Mutator Sets and their Application to Scalable Privacy

Alan Szepieniec Thorkil Værge
alan@neptune.cash thor@neptune.cash

Neptune

Abstract. A mutator set is a cryptographic data structure for authenticating operations on a changing set of data elements called *items*. Informally:

- There is a short commitment to the set.
- There are succinct membership proofs for elements of the set.
- It is possible to update the commitment as well as the membership proofs with minimal effort as new items are added to the set or as existing items are removed from it.
- Items cannot be removed before they were added.
- It is difficult to link an item's addition to the set to its removal from the set, except when using information available only to the party that generated it.

This paper formally defines the notion, motivates its existence with an application to scalable privacy in the context of cryptocurrencies, and proposes an instantiation inspired by Merkle mountain ranges and Bloom filters.

1 Introduction

Blockchain-based money has the potential to transform and disrupt the financial industry and even the global monetary system. However, its wider adoption and success are obstructed by two critical challenges: *privacy* and *scalability*. Privacy is the quality of a blockchain protocol that thwarts attempts to trace the source and destination of payments based on information present in the public ledger. Scalability is the quality of a blockchain protocol that resists feature degradation as a result of increased use or age. Accordingly, much research has been devoted to cryptographic techniques that promise to achieve one or both of these features.

Cryptographic accumulation schemes compress a large set of items into a compact commitment in order to dramatically reduce the complexity of elementary set operations at the modest cost of sacrificing perfect security for computational security. They have applications in timestamping, blacklists, public key infrastructure, anonymous credentials, group signatures, to name examples without being exhaustive. All these applications have one thing in common: they enable a lightweight agent to verify with cryptographic certainty the correct computation of elementary operations on relatively large sets.

Accumulation schemes have seen renewed interest in recent years in the context of blockchains, due to their potential to improve scalability by reducing the cost of participation. The lower barrier to entry and lower running costs incentivize more participation, benefiting the protocol's decentralization metrics and the features that are downstream from there such as resilience against legal attacks.

While accumulation schemes are extremely well suited to reduce the workload of verifying traceable ledgers, they are incapable of the same task when it comes to untraceable ledgers. This limitation comes from the static nature of the items accumulated by an accumulation scheme. Whenever the same item is added and later removed, or added and later modified, or modified once and later modified again, it is apparent to all observers that the items in question are one and the same. A basic requirement towards addressing privacy would stipulate items be hidden behind cryptographic commitments, and that these commitments *mutate* between operations such that external observers cannot link them.

However, up until now blockchains have relied on entirely different techniques from accumulation schemes to achieve privacy, namely decoys or mixnets. In either case, the transaction inputs and outputs are hidden behind cryptographic commitments that hide the amount, and every transaction comes with a zero-knowledge proof that establishes that no inflation is taking place and that all amounts are positive.

Decoys are plausible transaction inputs listed alongside the true origin inputs, which is where the funds that are being spent actually come from. The external observer cannot distinguish between decoys and true origins. In order to catch double-spending attempts, every transaction input that is being spent unlocks a short string of data called the *nullifier*, and a zero-knowledge proof hides which of the inputs to the transaction it came from. The blockchain maintains a set of nullifiers and whenever a transaction is confirmed, the nullifiers announced by it are added to the set. A transaction is only valid if its nullifiers are not already in the set.

The downside of this decoy-and-nullifier approach is its poor scalability: this nullifier set grows with the number of transactions in the history of the network. In a naïve implementation, the verifying protocol participant stores the entire set and incurs the associated storage cost. A more sophisticated implementation uses an accumulation scheme that admits proofs of non-membership. It shifts the burden of work but does not eliminate it: the *verifying protocol participant* only needs to store a compact set commitment and verify succinct non-membership proofs, but the *transaction initiator* must produce a non-membership proof relative to this set, which integrates knowledge of all nullifiers. An even more sophisticated implementation fixes a snapshot of the set commitment at the time the coin was generated. To spend it, the transaction initiator must prove that the nullifier was not added to the nullifier set since it was created. The user must therefore remain online to continually update his non-membership proofs, or else synchronize them upon rejoining. The complexity of this latter synchronization task scales linearly with the intervening time because the updated proofs must depend on all the nullifiers announced in the time spent offline. To adapt this approach for better scalability, either the nullifier set must be prunable somehow or else its *non-membership proofs should be practically independent of members.*

Mixnets are an alternative to the decoy-and-nullifier approach. In this construction, unspent transaction outputs (UTXOs) are hidden behind *publicly re-randomizable* commitments, which enable *anyone* to transform an old commitment into a new one that *a)* binds to the same information, and *b)* can be opened only by the same parties that could open the old one, but *c)* is unlinkable to the old commitment in the eyes of everyone else. Depending on the blockchain architecture, the entire UTXO-commitment set or a subset thereof is mixed at regular or variable intervals by miners, transaction initiators, or designated operators that are neither.

The first downside of this approach is that the owner of a UTXO must learn his new commitment by iterating over all mix outputs and either trying to unlock them or running some other ownership test on them. The workload of this task is linear in the size of the batch and thus proportional to the size of the anonymity set. Even if the UTXO-commitment passes through multiple smaller mixes instead, the workload merely shifts from tracing the UTXO across one large mix to tracing it across multiple smaller ones. While the anonymity set size does expand exponentially in the number of mixes, the user's workload scales linearly in this number, and thus once again either requires users to remain online or face a costly catch-up task upon rejoining.

The second downside of this mixnet approach is the fact that the operator selects the secret permutation and can be coerced or bribed to release it, or to select one that is not uniformly random. A construction for eliminating the operator's randomness would obviate this trust assumption and moreover make the mix outputs deterministic, thus simplifying the UTXO owner's tracing task.

In summary, the existing constructions for scalability and for privacy seem to induce tradeoffs that preclude achieving both *scalability and privacy* simultaneously. Specifically:

- accumulation schemes reduce the cost of participation but are inherently traceable;
- decoys and nullifiers generate untraceable transactions but require operations on an ever-growing nullifier set that cannot be pruned or truncated;
- mixnets generate untraceable transactions but require users to trace their coins across mixes and moreover require trust in the mixnet operator.

Mutator sets are cryptographically authenticated data structures that fix these deficiencies. Informally, mutator sets enable users to add items to a set and remove them later. The essential differences relative to accumulation schemes are that mutator sets accumulate *commitments* to data items rather than items directly; and that the commitment when an item is added is unlinkable to its commitment when it is removed. Phrased crudely, *mutator sets are accumulation schemes with addition-removal unlinkability.*

Mutator sets are not just idly reminiscent of nullifier sets and decoy commitments. What the full list of all historical commitments to transaction outputs and the full nullifier set jointly represent, is a set of *unspent* transaction outputs. A transaction is valid only if all its inputs are members of this UTXO set: the nullifier set keeps track of which items are removed from this set. As the commitment and nullifier are unlinkable, this construction provides addition-removal unlinkability. The relevant difference relative to mutator sets is the workload. Phrased crudely, *mutator sets are succinct decoy-and-nullifier sets.*

As for mixnets, the similarity is readily apparent: the items are removed in a (possibly) different order from the one in which they were added, and transformed untraceably in the process. The main difference is that there is no operator. Consequently, there is no leakable permutation or re-randomization randomness. Phrased crudely, *mutator sets are operator-free mixnets.*

Construction Overview. Our construction uses Merkle mountain ranges (MMRs) [38,16], which are sequences of Merkle trees that support adding elements to a list, in addition to inheriting the basic features of Merkle trees such as compact membership proofs and support for modifying leafs. MMRs are used to compactify two data structures, the *append-only commitment list (AOCL)* and the *sliding-window Bloom filter (SWBF).*

The AOCL tracks additions to the mutator set by recording hiding commitments to data items. These commitments are known as *addition records.*

The SWBF tracks removals from the mutator set by recording indices of bits that are announced when items are removed from the mutator set. These indices are uniquely determined by the data item, but *which* indices is impossible for the computationally bounded external observer to predict. A zero-knowledge proof certifies that a given set of bits corresponds to *some* addition record that lives in the AOCL. A *removal record* is a set of bit indices along with this zero-knowledge proof and along with the bits' membership proofs in the SWBF MMR. The removal record is valid only if some of the listed bits are not set yet.

The SWBF has infinite size in principle, but every addition record can only sample bits from a finite window. This window slides periodically.

Figure 1 captures key elements visually:

– the commitment to the set consists of the Merkle roots of both MMRs and the active window;
– an addition record consists of a commitment to the data item;
– the zero-knowledge proof establishes that for some addition record, *a)* the addition record lives in the AOCL MMR, and *b)* the given bit indices were derived correctly;
– a removal record consists of a set of announced bits, the zero-knowledge proof, and membership proofs in the SWBF MMR for all bits that no longer live in the active window.

The effect of the sliding window is to reduce the dependency of membership proofs and removal records for items that were added early on, on the removal records of items added later. Specifically, the MMR membership proofs for old chunks of the SWBF require only a logarithmic number of updates as a function of the number of items that were added (and maybe removed) after the item in

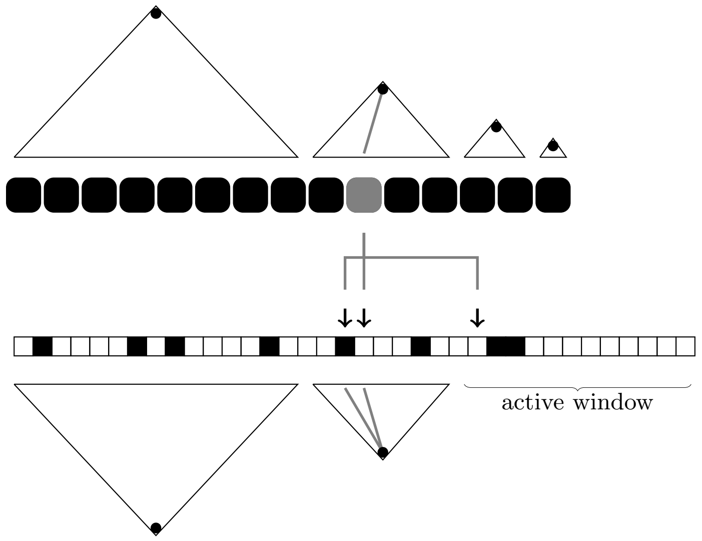

Fig. 1: Schematic overview of the present construction.

question was. Under the realistic assumption that at any point in time freshly added items are more likely to be removed than old ones, this property translates to a workload for updating membership proofs that decreases over time.

Related Work. *Accumulators.* Arguably Merkle trees [28] are the first accumulation schemes and when their leafs are sorted, they even admit non-membership proofs. Benaloh and de Mare [6] introduce the term "accumulator" and describe a construction where the order of accumulated items does not matter. Nyberg [33] proposed a similar function based on on symmetric primitives. Barić and Pfitzmann [4] argue that accumulators should be presented as tuples of algorithms, extending the term to accumulator *schemes*. Since then research has focused on finding new building blocks for instantiating accumulator schemes [32,26,27,1], extending their features and properties [10,13], and applying them to various protocols and tasks [34,23].

The combined use of Bloom filters and Merkle trees to construct cryptographic accumulator schemes was first proposed by Rambaja and Avdullahu [35]. Compared to just Merkle trees, the advantage is non-membership proofs without sacrificing updatability; and compared to just Bloom filters, the advantage is compactness. This construction does not provide addition-removal unlinkability.

Accumulators with Privacy. After finishing the bulk of this paper, the authors became aware of two papers extending accumulator schemes in such a way as to achieve a notion very similar to the one presented here. The first paper [2] presents a generic framework for building accumulator schemes with fancy properties from accumulator schemes without. It then proceeds to construct one from two additive RSA-based accumulators, one with membership proofs and one with non-membership proofs. The new construction features *join-revoke unlinkability* — which is essentially the same property as addition-removal unlinkability of the present paper.

The second paper [3] generically constructs an *oblivious* accumulator scheme, which is one with add-delete unlinkability (also essentially the same property) and add-delete indistinguishability (additionally hiding the set size). It builds this from a key-value commitment scheme, and the key-value commitment scheme from a universal accumulator and an extendable-length vector commitment scheme. What is missing relative to mutator sets is a polylogarithmic bound on the running times of *all* algorithms.

Light Clients. Starting with informal discussions [30,18], there have been a host of proposals for reducing the cost of participating in a blockchain network by shrinking the state of a client to a short cryptographic commitment [16,8]. A related objective is to shrink the blockchain itself by having miners produce succinct proofs of history verification, in addition to proof-of-work [24,25,22].

Privacy on Blockchains. Zerocoin [29] first proposed the use of accumulators to ensure privacy in cryptocurrencies. ZCash [5] was the first to use zk-SNARKs with the same objective. Monero uses ring signatures, a related tool to accumulators and zero-knowledge proofs, to achieve privacy [36]. All three fall in the decoy-and-nullifier category.

Mimblewimble [21,20,19] first constructed private UTXOs without nullifier sets. Unfortunately, this approach limits privacy to *amount confidentiality* (or *state confidentiality*) as UTXOs remain linkable across transactions.[1] Quisquis [17] first proposed to use mixnets to effect unlinkability without nullifier sets.

Mixnets. Mixnets were originally proposed by Chaum [11] for anonymous messaging. Since then they have been used as a building block for a host of privacy-enhancing technologies, including anonymous remailers [12], onion routing [15], electronic voting [37], etc.

