Trading Accumulation Size for Witness Size: A Merkle Tree Based Universal Accumulator via Subset Differences

Mahabir Prasad Jhanwar  
mahavir.jhawar@gmail.com  
Ashoka University, India

Pratyush Ranjan Tiwari  
pratyushranjan.tiwari@gmail.com  
Ashoka University, India

Abstract

Merkle-type trees are widely used to design cryptographic accumulators. The primary advantage in using Merkle tree for accumulators is that they only assume existence of collision-resistant hash functions. Merkle tree based accumulators produces constant size accumulation values. But, the size of the witness is always logarithmic in the number of values accumulated, opposed to the constant size witness as exhibited by some of the other popular accumulators that uses number theoretic techniques and problems. Surprisingly, there exists no Merkle tree based accumulator that provides a trade-off between accumulation size and witness size. Such a trade-off is warranted, as argued in this paper, in a situation where witnesses are stored in memory constrained devices and are being moved around continuously, as opposed to the accumulation values that remain stationary, often in devices with moderate storage capacity. In this paper we propose a Merkle-tree based accumulator scheme assuming only collision-resistant hash functions exist. Our scheme allows witness of size that is in general strictly less than logarithmic in the number of values accumulated, and in some cases reduces to constant size. The trade-off cost results in an increased accumulation size.

1 Introduction

A basic cryptographic accumulator scheme ($\mathcal{AC}$) facilitates optimal verification methods for set-membership relations [BdM93, BP97]. Briefly, given a set $X = \{x_1, \ldots, x_n\}$, an $\mathcal{AC}$ scheme simultaneously does the following tasks: (1) Accumulate: produces a short representation of $X$ denoted as $\text{Acc}_X$, and (2) Membership Witness Generation: for every $x \in X$, it produces an accompanying short membership witness $\text{wit}_x$. Later, by exhibiting the valid tuple $(x, \text{wit}_x, \text{Acc}_X)$, a prover can convince any third party that $x$ is indeed a member of a certain set whose short representation is given by $\text{Acc}_X$. The immediate security requirement is that it should be computationally infeasible to find a valid pair $(x^*, \text{wit}_{x^*})$ for any $x^* \notin X$. Accumulator schemes that in addition support non-membership witness proofs are called universal accumulators [LLX, CHKO12, BLL00]. In particular, for a fixed domain $M$, and for any set $X \subseteq M$, a universal accumulator scheme can produce both a membership witness for element $x \in X$ and a non-membership witness for an element $x^* \in M \setminus X$ that are to be validated against the succinct representation $\text{Acc}_X$ of $X$. Dynamic accumulators [CL02, Ngu05, DT08, CKS09, WWP08, CF13] are an extension that allows computation of $\text{Acc}_{X'}$ using only $\text{Acc}_X$ and $x$, where $X' = X \cup \{x\}$ (addition) or $X' = X \setminus \{x\}$ (removal) and update existing witnesses accordingly, without the need to fully recompute these values on each addition or removal.

Accumulators have proven to be a very strong mathematical tool with applications in a variety of privacy preserving technologies such as efficient time-stamping [BdM93], accountable certificate management [BLL00], authenticated dictionaries [GTH02]. Accumulators are also
used as building block in anonymous credential systems and group signatures [Nyb96, Ngu05, CKS09], ring signatures [DKNS04], redactable signatures [PS14], sanitizable signatures [CJ10], P-homomorphic signatures [ABC+12], and Zerocoin [MGGR] (an extension of the cryptographic currency Bitcoin), etc.

Building accumulator schemes whose security is based on cryptographic hardness assumptions that are both standard and minimal is an important goal in this area. Number theoretic problems such as the strong RSA problem [BdM93, BP97], discrete logarithm problem variants [Ngu05, DT08, CKS09, GOP+16, AN11], Paillier trapdoor permutation [WWP08], lattices and others [LLNW16, JS15, CF13, TX03, Lip12] have been used to construct several accumulator schemes. These schemes achieve better functionality and are compact i.e., they produce constant accumulation size and witness size. But, on the other hand, these schemes are also limited by the fact that the underlying assumptions are often non-standard and strong.

