How to Redact the Bitcoin Backbone Protocol

Mehmet Sabir Kiraz
De Montfort University
Leicester, UK
mehmet.kiraz@dmu.ac.uk

Enrique Larraia
nChain
Barcelona, Spain
e.larraia@nchain.com

Owen Vaughan
nChain
London, UK
o.vaughan@nchain.com

***Abstract*—We explain how to extend the Bitcoin backbone model of Garay et al. (Eurocrypt, 2015) to accommodate for redactable blockchains. Our extension captures fluid blockchain-based databases (with mutability requirements) and compliance with existing legislation, such as the GDPR right to be forgotten, or the need to erase offending data from nodes' databases that would otherwise provoke legal shutdowns. Our redactable backbone protocol retains the essential properties of blockchains. Leveraging zero-knowledge proofs, old data can be erased without requiring trusted third parties or heuristics about past chain validation. Our solution can be implemented on Bitcoin immediately without hard-forks, and it is scalable. It allows the redaction of data from UTXOs or unconfirmed transactions that have not yet flooded the network, while guaranteeing invariance of the Bitcoin state. Thus, offending data does not need to persist in the system, not even *temporarily*.**

I. Introduction

Public blockchains have in-built features of data integrity, transparency, verifiability, and a distributed network architecture. However, the same features that make the technology so attractive also pose a challenge for regulatory compliance. This is more important to large, risk-averse organisations who may not want to handle data that contains sensitive information without the ability to remove that data upon request [37], [54], [61]. This is enshrined in laws such as the European General Data Protection Regulation (GDPR) "right to be forgotten" [31], [47], [48]. It is the goal of this paper to make it easier to comply with this type of regulation without sacrificing the integrity of the underlying blockchain.

A Bitcoin transaction can be divided into payment data and non-payment data. Typical examples of non-payment data include image or text files. Such *validation-oblivious* data usually appears either in unspendable transaction outputs, or in unreachable script fragments of spendable outputs [44]. Consider a blockchain service provider that processes transactions on behalf of users. They give their users proof that their transactions have appeared on the blockchain[1]. If a user asks the service provider to remove their data from a transaction, then the service provider can no longer prove the integrity of the remaining fields in that transaction. The service provider is faced with the dilemma of either (1) keep the integrity of the transaction and do not remove user data, or (2) lose the integrity of the transaction and remove user data. Redactable blockchains aims to address this problem.

Redactable Blockchains: Ateniense, Magri, Venturi, and Andrade were the first to propose a blockchain with rewriting capabilities [2]. Their suggestion is to mine blocks using a chameleon hash (CH) [18]. The content of a block can be updated with the knowledge of the trapdoor key, which is secret-shared among a set of trustees. They can redact data by engaging in a multiparty protocol. CH-based redactions has spawned its own line of research [41], [42], [55], [56], [59], [60], [62], and more. For example, the authors of [42] use decentralized policy-based chameleon hashes to distribute redactions. In [36], the authors proposed another redactable blockchain scheme with a semi-trusted regulator who is assumed to follow the protocol.

Florian et al. in [26] propose the intriguing concept of *functionality-preserving* local erasures (FPLE) for Bitcoin. Offending or sensitive data is physically erased from storage or garbled, never again stored in a reconstructable form, and never shared with other nodes. To deal with validation, a set of pragmatic workarounds is proposed relying in SPV-like heuristics (e.g. if a transaction spending an output with erased data has been mined already, it is assumed the validation was correct) that effectively forces new joiners to trust past actions. Deuber et al. [24] introduce *blockchain-policy allowed* redactions. Miners decide with a PoW-based voting scheme if redactions are valid according to the policy (we note the miners need the original data to check policy compliance during the voting period).

For Ethereum, Puddu et al. present μchain [51]. It maintains encrypted versions of smart contract data, called "mutations", and only the unencrypted version is active. Upon user's request, miners can switch between versions via threshold decryption. Redactions via mutations may trigger a cascade of changes in other data: when a mutated transaction affects other transactions, they need to mutate all affected transactions. Manevich et al. [43] investigate redaction techniques for the execute-order-validate architecture of Hyperledger. Recently, [57] presented a new blockchain system called Strongly Synchronized Redactable Blockchain (SSRB) scheme which is based on verifiable delay functions (VDF).

More related to our work, Botta et. al. [16] propose redactions for Bitcoin using NIZKs to prove partial knowledge of SHA preimages. Their technique is quite similar to ours,

This is the full version of the paper. Differences with respect the conference version are: (i) expanded the review of the Bitcoin Backbone execution model in preliminaries, (ii) Bitcoin redactions (Section IV) has been extended to greater detail, (iii) we acknolwedge a related work, and (iv) we report benchmarks.

[1]This can be done efficiently with Simplified Payment Verification (SPV).

although we depart significantly from their work and improve on a number of aspects that we will ellaborate after detailing our contributions.

Summarising, most of the state of the art proposals for redactable blockchains are either in the permissioned setting, or require trusted parties or heuristics. More importantly, they also require a hard fork, which is not realistic for chains, especially Bitcoin and Ethereum (see Table I). In addition, many suffer from scalability issues, particularly those that are MPC-based or secret-share key material across miners.

A. Our Contributions and techniques

Our main contribution is an abstract framework to redact Bitcoin-like blockchains that allows nodes to store *different* validation-oblivious data locally. That is, in our framework GDPR-compliant mining nodes can coexist with nodes that insist in storing offending (again, validation-oblivious) data without breaking consensus or security at the application layer. We are the first to propose a novel solution for Bitcoin that is decentralised (miner-redactable), does not require trustees, heuristics, or hard-forks, and it is scalable. We use non-interactive zero-knowledge proofs (NIZKs) as building block, as in [16]. More concretely, our contributions are at the two levels of the Bitcoin blockchain.

a) Consensus level: We augment the Bitcoin backbone model of [29] to account for redactable blockchains. In our redactable model, the data set is partitioned into disjoint classes according to system-wide agreed-upon policies. In the augmented model, consensus pertains now to *classes* of blockchains, and consequently nodes are allowed to hold different data, or switch altogether from one blockchain representation to another representation of the same class. Let us emphasize once more that nodes can store locally different data and yet agree on the same history, namely, agree to the same blockchain *class*. Technically, we need to address two challenges:

- How can we maintain the existing proof of work? If a single bit of block data anywhere is altered, then a different block summary is produced. Thus, we cannot use the summary of the redacted block as it will invalidate the existing proof of work. We overcome this issue lifting the domain of the cryptographic hash G that is used to summarise blocks.
- How nodes that switch to a new representation of a blockchain class can erase the old representation from their local databases and yet check the redaction is compliant with the system-wide policy? In [26] this is achieved relying on an SPV-like heuristic, which undermines trustless validation in (full) mining nodes. We deal with block erasures leveraging NIZKs; if the setup of the NIZK is publicly-verifiable, no trust assumption is needed.

We provide a security analysis in the UC framework [19], with erasures, which fits perfectly with the model of the Bitcoin Backbone protocol [29]. The NIZK must be non-malleable. Indeed the non-malleability property ensures that from any redacted block proposed by (a possibly dishonest) mining node we are able to extract the unredacted block for which the proof of work was produced, even if the redacter has seen simulated proofs in the ideal process. If a SNARK is used, the size of the proof is (poly)logarithmic, sometimes constant, in the size of the redacted block.

b) Application level: We show how the redactable framework can be used in Bitcoin. Our policy $\mathsf{P}_{\bitcoin}$ only permits redaction of validation-oblivious data. This ensures the state of the blockchain remains unaltered, even if the redacted transaction is unconfirmed or part of the UTXO set. Compare this with [26] where this is not allowed at all, or with [24] where the unredacted transaction must persist in the system during the voting period. We achieve instant erasures of all types of transactions via enforcing transaction "templates" that ensures modifications only of *non-executable* portions of unlocking scripts. This also guarantees that no further changes in child transactions are required after the redaction happens, which fixes the "'cascade of changes" problem present in μchain [51]. For example, a valid redaction template has an unspendable output locked with script `OP_0 OP_RETURN` $\langle$`Data`$\rangle$. In analogy with regular expressions in matching patterns, the matched part or public pattern of the script above are the two opcodes, and the unmatched part or wildcard is the to-be-redacted `Data`. To guarantee a redaction is compliant with $\mathsf{P}_{\bitcoin}$ in the block erasure setting, were `Data` is kept private by the redacter node, we give two NIZKs to selectively prove *partial* equality to SHA256 preimages of *variable* length, that may be of independent interest.

B. Comparison with the work of Botta et. al. [16]

Botta Iovino and Visconti employ a similar technique to redact Bitcoin transactions using NIZKs. The main differences of their work and ours are the following.[2]

- We describe how to redact at the consensus layer of the Bitcoin backbone protocol, with or without privacy, by introducing the concept of "classes" of blockchains. Thus, unlike in [16] our approach is not hard-coded to Bitcoin, and works for any PoW-based blockchain.
- In [16] the security analysis is very sketchy and not formal. Importantly, they do not account for proof malleability. Thus, when simulating an honest proof it is conceivable that dishonest miners seeing such simulated proof can simulate proofs on their own. As commented in Section I-A we provide a formal security analysis in the UC framework provided the NIZK is simulation sound extractable.
- To redact Bitcoin spendable outputs, [16] needs to produce additional NIZKs (for consistency of the redacted locking script and its hash used internally by opcode `OP_CHECKSIG` to generate the signing message). We leverage `OP_CODESEPARATOR` to not produce any additional NIZK.

[2]In the conference version of this work, we missed comparing our work with [16]. We fill this gap in the full version.

Scheme	Setting	Redactable by	Decentralized	No hard-fork	Building block
[2]	Permissionless	Central Authority	✗	✗	CH
[51]	Permissioned	Central Authority	✗	✓	Encryption
[24]	Permissionless	Miners	✓	✗	PoW-voting
[26]	Permissionless	Miners	✓	✓	SPV heuristic
[36]	Permissioned	Central Authority	✗	✗	CH
[43]	Permissioned	Peers (Validators)	✗	✗	Unhashed data
[59]	Permissioned	Central Authority	✗	✗	Signatures, CH
[42]	Permissioned	Central Authority	✗	✗	DPCH
[55]	Permissioned	Central Authority	✗	✗	CH
[56]	Permissioned	Central Authority	✗	✗	Commitments, CH
[62]	Permissioned	Central Authority	✗	✗	CH
[57]	Permissioned	Central Authority	✗	✗	VDF
[16]	Permissionless	Miners	✓	✓	NIZK
Ours	Permissionless	Miners	✓	✓	NIZK

TABLE I: Comparison of redactable blockchains. (We see MPC or secret-shared based approaches as centralized because security relies on a fixed and small set of users.)

- Regarding proving partial equality to large SHA preimages. The approach of [16] is to reveal all the midstates of the original txid, and only verify redacted midstates. They claim this is a minimal leakage. We instead provide two approaches. A commit-and-prove approach for enforcing redactions in midstate individually, and, to reduce the number of midstate proofs, a recursive approach. Both of our approaches fully conceal the unredacted data. Thus, our approaches are applicable to other scenarios where partial equality to large SHA preimages is needed, whereas theirs is not truly zero-knowledge due to the leakage of the midstates.