Roadmap. Section 2 covers some basic preliminaries, and after that Section 3 presents formal definitions of both accumulation schemes and mutator sets. Section 4 details how a mutator set can be used in the context of a UTXO-based blockchain to achieve scalable privacy. Section 5 presents our construction in a sequence of incremental steps, and proves its security properties. Finally, Section 6 finishes with some concluding thoughts.

2 Preliminaries

Hash Function Security. We reduce the security of our construction to security of some concrete hash function $\mathsf{H} : \Sigma^* \rightarrow \mathsf{D}$ (the particular choice is irrelevant). To capture this notion we use two insecurity functions: 1) $\mathsf{InSec}^{\mathsf{H}}_{\mathsf{coll}}$, the insecurity function for collision resistance for H; and 2) $\mathsf{Adv}^{\mathsf{H}}_{\mathsf{PRF}}$, an adversary's

[1] Mimblewimble blockchains notably do not record individual transactions but only aggregates of them. Nevertheless, the individual transactions can be collected by the online but otherwise passive oberver as they are broadcasted.

advantage over a random guess at distinguishing H from a random function. In order to justify conflating a concrete function H with a family of functions (as the PRF game requires) we implicitly designate part of the input as key material and pretend as though the adversary cannot read it. These quantities are defined as

$$\mathsf{InSec}^{\mathsf{H}}_{\mathsf{coll}} \triangleq \max_{\mathsf{A}} \Pr\left[\mathsf{A}() \Rightarrow a, b \in \Sigma^* \wedge a \neq b \wedge \mathsf{H}(a) = \mathsf{H}(b)\right] \tag{1}$$

$$\mathsf{Adv}^{\mathsf{H}}_{\mathsf{PRF}} \triangleq \max_{\mathsf{A}} \left| \Pr\left[\mathsf{A}^{f_b}() \Rightarrow b\right] - \frac{1}{2} \right| \ , \tag{2}$$

where $f_0 = \mathsf{H}, f_1 \xleftarrow{\$} \{f \mid f : \Sigma^* \to \mathsf{D}\}, b \xleftarrow{\$} \{0,1\}$.

The omitted arguments of InSec and Adv can be the security parameter λ (whose unary representation is $\boldsymbol{\lambda}$), the allowance for adversaries with respect to running time, memory, or (in a black box model) number of queries. The maximum is taken over all adversaries A satisfying those constraints.

Commitment Scheme. A commitment scheme binds a party to a data item whose value can be revealed in a way that prevents equivocation. Specifically, a commitment scheme consists of two algorithms, presented here abstractly:

– $\mathsf{commit} : T\,[\times \mathsf{R}] \to \mathsf{C}$ takes a data item and possibly some randomness and outputs a *commitment*.
– $\mathsf{open} : T \times \mathsf{C}\,[\times \mathsf{O}] \to \{\mathsf{True}, \mathsf{False}\}$ takes a data item, the commitment, possibly some auxiliary opening information, and outputs True if the item was used to produce the commitment and False otherwise.

A commitment scheme generally satisfies two security properties, *hiding* and *binding*, although the former requirement is sometimes dropped. Hiding is equivalent to semantic security: no resource-bounded adversary should be able to compute any function of the committed message, and therefore hiding commitment functions require a randomness parameter. Binding captures non-equivocation: no adversary should be able to produce a commitment that it can open to distinct messages.

A commitment scheme $(\mathsf{commit}, \mathsf{open})$ is *hiding* if for all the polynomial-time adversaries $\mathsf{A} = (\mathsf{A}_0, \mathsf{A}_1)$ the advantage over a random guess in the semantic security game is negligible:

$$\mathsf{Adv}^{\mathsf{A}}_{\mathsf{ss}} \triangleq \left| \Pr\left[\mathsf{A}_1(st, c) \Rightarrow b\right] - \frac{1}{2} \right| \leq \epsilon(\lambda) \ , \tag{3}$$

where $m_0, m_1 \leftarrow \mathsf{A}_0(\boldsymbol{\lambda}); b \xleftarrow{\$} \{0,1\}; r \xleftarrow{\$} \mathsf{R}; c \leftarrow \mathsf{commit}(m_b; r)$.

A commitment scheme $(\mathsf{commit}, \mathsf{open})$ is *binding* if for all polynomial-time adversaries A the binding insecurity $\mathsf{InSec}_{\mathsf{binding}}$ is negligible:

$$\mathsf{InSec}^{\mathsf{A}}_{\mathsf{binding}} \triangleq \Pr\left[m_0 \neq m_1 \ \wedge \ \mathsf{open}(m_0, c, o_0) = \mathsf{open}(m_1, c, o_1)\right] \ , \tag{4}$$

where $(m_0, o_0, m_1, o_1, c) \leftarrow \mathsf{A}(\boldsymbol{\lambda})$.

A common commitment scheme is defined relative to a hash function $\mathsf{H} : \Sigma^* \rightarrow \mathsf{D}$ and known as the *canonical* commitment scheme. The commitment algorithm is defined as $\mathsf{commit} : T \times \mathsf{R} \rightarrow \mathsf{D}, (m, r) \mapsto \mathsf{H}(m \| r)$. In order to be hiding, the space of randomness R must be at least as large as the space of digests D. The binding property of the canonical commitment scheme reduces to the collision-resistance of the hash function.

The canonical opening algorithm is a special case of the *standard* opening algorithm, which is defined relative to a generic commitment algorithm $\mathsf{commit} : T \times \mathsf{R} \rightarrow \mathsf{C}$ as $\mathsf{open} : T \times \mathsf{C} \times \mathsf{R} \rightarrow \{\mathsf{True}, \mathsf{False}\}, (t, c, r) \mapsto \mathsf{commit}(t, r) \overset{?}{=} c$.

Merkle Tree. A Merkle tree [28] is a cryptographic data structure that compactly commits to a vector of 2^k items called *leafs*. This commitment, called the *root*, is derived by repeatedly hashing together two neighboring nodes in every layer of a balanced binary tree of *height* k, whose bottom layer consist of the leafs. The list of siblings of nodes on a path from a leaf to the root is known as that leaf's *authentication path* and can be used to verify the claim that the leaf belongs to the tree. We present only the interface here.

- $\mathsf{root} : [T] \rightarrow \mathsf{D}$ computes the root from the leafs.
- $\mathsf{open} : [T] \times \mathbb{N} \rightarrow [\mathsf{D}]$ computes the authentication path for an indicated leaf, from the list of all leafs and a leaf index.
- $\mathsf{verify} : \mathsf{D} \times \mathbb{N} \times [\mathsf{D}] \times T \rightarrow \{\mathsf{True}, \mathsf{False}\}$ verifies that a leaf belongs to a Merkle tree, given the tree's root, the leaf's index, its authentication path, and the leaf itself.

Additionally, we define two functions that are not traditionally associated with Merkle trees but nevertheless represent relatively trivial extensions.

- $\mathsf{modify} : \mathsf{D} \times \mathbb{N} \times [\mathsf{D}] \times T \times T \rightarrow \mathsf{D} \cup \{\perp\}$ modifies an indicated leaf, assuming its membership proof is valid. Specifically, it sends $(\mathit{old\_root}, \mathit{leaf\_index}, \mathit{authentication\_path}, \mathit{old\_item}, \mathit{new\_item})$ to $\mathit{new\_root}$ if the authentication path is valid, or to $\perp$ if it is not. Note that afterwards the same authentication path should be valid for $\mathit{new\_item}$ but relative to $\mathit{new\_root}$.
- $\mathsf{update} : T \times \mathbb{N} \times [\mathsf{D}] \times (\mathsf{D} \times \mathbb{N} \times [\mathsf{D}] \times T \times T) \rightarrow [\mathsf{D}] \cup \{\perp\}$ updates the authentication path for a given leaf, given all of the arguments for modifying another leaf.

Bloom Filter. A Bloom filter [7] is a probabilistic data structure for cheap probabilistic set membership tests. While the test can generate a false positive, it cannot produce false negatives. In other words, the result is either "the item might be in the set" or "the item is definitely not in the set". The cost of running the test only to get an undefinitive positive answer back, is compensated for by the definite negative alternative result, which allows the programmer to short-circuit whatever procedure comes next.

The state of a Bloom filter is defined by an array of w bits, which is initially all zero. Whenever an item $t \in T$ is added to the set, k indices in $[0, \ldots, w-1]$ are calculated from k independent hash functions $[\mathsf{H}_0, \ldots, \mathsf{H}_{k-1}]$ applied to t. The indicated bits are set.

To test an item $t \in T$ for membership, the same k hash functions are evaluated. If all indicated bits are set, the item might be in the set. But if some indicated bit is not set, then the item cannot have been added to the set, and as a result the item is definitely not in the set.

3 Definition

3.1 Accumulation Scheme

The following definition of accumulat*ion* schemes is different from formal definitions of accumula*tor* schemes existing in the literature [9]. The difference is motivated by the need for scaffolding to exposit the main construction and prove its security. Regardless, the notions are the same in spirit.

Definition 1 (accumulation scheme). *An* accumulation scheme *is a tuple of polynomial time algorithms relative to a item data type T.*

- $\mathsf{init} : [\{1\}] \rightarrow K$ *generates a commitment to the empty set.*
- $\mathsf{prove} : K \times T \rightarrow M$ *takes a set commitment and a new item, and outputs a* membership proof *that will be valid under the updated commitment if the item is added, but invalid if is removed thereafter.*
- $\mathsf{verify} : K \times T \times M \rightarrow \{\mathsf{True}, \mathsf{False}\}$ *verifies a membership proof.*
- $\mathsf{add} : K \times T \rightarrow K \times A$ *updates the set commitment so that the represented set includes the given item, and additionally outputs an* addition record.
- $\mathsf{remove} : K \times T \times M \rightarrow K \times R$ *updates the set commitment so that the set no longer includes the given item, and additionally outputs a* removal record.
- $\mathsf{update} : K \times T \times M \times (A \vee R) \rightarrow M$ *updates an item's membership proof given an addition or removal record.*

The main difference relative to existing definitions of accumula*tor* schemes comes from the explicit treatment of addition and removal records. Intuitively, records are the pieces of information that are associated with a change to the set commitment. As such, they are also needed to update a membership proof so that it is synchronized to the new set commitment. The addition and removal records expose a feature (or deficiency) of accumulation schemes: in all constructions the authors are aware of, the addition records and removal records pertaining to the same item are trivially linkable.

Another difference is that prove can only be invoked before the item in question was added. The scope of this paper does not cover generating proofs for items that were added in the past; it covers maintaining proofs and synchronizing them when they are out of date.

Some constructions enable modification of existing items. Strictly speaking, modification is implied by removing and adding. However, for practical use it is worthwhile to define the interface explicitly:

– $\mathsf{modify} : K \times M \times T \times T \to (K \times E) \cup \{\bot\}$ updates the set commitment so that the given item is replaced with the second given item, if the membership proof is valid, and if so outputs a *modification record* ("E" for edit).

When abstracting the type of record and intend only to capture some piece of data inducing an update to the set commitment, we use the term *update record* and reserve the symbol $\upsilon \in A \cup R \cup E$.

3.2 Security

An accumulation scheme needs to have the following security properties, defined in asymptotic terms relative to a security parameter λ. Additionally, the security games are defined with respect to a sequence $O = (o_i)_{i=1}^{\#O}$ of valid set operations $o_i : \{\mathsf{add}, \mathsf{remove}\} \times T$, corresponding to an underlying set[2] S recursively defined as $S(O \| (\mathsf{add}, t)) = S(O) \cup \{t\}$, $S(O \| (\mathsf{remove}, t)) = S(O) \backslash \{t\}$, and $S(\varnothing) = \varnothing$. Specifically, a sequence of operations $O = (o_i)_{i=1}^{\#O}$ is *valid* iff $\forall i \in \{1, \ldots, \#O\} \,.\, o_i = (\mathsf{remove}, t) \Rightarrow t \in S((o_j)_{j=1}^{i-1})$.

Definition 2 (completeness for accumulation schemes). *Consider the completeness game* $\mathsf{Game}^{\mathsf{AS}}_{\mathsf{cmplt}}$.

An accumulation scheme is complete *iff, for any valid sequence of operations* $O_1 \| (\mathsf{add}, t) \| O_2$*, the completeness error* $\mathsf{Err}_{\mathsf{cmpl}}$ *is negligible:*

$$\mathsf{Err}_{\mathsf{cmpl}}(\boldsymbol{\lambda}) \overset{\triangle}{=} 1 - \Pr\left[\mathsf{Game}_{\mathsf{cmplt}}(\boldsymbol{\lambda}, O_1, O_2, t) \to \mathsf{True}\right] \leq \epsilon(\lambda) \ . \tag{5}$$

It is tricky to define soundness for a proof system for set membership, when the set is determined by a compact commitment. Indeed: a single commitment may decommit to multiple sets, in which case soundness is a moot notion, and *knowledge-soundness* is preferred. We avoid definitional complexity by assuming instead that the commitment is computed honestly. The adversary can only provide the items to be added to the set and the randomnesses to be used when computing these additions. This definition of soundness fails in contexts where the adversary is capable of computing the commitment dishonestly. However, this dishonest calculation will be exposed at a later stage through the use of general-purpose zero-knowledge proofs.

Definition 3 (honest-commitment soundness). *Consider the honest-commitment soundness game* $\mathsf{Game}_{\mathsf{hcsnd}}$.