There exists a line of work that employ only collision-resistant hash functions to build secure accumulator schemes [BLL00, BLL02, CHKO12, BC14]. Unlike number theoretic constructions, these schemes do not require the accumulator manager to be trusted. But, a bottleneck for these schemes is that they are no longer compact - there is an increase in witness size which is logarithmic in the number of values accumulated. In this setting the schemes are designed based on Merkle type trees (also called hash trees). A Merkle tree is a labeled tree, with the leaves labeled by different values $H(x)$, where $x \in X$ and $H$ is a collision-resistant hash function. The labels of sibling nodes are hashed using $H$ in order to compute the label associated to their parent node, and so on and so forth, until a value for the root of the tree is obtained. The tree’s root value is then defined as the accumulator value of the set of values associated to the leaves of the tree. For example, given a set $X = \{x_1, x_2, x_3, x_4\}$, the short representation of $X$ is the value $Acc_X = H(H(H(x_1)||H(x_2))||H(H(x_3)||H(x_4)))$. A witness $wit_{x_i}$ that an element $x_i$ has in a set whose short representation is $Acc_X$ is the set of $O(\log |X|)$ nodes along the Merkle tree needed to trace the exact path from $H(x_i)$ to the root node.

1.1 Our Contributions

A typical straightforward application of accumulators is that they can be used to implement membership testing systems such as authenticated dictionaries. The later system involve three parties: a trusted source, an untrusted directory, and a user. The source defines a finite set $X \subseteq M$ of elements. A short representation $Acc_X$ of $X$ is published, and users can obtain it in an authenticated manner. The directory maintains the sets $\{wit_x \mid x \in X\}$ and $\{wit_x \mid x \in M \setminus X\}$. The user performs membership queries on the set $X$ of the type “is element $x$ in set $X$?”. To experience faster response and avoid network latency, instead of contacting the source directly, the users query the directory. The directory provides the user with a yes/no answer to the query together with a witness $wit_x$, which yields a proof of the answer. The user then verifies the proof based on $x, wit_x$, and $Acc_X$. Another typical membership testing system is the usual plain authentication system where the parties involve are users and a resource carrying system holding all user-credentials. The set $X$ defines the collection of all user-identities. The resource system stores $Acc_X$. Witnesses $wit_x$ are distributed to each user $x \in X$. Later, in order to access the resource system, a user can prove membership in $X$ by revealing their identity $x$ and the witness $wit_x$. It is important to note that, in both applications, the witnesses are continuously moved around, whereas the accumulation data $Acc_X$ remains static. To instantiate these systems, the number theoretic accumulators will prove to be a better choice then the simple Merkle tree based accumulators. The later schemes will make these systems communication heavy due to the increased witness sizes. An immediate question that can be asked here is that can we have a Merkle tree based accumulator scheme that can trade accumulation size for witness size. An
increase in accumulation size for such a trade-off scheme is tolerable given that the accumulation data is static and is stored in devices that are not resource constrained. The decrease in witness size on the other hand is welcome as witnesses are moved around continuously and often stored in resource constrained devices such as smart cards.

To our best knowledge, there exists no Merkle tree based accumulator scheme that trades accumulation size for witness size. In this paper, we propose a Merkle tree based accumulator scheme that achieves the same. Our scheme allows witness size that is in general strictly smaller than a size that is logarithmic in the number of values accumulated. This is achieved at a cost that increases the accumulation size. The novelty of our construction is that it employs, for the first time, a well known subset covering technique called subset difference method [NNL01] to the setting of Merkle tree to achieve this trade-off.

2 Preliminaries

Definition 1 (Collision-resistant Hash Family) A \( \lambda \)-bit hash function family is keyed function family \( \mathcal{H}_\lambda = \{ H_k : \{0,1\}^* \rightarrow \{0,1\}^\lambda \}_{k \in \mathbb{K}} \). The function family \( \mathcal{H}_\lambda \) is said to be collision resistant if, for every polynomial time adversaries \( A \), there exists a negligible function \( \text{negl}(\lambda) \) such that
\[
P[k \leftarrow \mathbb{K}; x_1, x_2 \leftarrow A(k) \mid x_1 \neq x_2 \text{ and } H_k(x_1) = H_k(x_2)] \leq \text{negl}(\lambda)
\]

2.1 Merkle Trees

A tree is a simple, undirected, connected graph without cycles. We particularly consider rooted trees, i.e., trees with a distinguished root node. The nodes adjacent to the root node are called its children; each child can be considered, in turn, the root of a subtree. Children of the same node are called siblings of each other. Nodes that have no children are called leaves, and all other nodes are called internal. A rooted tree is 2-regular or binary if each node is either a leaf or possesses exactly two child nodes. The level/depth \( L \) of a node indicates its distance to the root, where we assign level \( L = 0 \) to the root node. In this paper we focus on binary trees of constant depth \( d \), i.e., where all leaves have the same level \( L = d \). We denote such a tree by \( \mathbb{B}_d \). We let \( \mathbb{L}(\mathbb{B}_d) \) denotes the set of all leaf nodes of \( \mathbb{B}_d \) and its size is given by \( |\mathbb{L}(\mathbb{B}_d)| = 2^d \).