II. Preliminaries

A. Backbone Protocol

The Backbone protocol Π was introduced in the seminal work of Garay, Kiayias, and Leonardos [29] to formally asses the security of Nakamoto's Bitcoin protocol [46].

Blockchain Abstraction: Blocks in [29] are modelled as tuples $\mathsf{B} := \langle ctr, s, x\rangle$, where $ctr \in \mathbb{N}$ is a nonce for mining, $s \in \{0,1\}^\kappa$ is a pointer to another block, and $x \in \{0,1\}^*$ is the actual block data. A blockchain of length ℓ is any ordered sequence $\mathsf{C} := \mathsf{B}_1, \ldots, \mathsf{B}_\ell$. The ordering is settled by requiring that for any two consecutive blocks B_{i-1}, B_i it holds $s_i = H(ctr_{i-1}, G(s_{i-1}, x_{i-1}))$, where H, G are two cryptographic hash functions $H, G : \{0,1\}^* \mapsto \{0,1\}^\kappa$, both with range size set to the security parameter κ. H is used to mine blocks, and G to summarize block data before mining it. This distinction between H and G will be important to us. The prefix chain resulting from pruning the k rightmost blocks of C is typically denoted with $\mathsf{C}^{\lceil k}$, and the notation $\mathsf{C}' \preceq \mathsf{C}$ indicates that $\mathsf{C}' = \mathsf{C}^{\lceil k}$ for some $k \leq \ell$.

Backbone Protocol: The Backbone protocol Π is application-agnostic and its goal is to achieve proof-of-work based consensus. It consists in a two-layer protocol.

- *Application layer*: Semantics at this layer are unspecified. The protocol is parametrized with three (unimplemented) functions. The input-contribution function $I(\cdot)$ essentially prepares new candidate blocks, the chain-reading function $R(\cdot)$ interprets the blockchain data x_{C}, and the content validation predicate $V(\cdot)$ decides whether x_{C} is correct according to the application running on top.
- *Consensus layer:* Π fully specifies three functions $\mathsf{validate}$, $\mathsf{maxvalid}$, pow that are used for chain validation, chain comparison and proof of work, respectively.

In this work we do not explicitly write out the functions (pow, $\mathsf{validate}$, $\mathsf{maxvalid}$) nor the description of Π, and refer to [30] for full details. A high-level description is as follows. The Backbone protocol Π is an infinite loop where at each step (round) miners fetch new content, either from the application layer or from other miners, and either broadcast fresh mined blocks or move on to the next round if a new valid block arrives from another node. For a given difficulty target T, and assuming up to q hash tries, a block $\mathsf{B} := \langle ctr, s, x\rangle$ is considered 'mined', i.e. it has proof of work if $H(ctr, G(s,x)) \leq \mathsf{T} \ \wedge \ ctr < q$. In this case, predicate $\mathsf{validblock}^{\mathsf{T}}_{\mathsf{q}}(\mathsf{B})$ is set to true. A blockchain is deemed valid if all its blocks have been mined, and blockchain data x_{C} is semantically valid according to $V(\cdot)$. The function $\mathsf{validate}$ from [29] is parametrized with the hash functions H, G and with the content validation predicate $V(\cdot)$. To actually mine blocks, the function pow iterates over the nonce ctr till it finds a hash (of H) below the current difficulty target T.[3]

Execution Model: The execution model for the backbone protocol Π provided in [29] is inspired by the UC framework [19]. A protocol is a collection of programs run by a set of parties which are captured as interactive Turing Machines (ITMs). In the Bitcoin context, the n main parties $(\mathcal{P}_i)_{i\leq n}$ of the protocol are also called nodes or 'miners'. The model also incorporates two special ITMs, the environment $\mathcal{Z}$ and the adversary $\mathcal{A}$. The interactions between all machines (possibly including 'suboroutine' machines different than $\mathcal{P}_i, \mathcal{Z}, \mathcal{A}$) are governed by a control function C that dictates who can communicate with: at the very least, only the environment $\mathcal{Z}$ can specify the inputs of the main parties $\mathcal{P}_i$, and at the end of the protocol, each $\mathcal{P}_i$ must send back its output to $\mathcal{Z}$. Also, C allows bilateral backdoor communication between the adversary $\mathcal{A}$ and $\mathcal{P}_i$, and between $\mathcal{A}$ and $\mathcal{Z}$. We stress that the role of C in 19 is not meant to model communication channels across the main parties $\mathcal{P}_i$, instead it sets up the communication rules of system of ITMs that later enable simulation-based security analysis. In [29], the control function C is further restricted to only allow a 'round-robin' interaction between

[3] In practice, the difficulty target is inversely proportional to the collective computational power of the miners, and it is adjusted periodically.

the main parties $\mathcal{P}_i$, and the adversary is assumed to corrupt $t' \leq t$ parties for a fixed threshold t hard-coded in C; the later models the honest-majority assumption necessary for proof-of-work based blockchains.

Resources as ideal functionalities: Communication across the parties $\mathcal{P}_i$ is abstracted away with an ideal 'diffuse' functionality $\mathcal{F}_{\mathsf{DIFF}}$ with a twofold purpose: it models a non-reliable broadcast channel, and structures the communication into sequential rounds; the adversary $\mathcal{A}$ can spoof and send inconsistent messages (for example, change the order of the messages that parties send or receive to launch partition attacks) but $\mathcal{A}$ cannot avoid deliveries between the parties. Mining (hashing) is modelled with an ideal random oracle functionality $\mathcal{F}_{\mathsf{RO}}$, defined in the natural way: $\mathcal{F}_{\mathsf{RO}}$ maintains internal tables for the query/responses pairs, where responses are sampled uniformly on fresh queries. In [29] one table per hash function H, G is maintained. In our case, $\mathcal{F}_{\mathsf{RO}}$ is *only used to model hash* H (and possibly other queries, e.g. related to NIZKs), but we assume a description (circuit) of the hash G used to summarise blocks is known. The number of queries issued to H to mine blocks are less than a fixed bound q per each honest party and round, and less than $t' \cdot q$ adversarial queries per round (if $\mathcal{A}$ corrupts t' parties at the given round). Both functionalities $\mathcal{F}_{\mathsf{DIFF}}$, $\mathcal{F}_{\mathsf{RO}}$ share state but for clarity of exposition it is more convenient to describe them separately. The Backbone protocol Π is assumed to have access to these two functionalities as subroutines.

Refinements: The model we have sketched this far is known as the q-bounded synchronous model. We refer the reader to [29], [30] for full details. Later [50] extend it to the semi-synchronous model where the adversary $\mathcal{A}$ can delay messages up to Δ rounds, and in [3] it was extended to account for adversaries that can *temporarily* corrupt a majority of miners. To phrase it in a more UC style, one can think that the security of Π is carried over 'the $(\mathcal{F}_{\mathsf{RO}}, \mathcal{F}_{\mathsf{DIFF}})$-hybrid model', Badertscher et al. [4] provide a thorough formalization of this intuition in the UC framework of Bitcoin when seen as a transaction ledger.

Property-based security: For a given security parameter κ, let the random variable $\mathsf{view}^{q,t,n}_{\Pi,\mathcal{A},\mathcal{Z}}(1^\kappa)$ that describes the joint view of all parties $(\mathcal{P}_i)_{i\leq n}$ when running the Backbone protocol Π with functionalities $\mathcal{F}_{\mathsf{RO}}$, $\mathcal{F}_{\mathsf{DIFF}}$, and $\mathcal{Z}$'s input fixed to 1^κ. Note that the joint view is a function of κ, $\mathcal{Z}$ and $\mathcal{A}$.

In [29] a property Q is a binary predicate over the joint view, and we say Q holds for Π if for all $\mathcal{Z}$ and $\mathcal{A}$ it holds

$$Pr[Q(\mathsf{view}^{q,t,n}_{\Pi,\mathcal{A},\mathcal{Z}}(1^\kappa)) = \mathsf{false}] \leq neg(\kappa).$$

We emphasize that as defined, joint views correspond to single standalone executions of Π, and the probabilities are taken over the random choices made by all the system ITMs (including the hybrid subroutines modelling resources).

Regardless of the actions of $\mathcal{A}$, three properties hold over Π (proved in [30]).

- $Q_{\mathsf{commonPrefix}}$: Any pair of local chains C_1, C_2, adopted by any pair of honest miners at (possibly distinct) rounds, share the same prefix. That is, it holds $\mathsf{C}_1 = \mathsf{C}_2^{\lceil k}$, for a given parameter k .
- $Q_{\mathsf{chainQuality}}$: It has parameters ℓ, μ. In any ℓ consecutive blocks of C, the ratio of honest blocks contributed by honest parties is at least μ.
- $Q_{\mathsf{chainGrowth}}$: It has parameters τ, s. After s rounds, the chain will be at least τs blocks longer.

Formal definitions of the properties informally stated above can also be found in [30].

B. Bitcoin Blockchain

In Bitcoin the chain reading function $R(\cdot)$ parses block data x into an ordered set of transactions $x := (\mathsf{tx}_1, \ldots, \mathsf{tx}_n)$ interpreted as the entries of a double-entry accounting system. Each transaction specifies a list of inputs (debit) and a list of outputs (credit). The credit of an output is *spent* if it is referenced as an input of a transaction appearing in a future block. Regarding validation $V(\cdot)$, besides enforcing no double spending, Bitcoin comes with a stack-based programming language in which spending conditions (puzzles) can be coded up in *locking* scripts associated to outputs. The inputs of a transaction, reference (spend) previous outputs, and also provide an *unlocking* script, with the arguments (puzzle solutions) needed to execute the locking script of the referenced output. The spending is allowed if the execution resolves to `true`. An output is *spendable* if the conditions specified in the locking script can be met.

a) Transaction Identifiers.: There exists several implementations of Bitcoin [9]–[11] differing in details such as transaction format, block size, or expressiveness of the spending conditions, but they all generate transaction identifiers by double hashing with `SHA256`. Our techniques can be applied to any of these implementations.

b) Arbitrary Content Data.: In Bitcoin, besides financial data, a transaction can also contain arbitrary content. There are several insertion methods: (1) in locking scripts after opcode `OP_RETURN`, (2) in unreachable conditional branches of locking scripts, (3) in coinbase inputs (only available to miners), or (4) in public keys, public key hashes and pay-to-script hashes. We will collectively denote the data inserted with the first two techniques as *non-executable locking script* (NELS) data. According to [44], a vast majority of non-financial data in the BTC network is `OP_RETURN` data [4]. It is reasonable to assume that insertion via NELS data will become the *de facto* strategy. The permitted size varies between blockchain implementations, but for some it can be very large. For example, BTC can store data up to 80 bytes [12] whilst BSV has recorded mainnet transactions of 42 MB [58].

[4] [44] reports that $86,8\%$ of BTC non-financial data is `OP_RETURN` data, as per 2017.