An accumulation is honest-commitment sound *iff the honest-commitment soundness error is negligible, where the maximum is taken over all polynomial-time adversaries* A.

$$\mathsf{Err}_{\mathsf{hcsnd}}(\boldsymbol{\lambda}) \overset{\triangle}{=} \max_{\mathsf{A}} \Pr\left[\mathsf{Game}^{\mathsf{A}}_{\mathsf{hcsnd}}(\boldsymbol{\lambda})\right] \leq \epsilon(\lambda) \ . \tag{6}$$

[2] Throughout this paper we use the word "set" to denote multiset, or unordered list.

Game 1: Completeness

```
define $\mathsf{Game}^{\mathsf{AS}}_{\mathsf{cmplt}}(\boldsymbol{\lambda}, O_1, O_2, t)$ as:
    $\kappa \leftarrow \mathsf{init}(\boldsymbol{\lambda})$
    for $o \in O_1$ :
        if $o = (\mathsf{add}, s)$ :
            $\kappa, \mu^\star, \_ \leftarrow \mathsf{add}(\kappa, s)$
        if $o = (\mathsf{remove}, s)$ :
            $\kappa, \_ \leftarrow \mathsf{remove}(\kappa, s, \mu^\star)$
    $\kappa, \mu, \alpha \leftarrow \mathsf{add}(\kappa, t)$
    for $o \in O_2$ :
        if $o = (\mathsf{add}, s)$ :
            $\kappa, \mu^\star, \alpha^\star \leftarrow \mathsf{add}(\kappa, s)$
            $\mu \leftarrow \mathsf{update}(t, \mu, \alpha^\star)$
        if $o = (\mathsf{remove}, s)$ :
            $\kappa, \rho^\star \leftarrow \mathsf{remove}(\kappa, s, \mu^\star)$
            $\mu \leftarrow \mathsf{update}(t, \mu, \rho^\star)$
    return $\mathsf{verify}(\kappa, t, \mu)$
```

Game 2: Honest-Commitment Soundness

```
define $\mathsf{Game}^{\mathsf{A}}_{\mathsf{hcsnd}}(\boldsymbol{\lambda})$ as:
    $\kappa \leftarrow \mathsf{init}(\boldsymbol{\lambda})$
    ▷ adversary supplies:
    -set operations $O$,
    -randomnesses $\mathcal{R}$,
    -item $t$,
    -membership proof $\mu$
    $O, \mathcal{R}, t, \mu \leftarrow \mathsf{A}(\boldsymbol{\lambda})$ ;
    for $(o, r) \in \mathsf{zip}(O, \mathcal{R})$ :
        if $o = (\mathsf{add}, s)$ :
            $\kappa, \_, \_ \leftarrow \mathsf{add}(\kappa, s; r)$
        if $o = (\mathsf{remove}, s)$ :
            $\kappa, \_ \leftarrow \mathsf{remove}(\kappa, s; r)$
    return $\mathsf{verify}(\kappa, t, \mu) \wedge t \notin S(O)$
```

3.3 Mutator Set

Informally, a mutator set is an accumulation scheme that additionally satisfies *addition-removal unlinkability*, which asserts that it is difficult to relate addition records to removal records. However, the set operations for accumulation schemes are deterministic, giving rise to a trivial attack against unlinkability whereby the adversary runs through all possible permutations of the given set operations to see which one matches with his view. In order to achieve unlinkability, the set operations must involve some randomness that the adversary does not have access to. Mutator sets provide this interface.

To define the accumulator set interface we start from accumulation schemes and modify this interface as follows:

- Users must *commit* to items $t \in T$ using secret randomness $r \xleftarrow{\$} \mathsf{R}$ to generate an addition record α before they can be added. This function is achieved by the method $\mathsf{commit} : T \times \mathsf{R} \to A$, where R is a sufficiently large alphabet of random symbols. The addition record $\alpha \in A$ is a commitment to the item t (not to be confused with the commitment κ to the set).
- The public input to the add method is the addition record $\alpha \in A$, instead of the randomness-free item $t \in T$. (Note that in contrast to add for accumulation schemes, α is an input not an output.)

– To remove an item $t \in T$ from the set, the user must first generate a removal record $\rho \in R$, which observers can then use to update the set commitment or membership proofs with. This separation of concerns is captured by to methods, drop and remove.
  - $\mathsf{drop} : K \times T \times M \to R$ generates a removal record from a set commitment, an item, and its membership proof.
  - $\mathsf{remove} : K \times R \to K \cup \{\bot\}$ updates the set commitment with a valid removal record.

  (Note: symmetrically to α, ρ has moved from the output of remove in an accumulation scheme to the input of remove in a mutator set.)

Definition 4 (mutator set). *A* mutator set *is a tuple of polynomial time algorithms relative to a item data type T:*

– $\mathsf{init} : [\{1\}] \to K$ *generates a commitment to the empty set.*
– $\mathsf{commit} : T \times \mathsf{R} \to A$ *generates an addition record by using the supplied randomness to commit to an item.*
– $\mathsf{prove} : K \times T \times \mathsf{R} \to M$ *takes a set commitment, a new item, and randomness used to commit to it, and outputs a* membership proof *that will be valid under the updated commitment if the item is added, but invalid if is removed thereafter.*
– $\mathsf{verify} : K \times T \times M \to \{\mathsf{True}, \mathsf{False}\}$ *verifies a membership proof.*
– $\mathsf{add} : K \times A \to K$ *updates the set commitment so that the set includes the item committed to by the addition record.*
– $\mathsf{drop} : K \times T \times M \to R$ *generates a removal record from a set commitment, an item, and its membership proof.*
– $\mathsf{remove} : K \times R \to K \cup \{\bot\}$ *updates the set commitment so that the set no longer includes the item determined by the removal record, assuming it is a valid removal record.*
– $\mathsf{update} : K \times T \times M \times (A \vee R) \to M$ *updates an item's membership proof given an addition or removal record for another item, such that the updated membership proof is valid for the new set commitment resulting from applying the record.*

3.4 Security

Completeness for mutator sets is defined analogously to Definition 2: any sequence of valid set operations should give rise to valid membership proofs. Definition 5 adapts this intuition to the updated interface.

Definition 5 (completeness for mutator sets). *Consider the completeness game* $\mathsf{Game}^{\mathsf{MS}}_{\mathsf{cmplt}}$*. A mutator set is* complete *iff, for any valid sequence of operations O, and any $t \in S(O)$, the completeness error* $\mathsf{Err}_{\mathsf{cmpl}}$ *is negligible:*

$$\mathsf{Err}_{\mathsf{cmpl}}(\boldsymbol{\lambda}) \stackrel{\triangle}{=} 1 - \Pr\left[\mathsf{Game}^{\mathsf{MS}}_{\mathsf{cmplt}}(\boldsymbol{\lambda}, O, t) \to \mathsf{True}\right] \leq \epsilon(\lambda) \ . \quad (7)$$

Honest-commitment soundness is inadequate for mutator sets due to the changed interface. In particular, and in contrast to accumulation schemes, addition and removal records hide which items they pertain to. The set represented by the set commitment κ, cannot be defined *directly* in terms of the constituent items; but can only defined *by proxy*, in terms of the items to which the addition and removal records are bound to. It follows that an adequate definition of soundness for mutator sets must imply that both addition records and removal records are binding commitments to particular items.

Moreover, in order for the value κ to compactly represent a set of items it must be binding to that set. However, this variant of binding is tricky to capture formally because the interface does not provide a functionality for opening the commitment and revealing the entire set. Instead, we opt for a simpler and stronger quality: the set commitment must be binding to the set of all addition and removal records that produced it, starting from the commitment to the empty set.

At this point what remains for security is a means for outlawing invalid set updates, such as removing non-members or removing the same item twice. Note that the interface delegates to the security definition the choice as to what happens when an item is added twice. We opt here to disallow removing an item more often than it was added. Technically speaking, this choice implicitly counts multiplicities and thus corresponds to multisets rather than sets, but we ignore this distinction and overload the latter term with the meaning of the former.

Definition 6 (security for mutator sets). *A mutator set is* secure *iff*

- $(\mathsf{commit}, \mathsf{open})$ *where* $\mathsf{open}(\alpha, t; r) \overset{\triangle}{=} \left(\mathsf{commit}(t, r) \overset{?}{=} \alpha\right)$ *is a binding commitment scheme with negligible binding insecurity:* $\mathsf{InSec}_{\mathsf{bnd}}^{\mathsf{commit}}(\boldsymbol{\lambda}) \leq \epsilon(\lambda)$*; and*
- $(\mathsf{drop}, \mathsf{open})$ *where* $\mathsf{open}(\rho, t; \mu) \overset{\triangle}{=} \left(\mathsf{drop}(t, \mu) \overset{?}{=} \rho\right)$ *is a binding commitment scheme with negligible binding insecurity:* $\mathsf{InSec}_{\mathsf{bnd}}^{\mathsf{drop}}(\boldsymbol{\lambda}) \leq \epsilon(\lambda)$*; and*
- $(\mathsf{set\_commit}, \mathsf{open})$ *where* $\mathsf{set\_commit} : \{A, R\}^* \times \{\varnothing\} \rightarrow K$,

$$(O, \varnothing) \mapsto \begin{cases} \mathsf{init}(\boldsymbol{\lambda}) & \Leftarrow O = \varnothing \\ \mathsf{add}(\mathsf{set\_commit}(O^\star, \varnothing), \alpha) & \Leftarrow O = \alpha \| O^\star \wedge \alpha : A \\ \mathsf{remove}(\mathsf{set\_commit}(O^\star, \varnothing), \rho) & \Leftarrow O = \rho \| O^\star \wedge \rho : R \end{cases}$$

  and $\mathsf{open}(\kappa, O; \varnothing) \overset{\triangle}{=} \left(\mathsf{set\_commit}(O; \varnothing) \overset{?}{=} \kappa\right)$ *is a binding commitment scheme with negligible binding insecurity:* $\mathsf{InSec}_{\mathsf{bnd}}^{\mathsf{set\_commit}}(\boldsymbol{\lambda}) \leq \epsilon(\lambda)$*; and*
- *the* non-negativity insecurity $\mathsf{InSec}_{\mathsf{nonneg}}$ *is negligible, where this quantity is defined as the maximum probability over all polynomial-time adversaries* A *of winning the non-negativity game of Game 4:*

$$\mathsf{InSec}_{\mathsf{nonneg}}(\boldsymbol{\lambda}) \overset{\triangle}{=} \max_{\mathsf{A}} \mathsf{Pr}\left[\mathsf{Game}_{\mathsf{nonneg}}^{\mathsf{A}}(\boldsymbol{\lambda})\right] \leq \epsilon(\lambda) \enspace .$$

Game 3: Completeness

```
define $\mathsf{Game}^{\mathsf{MS}}_{\mathsf{cmplt}}(\boldsymbol{\lambda}, O, t)$ as:
    $\kappa \leftarrow \mathsf{init}(\boldsymbol{\lambda})$
    $\mathcal{T} \leftarrow \varnothing$
    for $o \in O$ :
        if $o = (\mathsf{add}, s)$ :
            $r \xleftarrow{\$} \mathsf{R}$
            $\alpha \leftarrow \mathsf{commit}(\kappa, t; r)$
            $\mu \leftarrow \mathsf{prove}(\kappa, t, r)$
            for $(s^*, \mu^*) \in \mathcal{T}$ :
                $\mu^* \leftarrow$
                  $\mathsf{update}(\kappa, s^*, \mu^*, \alpha)$
            $\mathcal{T} \leftarrow \mathcal{T} \cup \{(s, \mu)\}$
            $\kappa \leftarrow \mathsf{add}(\kappa, s)$
        if $o = (\mathsf{remove}, s)$ :
            find $\mu$ such that
              $(s, \mu) \in \mathcal{T}$
            $\rho \leftarrow \mathsf{drop}(\kappa, s, \mu)$
            $\mathcal{T} \leftarrow \mathcal{T} \backslash \{(s, \mu)\}$
            for $(s^*, \mu^*) \in \mathcal{T}$ :
                $\mu^* \leftarrow$
                  $\mathsf{update}(\kappa, s^*, \mu^*, \rho)$
            $\kappa \leftarrow \mathsf{remove}(\kappa, \rho)$
    find $\mu$ such that $(t, \mu) \in \mathcal{T}$
    return $\mathsf{verify}(\kappa, t, \mu)$
```

Game 4: Non-negativity

```
define $\mathsf{Game}^{\mathsf{A}}_{\mathsf{nonneg}}(\boldsymbol{\lambda})$ as:
    $\kappa \leftarrow \mathsf{init}(\boldsymbol{\lambda})$
    $ctr \leftarrow 0$
    $O, \mathcal{T}, \rho^\star \leftarrow \mathsf{A}(\boldsymbol{\lambda})$
    for $o \in O$ :
        if $o = (\mathsf{add}, \alpha)$ :
            $\kappa \leftarrow \mathsf{add}(\kappa, \alpha)$
            $ctr \leftarrow ctr + 1$
        if $o = (\mathsf{remove}, \rho)$ :
            $\kappa \leftarrow \mathsf{remove}(\kappa, \rho)$
            $ctr \leftarrow ctr - 1$
    return $ctr =$
      $\#\mathcal{T} \wedge \mathsf{remove}(\kappa, \rho^\star) \neq \perp \wedge$
      $\forall (t, \mu) \in \mathcal{T} \,.\, \mathsf{verify}(\kappa, t, \mu) \wedge$
      $\rho^* \neq \mathsf{drop}(\kappa, t, \mu)$
```

3.5 Unlinkability and Anonymity

Capturing unlinkability in cryptographic terms is tricky. As a first attempt, one might devise a game whereby the adversary sees two addition records and one removal record and nothing else, but with the guarantee that the removal record is linked to one of the addition records and not to the other, but the order in which they are supplied is random. The adversary wins if he guesses which one. No resource-bounded adversary should be able to win this game with more than a negligible probability.

