth, M. Kohlweiss, Square span programs with applications to succinct NIZK arguments, in: ASIACRYPT 2014, volume 8873 of *LNCS*, Springer, 2014, pp. 532–550.
[19] J. Groth, On the size of pairing-based non-interactive arguments, in: EUROCRYPT 2016, volume 9666 of *LNCS*, Springer, 2016, pp. 305–326.
[20] A. Kothapalli, S. Setty, I. Tzialla, Nova: Recursive zero-knowledge arguments from folding schemes, in: CRYPTO 2022, Springer, 2022, pp. 359–388.
[21] H. Krawczyk, T. Rabin, Chameleon signatures, in: NDSS 2000, The Internet Society, 2000.
[22] J. A. Garay, A. Kiayias, N. Leonardos, The bitcoin backbone protocol: Analysis and applications, in: EUROCRYPT 2015, volume 9057 of *LNCS*, Springer, 2015, pp. 281–310.