C. NIZKs

Non-Interactive Zero-Knowledge Arguments of Knowledge (NIZKs) were first introduced by Blum, Micali, and Fieldman in [13], [14]. Given an NP relation $\mathcal{R}$ and $\mathcal{L}$ its associated language. A (pre-processing) non-interactive zero-knowledge argument of knowledge (NIZK) for $\mathcal{R}$ is a triplet of algorithms $\Gamma := (\mathbf{S}, \mathbf{P}, \mathbf{V})$ defined as follows:

- $\mathbf{S}(1^\lambda, \mathcal{R}) \rightarrow (pk, vk)$ takes as input a security parameter λ and a description of relation R and it outputs a pair of keys pk, vk.
- $\mathbf{P}(pk, x, w) \rightarrow \pi$ takes the proving key pk, the public instance x and the private witness w as input and outputs a proof π.
- $\mathbf{V}(vk, x, \pi) \rightarrow \mathsf{b}$ takes the verification key vk, the public instance x, and the proof π as input, and outputs either $\mathsf{b} := \mathsf{true}$ (valid proof) or $\mathsf{b} := \mathsf{false}$ (invalid proof).

Γ is in the random oracle model if $\mathbf{S}$, $\mathbf{P}$, $\mathbf{V}$ are given oracle access to a random function (in our context, access to the $\mathcal{F}_{\mathsf{RO}}$ functionality). Informally, Γ is *complete* if $\mathbf{V}$ always accepts proofs π generated by the honest prover $\mathbf{P}$ on inputs $(x; w) \in \mathcal{R}$. It is *computationally sound* if no efficient cheating prover $\mathbf{P}^*$ can produce a proof π for a false statement $x \notin \mathcal{L}$. It is *zero-knowledge* if no information about the witness w is leaked from the proof π; formalised by requiring the existence of a simulator $\mathcal{S}$ that can forge proofs for fake statements $x \notin \mathcal{L}$, or for valid statements for which no witness is known. $\mathcal{S}$ has some extra power: in the common reference string (CRS) model it has access to a simulation trapdoor implicit in $crs := (pk, vk)$, and in the random oracle model it has the ability to program $\mathcal{F}_{\mathsf{RO}}$.

Simulation Sound Extractability (SSE): Γ is adaptive *knowledge sound* if for any efficient algorithm $\mathbf{P}^*(crs)$ that produces a valid proof for x we can extract a witness to the statement $x \in \mathcal{L}$. A stronger notion capturing NIZKs that are non-malleable is *simulation sound extractability* [52], [53], and it requires the above to hold even if $\mathbf{P}^*$ sees simulated proofs produced by the zero-knowledge simulator $\mathcal{S}$. In this work, we consider *black-box* SSE-NIZKs. Thus, we assume the existence of an efficient extractor that works for all efficient cheating provers. It is known that BB-SSE is necessary to construct UC-secure NIZKs [20], [33], [35].

zkSNARKs: If the NIZK is computational knowledge-sound and the size of the proof is $|\pi| = \mathcal{O}_\lambda(1)$, then it is a succinct non-interactive argument of knowledge [8]. There has been an explosion in the design of SNARKs partially fuelled by applications in cryptocurrencies [6], [15]. The constructions are either in the standard (CRS) model or in ROM. See for example [7], [21], [27], [32], [49], to name just a few. In [34], Groth and Maller construct white-box SSE SNARKs, however, it is usually believed that black-box SSE SNARKs cannot be attained in the standard model. The intuition is that a succinct proof cannot contain enough information about a witness, and hence the extractor should be hard-coded to a concrete $\mathbf{P}^*$. If the succinctness property is relaxed so that the size of π can depend quasi-linearly in the size of w (but still sublinear in the size of the circuit describing $\mathcal{R}$) then BB-SSE for SNARKs is possible [1], [38]. Recently, Ganesh et al. construct witness-succinct UC-secure SNARKs in the random oracle model [28] from succinct polynomial commitment schemes and using the Fischlin transform [25] to avoid rewinds.

III. Redactable Backbone Protocol

A. The extended model for redactable blockchains

Policies: We extend the backbone protocol to redactable blockchains by partitioning the set of possible blockchains $\mathcal{C}$ into disjoint classes according to some policy P. We formalize the notion of policy through an equivalence relation over $\{0,1\}^*$ that is efficiently testable.

Definition 1 (P-equivalent data, blocks, and chains): Let $\sim$ be an equivalence relation (reflexive, symmetric, transitive) over $\{0,1\}^*$ and a probabilistic polynomial-time (PPT) algorithm P such that $x \sim x^*$ if and only if exists $\sigma \in \{0,1\}^*$ such that $\mathsf{P}(x, x^*, \sigma) = 1$. Here σ is auxiliary information, possibly set to empty. The class $[x]$ is the set of all x^* related to x.

- Two blocks $\mathsf{B} := \langle ctr, s, x \rangle$, $\mathsf{B}^* := \langle ctr, s, x^* \rangle$ are equivalent if $x \sim x^*$. The class $[\mathsf{B}]$ is the set of all B^* related to B.
- Two chains of the same length $\mathsf{C} := (\mathsf{B}_1, \ldots, \mathsf{B}_\ell)$, $\mathsf{C}^* := (\mathsf{B}_1^*, \ldots, \mathsf{B}_\ell^*)$ are equivalent, if $\mathsf{B}_i \sim \mathsf{B}_i^*$ for $i = 1, \ldots, \ell$. The class $[\mathsf{C}]$ is the set of all C^* related to x.

In the redactable model, we are interested in the system agreeing in the same blockchain class $[\mathsf{C}]$ of the quotient space $\mathcal{C}/\sim$. In particular, two nodes may store locally two different blockchain representations $\mathsf{C}', \mathsf{C}'' \in [\mathsf{C}]$, or at some point in time, all nodes may switch from the first representation C' to the second one C''.

Redactions via policies may render validation at the application layer of the backbone protocol invalid. Although this is acceptable, it requires changes in the logic of $V(\cdot)$. If this is not the case, we say the policy P is compatible with $V(\cdot)$. Recall that x_{C} denotes the data held in blockchain C.

Definition 2 (V-compatible policies): A policy P as in Defn. 1 is *compatible* with a validation-content function $V(\cdot)$ if, for any two related blockchains $\mathsf{C}_1 \sim_{\mathsf{P}} \mathsf{C}_2$, we have that $V(x_{\mathsf{C}_1}) = \mathsf{true} \Leftrightarrow V(x_{\mathsf{C}_2}) = \mathsf{true}$.

Importantly, V-compatible policies guarantee that after redacting into block B^*, a cascade of changes in other blocks is *not* triggered. Our policy for Bitcoin in Section IV is compatible with the existing content validation logic (i.e. with Script execution).

Redactable Hashes: Data recorded in a redactable blockchain is the class $[x]$. Therefore, we need to lift the domain of the cryptographic hash that is used to summarise block data. Given a cryptographic hash G, define the hash of $(s, [x])$ as the *set* of all hashes of bitstrings x^* related to x. This mapping is well-defined over domain $\mathcal{D} := \{0,1\}^\kappa \times \{0,1\}^*/\sim$ and it is hard to find collisions over $\mathcal{D}$.

Lemma 1: Let an equivalence relation $\sim$ over $\{0,1\}^*$, and let a cryptographic hash $G : \{0,1\}^\kappa \times \{0,1\}^* \rightarrow \{0,1\}^\kappa$.

The function $\tilde{G}$ with domain $\{0,1\}^\kappa \times \{0,1\}^*/\sim$ given by $\tilde{G}(s,[x]) := \{g^* \mid \exists x^* \sim x \text{ s.t. } g^* = G(s,x^*)\}$ has the following properties:

(i) $\tilde{G}$ is well defined. That is, $\tilde{G}(s,[x_1]) = \tilde{G}(s,[x_2])$ for all $x_1 \sim x_2$.
(ii) $\tilde{G}$ is collision resistant in the following sense: for any $s \in \{0,1\}^\kappa$ and $x_1 \not\sim x_2 \in \{0,1\}^*/\sim$, it is unfeasible to find $g \in \tilde{G}(s,[x_1]) \cap \tilde{G}(s,[x_2])$.
(iii) $\tilde{G}$ is efficiently computable. Thus, $G(s,x) \in \tilde{G}(s,[x])$.

Proof: We start proving that $\tilde{G}$ is well-defined. Let $x_1 \sim x_2$, and $g^* \in \tilde{G}(s,[x_1])$. There exists $x_1^* \sim x_1$ such that $g^* = G(s,x_1^*)$. By transitivity $x_1^* \sim x_2$, and therefore $g^* \in \tilde{G}(s,[x_2])$. This shows the '$\subseteq$' direction. The other direction '$\supseteq$' is argued similarly and using the symmetry of $\sim$.

Collision resistance is straightforward. If there exists a collision finder $\tilde{\mathcal{A}}$ that outputs s, $x_1 \not\sim x_2$ with $G(s,x_1) = G(s,x_2)$, then $\tilde{\mathcal{A}}$ also finds collisions for G.

Last, using the reflexive property of $\sim$ we have that $g := G(s,x) \in \tilde{G}(s,[x])$. ∎

Redacting Data and Proof of Work: We want to mine class blocks $[B = \langle ctr, s, x\rangle]$ only once, and not every time a new representation $x^* \sim x$ of the data comes in. We reconcile these two apparent conflicting requirements by deeming a redacted block $\mathsf{B} := \langle ctr, s, x^*\rangle$ mined if there exists proof of work for an old representation $\mathsf{B} \in [\mathsf{B}^*]$. More concretely, we set predicate $\mathsf{validblock}_{\mathsf{q}}^{\mathsf{T}}(\mathsf{B}^*)$ to true if:

$$\exists x \in [x^*] \text{ s.t. } H(ctr, G(s,x)) < \mathsf{T} \wedge (ctr \leq q),$$

where q is a bound on the number of queries to $\mathcal{F}_{\mathsf{RO}}$. To be able to evaluate the above predicate we use an ideal functionality $\mathcal{F}_{\mathsf{redact}}$ that allows miners to validate whether or not a hash g is in the set $\tilde{G}(s,[x^*])$. The functionality also allows miners to register unredacted data $x \sim x^*$ with other miners under a handle τ. Since we want to reuse proof-of-work, the hash $g = G(s,x)$ is always given away after redaction, however, registering x can potentially leak more information about x (e.g. register by broadcasting handle $\tau := x$). To capture this, $\mathcal{F}_{\mathsf{redact}}$ is parametrized with a leakage function $\mathcal{L}$ such that $\mathcal{L}(x,\sigma)$ is always given to the ideal adversary $\mathcal{S}$. See Figure 1 for the full description of commands registerRedaction, and validateRedaction of $\mathcal{F}_{\mathsf{redact}}$.

Functions of the Redactable Backbone Protocol: The functions $\mathsf{maxvalid}^*$ and pow^* of Π^* are exactly the same as in the standard backbone protocol Π. The syntax of function $\mathsf{validate}^*$ is slightly different than validate from [29]. The changes are only related to setting the right value of g^* (either $G(s,x^*)$ or old mined digest g).

We do not specify how the handle τ and the old mined digest g are transmitted between peers, nor where handlers are stored at the receiving node. We just assume they are part of the internal state st of $\mathcal{P}_i$ at the moment of validating. In practice, the old digest g can be retrieved from the block headers, and the handler τ can be encoded as part of the redacted block data[5], or sent separately by other means. Thus, in our redactable model with $\mathcal{F}_{\mathsf{redact}}$, block validity is implemented as follows:

Functionality $\mathcal{F}_{\mathsf{redact}}^{G,\mathcal{L}}$

The functionality is parametrized by a cryptographic hash G and a leakage function $\mathcal{L}$.

Register redactions:
Upon receiving $(\mathsf{registerRedaction}, x^*, x, \sigma, s)$ from miner $\mathcal{P}_i$.