However, this approach is rather moot because realistic adversaries have access to more information than just three records. They see the set commitment and the entire sequence of addition and removal records that generated it.

As a second attempt, one might devise a game whereby the adversary sees an entire mutator set history play out, and is then tasked with finding the addition record that matches with a given removal record. The baseline is the success probability of the optimal algorithm for this task that reads only the set operations and no cryptographic data. The concrete adversary's advantage when attacking a concrete scheme is the difference between its success probability and this baseline. A mutator set may be defined as *unlinkable* if for all bounded resource adversaries, this *unlinkability advantage* is negligible in the security parameter.

While the above intuition captures the essence of an ideal mutator set in regards to linkability, it is still rather moot because in practice the produced privacy might be constrained by the sequence of set operations. In the extremal case where every addition except for the last is followed immediately by a removal, the adversary in this game can have a 100% success probability even without breaking any cryptography.

An alternative to capturing unlinkability in cryptographic terms, is to capture anonymity in entropic terms [14]. In the present context, the attacker observes one removal record and assigns probabilities to all possible addition records representing the likelihood, in the attacker's estimation, that they are linked to the same item. The degree of anonymity is the entropy of this probability distribution. We refer to this notion as *removal entropy*.

3.6 Succinctness

In order for a mutator set to be practically useful, it must be *succinct*. Specifically, the commitments and records must be compact, and the operations on them efficient. Informally, *compactness* denotes a polylogarithmic size limit (as a function of the number of historical set operations) on the set commitments, membership proofs, and update records. Likewise, the operations are *efficient* if they have polylogarithmic running time.

Definition 7 (succinctness). *A mutator set is* succinct *relative to a number of set operations N iff both*

- *the set commitments $\kappa \in K$, addition records $\alpha \in A$, and removal records $\rho \in R$, are* compact, i.e., *all have a representation that grows at most polylogarithmically:* $\mathsf{max}(|\kappa|, |\alpha|, |\rho|) \leq \mathsf{polylog}(N)$*;*
- *all algorithms comprising the scheme are* efficient*: their running time is bounded from above by* $\mathsf{polylog}(N)$.

Succinct Synchronization Suppose the set was updated N times in some period of time spent offline. Once online again, how much work does it take to update a membership proof accordingly? Ideally, the answer ought to be *much less than N*.

We opted to omit this feature from the formal definition of succinctness and discuss it only informally, because capturing it formally in complexity theoretic terms is rather tricky. Surely the user wanting to update a membership proof needs to read all the updates, even if only to come to the conclusion that some or perhaps most of them are not necessary for updating *his* membership proofs. We leave the proper formalization of this intuition as an open question.

4 Scalable Privacy

In UTXO-based cryptocurrencies like Bitcoin [31], every transaction consumes existing coins and produces new ones. The consumed coins determine a public key, and the transaction is only valid if it is signed by the consumed coins. Synchronized nodes only need to keep track of the set of unspent transaction outputs (UTXO set) to verify new transactions. Once a coin is consumed, and after this expenditure confirmed by enough blocks, old coins can be discarded safely.

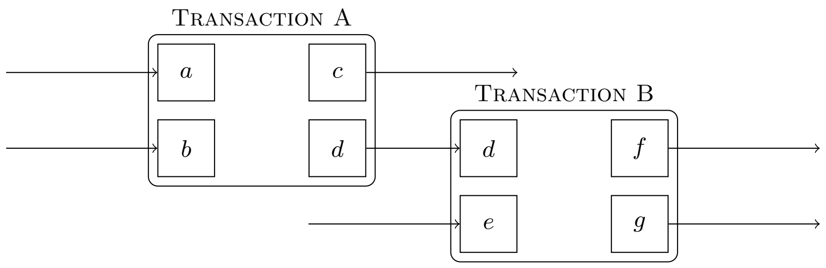

Fig. 2: Fragment of a transaction graph in a transparent UTXO-based ledger.

Figure 2 illustrates the mechanics. UTXOs a, b, d, and e are consumed and can thus be discarded to save disk space. UTXOs c, f, and g have not been spent yet and can therefore be used as inputs in future transactions. Every coin can be consumed at most once. The ledger is transparent in two senses. First, the UTXOs themselves are public and so the observer can see their public keys[3] even if he cannot produce a signature for them. Second, the UTXOs are traceable across transactions. While this transparency is beneficial for auditability, it also undermines privacy.

Mutator sets serve as a drop-in replacement for the transparent UTXO set. The consumed coins in a transaction are replaced by removal records and the generated coins by addition records. Both the removal records and the addition records are binding commitments to coins, and so it is possible to prove statements about the committed coins. Specifically, the transaction contains zero-knowledge proofs establishing that a) the spending policies of all inputs are satisfied (for instance, because the owner signed off on the transaction); and b) the coin logic is satisfied (for instance, all outputs have positive amounts and their sum equals that of of all inputs). The mutator set guarantees that every coin can be removed at most once.

As coins are hidden behind commitments, and as removal records are unlinkable with addition records, it is impossible to track them. All the while, the ledger's soundness is guaranteed with zero-knowledge proofs.

[3] The phrase "public key" is used loosely to refer to a binding commitment to a spending policy that must be satisfied by the transaction initiator.

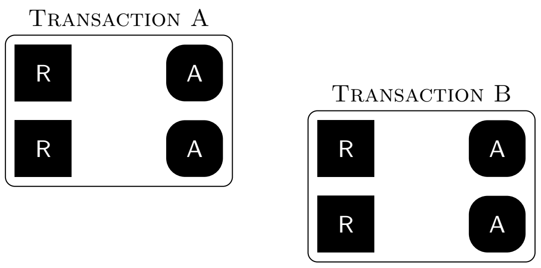

Fig. 3: Fragment of an opaque ledger based on mutator sets.

5 Construction

5.1 Merkle Mountain Range

A Merkle mountain range (MMR) [38,16] is an authenticated data structure that compactly represents a growing set of items along with generating membership proofs for elements of this set. We provide here only an informal overview. The novelty of this section consists of fitting the folklore notion into our definition of an accumulation scheme.

Informally, a Merkle mountain range is a list of Merkle trees in descending order by size. Whenever two trees of equal height exist, they are folded into one. To add an item to the MMR, add it as a singleton to the small end of the list, and apply the folding operation as often as necessary. Every item ever added is a leaf in *some* tree, and so every item has an authentication path relative to *some* Merkle root.

To fit this object into our definition of accumulation schemes, we populate the types required by it:

– The data structure accumulates items of data type T.
– A commitment consists of a list of Merkle roots and a leaf count: $K = [\mathsf{D}] \times \mathbb{N}$.
– A membership proof is an index and an authentication path, the latter of which is a list of hash digests: $M = \mathbb{N} \times [\mathsf{D}]$.
– An addition record is given by the hash digest of the added element along with its index: $A = \mathsf{D} \times \mathbb{N}$.
– A modification record is given by the the new leaf, its index, and its authentication path: $E = T \times \mathbb{N} \times [\mathsf{D}]$.

Likewise, we populate the functions defined by the accumulation scheme interface:

– $\mathsf{init} : [\{1\}] \rightarrow K$ generates an empty list of Merkle roots along with a leaf count set to zero. This commitment represents the empty list.
– $\mathsf{prove} : K \times T \rightarrow M$ produces a membership proof for an item $t \in T$ that will be valid under the new set commitment if the item is added to the data structure.

- $\mathsf{add} : K \times T \rightarrow K \times A$ increments the leaf count, appends the item to the peaks list, reduces the number of peaks by folding equal-height roots as necessary, and outputs a new set commitment along with an addition record.
- $\mathsf{verify} : K \times T \times M \rightarrow \{\mathsf{True}, \mathsf{False}\}$ verifies the membership proof which supposedly establishes that the given item is a member of the data structure at the given index.
- $\mathsf{modify} : K \times M \times T \times T \rightarrow K \times E$ takes a commitment, a membership proof, an old element, and a new element; and outputs a set commitment list obtained by replacing the indicated element with the new one, along with a modification record. Note that the membership proof for the new item is the same as for the old one.
- $\mathsf{update} : K \times T \times M \times (A \vee E) \rightarrow M$ takes a set commitment, the item, its membership proof, and an addition or modification record, and produces a new membership proof. It does this by extending the old authentication path by zero or more digests if an item was added, or modifying some suffix of it if an item in the same tree was modified.

Removals can be simulated by modifying items to set a deletion bit.

Theorem 1 (MMR Completeness). MMR *has perfect completeness.*

Proof. The statement follows from construction. □

Theorem 2 (MMR Soundness). MMR *is honest-commitment sound with error bounded by the hash collision resistance insecurity of* H*:* $\mathsf{Err}^{\mathsf{MMR}}_{\mathsf{hcsnd}} \leq \mathsf{InSec}^{\mathsf{H}}_{\mathsf{coll}}$.

Proof. In order for $\mathsf{MMR.verify}(\kappa, t, \mu)$ to output True for an element $t \notin S(O)$, $\mathsf{Merkle.verify}(\kappa.\boldsymbol{p}_{[j]}, g, t, \mu.auth\_path)$ has to output True for some tree index j that corresponds to $\kappa.n$, and for some (local-tree) leaf index g corresponding to (MMR-wide) item index j. Since $t \notin S(O)$, no pair (g, j) satisfies that t is the gth leaf of the jth Merkle tree. So the valid Merkle verification must come from a collision of hashes somewhere.

A successful adversary A in the honest-commitment soundness game gives rise to a successful collision finder B. The description of this collision-finder is identical to the description of Game 2 except that it collects all preimage-digest pairs of the hash function as they are computed. After the execution is done, B simply sorts for digest. This sorted list will contain one pair with distinct preimages and identical digests, so B outputs the preimages. The overhead of B is $\log Q$ where Q is the number of times the hash function is called. This fact establishes that B still runs in polynomial time and therefore has success probability bounded by $\mathsf{InSec}^{\mathsf{H}}_{\mathsf{coll}}$. □

5.2 Append-Only Commitment List

An append-only commitment list is a data structure that contains a growing list of *commitments* to data items, with the obvious remark that these commitments are themselves data items as well. The state of an append-only commitment list is fully determined by one object:

– a list $V : [\mathsf{D}]$ whose elements are hiding commitments to the accumulated items.

The commitment scheme is defined as follows. To commit to a message $t \in T$, obtain randomness $r \xleftarrow{\$} \mathsf{R}$ drawn uniformly at random from a sufficiently large set of symbols R, and compute the commitment $c \in \mathsf{D}$ as $c \leftarrow \mathsf{H}(t \| r)$. To verify that a commitment c binds to a message t, it suffices to supply r as it enables the test $c \overset{?}{=} \mathsf{H}(t \| r)$.

Adding an item $t \in T$ to the set consists of simply appending a new commitment. Having added an item, it is possible to prove membership of the item t to the set in a way that can be verified efficiently, provided that the item's *index of addition* $m \in \mathbb{N}$ is known. Therefore, this index must be part of the item's membership proof, along with the randomness.

The scheme AOCL has the interface of an accumulation scheme, with the exception that removals are disallowed. The functions, add and verify include logic specific to the commitments. The remaining functions are omitted because obvious.

Algorithm 1: AOCL.add

```
1 define AOCL.add($\kappa = V, t; r$) as:
2     $m \leftarrow \#V$
3     $c \leftarrow \mathsf{H}(t \| r)$
4     $V \leftarrow V \| c$
5     return $\kappa = V, \mu = (m, r), \alpha = c$
```

Algorithm 2: AOCL.verify

```
1 define AOCL.verify($\kappa = V, t, \mu = (m, r)$) as:
2     $c \leftarrow \mathsf{H}(t \| r)$
3     return $V_{[m]} \overset{?}{=} c$
```

Security. The completeness of the append-only timestamped commitment list follows from constuction. Its honest-commitment soundness is less straightforward. We consider Game 2 with the restriction that O does not contain any remove operations.

Theorem 3. *As an accumulation scheme,* AOCL *has honest-commitment soundness error bounded by*

$$\mathsf{Err}_{\mathsf{snd}}^{\mathsf{AOCL}} \leq \mathsf{InSec}_{\mathsf{coll}}^{\mathsf{H}} \ . \tag{8}$$

Proof. Assume that $(\mathsf{add}, t) \notin O$ but verify returns True. This event implies a collision since for some $t^* \neq t$ and some r^*, $c = \mathsf{H}(t \| r) = \mathsf{H}(t^* \| r^*)$. The pair t^*, r^* is obtained by recording all preimage-digest pairs as the hash function is computed. A successful adversary can be transformed into a successful collision-finder, whose success probability is bounded by definition by $\mathsf{InSec}_{\mathsf{coll}}^{\mathsf{H}}$. □

5.3 Sliding Window Bloom Filter

A *sliding window Bloom filter* represents a growing set of *timestamped* items $(l, t) \in \mathbb{N} \times T = T^\star$. The state of a filter is fully defined by an infinite array of bits $F \in \{0,1\}^*$. The bits in the array are initially set to zero but are set to one as elements are added to the set. Since only finite sets are ever represented, only a finite section of the bit array is ever represented.