The total number of nodes in \( \mathbb{B}_d \) is \( 2^{d+1} - 1 \). We let \( \mathbb{N} = 2^d \) and label nodes of \( \mathbb{B}_d \) using numbers in \( 1, \ldots, 2\mathbb{N} - 1 \) as follows: the root node is labeled with 1, if parent is labeled with \( i \) then the left child is labeled with \( 2i \) and the right child is labeled with \( 2i + 1 \). We let \( v_i \) denote the node labeled with \( i \), \( 1 \leq i \leq 2\mathbb{N} - 1 \); for example the root node is denoted by \( v_1 \). For a node \( v_i \), \( \text{sib}(v_i) \) denotes its sibling, \( \text{parent}(v_i) \) denotes the parent node of \( v_i \), and \( \text{child}(v_i) \) (when \( v_i \) is a non-leaf node) denotes a child node of \( v_i \).

For a node \( v_i \in \mathbb{B}_d \), the subtree rooted at \( v_i \) is denoted by \( T_i \), and \( S_i \) denotes the set of all leaf nodes of \( T_i \), i.e., \( S_i = \mathbb{L}(T_i) \). For any \( v_i, v_j \in \mathbb{B}_d \), we let \( T_{i,j} \) denote the subtree \( T_i \setminus T_j \) - the subgraph obtained by taking away \( T_j \) from \( T_i \), and subsequently \( S_{i,j} = \mathbb{L}(T_{i,j}) \).

Definition 2 (\( \mathbb{B}_{d,M} \)) Let \( M = \{x_0, \ldots, x_{2d-1}\} \) and \( H : M \times M \rightarrow M \) be a collision-resistant hash function. A perfect full binary tree \( \mathbb{B}_d \) is said to model \( M \) under \( H \) if leaf nodes of \( \mathbb{B}_d \) are set to the following values: \( v_i = H(x_i \mod (2^d)) \), \( 2^d \leq i \leq 2^{d+1} - 1 \). The resulting tree will be denoted by \( \mathbb{B}_{d,M} \).

Definition 3 (Merkle Tree) Let \( M = \{x_0, \ldots, x_{2d-1}\} \) and \( H : M \times M \rightarrow M \) be a collision-resistant hash function. Let \( \mathbb{B}_{d,M} \) models \( M \) under \( H \). The \( \mathbb{B}_{d,M} \) is said to be Merkle tree if
the internal nodes of \( BT_{d,M} \) are set recursively as follows

\[
v_i = H(v_{2i}||v_{2i+1}), \quad i = 2^d - 1, \ldots, 1
\]

The resulting tree is denoted by \( MT_{d,M} \). For example, the Figure 1 represents \( MT_{3,M} \).

Definition 4 (Exact Path) Let \( u, u' \in MT_{d,M} \) be two distinct nodes such that \( u \) is an ancestor of \( u' \). The exact path from \( u' \) to \( u \), denoted as \( EP_{u' \rightarrow u} \), is a sequence of nodes \( EP_{u' \rightarrow u} = (u_{\ell - 1} = u', u_{\ell - 2}, \ldots, u_1, u_0 = u) \) such that, for \( i = 1, \ldots, \ell \), \( u_{i-1} \) is a child of \( u_i \). For example, in Figure 1, the exact path \( EP_{v_{10} \rightarrow v_1} = (v_{10}, v_5, v_2, v_1) \).

Definition 5 (Pseudo Path) For an exact path \( EP_{u' \rightarrow u} = (u_{\ell - 1} = u', u_{\ell - 2}, \ldots, u_1, u_0 = u) \) in \( MT_{d,M} \), its corresponding pseudo path, denoted as \( PP_{u' \rightarrow u} \), is a sequence of nodes \( PP_{u' \rightarrow u} = (v_{\ell - 1}, v_{\ell - 2}, \ldots, v_1) \) such that

\[
H(...H(H(u_{\ell-1}||v_{\ell-2})||v_1)) = u
\]

For example, in Figure 1, the pseudo path \( PP_{v_{10} \rightarrow v_1} = (v_{11}, v_4, v_3) \).

![Figure 1: Merkle Tree](image)

Definition 6 (Steiner Tree) Let \( R \subseteq LN(BT_d) \). The Steiner tree \( ST(R) \) induced by \( R \) on \( BT_d \) is a subgraph of \( BT_d \) that only retains nodes and edges present on the exact paths from the root node \( v_1 \) to leaf nodes in \( R \) respective.