- Do nothing if $P(x^*, x, \sigma) \neq 1$. (Thus, if $x \not\sim x^*$.)
- Else:
  - Send $(\mathsf{leak}, x^*, G(s,x), \mathcal{L}(x,\sigma))$ to adversary $\mathcal{S}$ and wait for response τ.
  - Store (x^*, s, x, τ)
  - Output τ to $\mathcal{P}_i$

Validate redactions:
Upon receiving $(\mathsf{validateRedaction}, x^*, s, g, \tau)$ from miner $\mathcal{P}_i$:

- If $(x^*, s, \star, \tau)$ not stored send $(\mathsf{unredactedData}, x^*, s, g, \tau)$ to $\mathcal{S}$ and wait for response (x, σ). If $P(x^*, x, \sigma) = 1$ and $g = G(s,x)$, store (x^*, s, x, τ).
- If $(x^*, s, \star, \tau)$ is stored, output true to $\mathcal{P}_i$, else output false.

Fig. 1: Ideal functionality to register and validate redactions.

Predicate $\mathsf{validblock}_{\mathsf{q}}^{\mathsf{T}}(\mathsf{B}, \mathsf{st})$:

1: $\langle ctr, s, x^*\rangle \leftarrow \mathsf{B}$
2: $(g, \tau) \leftarrow \mathsf{getUnredactedBlockSummary}(x^*, \mathsf{st})$
3: **if** $(g, \tau) = \emptyset$ **then**
4: $\quad g^* \leftarrow G(s, x^*)$
5: $\quad \mathsf{validRedaction} \leftarrow \mathsf{true}$
6: **else**
7: $\quad g^* \leftarrow g$
8: $\quad \mathsf{validRedaction} \leftarrow \mathcal{F}_{\mathsf{redact}}^{G,\mathcal{L}}(\mathsf{validateRedaction}, x^*, s, g, \tau)$
9: **end if**
10: $\mathsf{validPoW} \leftarrow (H(ctr, g^*) \overset{?}{<} \mathsf{T}) \wedge (ctr \overset{?}{<} q)$
11: **return** $(\mathsf{validPoW} \wedge \mathsf{validRedaction})$

We emphasize that since in the redactable model we deal with *class* blocks $[\mathsf{B}]$, beyond the restatement of predicate $\mathsf{validblock}_{\mathsf{q}}^{\mathsf{T}}$, the function validate from [29] and our own $\mathsf{validate}^*$ have the same semantics.

B. Properties in the Redactable Model

The notion of chain prefix is extended to the quotient $\mathcal{C}/\sim$ in the natural way. Thus, $[\mathsf{C}]^{\lceil k}$ denotes the chain class resulting from pruning the k rightmost class blocks from $[\mathsf{C}]$. We write $[\mathsf{C}_1] \preceq [\mathsf{C}_2]$ if $[\mathsf{C}_1]$ is a prefix of $[\mathsf{C}_2]$ in the above sense. The common prefix property $Q_{\mathsf{commonPrefix}}$ for Π^* is defined using this partial ordering in $\mathcal{C}/\sim$.

Similarly, we can restate properties $Q_{\mathsf{chainQuality}}$ and $Q_{\mathsf{chainGrowth}}$. For chain quality, what is demanded is that a fraction of *class* blocks are honest; therefore, in ℓ consecutive blocks, the adversary $\mathcal{A}$ is allowed to contribute with a fraction larger than μ as long as the original data and the adversarial data are related through $\sim$. Note that the redaction can even happen in the node that receives or compiles the block data in the first place; in this sense, it is responsibility of the agreed-upon policy to decide what is fine to 'censor' (i.e. to redact) in the network and what not.

[5]For example, in Bitcoin one can embed τ in an unspendable UTXO of a transaction of x^*.

Following [29], we consider the redactable Backbone protocol Π^* in the $(\mathcal{F}_{\mathsf{redact}}, \mathcal{F}_{\mathsf{RO}}, \mathcal{F}_{\mathsf{DIFF}})$-hybrid model, where recall $\mathcal{F}_{\mathsf{RO}}$ models calls to the hash H and $\mathcal{F}_{\mathsf{DIFF}}$ models network communication.

Now, let $\{\mathsf{view}^{q,t,n}_{\Pi^*,\mathcal{A},\mathcal{Z}}(1^\kappa)\}_{\kappa\in\mathbb{N}}$ be the random variable ensemble that describes the joint view of the nodes when running the hybrid redactable backbone protocol Π^*. Recall from Section II that property Q holds for Π^* if for all environments $\mathcal{Z}$ and adversaries $\mathcal{A}$ we have $Pr[Q(\mathsf{view}^{q,t,n}_{\Pi^*,\mathcal{A},\mathcal{Z}}(1^\kappa)) = \mathsf{false}] \leq neg(\kappa)$.

Theorem 1: The properties $Q_{\mathsf{commonPrefix}}$, $Q_{\mathsf{chainQuality}}$, $Q_{\mathsf{chainGrowth}}$ hold for runs of the redactable backbone protocol Π^* in the $(\mathcal{F}_{\mathsf{redact}}, \mathcal{F}_{\mathsf{RO}}, \mathcal{F}_{\mathsf{DIFF}})$-hybrid model.

Proof (sketch): The proof strategy in [29] crucially relies on the properties of '*typical executions*' of the Backbone protocol Π, which in turn rely on the properties of the random oracle $\mathcal{F}_{\mathsf{RO}}$ modelling H and G (cf. [30, Defn. 9, Thm. 10]). The same proof strategy can be extended to the redactable Backbone protocol Π^* with the natural restatements of block insertion, block copy and block prediction for blocks $\mathsf{B}, \mathsf{B}^*, \mathsf{B}'$ [30, Defn. 8] for class blocks $[\mathsf{B}], [\mathsf{B}^*], [\mathsf{B}']$, and leveraging the collision resistance of the redactable hash $\tilde{G}$ shown in Lemma 1. ∎

C. Implementing $\mathcal{F}_{\mathsf{redact}}$ with Block Erasures

$\mathcal{F}_{\mathsf{redact}}$ can be trivially implemented if the redacter leaks the original (unredacted) block data x —i.e broadcasts handler $\tau := x$ along with redaction x^*. To validate x^*, miners simply check that $x^* \sim x$ and $g = G(s, x)$. However, this approach is problematic if x contains offending data or conflicts with privacy regulations. We observe that the node preparing a redaction x^* can also prove in zero-knowledge the existence of x such that $x^* \sim x$. This allows to remove x from validating nodes, as we shall see.

Recall we are assuming $\sim$ is an equivalence relation efficiently testable with policy algorithm P (cf. Defn. 1).

Definition 3 (Redaction language): Consider the following NP relation parametrized by a cryptographic hash G and policy P:

$$\mathcal{R}_{\mathsf{red}} := \left\{ ((x^*, \sigma, s, g); x) \;\middle|\; \begin{matrix} g = G(s, x) \\ \mathsf{P}(x, x^*, \sigma) = 1 \end{matrix} \right\}.$$

The associated language $\mathcal{L}_{\mathsf{red}}$ is any redacted block data $x^* \in [x]$, the old pointer s, and the digest g; the witness is the original block data x for which the digest g was computed.

Let $\Gamma_{\mathsf{red}} := (\mathbf{S}_{\mathsf{red}}, \mathbf{P}_{\mathsf{red}}, \mathbf{V}_{\mathsf{red}})$ be a NIZK scheme for $\mathcal{R}_{\mathsf{red}}$. The redactor node can produce a proof π to the statement "$(x^*, \sigma, s, g) \in \mathcal{L}_{\mathsf{red}}$" with $\mathbf{P}_{\mathsf{red}}$, and broadcast $\mathsf{B}^* := \langle ctr, s, x^* \rangle$ to the system along with the tuple π, σ, g. The other nodes, after verifying with $\mathbf{V}_{\mathsf{red}}$ the validity of π, replace $\mathsf{B} := \langle ctr, s, x \rangle$ with B^* in their local databases. Importantly, nodes that have not seen the original block B and only receive $\mathsf{B}^*, \sigma, g, \pi$, for example new joiners, will also be convinced that the redaction B^* adheres to the agreed policy P, and validate the proof of work by themselves using g, without relying on SPV heuristics or on trusted third parties.

Function $\Pi^{G,\Gamma_{\mathsf{red}}}_{\mathsf{redact}}$

The function is parametrized with a cryptographic hash G, and NIZK Γ_{red}. The function makes calls to an ideal functionality $\mathcal{F}^{\lambda,\Gamma_{\mathsf{red}}}_{\mathsf{KEYGEN}}$ to get the NIZK keys.

On input (cmd, q) **do the following:**

1: **if** $\mathsf{cmd} = \mathsf{registerRedaction}$ **then**
2: $(x^*, x, \sigma, s) \leftarrow q$
3: $g \leftarrow G(s, x)$
4: $pk \leftarrow \mathcal{F}^{\lambda,\Gamma_{\mathsf{red}}}_{\mathsf{KEYGEN}}(\mathsf{provingkey})$
5: $\pi \leftarrow \mathbf{P}_{\mathsf{red}}(pk, (x^*, \sigma, s, g), x)$
6: Erase x and all internal randomness used in $\mathbf{P}_{\mathsf{red}}$
7: **return** $\tau := (\sigma, \pi)$ ▷ Handle set to nizk π and auxiliary information σ
8: **end if**
9: **if** $\mathsf{cmd} = \mathsf{validateRedaction}$ **then**
10: $(x^*, s, g, \tau := (\sigma, \pi)) \leftarrow q$
11: $vk \leftarrow \mathcal{F}^{\lambda,\Gamma_{\mathsf{red}}}_{\mathsf{KEYGEN}}(\mathsf{verificationkey})$
12: **return** $(\mathsf{true} \stackrel{?}{=} \mathbf{V}_{\mathsf{red}}(vk, (x^*, \sigma, s, g), \pi))$
13: **end if**

Fig. 2: The function used to register and validate redactions $x^* \sim x$ with block erasures.

The Redact Function: We assume the keys of Γ_{red} are generated as prescribed by the setup algorithm. This assumption is realistic if Γ_{red} does not have a trusted setup, as in this case one can publicly verify that the keys are correct. Otherwise, the trust posed on the keys generation can be mitigated if the scheme is updatable —essentially, by running a multiparty computation protocol to generate a new key pair deemed secure. The ideal functionality $\mathcal{F}^{\lambda,\Gamma_{\mathsf{red}}}_{\mathsf{KEYGEN}}$ gives access to honestly generated keys. On initialization runs $(pk, vk) \leftarrow \mathbf{S}_{\mathsf{red}}(1^\lambda, \mathcal{R}_{\mathsf{red}})$. Then, on query $\mathsf{provingkey}$ returns pk, and on query $\mathsf{verificationkey}$ returns vk.

In Figure 2 we define the function Π_{redact} that uses a NIZK scheme Γ_{red} for $\mathcal{R}_{\mathsf{red}}$ with access to the verification key vk via $\mathcal{F}^{\lambda,\Gamma_{\mathsf{red}}}_{\mathsf{KEYGEN}}$. We have the following result.