A sliding window Bloom filter is parameterized by four integers:

- the *window width* $w \in \mathbb{N}$, which determines the size of the interval where bits can be set as items are added;
- the *pointer count* $k \in \mathbb{N}$, which determines the maximum number of bits flipped by any item;
- the *batch size* $b \in \mathbb{N}$, which determines the number of items in between window slides; and
- the *step size* $s \in \mathbb{N}$. Assume the step size divides the window width.

Additionally, we associate a hash function $\mathsf{H}_w : \mathbb{N} \times T \times \mathsf{R} \times \mathbb{N} \to \{0, \ldots, w-1\} \subset \mathbb{N}$ with the window size w.

A filter defines two set operations:

- $\mathsf{add} : \{0,1\}^* \times T^\star \times \mathsf{R} \to (\{0,1\}^* \times \mathbb{N}) \times \mathbb{N}^k$, which adds a timestamped and randomized item to the set represented by the filter.
- $\mathsf{probe} : \{0,1\}^* \times T^\star \times \mathsf{R} \to \{\mathsf{True}, \mathsf{False}\}$, which tests if a given item is a member of the set with true positives, false positives, and true negatives, but no false negatives.

What sets a sliding window Bloom filter apart from a regular Bloom filter is that the array of bits has infinite length but the window of settable bits shifts in accordance with the timestamp l that items come with. Specifically, the add operation consists of sampling k indices in $\{0, \ldots, w-1\}$ pseudorandomly, translating the indices by $\lfloor \frac{l}{b} \rfloor \cdot s$, and setting the indicated bits.

Algorithm 3: SWBF.add

```
1 define SWBF.add($F, (l, t), r$) as:
2     $J \leftarrow \varnothing$
3     for $i \in \{0, \ldots, k-1\}$ :
4         $j \leftarrow \mathsf{H}_w(l \,\|\, t \,\|\, r \,\|\, i)$
5         $J \leftarrow J \cup j$
6         $F_{[j+\lfloor \frac{l}{b} \rfloor \cdot s]} \leftarrow 1$
7     return $F, J$
```

Algorithm 4: SWBF.probe

```
1 define SWBF.probe($F, (l, t), r$) as:
2     for $i \in \{0, \ldots, k-1\}$ :
3         $j \leftarrow \mathsf{H}_w(l \,\|\, t \,\|\, r \,\|\, i)$
4         if $F_{[j+\lfloor \frac{l}{b} \rfloor \cdot s]} = 0$ :
5             return false
6     return true
```

To probe whether an element is a member of the set, check the indicated array elements. If any one element is 0, then the element definitely was not added. Crucially, this operation requires the timestamp l because otherwise the offset is not known.

The false positive probability corresponds to the probability that k given indices are nonzero. This probability is

$$\left(1 - \left(1 - \frac{1}{w}\right)^{\frac{kbw}{s}}\right)^k \approx \left(1 - e^{-\frac{kb}{s}}\right)^k \quad . \tag{9}$$

By setting s to a significant fraction of kb, and w much larger than kb, it is possible to engineer a cryptographically small false positive probability. The optimal value of k as a function of b and s is $k = \frac{s \cdot \ln 2}{b}$. Plugging that ratio into the expression (9) gives a probability of roughly 2^{-k}.

5.4 Non-Compact Mutator Sets

In this section we build a mutator set from an append-only commitment list and a sliding window Bloom filter. This mutator set lacks a crucial property: compactness. The next section remedies this deficiency.

The general strategy is to use canonical commitments to items as addition records and keep track of them inthe append-only commitment list. Simultaneously, the removal records contain a list of pseudorandom indices of set bits in the sliding window Bloom filter, which tracks which items are being removed. A membership proof to the mutator set consists of a membership proof to the append-only commitment list, and a *non-membership* proof to the sliding window Bloom filter. This construction turns the cryptographically small false positive probability into a negligible completeness error.

The item acquires a timestamp when it is added to the set. The timestamp is part of the membership proof μ. The timestamp is necessary to slide the Bloom filter window, which in turn is necessary to avoid saturating the filter. Therefore, the further apart items are added, the more noticeably will their removal records reflect this distance and relative order. Phrased differently, unlinkability is optimal for items added immediately after one another; and degrades with their distance.

To ensure non-negativity, the property that prevents the set commitment from being updated with a removal record for a non-member, it is necessary to add a non-interactive zero-knowledge proof $\pi \in \mathit{\Pi}$ to the removal record. Informally, this proof establishes that the index set J of set bits was derived from some item, timestamp, randomizer, and matching canonical commitment that lives in the AOCL. Moreover, it is important that this proof system has knowledge soundness with respect to the secrets: specifically, $\mathsf{ZKPoK}\{(l, t, r, c) : c = \mathsf{H}(t \| r) \wedge c \in V \wedge J = \{\mathsf{H}_w(l \| t \| r \| i) + \lfloor \frac{l}{b} \rfloor \cdot s \,|\, 0 \leq i < k\}\}$. Let ς denote the soundness error for this proof system; but note that completeness and zero-knowledge are taken for granted. For the purpose of comparing equality between removal records, the proof π is ignored; in particular, this means that malleating the proof is not considered an attack on the binding property of drop.

The types are defined as follows.

– $\mathsf{MS}.T = T$
– $\mathsf{MS}.K = [\mathsf{C}] \times \{0,1\}^*$
– $\mathsf{MS}.M = \mathbb{N} \times \mathsf{R}$
– $\mathsf{MS}.A = \mathsf{C} \times \mathbb{N}$
– $\mathsf{MS}.R = \mathbb{N}^k \times \mathit{\Pi}$

The following algorithm definitions use subscript 1 in anticipation of another variant with improved features.

Algorithm 5: $\mathsf{MS}_1.\mathsf{init}$

```
1 define $\mathsf{MS}_1.\mathsf{init}(i)$ as:
2     $\lambda \leftarrow \#i$
3     return $\kappa = ([\,], [0]^\infty)$
```

Algorithm 6: $\mathsf{MS}_1.\mathsf{commit}$

```
1 define $\mathsf{MS}_1.\mathsf{commit}(t, r)$ as:
2     $\_, \_\alpha \leftarrow \mathsf{AOCL.add}([\,], t; r)$
3     return $\alpha$
```

Algorithm 7: $\mathsf{MS}_1.\mathsf{prove}$

```
1 define
  $\mathsf{MS}_1.\mathsf{commit}(\kappa = (V, F), t, r)$
  as:
2     $\_, \mu, \_ \leftarrow \mathsf{AOCL.add}(V, t; r)$
3     return $\mu$
```

Algorithm 8: $\mathsf{MS}_1.\mathsf{verify}$

```
1 define $\mathsf{MS}_1.\mathsf{verify}(\kappa =$
  $(V, F), t, \mu = (l, r))$ as:
2     return
        $\mathsf{AOCL.verify}(V, t, (l, r)) =$
        $\mathsf{True} \wedge$
        $\mathsf{SWBF.probe}(F, (l, t), r) =$
        $\mathsf{False}$
```

Algorithm 9: $\mathsf{MS}_1.\mathsf{add}$

```
1 define $\mathsf{MS}_1.\mathsf{add}(\kappa = (V, F), \alpha)$
  as:
2     $V \leftarrow V \| \alpha$
3     return $\kappa = (V, F)$
```

Algorithm 10: $\mathsf{MS}_1.\mathsf{drop}$

```
1 define $\mathsf{MS}_1.\mathsf{drop}(\kappa =$
  $(V, F), t, \mu = (l, r))$ as:
2     $\_, J \leftarrow$
        $\mathsf{SWBF.add}(F, (l, t), r)$
3     $\pi \leftarrow \mathit{\Pi}.\mathsf{prove}(\{(l, t, r, c\} :$
        $c = \mathsf{H}(t \| r) \wedge c \in V \wedge J =$
        $\{\mathsf{H}_w(l \| t \| r \| i) + \lfloor \frac{l}{b} \rfloor \cdot s \mid 0 \leq$
        $i < k\})$
4     return $\kappa = (V, F)$
```

While there are records α and ρ, neither record is necessary to update the membership proof μ. Therefore the update function is just the identity.

Theorem 4 (completeness of MS_1). *As a mutator set, MS_1 is complete with completeness error* $\left(1 - \left(1 - \frac{1}{w}\right)^{\frac{kbw}{s}}\right)^k \approx \left(1 - e^{-\frac{kb}{s}}\right)^k$.

Proof. Suppose $\mathsf{Game}_{\mathsf{cmplt}}(\boldsymbol{\lambda}, O_1, O_2, t)$ output False in some execution for a valid sequence op operations $O_1 \| (\mathsf{add}, t) \| O_2$. This proposition implies that $\mathsf{verify}(\kappa, t, \mu)$ on line Line 21 of Algorithm 1 fails, which in turn implies that either $\mathsf{AOCL.verify}(V, t, (m, r)) = \mathsf{False}$ or $\mathsf{SWBF.probe}(F, (l, t)) = \mathsf{True}$ on line Line 2 of Algorithm 8. The AOCL has perfect completeness and so the left clause

Algorithm 11: MS_1.remove

```
1 define $\mathsf{MS}_1.\mathsf{remove}(\kappa = (V, F), \rho)$ as:
2     if $\Pi.\mathsf{verify}(\rho.\pi, \{(l, t, r, c\} : c = \mathsf{H}(t\|r) \wedge c \in V \wedge J = \{\mathsf{H}_w(l\|t\|r\|i) + \lfloor \frac{l}{b} \rfloor \cdot s | 0 \leq i < k\}) = \mathsf{False}$ :
3         return $\perp$
4     else
5         $F^*, \_ \leftarrow \mathsf{SWBF.add}(F, (l, t), r)$
6         if $F^* \neq F$ :
7             return $\kappa = (V, F^*)$
8         else
9             return $\perp$
```

of this conjunction is guaranteed to return True. The right clause returning true corresponds to the false positive probability of the SWBF, which is given by Eqn. 9. □

Theorem 5 (security of MS_1). MS_1 *is secure; specifically:*

- commit*, in combination with the standard opening algorithm, has bounded binding insecurity* $\mathsf{InSec}^{\mathsf{commit}}_{\mathsf{bnd}} \leq \mathsf{InSec}^{\mathsf{H}}_{\mathsf{coll}}$*;*
- drop*, in combination with the standard opening algorithm, has bounded binding insecurity* $\mathsf{InSec}^{\mathsf{drop}}_{\mathsf{bnd}} \leq \frac{Q(Q-1)k!}{2w^k} + \mathsf{Adv}^{\mathsf{H}_w}_{\mathsf{PRF}}$*, where Q is the number of queries to* H_w*;*
- *the function* $\mathsf{set\_commit} : \{A, R\}^* \times \{\varnothing\} \to K$*,*

$$(O, \varnothing) \mapsto \begin{cases} \mathsf{init}(\boldsymbol{\lambda}) & \Leftarrow O = \varnothing \\ \mathsf{add}(\mathsf{set\_commit}(O^\star, \varnothing), \alpha) & \Leftarrow O = \alpha \| O^\star \wedge \alpha : A \\ \mathsf{remove}(\mathsf{set\_commit}(O^\star, \varnothing), \rho) & \Leftarrow O = \rho \| O^\star \wedge \rho : R \end{cases}$$

in combination with the standard opening algorithm, has bounded binding insecurity $\mathsf{InSec}^{\mathsf{set\_commit}}_{\mathsf{bnd}} \leq \left(1 - \left(1 - \frac{1}{w}\right)^{\frac{kbw}{s}}\right)^k + \mathsf{Adv}^{\mathsf{H}_w}_{\mathsf{PRF}}$*;*
- MS_1 *has bounded non-negativity insecurity* $\mathsf{InSec}^{\mathsf{MS}_1}_{\mathsf{nonneg}} \leq \varsigma + \mathsf{InSec}^{\mathsf{H}}_{\mathsf{coll}}$.

Proof. Commit. The bounded binding insecurity of commit follows from the fact that its definition coincides with that of a canonical commitment scheme. The adversary who successfully equivocates on a given commitment can be used to build a collision-finder, so $\mathsf{InSec}^{\mathsf{commit}}_{\mathsf{bnd}}(\boldsymbol{\lambda}) \leq \mathsf{InSec}^{\mathsf{H}}_{\mathsf{coll}}$.

Drop. If the hash function $\mathsf{H}_w : \Sigma^* \to \{0, \ldots, w-1\}$ is uniformly random, then so is the implicit mapping from (l, t, r, i) to $J \in \{0, \ldots, w-1\}^k$ as a *list* of indices, *i.e.*, where the order matters. In the worst case there are k unique indices and then there are $k!$ ways to order them. So every index *set* defines up to $k!$ different but equivalent *lists*. Therefore, the probability of observing a collision of *sets* is bounded by $\frac{Q(Q-1)k!}{2w^k}$, where Q is the number of queries to H_w. A successful adversary in the binding game for drop can be transformed into an algorithm that finds collisions of index sets, and so its success probability must be bound by this quantity. An adversary with success probability in excess of this amount can be used to distinguish the hash function (interpreted as a PRF with implicit key r) from random. The advantage of this distinguisher is bounded by $\mathsf{Adv}^{\mathsf{H}_w}_{\mathsf{PRF}}$ by assumption. By the union bound, the binding adversary's success probability is bounded by $\frac{Q(Q-1)k!}{2w^k} + \mathsf{Adv}^{\mathsf{H}_w}_{\mathsf{PRF}}$.