### 2.2 Subset Covering

The construction of our accumulator scheme uses a well known subset covering algorithm, called subset difference (SD) method, due to Naor, Naor and Lotspiech (NNL) [NNL01]. In [NNL01], a broadcast encryption was proposed using subset difference method. In the following we first recall the concept of subset covering and then present the NNL subset difference method.

Definition 7 (Subset Covering) For a non-empty set \( M \), a subset-cover algorithm defines a collection of subsets \( S \subseteq 2^M \) such that the following holds: for any \( R \subseteq M \) there exits a sub collection \( CV_R = \{ S_1, \ldots, S_m \} \) that partition \( M \setminus R \), i.e.,

\[
M \setminus R = \bigcup_{S_i \in CV_R} S_i \text{ and } S_i \cap S_j = \emptyset \text{ for every } S_i, S_j \in CV_R, i \neq j.
\]
2.2.1 Subset Difference [NNL01]: A Subset Covering Method

The subset difference method is given by the algorithms (SD.SetUp, SD.Cover):

- **SD.SetUp**(M): The input to the algorithm is a set \( M = \{x_0, \ldots, x_{N-1}\} \), where \( N = 2^d \) and \( d \in \mathbb{N} \). It defines a collection of subsets \( S \subseteq 2^M \) as follows:
  - Construct a binary tree \( BT_d \) of constant depth \( d \). Assign elements of \( M \) to its leaf nodes, i.e., for \( i \) in \( 2^d \leq i \leq 2^{d+1} - 1 \), leaf node \( v_i = x_{i \mod (2^d)} \).
  - Define \( S = \{S_{i,j} \mid (v_i, v_j \in BT_d) \land (v_i \text{ is an ancestor of } v_j)\} \), where \( S_{i,j} = LN(T_i \setminus T_j) \).

- **SD.Cover**(BT\(_d\), R): This algorithm takes a set \( R \subseteq M \) as input and computes a cover \( CV_R \subseteq S \) for \( M \setminus R \) i.e., \( CV_R \) partitions \( M \setminus R \). The algorithm works iteratively: at each stage it removes nodes from the Steiner tree \( ST_R \) induced by \( R \) while building \( CV_R \) and finally stops when the updated \( ST_R \) is left with only one node.
  1. Set \( CV_R = \phi \), the empty set.
  2. Find two leaves \( v_i, v_j \) in \( ST_R \) such that the least common ancestor \( lca(v_i, v_j) \) of \( v_i \) and \( v_j \) satisfies the following property: the set of leaf nodes of the subtree, rooted at \( lca(v_i, v_j) \), does not contain any leaf node from \( ST_R \) other than \( v_i, v_j \). Suppose \( lca(v_i, v_j) = v_t \), where \( t \in [2N - 1] \). Then \( v_{2t} \) and \( v_{2t+1} \) are respectively the left and the right child of \( v_t \).
  2a. If \( v_{2t} \neq v_i \), then update \( CV_R = CV_R \cup \{S_{2t,i}\} \); likewise, if \( v_{2t+1} \neq v_j \), update \( CV_R = CV_R \cup \{S_{2t+1,j}\} \).
  2b. Update \( ST_R \) by removing all descendants of \( v_t \). The updated \( ST_R \) now has \( v_i \) as its leaf.
  2c. Go to Step 2.
  3. If \( ST_R \) is left with only one leaf, then do the following. Let \( v_1 \) be the leaf. \( ST_R \) is set to \( v_1 \), the root. Update \( CV_R = CV_R \cup \{S_{1,i}\} \).
  4. The algorithm stops by outputting the updated \( CV_R \).

**Lemma 1 (Correctness [NNL01])** The sub-collection \( CV_R \), as described above, partitions \( M \setminus R \), i.e., \( M \setminus R = \cup_{i,j} CV_R \cap S_{i,j} \) and \( S_{i,j} \cap S_{i',j'} = \phi \) for every \( S_{i,j}, S_{i',j'} \in CV_R \).

**Lemma 2 (Covering Size [NNL01])** For any \( R \subseteq LN(BT_d) \), the size of the covering set is \( |CV_R| \leq 2r - 1 \), where \( r = |R| \). Furthermore, if the set \( M \) is random, then the average number of subsets in \( CV_R \) is 1.25\( r \).

In Figure 2 below, we depict a sample run of the subset difference method.