Theorem 2: Let Γ_{red} be a NIZK for relation $\mathcal{R}_{\mathsf{red}}$ with black-box simulation sound extractability. Then, $\Pi^{G,\Gamma_{\mathsf{red}}}_{\mathsf{redact}}$ UC-realizes the functionality $\mathcal{F}^{G,\mathcal{L}}_{\mathsf{redact}}$ in the $\mathcal{F}^{\lambda,\Gamma_{\mathsf{red}}}_{\mathsf{KEYGEN}}$-hybrid world for leakage function $\mathcal{L}(x, \sigma) = \sigma$. Thus, for every real adversary $\mathcal{A}$ against $\Pi^{G,\Gamma_{\mathsf{red}}}_{\mathsf{redact}}$, there exists an ideal adversary $\mathcal{S}$ against $\mathcal{F}^{G,\mathcal{L}}_{\mathsf{redact}}$ such that it holds $\mathsf{exec}_{\Pi_{\mathsf{redact}},\mathcal{A},\mathcal{Z}}(1^\lambda) \approx \mathsf{exec}_{\mathcal{F}_{\mathsf{redact}},\mathcal{S},\mathcal{Z}}(1^\lambda)$.

Proof: In the UC experiment, the ideal execution involves an ideal adversary $\mathcal{S}$ that simulates a real execution towards its environment $\mathcal{Z}$. $\mathcal{S}$ is connected to $\mathcal{Z}$ and $\mathcal{F}_{\mathsf{redact}}$, but has no access to the dummy parties $\tilde{\mathcal{P}}_i$ (which simply relay messages between $\mathcal{Z}$ and $\mathcal{F}_{\mathsf{redact}}$). $\mathcal{S}$ runs Π_{redact} internally with a copy of the real adversary $\mathcal{A}$ and parties $\mathcal{P}_i$. If $\mathcal{A}$ corrupts $\mathcal{P}_i$, $\mathcal{S}$ tells $\mathcal{F}_{\mathsf{redact}}$ to corrupt dummy $\tilde{\mathcal{P}}_i$, and in response gets the internal state of $\tilde{\mathcal{P}}_i$; also $\mathcal{S}$ can specify the output of corrupted $\tilde{\mathcal{P}}_i$.

The ideal adversary $\mathcal{S}$ is defined as follows:

- When $\mathcal{S}$ receives $(\mathsf{leak}, x^*, g, \mathcal{L}(x, \sigma) := \sigma)$ from $\mathcal{F}_{\mathsf{redact}}$, it simulates a proof π for statement $(x^*, \sigma, s, g) \in \mathcal{L}_{\mathsf{red}}$. It sends handle $\tau := (\sigma, \pi)$ to $\mathcal{F}_{\mathsf{redact}}$.
- When $\mathcal{S}$ receives $(\mathsf{unredactedData}, x^*, s, g, \tau := (\sigma, \pi))$ from $\mathcal{F}_{\mathsf{redact}}$. If the proof π is not valid for statement

$(x^*, \sigma, s, g,)$, it sets $x := \perp$. Else, it extracts witness preimage x. $\mathcal{S}$ sends (x, σ) to $\mathcal{F}_{\mathsf{redact}}$.

- When $\mathcal{A}$ corrupts real party $\mathcal{P}_i$, $\mathcal{S}$ gets from $\mathcal{F}_{\mathsf{redact}}$ all registrations and validations queried by dummy party $\tilde{\mathcal{P}}_i$. $\mathcal{S}$ can simulate validations towards $\mathcal{A}$ by simply running $\mathbf{V}_{\mathsf{red}}$. To simulate registration queries, $\mathcal{S}$ only sends (simulated) proofs π to $\mathcal{A}$, but not the unredacted data x (which $\mathcal{S}$ does not have). This is allowed because in step (6) of Π_{redact}, the real party $\mathcal{P}_i$ has erased its internal tape where it supposedly runs $\mathbf{P}_{\mathsf{red}}$.

It is not difficult to see with a hybrid argument that the ideal execution involving $\mathcal{F}_{\mathsf{redact}}$ described above is indistinguishable from the real execution of Π_{redact}. The hybrid argument leverages black-box simulation extractability and is the same as the one used by Groth in [33] to implement the functionality $\mathcal{F}_{NIZK}$ in a model with erasures. We refer to Theorem 20 of the full version of [33] for details. ∎

On simulation soundness: To avoid malleability attacks where a cheating redaction node mauls proofs of non-compliant redactions we need the NIZK Γ_{red} to be simulation sound. Black-box SSE is necessary to argue composable UC-security [19] of Π_{redact} as in Thm. 2. However, the execution model of the backbone protocol is in the *standalone* setting (see Section II-A), and therefore it may be the case that white-box SSE, or just simulation soundness, suffices. This affects what schemes can be used. For example, the SNARK of [34] and variants of Groth16 [5] are white-box SSE, and do not need compilers [1], [28], [38] to lift to black-box SSE.

IV. Content Redaction in Bitcoin Transactions

A. The Redaction Policy

We will only allow redaction of non-executable portions of locking scripts. This guarantees that the state of the Bitcoin blockchain, such as the UTXO set, and the traceability of spent coins remains the same after a redaction happens.

Definition 4 (NELS data): Let $x := (\mathsf{tx}_1, \ldots, \mathsf{tx}_n) \in \{0,1\}^*$ be a set of Bitcoin transactions forming a block. We say that `Data` *forms part of the non-executable locking script data (NELS) of block* x if some output $\mathsf{tx}_i.\mathsf{out}_j \in \{0,1\}^*$ is of the form:

$$\mathsf{tx}_i.\mathsf{out}_j := \text{“}\texttt{OP\_0 OP\_RETURN } \langle \texttt{Data} \rangle \text{”},$$

or it has pattern:[6]

$$\mathsf{tx}_i.\mathsf{out}_j := \text{“}\texttt{ OP\_0 OP\_IF } \langle \texttt{Data} \rangle \texttt{ OP\_ENDIF OP\_CODESEPARATOR } *** \text{”}.$$

The first pattern is a standard unspendable UTXO. The second pattern can contain any succeeding script code (the *** part above); namely it is to redact spendable outputs. More complex patterns can be added to define NELS data. We say that two data blocks x, x^* are related in Bitcoin, if they have the same number n of transactions, and each of their (ordered) transactions have same non-NELS data. In simple words, two blocks are related if the *financial* data that miners use for validation is the same in both blocks.

Definition 5 (NELS-equivalent data): We say that two transactions tx, tx^* of the same length ℓ are related if for some subset $\sigma \subseteq \{1, \ldots, \ell\}$ it holds:

(i) The bits of tx^* at positions $\{1, \ldots, \ell\} \backslash \sigma$ form NELS data.
(ii) The bits of tx and tx^* at positions $\sigma \subseteq \{1, \ldots, \ell\}$ are equal.

We will write $\mathsf{P}_{\text{₿}}(\mathsf{tx}, \mathsf{tx}^*, \sigma) = 1$ if this is the case. Two data blocks $x := (\mathsf{tx}_1, \ldots, \mathsf{tx}_n)$, $x^* := (\mathsf{tx}_1^*, \ldots, \mathsf{tx}_n^*)$ with the same number of transactions are related if tx_i is related to tx_i^* for all $i \leq n$. With slight abuse of notation we will also write $\mathsf{P}_{\text{₿}}(x, x^*, \boldsymbol{\sigma}) = 1$ if this is the case, where $\boldsymbol{\sigma} = (\sigma_i)_{i=1}^n$.
It is not difficult to see that $\mathsf{P}_{\text{₿}}$ defines an efficiently testable equivalence relation (cf. Defn. 1) over the set of Bitcoin blocks.

Bitcoin Validation Compatibility: In the second pattern of NELS data (cf. Defn. 4) we add opcode `OP_CODESEPARATOR` to make sure that if the output has been spent already, and the succeeding script code involves a signature check (for example a P2PKH script), the signed message does not include the unredacted data `Data` [17] (see also footnote 6). With this trick we can redact `Data` without invalidating the signature already present in the spending transaction, and hence no further changes in other transactions are needed. Thus, $\mathsf{P}_{\text{₿}}$ is compatible with the Bitcoin content validation content $V_{\text{₿}}(\cdot)$ as per Defn. 2.

B. Redacting Bitcoin blocks

In Bitcoin, the hash G ignores the pointer s and computes the root of the Merkle tree whose leaves are the transaction IDs $(\mathsf{txid}_i)_{i=1}^n$. Thus $g := \mathsf{Merkle.getRoot}((\mathsf{tx}_i)_i)$

The partial equality relation $\mathcal{R}_{\mathsf{peq}}$: Say that a subset of r transactions $(\mathsf{tx}_{i_j})_{j=1}^r$ of a block B are redacted as transactions $(\mathsf{tx}_{i_j}^*)_{j=1}^r$. A redaction Bitcoin node needs to prove that they are related to the original transactions $(\mathsf{tx}_{i_j})_j$. He will do so by disclosing the original transaction identifier txid_{i_j} and proving partial equality of $\mathsf{tx}_{i_j}^*$ to preimage tx_{i_j} of txid_{i_j}. Thus, we consider the following relation:

$$\mathcal{R}_{\mathsf{peq}} := \left\{ ((\mathsf{tx}^*, \mathsf{txid}, \sigma); \mathsf{tx}) \;\middle|\; \begin{matrix} \mathsf{txid} = \mathsf{SHA256d}(\mathsf{tx}), \\ \mathsf{tx}^*[k] = \mathsf{tx}[k] \;\forall k \in \sigma \end{matrix} \right\}$$

where $\mathsf{SHA256d}$ denotes double hashing and $\mathsf{tx}[k]$ denotes the k-th bit of transaction tx. Observe that this covers item (ii) of policy $\mathsf{P}_{\text{₿}}$.

Parsing redacted blocks: We augment each redacted $\mathsf{tx}_{i_j}^*$ with an extra unspendable output containing the positions of the unchanged bits σ_{i_j}, the original transaction identifier txid_{i_j}, and the proof π_j for the partial equality statement. If the the original and redacted transactions $\mathsf{tx}_{i_j}, \tilde{\mathsf{tx}}_{i_j}^*$ have m outputs, the last output of the augmented redacted transaction $\tilde{\mathsf{tx}}_{i_j}^*$ is

[6] In the conference version, an incorrect pattern that would invalidate content validation was given: “‘*** `OP_RETURN` ⟨`Data`⟩ `OP_CODESEPARATOR`”. The unlocking signature of any preceeding P2PK/P2PKH script would be for a message that includes the unredacted data `Data`.

$$\tilde{\mathsf{tx}}^*_{i_j}.\mathsf{out}_{m+1} := \texttt{OP\_0 OP\_RETURN } \langle \mathsf{red}, \mathsf{txid}_{i_j}, \sigma_j, \pi_j \rangle,$$

where red is a flag that marks the transaction as redacted. The redacted block is $\mathsf{B}^* := \langle ctr, s, (\tilde{\mathsf{tx}}^*_i)_i \rangle$. The parser getUnredactedBlockSummary identifies the redacted transactions $\tilde{\mathsf{tx}}^*_i$ of B^* searching for flag red, and extracts $(\mathsf{txid}_{i_j}, \pi_j)$ from its last output. In case tx^*_i is not marked as redacted (thus $\mathsf{tx}^*_i = \mathsf{tx}_i$), it just sets $(\mathsf{txid}^*_i, \emptyset)$. Thus the parser outputs the (implicit) original block summary $g := (\mathsf{txid}_i)_i$, and handle τ set to the pair of vectors $\boldsymbol{\sigma} := (\sigma_i)_i$, $\boldsymbol{\pi} := (\pi_i)_i$. Note that the only information leaked about the original (unredacted) block data x are the redacted bit positions $\boldsymbol{\sigma}$.