Set commit. The AOCL is perfectly binding to the set of addition records because it is a list. A pair of different sets of removal records $O_1 \neq O_2$ that induce the same bits in the SWBF can be reduced to a pair of non-intersecting sets with the same properties in addition to $O_1 \cap O_2 = \varnothing$. Assuming H_w is uniformly random it is possible to calculate the probability that all bits indicated by some $\rho \in O_1$ are already set by the removal records in O_2. In fact, this calculation repeats the derivation of the completeness error, which is $\left(1 - \left(1 - \frac{1}{w}\right)^{\frac{kbw}{s}}\right)^k \approx \left(1 - e^{-\frac{kb}{s}}\right)^k$. Furthermore, the adversary who succeeds in excess of this probability can be used to build a distinguisher in the PRF game to distinguish H_w from random, whose advantage is bounded by $\mathsf{Adv}^{\mathsf{H}_w}_{\mathsf{PRF}}$. By the union bound, the binding adversary's success probability is bounded by

$$\left(1 - \left(1 - \frac{1}{w}\right)^{\frac{kbw}{s}}\right)^k + \mathsf{Adv}^{\mathsf{H}_w}_{\mathsf{PRF}} \ .$$

Non-negativity. Suppose the adversary wins Game 4, meaning that $\mathsf{remove}(\kappa, \rho^*) \neq \bot$ even though all items still in the mutator set generate removal records distinct from ρ^*. By using the extractor implied by the knowledge-soundness of π one extracts the witness (l, t, r, c) such that $\mathsf{H}(t \| r) = c \in V$. There are two possibilities:

1. $(t, (l, r)) \notin \mathcal{T}$. In this case we have an attack on the binding property of commit because $\mathcal{T}$ must contain some other element that also generates $c \in V$. This attack on binding in turn implies an attack on the collision resistance of H.
2. $(t, (l, r)) \in \mathcal{T}$. This is a contradiction because on the one hand $\rho^* = \mathsf{drop}(\kappa, t, \mu = (l, r))$ because this is implied by $\mathsf{remove}(\kappa, \rho^*) \neq \bot$ in the middle of Line 12 in Algorithm 4; on the other hand $\rho^* \neq \mathsf{drop}(t, \mu = (l, r))$ is asserted in the same line.

In summary, if the adversary A is successful, and if the extractor E is successful, then some collision-finder C for H is successful. One derives a bound on the success probability of A from this:

$$\begin{aligned}
1 &= \Pr[\mathsf{A}\checkmark \wedge \mathsf{E}\checkmark \Rightarrow \mathsf{C}\checkmark] && (10)\\
&= \Pr[\neg(\mathsf{A}\checkmark \wedge \mathsf{E}\checkmark) \vee \mathsf{C}\checkmark] && (11)\\
&= \Pr[\mathsf{A}\mathbf{x} \vee \mathsf{E}\mathbf{x} \vee \mathsf{C}\checkmark] && (12)\\
&\leq \Pr[\mathsf{A}\mathbf{x}] + \Pr[\mathsf{E}\mathbf{x}] + \Pr[\mathsf{C}\checkmark] && (13)\\
&\leq 1 - \Pr[\mathsf{A}\checkmark] + \varsigma + \mathsf{InSec}^{\mathsf{H}}_{\mathsf{coll}} && (14)
\end{aligned}$$

$$\Pr[\mathsf{A}\checkmark] \leq \varsigma + \mathsf{InSec}^{\mathsf{H}}_{\mathsf{coll}} \ . \qquad (15)$$

$\square$

5.5 Compact Mutator Sets

A pair of MMRs transform the non-compact mutator set MS_1 into a compact one (MS_2). The general idea behind the transformation is this:

- Use one MMR to compactify the AOCL V, and adapt the algorithms that access it accordingly.
- Use another MMR to compactify the *inactive part* of the SWBF F, and adapt the algorithms that access it accordingly. For the purpose of computing Merkle trees, the chunk size is s. In other words, each leaf contains a contiguous chunk of s bits from F. The active part is represented explicitly as an array of w bits. The dividing line between inactive and active lies at index $\lfloor \frac{N}{b} \rfloor \cdot s$, where N is the total number of items added to the mutator set.
- Replace the commitment $\kappa = (V, F) : [\mathsf{D}] \times \{0,1\}^*$ with $(\kappa_V, \kappa_F, F_A) : \mathsf{MMR}.K \times \mathsf{MMR}.K \times \{0,1\}^w = K$.
- Expand the membership proof $\mu = (l, r) : \mathbb{N} \times \mathsf{R}$ with:
  - an authentication path $\mathit{auth\_path}_V : [\mathsf{D}]$ which is the authentication path for element m in the MMR of V; and
  - a dictionary sending indices to chunk-and-authentication-path pairs: $\mathit{target\_chunks} : \mathbb{N} \rightarrow (\{0,1\}^s \times [\mathsf{D}])$. This dictionary is used to access bits in the inactive part of the filter, specifically *those* bits that *will be* set if the item is removed: $J = \{\lfloor \frac{l}{b} \rfloor \cdot s + \mathsf{H}_w(l \| t \| r \| i) \,|\, 0 \leq i < k\}$. The keys in this dictionary correspond to *chunk indices* c_i derived from the *bit indices* b_i via $c_i = \lfloor \frac{b_i}{s} \rfloor$. There are no entries in this dictionary for bits in the active portion of the filter.

  The new membership proof is therefore $(l, r, \mathit{auth\_path}_V, \mathit{target\_chunks}) : \mathbb{N} \times \mathsf{R} \times [\mathsf{D}] \times (\mathbb{N} \rightarrow (\{0,1\}^s \times [\mathsf{D}])) = M$.
- Modify the zero-knowledge proof system $\mathit{\Pi}$ to prove the correct execution of $\mathsf{MMR.verify}(\kappa_V, c, \mu_V)$ rather than the membership $c \in V$.
- Expand the removal record $\rho = (J, \pi) : \mathbb{N}^k \times \mathit{\Pi}$ with a dictionary of leaf-and-authentication-path pairs where the leaf represents the chunk of the filter F needed to set the inactive bits indicated by J: $\rho = (J, \pi, \mathit{target\_chunks}) : \mathbb{N}^k \times \mathit{\Pi} \times (\mathbb{N} \rightarrow ([\{0,1\}]^s \times [\mathsf{D}])) = R$. There are no leafs or authentication paths for bits in the active part of the filter.

The algorithms are defined explicitly below, except for $\mathsf{MS}_2.\mathsf{commit}$ which is identical to $\mathsf{MS}_1.\mathsf{commit}$. Unfortunately, using the MMR to compress commitments requires unrolling the calls to the proper functions of AOCL and SWBF.

Security. The use of MMRs does not affect completeness. In terms of security, if the MMRs are sound then a successful adversary against the compact mutator set MS_2 can be reduced to a successful adversary against the non-compact mutator set MS_1.

Algorithm 12: $\mathsf{MS}_2.\mathsf{init}$

```
define $\mathsf{MS}_2.\mathsf{init}(\boldsymbol{\lambda})$ as:
    $\lambda \leftarrow \#\boldsymbol{\lambda}$
    $\kappa_V \leftarrow \mathsf{MMR.init}(\boldsymbol{\lambda})$
    $\kappa_F \leftarrow \mathsf{MMR.init}(\boldsymbol{\lambda})$
    $F_A \leftarrow [0]^w$
    return $\kappa = (\kappa_V, \kappa_F, F_A)$
```

Algorithm 13: $\mathsf{MS}_2.\mathsf{add}$

```
define
  $\mathsf{MS}_2.\mathsf{add}(\kappa = (\kappa_V, \kappa_F, F_A), \alpha)$
  as:
    $\kappa_V, \_ \leftarrow \mathsf{MMR.add}(\kappa_V, \alpha)$
    if $\kappa_V \equiv 0 \mod b$ :
        // window slides
        $slid\_chunk \leftarrow F_{A[0:s]}$
        $F_A \leftarrow F_{A[s:]} \| [0]^{w-s}$
        $\kappa_F, \_ \leftarrow$
          $\mathsf{MMR.add}(chunk)$
    return $\kappa = (\kappa_V, \kappa_F, F_A)$
```

Algorithm 14: $\mathsf{MS}_2.\mathsf{prove}$

```
define $\mathsf{MS}_2.\mathsf{prove}(\kappa = (\kappa_V, \kappa_F, F_A), t, r)$ as:
    // compute commitment to item
    $l \leftarrow \kappa_V.n$
    $c \leftarrow \mathsf{H}(t \| r)$
    // add commitment to list
    $\kappa_V, \mu_V, \alpha \leftarrow \mathsf{MMR.add}(\kappa_V, c)$
    // test if window slides; if so update filter MMR
    $target\_chunks \leftarrow \mathsf{dict}()$
    if $\kappa_V.n \equiv 1 \mod b$ :
        $chunk \leftarrow F_{A[:s]}$
        $F_A \leftarrow F_{A[s:]} \| [0]^s$
        $\kappa_F, \mu_F, \_ \leftarrow \mathsf{MMR.add}(\kappa_F, chunk)$
        // prepare filter MMR authentication paths
        if $j < \lfloor \frac{\kappa_V.n}{b} \rfloor \cdot s$ for some $j \in J = \{\mathsf{H}_w(l \| t \| i) + l \cdot s \mid 0 \leq i < k\}$ :
            $target\_chunks(j) \leftarrow (chunk, \mu_F)$
    return $\mu = (l, r, \mu_V, target\_chunks)$
```

Algorithm 15: $\mathsf{MS}_2.\mathsf{verify}$

```
define $\mathsf{MS}_2.\mathsf{verify}(\kappa = (\kappa_V, \kappa_F, F_A), t, \mu = (l, r, \mu_V, target\_chunks)$ as:
    $c \leftarrow \mathsf{H}(t \| r)$
    if $\mathsf{MMR.verify}(\kappa_V, auth\_path_V, c) = \mathsf{False}$ : return false
    for $j \in J = \{\mathsf{H}_w(l \| t \| i) + \lfloor \frac{l}{b} \rfloor \cdot s \mid 0 \leq i < k\}$ :
        if $j > \lfloor \frac{\kappa_V.n}{b} \rfloor \cdot s$ :
            if $F_{A[j - \lfloor \frac{l}{b} \rfloor \cdot s]} = 1$ : return False
        else
            $chunk, path \leftarrow target\_chunks(j)$
            if $chunk_{[j \mod s]} = 1$ : return False
            if $\mathsf{MMR.verify}(\kappa_F, path, chunk) = \mathsf{False}$ : return False
    return True
```

Algorithm 16: $\mathsf{MS}_2.\mathsf{drop}$

```
define $\mathsf{MS}_2.\mathsf{drop}(\kappa = (\kappa_V, \kappa_F, F_A), t, \mu = (l, r, \mu_V, \mathit{target\_chunks})$ as:
    $J \leftarrow \{\mathsf{H}_w(l \| t \| r \| i) + \lfloor \frac{l}{b} \rfloor \cdot s \mid 0 \leq i < k\}$
    $\pi \leftarrow \Pi.\mathsf{prove}(\{(l, t, r, c, \mu_V) : c = \mathsf{H}(t \| r) \wedge \mathsf{MMR.verify}(\kappa_V, c, \mu_V) \wedge J = \{\mathsf{H}_w(l \| t \| r \| i) + \lfloor \frac{l}{b} \rfloor \cdot s \mid 0 \leq i < k\}\})$
    return $\rho = (J, \pi, \mathit{target\_chunks})$
```

Algorithm 17: $\mathsf{MS}_2.\mathsf{remove}$

```
define $\mathsf{MS}_2.\mathsf{remove}(\kappa = (\kappa_V, \kappa_F, F_A), \rho = (\rho.J, \rho.\pi, \rho.\mathit{target\_chunks}))$ as:
    $\mathit{set\_some\_bit} \leftarrow \mathsf{False}$
    if $\Pi.\mathsf{verify}(\rho.\pi, \{(l, t, r, c, \mu_V) : c = \mathsf{H}(t \| r) \wedge \mathsf{MMR.verify}(\kappa_V, c, \mu_V) \wedge J = \{\mathsf{H}_w(l \| t \| r \| i) + \lfloor \frac{l}{b} \rfloor \cdot s \mid 0 \leq i < k\}\}) = \mathsf{False}$ :
        return $\bot$
    for $j \in J$ :
        if $j > \lfloor \frac{\kappa_V.n}{b} \rfloor \cdot s$ :
            if $F_{A_{[j - \lfloor \frac{l}{b} \rfloor \cdot s]}} = 0$ :
                $\mathit{set\_some\_bit} \leftarrow \mathsf{True}$
                $F_{A_{[j - \lfloor \frac{l}{b} \rfloor \cdot s]}} \leftarrow 1$
        else
            $\mathit{path}, \mathit{chunk} \leftarrow \mathit{target\_chunks}(\lfloor \frac{j}{b} \rfloor)$
            if $\mathsf{MMR.verify}(\kappa_F, \mathit{chunk}, \mathit{path}) = \mathsf{False}$ :
                return $\bot$
            $\mathit{new\_chunk} \leftarrow \mathit{chunk}$
            if $\mathit{new\_chunk}_{[j \mod s]} = 0$ :
                $\mathit{set\_some\_bit} \leftarrow \mathsf{True}$
                $\mathit{new\_chunk}_{[j \mod s]} \leftarrow 1$
                $\kappa_F \leftarrow \mathsf{MMR.modify}(\kappa_F, \mathit{path}, \mathit{chunk}, \mathit{new\_chunk})$
    if $\mathit{set\_some\_bit}$ :
        return $\kappa = (\kappa_V, \kappa_F, F_A)$
    else
        return $\bot$
```