Proving and verifying redactions in Bitcoin: Using any NIZK $\Gamma_{\mathsf{peq}} := (\mathbf{S}_{\mathsf{peq}}, \mathbf{P}_{\mathsf{peq}}, \mathbf{V}_{\mathsf{peq}})$ for the partial euqality relation $\mathcal{R}_{\mathsf{peq}}$ we can build a NIZK $\Gamma_{\mathsf{red}} := (\mathbf{S}_{\mathsf{red}}, \mathbf{P}_{\mathsf{red}}, \mathbf{V}_{\mathsf{red}})$ to prove and verify redactions in Bitcoin.

Prover $\mathbf{P}_{\mathsf{red}}$: On input an original block $\mathsf{B} := \langle ctr, s, (\tilde{\mathsf{tx}}_i)_i \rangle$ and redacted block $\mathsf{B}^* := \langle ctr, s, (\tilde{\mathsf{tx}}^*_i)_i \rangle$

1) for each redacted transaction pair $(\mathsf{tx}_{i_j}, \mathsf{tx}^*_{i_j})$ generate a proof for the partial equality statement $(\mathsf{tx}^*_{i_j}, \mathsf{txid}_{i_j}, \sigma_j; \mathsf{tx}_{i_j}) \in \mathcal{R}_{\mathsf{peq}}$:

$$\pi_j \leftarrow \mathbf{P}_{\mathsf{peq}}((\mathsf{tx}^*_{i_j}, \mathsf{txid}_{i_j}, \sigma_j), \mathsf{tx}^*_{i_j})$$

2) output $\pi := (\pi_j)_j$

Verifier $\mathbf{V}_{\mathsf{red}}$: On input a redacted block $\mathsf{B}^* := \langle ctr, s, (\tilde{\mathsf{tx}}^*_i)_i \rangle$ and proofs $\pi := (\pi_j)_j$

1) extract proof π_j, original txid_{i_j} and σ_j from augmented transaction $\tilde{\mathsf{tx}}^*_{i_j}$
2) check proof:

$$\mathsf{true} \stackrel{?}{=} \mathbf{V}_{\mathsf{peq}}((\mathsf{tx}^*_{i_j}, \mathsf{txid}_{i_j}, \sigma_j), \pi_j)$$

3) check item (i) of Defn. 4 explicitly using $\mathsf{tx}^*_{i_j}, \sigma_j$.

C. Partial equality to SHA256 preimages

SHA256 follows the Merkle-Damgard construction with a compression function $\mathsf{CF}_{\mathsf{sha}}$ over 512-bit words messages and 256-bit digests. We will start assuming that the size of the transaction that needs to be redacted is less than 512 bits, as in this case a single call to $\mathsf{CF}_{\mathsf{sha}}$ is needed[7] and our technique is easier to understand. Later, we will explain how to deal with larger preimages.

The main difficulty strives in the fact that Bitcoin transactions are customisable in the number of inputs, the number of outputs, and their script patterns. This means that one cannot predict in advance which substring of a serialised transaction will correspond to NELS data.

We design a circuit that takes σ as part of its public input. That is, a circuit that allows to prove *dynamically* what bits have been not been modified. If the transaction tx has ℓ bits, we will represent the subset σ of non-redacted bit positions as a vector of ℓ bits such that:

$$\sigma[k] = \begin{cases} 0 & \text{if } k\text{-th bit of tx is changed (redacted) in tx}^* \\ 1 & \text{if } k\text{-th bit of tx is not changed in tx}^* \end{cases}$$

This ℓ-bit vector σ acts as a " bit selector" between the bits of the original non-redacted tx and the new redacted $\tilde{\mathsf{tx}}$:

$$\mathsf{tx} = \sigma \cdot \tilde{\mathsf{tx}}^* + (\mathbf{1} - \sigma) \cdot \mathsf{tx}, \tag{1}$$

where above '$+$', '$-$', and '$\cdot$' denote component-wise addition, subtraction, and multiplication, respectively, and $\mathbf{1}$ is the ℓ-bit vector of ones. In other words, for preimages of sizes less than 512 bits the checks that need to be enforced in zero-knowledge are:

1) Check that $\mathbf{0} = (\tilde{\mathsf{tx}}^* - \mathsf{tx}) \cdot \sigma$
2) Check that $\mathsf{txid} = \mathsf{CF}_{\mathsf{sha}}(\mathsf{CF}_{\mathsf{sha}}(\mathsf{tx}))$

Above $\mathbf{0}$ is the ℓ-bit vectors of zeros. Check #1 is equivalent to equation (1) but slightly optimized, requiring four bit decompositions and two component-wise arithmetic operations (as opposed to equation (1) that requires five bit decompositions and three operations). The double call to $\mathsf{CF}_{\mathsf{sha}}$ is there because txid is the double hash of tx —In reality, we need to account for padding and work with 512-bit vectors, but this is an implementation detail that can be easily dealt with.

D. Dealing with large transactions

It is not possible to express $\mathcal{R}_{\mathsf{peq}}$ as a monolithic circuit if we want to account for variable transaction lengths: two transactions of different length would require different number of calls to the compression function $\mathsf{CF}_{\mathsf{sha}}$. We propose two different strategies to deal with arbitrarily large transactions yielding two different circuits. Each has its own advantages and disadvantages.

a) Approach #1: Commit to midstates: The idea is to commit in cm to the N midstates h_i of SHA256 and produce N proofs attesting to (i) correctness of the selector equation (1), for the Nbit i-th chunk σ_i of the selector vector σ, (ii) the correctness of the N calls of the compression function $\mathsf{CF}_{\mathsf{sha}}$ and (iii) the correctness of the commitments cm_{i-1}, cm_i to the input and output midstates h_i, h_{i-1}, respectively. We commit to midstates to not leak information about the original transaction tx.

Circuit $\mathsf{C}_{\mathsf{mid}}[\mathsf{ck}]$:

Public Input: $m^*_i, \mathsf{cm}_i, \mathsf{cm}_{i-1}, \sigma$
Private Input: $m_i, h_i, h_{i-1}, r_i, r_{i-1}$
Steps:

1: Check that $\mathbf{0} = (m^*_i[k] - m_i[k]) \cdot \sigma[k]$ ▷ k-th bit, $k \leq 512$
2: Check that $h_i = \mathsf{CF}_{\mathsf{sha}}(m_i, h_{i-1})$
3: Check that $\mathsf{cm}_{i-1} = \mathsf{Commit}(\mathsf{ck}, h_{i-1}, r_{i-1})$
4: Check that $\mathsf{cm}_i = \mathsf{Commit}(\mathsf{ck}, h_i, r_i)$

The commitment key ck is hard-coded in the circuit description, and h_0 is the intialization vector (IV) of SHA256. This approach adds a relatively mild overhead to the resulting circuit if we use a zero-knowledge friendly commitment

[7] Actually, with padding, transactions of more than 447 bits need more calls to $\mathsf{CF}_{\mathsf{sha}}$.

NIZK $(\mathbf{S}^{(\mathsf{mid})}_{\mathsf{peq}}, \mathbf{P}^{(\mathsf{mid})}_{\mathsf{peq}}, \mathbf{V}^{(\mathsf{mid})}_{\mathsf{peq}})$

$\mathbf{S}^{(\mathsf{mid})}_{\mathsf{peq}}(1^\lambda)$:
1: $\mathsf{ck} \leftarrow \mathsf{GenCommKey}(1^\lambda)$
2: $(pk_{\mathsf{mid}}, vk_{\mathsf{mid}}) \leftarrow \mathbb{S}_{\mathsf{mid}}(1^\lambda, \mathsf{C}_{\mathsf{mid}}[\mathsf{ck}])$
3: Output $pk := (pk_{\mathsf{mid}}, \mathsf{ck})$, $vk := (vk_{\mathsf{mid}}, \mathsf{ck})$

$\mathbf{P}^{(\mathsf{mid})}_{\mathsf{peq}}(pk, (\mathsf{tx}^*, \mathsf{txid}, \sigma), \mathsf{tx})$:
1: parse $(m_i, m_i^*, \sigma_i)_{1 \le i \le N} \leftarrow (\mathsf{tx}, \mathsf{tx}^*, \sigma)$
2: parse $(pk_{\mathsf{mid}}, \mathsf{ck}) \leftarrow pk_{\mathsf{mid}}$
3: $h_0 \leftarrow IV$ ▷ For IV of SHA256
4: $r_0 \leftarrow \mathsf{Random}(1^\lambda)$ ▷ Random element from appropriate domain
5: $\mathsf{cm}_0 \leftarrow \mathsf{Commit}(\mathsf{ck}, h_0, r_0)$
6: **for** $i \leftarrow 1$ to N **do**
7: $h_i \leftarrow \mathsf{CF}_{\mathsf{sha}}(m_i, h_{i-1})$
8: $r_i \leftarrow \mathsf{Random}(1^\lambda)$
9: $\mathsf{cm}_i \leftarrow \mathsf{Commit}(\mathsf{ck}, h_i, r_i)$
10: $z_i \leftarrow (m_i^*, \mathsf{cm}_i, \mathsf{cm}_{i-1}, \sigma_i)$ ▷ Current public input
11: $w_i \leftarrow (m_i, h_i, h_{i-1}, r_i, r_{i-1})$ ▷ Current witness
12: $\pi_i \leftarrow \mathbb{P}_{\mathsf{mid}}(pk_{\mathsf{mid}}, z_i, w_i)$
13: **end for**
14: Output $\pi := ((\pi_i)_{1 \le i \le N}, (\mathsf{cm}_i)_{0 \le i \le N}, h_N, r_N)$ ▷ $h_N = \mathsf{SHA256}(\mathsf{tx})$

$\mathbf{V}^{(\mathsf{mid})}_{\mathsf{peq}}(vk, (\mathsf{tx}^*, \mathsf{txid}, \sigma), \pi)$:
1: parse $(m_i^*, \sigma_i)_{1 \le i \le N} \leftarrow (\mathsf{tx}^*, \sigma)$
2: parse $(vk_{\mathsf{mid}}, \mathsf{ck}) \leftarrow vk$
3: parse $((\pi_i)_{1 \le i \le N}, (\mathsf{cm}_i)_{0 \le i \le N}, h_N, r_N) \leftarrow \pi$
4: **if** $\mathsf{txid} \neq \mathsf{SHA256}(h_N) \ \vee \ \mathsf{cm}_N \neq \mathsf{Commit}(\mathsf{ck}, h_N, r_N)$ **then**
5: **return** false
6: **end if**
7: **for** $i \leftarrow 1$ to N **do**
8: $z_i \leftarrow (m_i^*, \mathsf{cm}_i, \mathsf{cm}_{i-1}, \sigma_i)$
9: **if** false $= \mathbb{V}_{\mathsf{mid}}(vk_{\mathsf{mid}}, z_i, \pi_i)$ **then**
10: **return** false
11: **end if**
12: **end for**
13: **return** true

Fig. 3: The NIZK Γ_{peq} for relation $\mathcal{R}_{\mathsf{peq}}$ implemented with the commit-to-midstate approach.