Algorithm 18: MS_2.update

```
1  define $\mathsf{MS}_2.\mathsf{update}(\kappa = (\kappa_V, \kappa_F, F_A), t, \mu = (l, r, \mu_V, \mathit{target\_chunks}), \upsilon)$ as:
2      if $\upsilon : A$ :
3          $\_, \mu_V \leftarrow \mathsf{MMR.update}(\kappa_V, t, (l, \mu_V), \alpha = (\upsilon, \kappa_V.n))$
4          if $\kappa_V.n + 1 \equiv 0 \mod b$ :
               // chunk slides
5              $J \leftarrow \{\mathsf{H}_w(l \| t \| r \| i) + \lfloor \frac{l}{b} \rfloor \cdot s \mid 0 \leq i < k\}\}$
6              if $J \cap \{\lfloor \frac{\kappa.n}{b} \rfloor \cdot s, \ldots, \lfloor \frac{\kappa.n}{b} \rfloor \cdot s + s - 1\} \neq \varnothing$ :
                   // need to keep track of this chunk
7                  $\mathit{slid\_chunk} \leftarrow F_{A[0:w]}$
8                  $\mathit{slid\_mp} \leftarrow \mathsf{MMR.prove}(\kappa_F, \mathit{slid\_chunk})$
9                  $\mathit{target\_chunks}(\kappa_F) \leftarrow (\mathit{slid\_mp}, \mathit{slid\_chunk})$
10     else
           // $\upsilon$ is a removal record
11         $\_, \_, \rho.\mathit{target\_chunks} \leftarrow \upsilon$
12         for $(\rho.\mathit{chunk\_index}, \rho.\mathit{chunk\_mp}, \rho.\mathit{chunk}) \in \rho.\mathit{target\_chunks}$ :
               // prepare modification in SWBF MMR
13             $\varepsilon \leftarrow (\rho.\mathit{chunk}, \rho.\mathit{chunk\_index}, \rho.\mathit{chunk\_path})$
14             for
                $(\mathit{own\_chunk\_index}, \mathit{own\_chunk\_mp}, \mathit{own\_chunk}) \in \mathit{target\_chunks}$ :
15                 if $\rho.\mathit{chunk\_index} = \mathit{own\_chunk\_index}$ :
16                     $\mathit{own\_chunk} \leftarrow \rho.\mathit{chunk}$
17                 else
18                     $\mathit{own\_chunk\_mp} \leftarrow$
                         $\mathsf{MMR.update}(\kappa_F, \mathit{own\_chunk}, \mathit{own\_chunk\_mp}, \varepsilon)$
                   // update own dictionary
19                 $\mathit{target\_chunks}.(\mathit{own\_chunk\_index}) \leftarrow$
                     $(\mathit{own\_chunk\_mp}, \mathit{own\_chunk})$
20     return $\mu = (l, r, \mu_V, \mathit{target\_chunks})$
```

Theorem 6 (security of MS_2). MS_2 *is secure; specifically:*

- commit*, in combination with the standard opening algorithm, has bounded binding insecurity* $\mathsf{InSec}^{\mathsf{commit}}_{\mathsf{bnd}} \leq \mathsf{InSec}^{\mathsf{H}}_{\mathsf{coll}}$*;*
- drop*, in combination with the standard opening algorithm, has bounded binding insecurity* $\mathsf{InSec}^{\mathsf{drop}}_{\mathsf{bnd}} \leq \frac{Q(Q-1)k!}{2w^k} + \mathsf{Adv}^{\mathsf{H}_w}_{\mathsf{PRF}}$*, where Q is the number of queries to* H_w*;*
- *the function* $\mathsf{set\_commit} : \{A, R\}^* \times \{\varnothing\} \rightarrow K$,

$$(O, \varnothing) \mapsto \begin{cases} \mathsf{init}(\boldsymbol{\lambda}) & \Leftarrow O = \varnothing \\ \mathsf{add}(\mathsf{set\_commit}(O^\star, \varnothing), \alpha) & \Leftarrow O = \alpha \| O^\star \wedge \alpha : A \\ \mathsf{remove}(\mathsf{set\_commit}(O^\star, \varnothing), \rho) & \Leftarrow O = \rho \| O^\star \wedge \rho : R \end{cases}$$

  in combination with the standard opening algorithm, has bounded binding insecurity $\mathsf{InSec}^{\mathsf{set\_commit}}_{\mathsf{bnd}} \leq \left(1 - \left(1 - \frac{1}{w}\right)^{\frac{kbw}{s}}\right)^k + \mathsf{Adv}^{\mathsf{H}_w}_{\mathsf{PRF}} + 2 \cdot \mathsf{InSec}^{\mathsf{H}}_{\mathsf{coll}}$*;*
- MS_2 *has bounded non-negativity insecurity* $\mathsf{InSec}^{\mathsf{MS}_2}_{\mathsf{nonneg}} \leq 2 \cdot \varsigma + 3 \cdot \mathsf{InSec}^{\mathsf{H}}_{\mathsf{coll}}$.

Proof. Commit. $\mathsf{MS}_2.\mathsf{commit}$ is identical to $\mathsf{MS}_1.\mathsf{commit}$ and so is its security.

Drop. For the purpose of testing equality both the proof π and the dictionary *target_chunks* are ignored. As a result, the security argument is identical to that of $\mathsf{MS}_1.\mathsf{drop}$.

Set commit. Suppose the adversary finds two sequences of operations $O_1 \neq O_2$ resulting in the same set commitment $\kappa = (\kappa_V, \kappa_F, F_A)$. Then one (or more) of three possibilities must hold:

1. O_1 and O_2 define two different AOCLs even though their MMR set commitment κ_V is identical. This implies a break of the MMR's honest-commitment soundness.
2. O_1 and O_2 define two different SWBFs even though the MMR set commitment to the inactive parts are identical. This implies a break of the MMR's honest-commitment soundness.
3. O_1 and O_2 define identical AOCLs and SWBFs. In this case O_1 and O_2 give rise to an attack against $\mathsf{MS}_1.\mathsf{set\_commit}$.

So relative to $\mathsf{MS}_1.\mathsf{set\_commit}$, $\mathsf{MS}_2.\mathsf{set\_commit}$ incurs a term $2 \cdot \mathsf{Err}^{\mathsf{MMR}}_{\mathsf{hcsnd}} \leq 2 \cdot \mathsf{InSec}^{\mathsf{H}}_{\mathsf{coll}}$ security degradation.

Non-negativity. Suppose that an adversary A_2 wins the non-negativity game for MS_2. Build an adversary A_1 against the non-negativity game for MS_1 as follows. Take the output from A_2 and strip all the Merkle authentication paths and inactive SWBF chunks. Modify the proof π_2 on the removal record by *a)* extracting the witness (l, t, r, c, μ_V) using the extractor E; and *b)* generating π_1 with $\mathit{\Pi}.\mathsf{prove}(\{(l, t, r, c\} : c = \mathsf{H}(t \| r) \wedge c \in V \wedge J = \{\mathsf{H}_w(l \| t \| r \| i) + \lfloor \frac{l}{b} \rfloor \cdot s \,|\, 0 \leq i < k\})$. There are three conditions on which A_1 fails even though A_2 succeeds:

- The extractor E fails to produce a valid witness; the probability of this event is ς.
- The witness produced by E is valid $c \notin V$ even though $\mathsf{MMR.verify}(\kappa_V, c, \mu_V)$. This implies a break of the honest-commitment soundness of the MMR.

– The chunks with bits that are flipped in Line 17 of MS_2.remove do not correspond to the actual SWBF because of a malicious membership proof in the SWBF MMR. This implies a break of the honest-commitment soundness of the MMR.

If none of these conditions are met, then A_1 must be successful. This leads to a bound on the success probability of A_2:

$$\begin{aligned} \Pr[\mathsf{A}_2\checkmark] &\leq \varsigma + 2 \cdot \mathsf{Err}^{\mathsf{MMR}}_{\mathsf{hcsnd}} + \Pr[\mathsf{A}_1\checkmark] && (16) \\ &\leq 2 \cdot \varsigma + 3 \cdot \mathsf{InSec}^{\mathsf{H}}_{\mathsf{coll}} && (17) \end{aligned}$$

□

Theorem 7 (succinctness). MS_2 *is succinct.*

Proof. By construction. All algorithms touch at most a number of nodes in the MMRs that scales logarithmically with the number of addition and removal records. □

Synchronization is definitely succinct when only additions are concerned because in this case the MMRs are only added to. When there can also be removal updates, the SWBF membership proofs must be updated to accomodate for modifications as well. This fact would undermine succinct synchronization if the modifications were arbitrary, but in the case of mutator sets the modifications induced by removal records are not arbitrary. The range of chunks that could be modified is contiguous and can even be small relative to a large number of chunks. Moreover, and specifically to a blockchain context, old UTXOs are less likely to be spent than new UTXOs, and so the older the chunk the less likely it is to change.

5.6 Unlinkable Mutator Sets

The security theorem statements enable a search for parameters in pursuit of a target security level. What they do not do is quantify the amount of privacy generated by a given set of parameters. That is the purpose of this section.

Within one batch, the addition records have identically distributed removal records. It may be a viable strategy to increase the step size s and decrease the active window size w until they meet. At this point, the windows of settable bits of distinct batches are disjunct and there is essentially one Bloom filter per batch.

However, when $s < w$ it is possible that the set of set bits does not uniquely determine which batch a removal record must have come from. This situation is qualitatively more appealing because from the adversary's point of view the batches themselves are only probabilistically delineable. In other words, the adversary is not only incapable of assigning a given removal record (provided it has this property) to its batch index with certainty; but he is also incapable of listing a given record's would-be siblings assuming its batch index is given.

In general, the privacy increases with the active window size w. However, practice requires w be small enough to be stored on the intended machinery. A natural question that arises is, what is the effect of w on privacy? One way to answer this question is to plot a histogram of removal entropy (see Section 3.5).

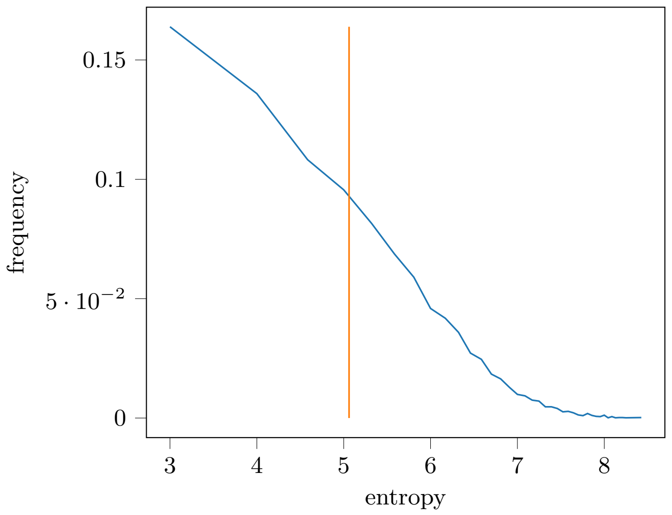

Fig. 4: Frequency of observed entropy.

Figure 4 was generated for parameter set ($w = 2^{20}, b = 8, s = 2^{12}, k = 45$) which incidentally achieves completeness error 2^{-160}. Note that the batch size is $b = 8$, and as removal records are identically distributed with their batch-siblings, the entropy can never be lower than $\log_2 8 = 3$. However, as the entropy is frequently larger, the expectation (indicated by the orange line) is substantially higher. For these parameters, the expected entropy of removal records is slightly larger than 5, roughly equivalent to hiding perfectly behind 31 decoys.

5.7 Implementation Aspects

Reversibility. In a blockchain context, it is possible that what seems to be the consensus now is later overturned as a longer chain is disseminated. In this case nodes must revert the changes induced by the orphaned blocks and apply those induced by the new canonical chain tip. If the mutator set is part of the consensus rules, as described in Section 4, some set bits need to be unset, but only if no other historical removal record set the bit in question. It is easy to support these reversions when the Bloom filter in invertible.

Sparsity. The calculation of the false positive probability in Section 5.3 suggests that the optimal value of k in terms of s and b is $k = \frac{s \cdot \ln 2}{b}$. But k corresponds to the number of indices of set bits sampled pseudorandomly and, importantly, whose integral sampling must be proved. Even if an arithmetization-oriented hash function is used for this purpose, it may still be desirable to reduce the number of required invocations of this primitive at the cost of other trade-offs. In particular, when k is smaller than this optimal value, the Bloom filter may be *sparse* — containing far more zeros than ones even in the inactive part. A straightforward way to represent this *sparse Bloom filter* efficiently is to store the indices of the set bits, and not the bit array itself. Note that this representation affects only performance and does not affect any calculations regarding security or completeness. For the parameters suggested in Section 5.6, the active window consists of at most $\frac{w}{s} \cdot k = 11520$ integers of at most 20 bits, and on average half that. In our implementation this data structure is around 20 kiB in size.

Batching. Updates tend to come in batches rather than individually, for instance because a new block confirms a bunch of transactions rather than one at time. While it is possible to separate a batch of updates into its constituent elements and apply them individually, the update operations stand to benefit from batching them together and reusing intermediate calculations. There are compelling use cases for all permutations of the regular expression $/\mathsf{update}\,[((\mathsf{batch\ of})?(\mathsf{membership\ proof})),\ ((\mathsf{batch\ of})?(\mathsf{removal\ record})),\ (\mathsf{set\ commitment})]\ \mathsf{for}(\mathsf{batch\ of})?\ [(\mathsf{addition}),\ (\mathsf{removal})]\ (\mathsf{reversion})?/$.

Sender and Receiver Randomness. As it is described in Sections 5.4 and 5.5, the randomizer r serves two purposes: *a)* it gives the canonical commitment, and thus the addition record, semantic security; and *b)* it randomizes the set of set bits when a removal record is generated. A non-interactive proof establishes that r links the removal record to the addition record, but as this proof is zero-knowledge, external observers do not learn this link. This construction suffices for exposition, but has two drawbacks. First, the party who chooses r can determine (with reasonable likelihood) the pattern of set bits, giving rise to maliciously increased false positive probability. Second, in a blockchain context, while the sender of a transaction cannot spend the UTXO (after all, the UTXO requires its own separate unlocking logic) he does know the pattern of set bits and can therefore observe when the receiver spends it. Both deficiencies are remedied with the following fix: the receiver selects a secret random preimage $r_p \xleftarrow{\$}$ and disseminates its hash digest $r_d = \mathsf{H}(r_p)$, for instance as part of his receiving address. The sender samples his own secret randomness $r_s \xleftarrow{\$} \mathsf{D}$. The canonical commitment is computed as $\alpha = \mathsf{H}(t \| r_s \| r_d)$, but the bit indices are derived from $\mathsf{H}_w(l \| t \| r_s \| r_p \| i)$ and the zero-knowledge proof additionally establishes that r_p and r_d are correctly related. The sender communicates r_s via an off-chain protocol or uploads a public key encryption of this value to the blockchain such that only the intended receiver can read this secret.

6 Conclusion

Mutator sets capture a notion conceptually in between accumulation schemes and mixnets. While this exposition in this paper focused on the application to blockchain protocols, mutator sets likely also apply to various other schemes and protocols that also rely on an accumulator or on a mixnet. For instance:

– Accumulation schemes can generate dynamic ring and group signature schemes. While there is perfect anonymity among signers that belong to the group, the group itself well defined even from the point of view of external observers. Using a mutator set instead, the group itself can be made fuzzy as external observers can no longer observe who joins and who leaves the group. Mutator sets can thus generate *anonymous group and ring signatures*.
– Accumulation schemes can generate dynamic cryptographically authenticated access control policies, or whitelists. As before, the set of users with access is well defined even from the external observer's point of view. Using mutator sets, the whitelist can hide who is granted access and for whom it is revoked. In other words, mutator sets can generate *anonymous dynamic whitelists*.
– Mixnets are one of the two main constructions for private electronic voting; the other one being additively homomorphic encryption. In traditional mixnet-based voting, authorities are required to shuffle and rerandomize the set of encrypted votes. They are not trusted for integrity because they are required to prove it, but they are trusted for availability. Moreover, the secret decryption key must exist somewhere, even if it is spread across multiple shards. Using a mutator set instead of a mixnet addresses both of these deficiencies: voters can mix the votes themselves, and open their own committed votes. This sketch gives rise to *cryptographic voting without encryption*.

Our construction for mutator sets relies only on hash functions for security and in particular does not use any number-theoretic hard problems. While we do take the zero-knowledge proof for granted, we do note that there adequate proof systems that likewise rely only on hash functions for security. As a result, our construction is plausibly secure against quantum adversaries in addition to classical ones. An interesting open question remains what performance improvements or security tradeoffs come about as a result of migrating to a construction that does rely on number-theoretic hard problems, if such a construction can be found.

References

1. Ayebie, E.B., Souidi, E.M.: New code-based cryptographic accumulator and fully dynamic group signature. Des. Codes Cryptogr. **90**(12), 2861–2891 (2022), https://doi.org/10.1007/s10623-022-01007-5
2. Baldimtsi, F., Camenisch, J., Dubovitskaya, M., Lysyanskaya, A., Reyzin, L., Samelin, K., Yakoubov, S.: Accumulators with applications to anonymity-preserving revocation. In: IEEE EuroS&P, 2017. pp. 301–315. IEEE (2017), https://doi.org/10.1109/EuroSP.2017.13
3. Baldimtsi, F., Karantaidou, I., Raghuraman, S.: Oblivious accumulators. IACR Cryptol. ePrint Arch. p. 1001 (2023), https://eprint.iacr.org/2023/1001

4. Baric, N., Pfitzmann, B.: Collision-free accumulators and fail-stop signature schemes without trees. In: Fumy, W. (ed.) EUROCRYPT '97. Lecture Notes in Computer Science, vol. 1233, pp. 480–494. Springer (1997). https://doi.org/10.1007/3-540-69053-0_33, https://doi.org/10.1007/3-540-69053-0_33
5. Ben-Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from bitcoin. In: IEEE SP 2014. pp. 459–474. IEEE Computer Society (2014). https://doi.org/10.1109/SP.2014.36, https://doi.org/10.1109/SP.2014.36
6. Benaloh, J.C., de Mare, M.: One-way accumulators: A decentralized alternative to digital sinatures (extended abstract). In: Helleseth, T. (ed.) EUROCRYPT '93. Lecture Notes in Computer Science, vol. 765, pp. 274–285. Springer (1993). https://doi.org/10.1007/3-540-48285-7_24, https://doi.org/10.1007/3-540-48285-7_24
7. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Commun. ACM **13**(7), 422–426 (1970). https://doi.org/10.1145/362686.362692, https://doi.org/10.1145/362686.362692
8. Bünz, B., Kiffer, L., Luu, L., Zamani, M.: Flyclient: Super-light clients for cryptocurrencies. In: IEEE SP 2020. pp. 928–946. IEEE (2020), https://doi.org/10.1109/SP40000.2020.00049
9. Camenisch, J., Kohlweiss, M., Soriente, C.: An accumulator based on bilinear maps and efficient revocation for anonymous credentials. In: Jarecki, S., Tsudik, G. (eds.) PKC 2009. Lecture Notes in Computer Science, vol. 5443, pp. 481–500. Springer (2009). https://doi.org/10.1007/978-3-642-00468-1_27, https://doi.org/10.1007/978-3-642-00468-1_27
10. Camenisch, J., Lysyanskaya, A.: Dynamic accumulators and application to efficient revocation of anonymous credentials. In: Yung, M. (ed.) Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings. Lecture Notes in Computer Science, vol. 2442, pp. 61–76. Springer (2002). https://doi.org/10.1007/3-540-45708-9_5, https://doi.org/10.1007/3-540-45708-9_5
11. Chaum, D.: Untraceable electronic mail, return addresses, and digital pseudonyms. Commun. ACM **24**(2), 84–88 (1981), https://doi.org/10.1145/358549.358563
12. Danezis, G., Dingledine, R., Mathewson, N.: Mixminion: Design of a type III anonymous remailer protocol. In: IEEE S&P 2003. pp. 2–15. IEEE Computer Society (2003), https://doi.org/10.1109/SECPRI.2003.1199323
13. Derler, D., Hanser, C., Slamanig, D.: Revisiting cryptographic accumulators, additional properties and relations to other primitives. In: Nyberg, K. (ed.) CT-RSA 2015. Lecture Notes in Computer Science, vol. 9048, pp. 127–144. Springer (2015), https://doi.org/10.1007/978-3-319-16715-2_7
14. Díaz, C., Seys, S., Claessens, J., Preneel, B.: Towards measuring anonymity. In: Dingledine, R., Syverson, P.F. (eds.) Privacy Enhancing Technologies PET 2002. Lecture Notes in Computer Science, vol. 2482, pp. 54–68. Springer (2002). https://doi.org/10.1007/3-540-36467-6_5, https://doi.org/10.1007/3-540-36467-6_5
15. Dingledine, R., Mathewson, N., Syverson, P.F.: Tor: The second-generation onion router. In: Blaze, M. (ed.) USENIX 2004. pp. 303–320. USENIX (2004), http://www.usenix.org/publications/library/proceedings/sec04/tech/dingledine.html
16. Dryja, T.: Utreexo: A dynamic hash-based accumulator optimized for the Bitcoin UTXO set. Cryptology ePrint Archive (2019)

17. Fauzi, P., Meiklejohn, S., Mercer, R., Orlandi, C.: Quisquis: A new design for anonymous cryptocurrencies. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019, Part I. Lecture Notes in Computer Science, vol. 11921, pp. 649–678. Springer (2019). https://doi.org/10.1007/978-3-030-34578-5_23, https://doi.org/10.1007/978-3-030-34578-5_23
18. Friedenbach, M.: Compact spv proofs via block header commitments, https://www.mail-archive.com/bitcoin-development@lists.sourceforge.net/msg04318.html
19. Fuchsbauer, G., Orrù, M.: Non-interactive mimblewimble transactions, revisited. In: Agrawal, S., Lin, D. (eds.) ASIACRYPT 2022, Part I. Lecture Notes in Computer Science, vol. 13791, pp. 713–744. Springer (2022), https://doi.org/10.1007/978-3-031-22963-3_24
20. Fuchsbauer, G., Orrù, M., Seurin, Y.: Aggregate cash systems: A cryptographic investigation of mimblewimble. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part I. Lecture Notes in Computer Science, vol. 11476, pp. 657–689. Springer (2019), https://doi.org/10.1007/978-3-030-17653-2_22
21. Jedusor, T.E.: Mimblewimble (2016), https://docs.beam.mw/Mimblewimble.pdf
22. Kattis, A., Bonneau, J.: Proof of necessary work: Succinct state verification with fairness guarantees. IACR Cryptol. ePrint Arch. p. 190 (2020), https://eprint.iacr.org/2020/190
23. Khedr, W.I., Khater, H.M., Mohamed, E.R.: Cryptographic accumulator-based scheme for critical data integrity verification in cloud storage. IEEE Access **7** (2019). https://doi.org/10.1109/ACCESS.2019.2917628
24. Kiayias, A., Lamprou, N., Stouka, A.: Proofs of proofs of work with sublinear complexity. In: Clark, J., Meiklejohn, S., Ryan, P.Y.A., Wallach, D.S., Brenner, M., Rohloff, K. (eds.) FC 2016. Lecture Notes in Computer Science, vol. 9604, pp. 61–78. Springer (2016), https://doi.org/10.1007/978-3-662-53357-4_5
25. Kiayias, A., Miller, A., Zindros, D.: Non-interactive proofs of proof-of-work. In: Bonneau, J., Heninger, N. (eds.) FC 2020. Lecture Notes in Computer Science, vol. 12059, pp. 505–522. Springer (2020), https://doi.org/10.1007/978-3-030-51280-4_27
26. Libert, B., Ling, S., Nguyen, K., Wang, H.: Zero-knowledge arguments for lattice-based accumulators: Logarithmic-size ring signatures and group signatures without trapdoors. In: Fischlin, M., Coron, J. (eds.) EUROCRYPT 2016, Part II. Lecture Notes in Computer Science, vol. 9666, pp. 1–31. Springer (2016), https://doi.org/10.1007/978-3-662-49896-5_1
27. Libert, B., Ramanna, S.C., Yung, M.: Functional commitment schemes: From polynomial commitments to pairing-based accumulators from simple assumptions. In: Chatzigiannakis, I., Mitzenmacher, M., Rabani, Y., Sangiorgi, D. (eds.) ICALP 2016. LIPIcs, vol. 55, pp. 30:1–30:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2016), https://doi.org/10.4230/LIPIcs.ICALP.2016.30
28. Merkle, R.C.: A digital signature based on a conventional encryption function. In: Pomerance, C. (ed.) CRYPTO '87. Lecture Notes in Computer Science, vol. 293, pp. 369–378. Springer (1987). https://doi.org/10.1007/3-540-48184-2_32, https://doi.org/10.1007/3-540-48184-2_32
29. Miers, I., Garman, C., Green, M., Rubin, A.D.: Zerocoin: Anonymous distributed e-cash from bitcoin. In: IEEE SP 2013. pp. 397–411. IEEE Computer Society (2013). https://doi.org/10.1109/SP.2013.34, https://doi.org/10.1109/SP.2013.34
30. Miller, A.: The high value hashway, https://bitcointalk.org/index.php?topic=98986.0

31. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. Metzdowd Cryptography Mailing List (2008)
32. Nguyen, L.: Accumulators from bilinear pairings and applications. In: Menezes, A. (ed.) CT-RSA 2005. Lecture Notes in Computer Science, vol. 3376, pp. 275–292. Springer (2005), https://doi.org/10.1007/978-3-540-30574-3_19
33. Nyberg, K.: Fast accumulated hashing. In: Gollmann, D. (ed.) FSE 1996. Lecture Notes in Computer Science, vol. 1039, pp. 83–87. Springer (1996), https://doi.org/10.1007/3-540-60865-6_45
34. Papamanthou, C., Tamassia, R., Triandopoulos, N.: Authenticated hash tables based on cryptographic accumulators. Algorithmica **74**(2), 664–712 (2016), https://doi.org/10.1007/s00453-014-9968-3
35. Ramabaja, L., Avdullahu, A.: The bloom tree (2020)
36. Saberhagen, N.V.: Cryptonote v 2.0 (2013), https://www.getmonero.org/resources/research-lab/pubs/cryptonote-whitepaper.pdf
37. Sako, K., Kilian, J.: Receipt-free mix-type voting scheme - A practical solution to the implementation of a voting booth. In: Guillou, L.C., Quisquater, J. (eds.) EUROCRYPT '95. Lecture Notes in Computer Science, vol. 921, pp. 393–403. Springer (1995), https://doi.org/10.1007/3-540-49264-X_32
38. Todd, P.: Merkle mountain ranges, https://github.com/opentimestamps/opentimestamps-server/blob/master/doc/merkle-mountain-range.md