NIZK $(\mathbf{S}^{(\mathsf{sel})}_{\mathsf{peq}}, \mathbf{P}^{(\mathsf{sel})}_{\mathsf{peq}}, \mathbf{V}^{(\mathsf{sel})}_{\mathsf{peq}})$

$\mathbf{S}^{(\mathsf{sel})}_{\mathsf{peq}}(1^\lambda)$:
1: $(pk_{\mathsf{sel}}, vk_{\mathsf{sel}}) \leftarrow \mathbb{S}_{\mathsf{sel}}(1^\lambda, \mathsf{C}_{\mathsf{sel}})$
2: $z_0 \leftarrow (h_0 := IV, a_0)$ ▷ For IV of SHA256, and known a_0
3: Output $pk := (pk_{\mathsf{sel}}, z_0)$, $vk := (vk_{\mathsf{sel}}, z_0)$

$\mathbf{P}^{(\mathsf{sel})}_{\mathsf{peq}}(pk, (\mathsf{tx}^*, \mathsf{txid}, \sigma), \mathsf{tx})$:
1: parse $(m_i, m_i^*, \sigma_i)_{1 \le i \le N} \leftarrow (\mathsf{tx}, \mathsf{tx}^*, \sigma)$
2: parse $(pk_{\mathsf{sel}}, (h_0, a_0)) \leftarrow pk$
3: $\pi_0 \leftarrow \emptyset$
4: **for** $i \leftarrow 1$ to N **do**
5: $h_i \leftarrow \mathsf{CF}_{\mathsf{sha}}(m_i, h_{i-1})$
6: $a_i \leftarrow \mathsf{Hash}(a_{i-1}, m_i^*, \sigma_i)$
7: $z_{i-1} \leftarrow (h_{i-1}, a_{i-1})$ ▷ Input of the incremental step
8: $z_i \leftarrow (h_i, a_i)$ ▷ Output of the incremental step
9: $\omega_i \leftarrow (m_i, m_i^*, \sigma_i)$ ▷ Non-deterministic advice
10: $\pi_i \leftarrow \mathbb{P}_{\mathsf{sel}}(pk_{\mathsf{sel}}, (i, z_0, z_i), (\omega_i, z_{i-1}, \pi_{i-1}))$
11: **end for**
12: Output $\pi := (\pi_N, h_N)$ ▷ $h_N = \mathsf{SHA256}(\mathsf{tx})$

$\mathbf{V}^{(\mathsf{sel})}_{\mathsf{peq}}(vk, (\mathsf{tx}^*, \mathsf{txid}, \sigma), \pi)$:
1: parse $(m_i^*, \sigma_i)_{1 \le i \le N} \leftarrow (\mathsf{tx}^*, \sigma)$
2: parse $(vk_{\mathsf{sel}}, z_0 := (h_0, a_0)) \leftarrow vk$
3: parse $(\pi_N, h_N) \leftarrow \pi$
4: **if** $\mathsf{txid} \neq \mathsf{SHA256}(h_N)$ **then**
5: **return** false
6: **end if**
7: **for** $i \leftarrow 1$ to N **do**
8: $a_i \leftarrow \mathsf{Hash}(a_{i-1}, m_i^*, \sigma_i)$
9: **end for**
10: $z_N \leftarrow (h_N, a_N)$
11: **if** false $= \mathbb{V}_{\mathsf{sel}}(vk_{\mathsf{sel}}, (N, z_0, z_N))$ **then**
12: **return** false
13: **end if**
14: **return** true

Fig. 4: The NIZK Γ_{peq} for relation $\mathcal{R}_{\mathsf{peq}}$ implemented with the commit-to-selector approach.

scheme. However, on the downside, it requires to verify N proofs during block validation.

Specifically, using as building block a NIZK $(\mathbb{S}_{\mathsf{mid}}, \mathbb{P}_{\mathsf{mid}}, \mathbb{V}_{\mathsf{mid}})$ to prove satisfiability of N different instantiations of circuit $\mathsf{C}_{\mathsf{mid}}[\mathsf{ck}]$, we can build $\Gamma_{\mathsf{peq}} := (\mathbf{S}_{\mathsf{peq}}, \mathbf{P}_{\mathsf{peq}}, \mathbf{V}_{\mathsf{peq}})$ for relation $\mathcal{R}_{\mathsf{peq}}$. See Figure 3 for details. Note that during proofs validation, the commitment of the current midstate in the i-th proof (i.e. used as the first commitment of the public input of $\mathsf{C}_{\mathsf{mid}}$) must be used as the commitment of the previous midstate in the $(i+1)$-th proof (i.e. used as the second commitment of the public input).

b) Approach #2: Commit to selector vector: The key idea is to accumulate a hash of the selector vector σ and the redacted transaction tx^* with a collision-resistant hash function Hash, while simultaneously ensuring partial equality for each chunk of N bits of tx^* and tx. The circuit $\mathsf{C}_{\mathsf{sel}}$ has as public inputs the three midstates corresponding to $\mathsf{tx}, \mathsf{tx}^*, \sigma$.

Circuit $\mathsf{C}_{\mathsf{sel}}$:

Public Input: h_i, a_i
Private Input: $h_{i-1}, a_{i-1}, m_i, m_i^*, \sigma$
Steps:
1: Check that $0 = (m_i^*[k] - m_i[k]) \cdot \sigma[k]$ ▷ k-th bit, $k \le 512$
2: Check that $h_i = \mathsf{CF}_{\mathsf{sha}}(m_i, h_{i-1})$
3: Check that $a_i = \mathsf{Hash}(a_{i-1}, m_i^*, \sigma)$

Above a_0, is set to an arbitrary known value. The satisfiability of $\mathsf{C}_{\mathsf{sel}}$ can be proved with an *incrementally verifiable computation* scheme $(\mathbb{S}_{\mathsf{sel}}, \mathbb{P}_{\mathsf{sel}}, \mathbb{V}_{\mathsf{sel}})$. Namely circuit $\mathsf{C}_{\mathsf{sel}}$ models a single step of an iterated function $z_i = F(z_{i-1}; \omega_i)$. The current input is $z_i := (h_i, a_i)$, the previous input $z_{i-1} := (h_{i-1}, a_{i-1})$, and the non-deterministic advice is $\omega_i := (m_i, m_i^*, \sigma_i)$. At each step, a proof for the previous iteration or for correct proof/instance accumulation is verified as well. As before, the redactor only needs to reveal the unredacted transaction digest $h_N := \mathsf{SHA256}(\mathsf{tx})$, so tx and midstates h_i remain private.

See Figure 4 for the SNARK Γ_{peq} implemented with this approach. The advantage with respect the previous approach is that now only the last proof is verified, since this proof attests to the partial equality of all the N chunks m_i, m_i^* according to a selector and tx^* accumulated in a_N. However, the disadvantage is the recursion overhead incurred by the augmented circuit, stemming from the extra step that verifies the previous proof or the correctness of the accumulation in zero-knowledge.

E. Benchmarks

We have implemented the NIZK Γ_{peq} with the two approaches described in the previous section. The code is in

Gadget	R1CS constraints	
	arkworks	bellpeper
SHA256CF	42415	29544
Partial equality (of blocks)	2175	2624
Commit to midstate (twice with Pedersen)	5102	N/A
Accumulated hash (Poseidon)	N/A	436

TABLE II: R1CS constraints of the gadgets used in circuit $\mathsf{C}_{\mathsf{mid}}$ (arkworks) and circuit $\mathsf{C}_{\mathsf{sel}}$ (bellpepper).

GitHub [40]. The commit-and-prove approach (commit to midstates, cf. Fig. 3) with Groth16 [32] over BLS-381. We have used the Groth16 implementation from the arkworks library [22]. The IVC-based approach (commit to selector, cf. Fig. 4) uses Nova [39] with its default implementation [45], namely over Grumpkin-BN254 curve cycle. We have benchmarked transactions up to 128KB. This means that for the commit-and-prove approach up to 2048 proofs (the number of SHA256 blocks) are produced and verified in parallel. The tests have been executed on a Lenovo ThinkPad with an intel i5 13th generation processor with 10 cores, 12 threads, and 15GB of memory.

Arithmetization: The R1CS for the commit-and-prove circuit $\mathsf{C}_{\mathsf{mid}}$ is built using arkworks [22] over the scalar field of BLS-381. SHA256 midstates are committed with Pedersen hash defined over JubJub, and the commitment key hard-coded in the circuit description. For the IVC approach, we use bellpepper [23] to build the R1CS of the circuit $\mathsf{C}_{\mathsf{sel}}$ over the scalar field of BN254. The accumulated hash is set to Poseidon. See Table II. The first two gadgets of the circuits are the same. The main overhead is in the SHA256 gadget, as expected.

Comparison of both approaches: During proof generation, the commit-and-prove approach with Groth16 is slightly better up to transactions of size roughly 384 bytes (6 SHA256 blocks). After that, the IVC-based approach with Nova outperforms. At 128KB transactions, the Nova prover is roughly 3x faster than Groth16. See Figure 5. The Groth16 verifier, although is constant for a single proof, it must verify many proofs. Also, the Nova verifier has only one proof to verify but its runtime is also constant[8], however it must recompute the accumulated hash a_N. Therefore, the runtime of both verifiers increase with the transaction size. Groth16 is one order of magnitude faster; we note that most of the time of the Nova verifier is dedicated to recompute the accumulated Poseidon hash. See Figure 6. Last, the proof size of Nova is constant, whereas for Groth16 increases linearly with the size of the transaction. The crossover occurs at transactions of roughly 4KB. See Figure 7.

We believe the benchmarks can be improved significantly. For example, the commit-and-prove approach could be optimized using Poseidon hashes to commit to midstates. In general, there is room for improvement in the code. Also, the weird peaks of the groth16 curves are most likely due to not enough sample averaging during benchmarks.

[8]It is polylogarithmic in the size of the NP statement, i.e. knowledge of a valid IVC proof. Its size is *fixed* by the SHA256 block size N.

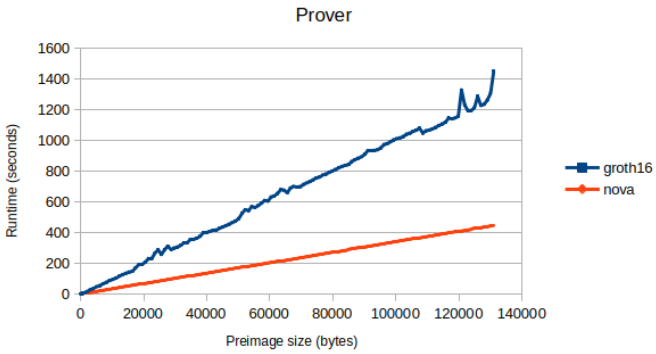

Fig. 5: Comparison of the Groth16 and Nova provers.

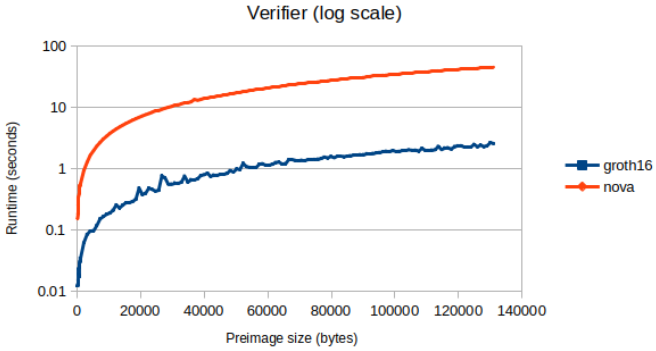

Fig. 6: Comparison of the Groth16 and Nova verifiers.

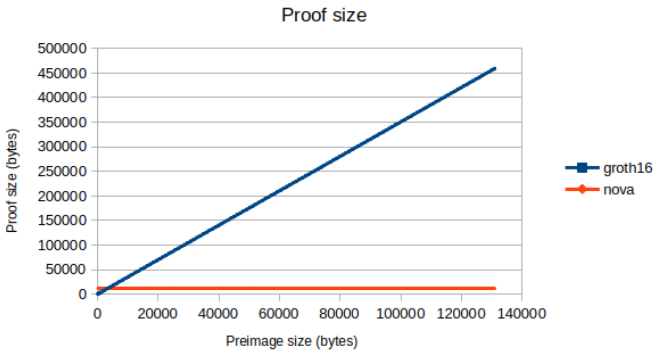

Fig. 7: Comparison of the Groth16 and Nova proofs.

References

[1] Abdolmaleki, B., Ramacher, S., Slamanig, D.: Lift-and-shift: Obtaining simulation extractable subversion and updatable snarks generically. In: CCS '20: 2020 ACM SIGSAC

[2] Ateniese, G., Magri, B., Venturi, D., Andrade, E.: Redactable blockchain – or – rewriting history in bitcoin and friends. In: 2017 IEEE European Symposium on Security and Privacy

[3] Avarikioti, G., Käppeli, L., Wang, Y., Wattenhofer, R.: Bitcoin security under temporary dishonest majority. In: Financial Cryptography and Data Security 2019

[4] Badertscher, C., Maurer, U., Tschudi, D., Zikas, V.: Bitcoin as a transaction ledger: A composable treatment. In: Advances in Cryptology - CRYPTO 2017

[5] Baghery, K., Pindado, Z., Ràfols, C.: Simulation extractable versions of groth's zk-snark revisited. IACR Cryptol. ePrint Arch.
[6] Ben Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from bitcoin. In: 2014 IEEE Symposium on Security and Privacy
[7] Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Succinct Non-Interactive Zero Knowledge for a von Neumann Architecture. In: USENIX 2014
[8] Bitansky, N., Canetti, R., Chiesa, A., Goldwasser, S., Lin, H., Rubinstein, A., Tromer, E.: The hunting of the SNARK. IACR Cryptol. ePrint Arch. (2014)
[9] Bitcoin Cash (BCH). https://github.com/bitcoin
[10] Bitcoin Core (BTC). https://github.com/bitcoincashbch
[11] Bitcoin Satoshi Vision (BSV). https://github.com/bitcoin-sv
[12] Bitcoin.org: https://developer.bitcoin.org/devguide/transactions.html#null-data
[13] Blum, M., Feldman, P., Micali, S.: Non-interactive zero-knowledge and its applications (extended abstract). In: Simon, J. (ed.) ACM Symposium on Theory of Computing, 1988
[14] Blum, M., Santis, A.D., Micali, S., Persiano, G.: Noninteractive zero-knowledge. SIAM J. Comput.
[15] Bonneau, J., Meckler, I., Rao, V., Shapiro, E.: Coda: Decentralized cryptocurrency at scale. IACR Cryptol. ePrint Arch. p. 352 (2020)
[16] Botta, V., Iovino, V., Visconti, I.: Towards data redaction in bitcoin. IEEE Transactions on Network and Service Management **19**(4) (2022)
[17] BSV Wiki: https://wiki.bitcoinsv.io/index.php/OP_CODESEPARATOR
[18] Camenisch, J., Derler, D., Krenn, S., Pöhls, H.C., Samelin, K., Slamanig, D.: Chameleon-Hashes with Ephemeral Trapdoors. In: Public-Key Cryptography – PKC 2017
[19] Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: FOCS 2001
[20] Canetti, R., Lindell, Y., Ostrovsky, R., Sahai, A.: Universally composable two-party and multi-party secure computation. In: ACM Symposium on Theory of Computing, 2002
[21] Chen, B., Bünz, B., Boneh, D., Zhang, Z.: Hyperplonk: Plonk with linear-time prover and high-degree custom gates. In: Advances in Cryptology – EUROCRYPT 2023
[22] arkworks contributors: `arkworks` zksnark ecosystem (2022), https://arkworks.rs
[23] argument computer corporation: `bellpepper`, https://github.com/argumentcomputer/bellpepper
[24] Deuber, D., Magri, B., Thyagarajan, S.A.K.: Redactable blockchain in the permissionless setting. In: 2019 IEEE Symposium on Security and Privacy (SP) (2019)
[25] Fischlin, M.: Communication-efficient non-interactive proofs of knowledge with online extractors. In: Advances in Cryptology - CRYPTO 2005
[26] Florian, M., Henningsen, S., Beaucamp, S., Scheuermann, B.: Erasing data from blockchain nodes. In: 2019 IEEE European Symposium on Security and Privacy Workshops
[27] Gabizon, A., Williamson, Z.J., Ciobotaru, O.: PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge. IACR Cryptology ePrint Archive (2019)
[28] Ganesh, C., Kondi, Y., Orlandi, C., Pancholi, M., Takahashi, A., Tschudi, D.: Witness-succinct universally-composable snarks. In: Advances in Cryptology - EUROCRYPT 2023
[29] Garay, J.A., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol: Analysis and applications. In: Advances in Cryptology - EUROCRYPT 2015
[30] Garay, J.A., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol: Analysis and applications. IACR Cryptol. ePrint Arch. p. 765 (2014)
[31] GDPR: Principles relating to processing of personal data. https://gdpr-info.eu/art-5-gdpr
[32] Groth, J.: On the size of pairing-based non-interactive arguments. In: Advances in Cryptology – EUROCRYPT 2016
[33] Groth, J.: Simulation-sound NIZK proofs for a practical language and constant size group signatures. In: Advances in Cryptology - ASIACRYPT 2006
[34] Groth, J., Maller, M.: Snarky signatures: Minimal signatures of knowledge from simulation-extractable snarks. In: Advances in Cryptology - CRYPTO 2017
[35] Groth, J., Ostrovsky, R., Sahai, A.: Perfect non-interactive zero knowledge for NP. In: Advances in Cryptology - EUROCRYPT 2006
[36] Jia, Y., Sun, S.F., Zhang, Y., Liu, Z., Gu, D.: Redactable blockchain supporting supervision and self-management. ASIA CCS '21
[37] Karasek-Wojciechowicz, I.: Reconciliation of anti-money laundering instruments and European data protection requirements in permissionless blockchain spaces. Journal of Cyber Security **7**(1), tyab004 (03 2021)
[38] Kosba, A., Zhao, Z., Miller, A., Qian, Y., Chan, H., Papamanthou, C., Pass, R., abhi shelat, Shi, E.: C∅c∅: A framework for building composable zero-knowledge proofs. Cryptology ePrint Archive, Paper 2015/1093
[39] Kothapalli, A., Setty, S., Tzialla, I.: Nova: Recursive zero-knowledge arguments from folding schemes. In: Advances in Cryptology – CRYPTO 2022
[40] Larraia, E.: `large_preimages_snarks` (2024), https://github.com/nchain-innovation/large_preimages_snarks
[41] Li, J., Ma, H., Wang, J., Song, Z., Xu, W., Zhang, R.: Wolverine: A scalable and transaction-consistent redactable permissionless blockchain. IEEE Transactions on Information Forensics and Security (2023)
[42] Ma, J., Xu, S., Ning, J., Huang, X., Deng, R.H.: Redactable blockc hain in decentralized setting. IEEE Transactions on Information Forensics and Security (2022)
[43] Manevich, Y., Barger, A., Assa, G.: Redacting transactions from execute-order-validate blockchains. In: 2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)
[44] Matzutt, R., Hiller, J., Henze, M., Ziegeldorf, J.H., Müllmann, D., Hohlfeld, O., Wehrle, K.: A quantitative analysis of the impact of arbitrary blockchain content on bitcoin. In: Financial Cryptography and Data Security 2018
[45] Microsoft: `nova`, https://github.com/microsoft/Nova
[46] Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. http://bitcoin.org/bitcoin.pdf (2008)
[47] Office, I.I.C.: Data minimisation. https://ico.org.uk/for-organisations/guide-to-data-protection/guide-to-the-general-data-protection-regulation-gdpr/principles/data-minimisation/, last accessed 14/09/2023
[48] Pagallo, U., Bassi, E., Crepaldi, M., Durante, M.: Chronicle of a clash foretold: Blockchains and the gdpr's right to erasure. In: Legal Knowledge and Information Systems - JURIX (12 2018)
[49] Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: Nearly Practical Verifiable Computation. In: 2013 IEEE Symposium on Security and Privacy
[50] Pass, R., Seeman, L., Shelat, A.: Analysis of the blockchain protocol in asynchronous networks. In: Coron, J., Nielsen, J.B. (eds.) Advances in Cryptology - EUROCRYPT 2017
[51] Puddu, I., Dmitrienko, A., Capkun, S.: μchain: How to forget without hard forks. Cryptology ePrint Archive, Paper 2017/106 (2017)
[52] Sahai, A.: Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In: FOCS 1999
[53] Santis, A.D., Crescenzo, G.D., Ostrovsky, R., Persiano, G., Sahai, A.: Robust non-interactive zero knowledge. In: Advances in Cryptology - CRYPTO 2001
[54] Schellinger, B., Völter, F., Urbach, N., Sedlmeir, J.: Yes, i do: Marrying blockchain applications with gdpr (09 2021)
[55] Shen, J., Chen, X., Liu, Z., Susilo, W.: Verifiable and redactable blockchains with fully editing operations. IEEE Transactions on Information Forensics and Security (2023)
[56] Tian, G., Wei, J., Kutyłowski, M., Susilo, W., Huang, X., Chen, X.: Vrbc: A verifiable redactable blockchain with efficient query and integrity auditing. IEEE Transactions on Computers (2023)
[57] Wang, W., Duan, J., Wang, L., Hu, X., Peng, H.: Strongly synchronized redactable blockchain based on verifiable delay functions. IEEE Internet of Things Journal (2023)
[58] Whatsonchain: https://shorturl.at/axzEY
[59] Xu, S., Ning, J., Ma, J., Huang, X., Deng, R.H.: K-time modifiable and epoch-based redactable blockchain. IEEE Transactions on Information Forensics and Security (2021)
[60] Xu, S., Ning, J., Ma, J., Xu, G., Yuan, J., Deng, R.H.: Revocable policy-based chameleon hash. In: Computer Security – ESORICS 2021
[61] Zhang, D., Le, J., Lei, X., Xiang, T., Liao, X.: Exploring the redaction mechanisms of mutable blockchains: A comprehensive survey. International Journal of Intelligent Systems (2021)
[62] Zhang, D., Le, J., Lei, X., Xiang, T., Liao, X.: Secure redactable blockchain with dynamic support. IEEE Transactions on Dependable and Secure Computing (2023)