### 2.3 Cryptographic Accumulators

A static universal accumulator scheme allows to produce a short representation of a large set \( X \subseteq M \), called accumulator/accumulation of \( X \) and is denoted by \( \text{Acc}_X \), subject to which the scheme can also produce, for every \( x \in M \), a witness \( \text{wit}_x \) attesting to the fact that \( x \in X \). Based on \( x, \text{wit}_x \) and \( \text{Acc}_X \), anyone can later verify whether \( x \in X \) or \( x \in M \setminus X \). We now give a formal definition of a universal cryptographic accumulator scheme.

**Definition 8 (Static Universal Accumulator Scheme)** A static universal accumulator scheme \( \mathcal{A} \) for a domain \( M \) is a tuple of PPT algorithms (Setup, Accumulate : \( \mathcal{P}(M) \to A \), WitGen : \( M \times \mathcal{P}(M) \to W \), Verify : \( X \times W \times A \to \{\text{mem}, \text{non-mem}, \perp\} \), where \( \mathcal{P}(M) = \{X \mid X \subseteq M\} \) is the power set of \( M \), \( A \) is the domain for accumulation values, and \( W \) is the domain for witnesses. The algorithms work as follows.
Figure 2: Subset Difference Method: For $M = \{x_0, \ldots , x_7\}$, $BT_3$ is constructed. For every $i$ in $1 \leq i \leq 15$, the node with label $i$ is denoted by $v_i$. The elements in $M$ are assigned to leaf nodes as discussed above. Take $R = \{v_8, v_{11}, u_{13}, v_{14}\}$. Then, the Steiner tree $ST_R$ induced by $R$ is given by color-filled nodes. The covering set $CV_R$ is constructed iteratively as follows. Consider a pair of leaf nodes $(v_8, v_{11})$ in $ST_R$. The $lca(v_8, v_{11}) = v_2$ and the subtree (of $ST_R$) rooted at $v_2$ has no leaf nodes from $ST_R$ other than $v_8, v_{11}$. For the children $v_4, v_5$ of $v_2$, the $CV_R$ is set to $CV_R = \{S_{4,8}, S_{5,11}\}$. The $ST_R$ is updated by removing all descendants of $v_2$ and making it a leaf node. Next, for $(v_{13}, v_{14})$, similarly $S_{6,13}$ and $S_{7,14}$ are added to $CV_R$ and making $v_6$ a leaf node for $ST_R$. The updated $ST_R$ is now left with leaf nodes $v_2, v_3$ and the root $v_1$. Step 2 of SD.Cover($BT_d, R$) finally updates $ST_R$ such that it is left with only $v_1$. The cover is finally given by $CV_R = \{S_{4,8}, S_{5,11}, S_{6,13}, S_{7,14}\}$.

- **Setup($\lambda$):** This algorithm is run by the accumulation manager. It takes as input a security parameter $\lambda$, and the domain set $M$. The algorithm outputs an auxiliary information $aux_M$. The $aux_M$ will be an implicit input to both Accumulate and WitGen.
- **Accumulate($X, aux_M$):** This algorithm is run by the accumulator manager. It takes as input a set $X \subseteq M$ and produces a succinct representation $Acc_X$ of $X$, also called the accumulation of $X$.
- **WitGen($x, X, aux_M$):** This algorithm is run by the accumulation manager. It takes as input an element $x \in M$, a set $X \subseteq M$ and produces a witness $wit_x \in W$.
- **Verify($x, w_x, c$):** This algorithm is run by any third party. It takes as input an element $x \in M$, a witness $wit_x$ and an accumulation value $Acc_X$, and it outputs “mem/non-mem/⊥”.

**Definition 9 (Correctness)** The correctness of a universal accumulator scheme requires that for valid $aux_M \leftarrow$ Setup($\lambda$), $X \subseteq M$, and $x \in M$, the following holds true:

$$
\text{Verify}(x, \text{WitGen}(x, X, aux_M), \text{Accumulate}(X, aux_M)) = \begin{cases} 
\text{mem}, & \text{if } x \in X \\
\text{non-mem}, & \text{if } x \in M \setminus X \\
\bot, & \text{otherwise}
\end{cases}
$$

**Definition 10 (Security)** A universal accumulator scheme is called secure if, for all domain sets $M$, all $\lambda \in \mathbb{N}$, and for all polynomial time (in $\lambda$) adversaries $A$, there exists a negligible function $\text{negl}(\lambda)$ such that:

$$
P[aux_M, \leftarrow \text{Setup}(\lambda); (X^*, x^* \in M, \text{wit}_{x^*}, \text{wit}'_{x^*}) \leftarrow A(aux_M) \mid \text{Verify}(x^*, \text{wit}_{x^*}, \text{Acc}_{X^*}) = \text{mem} \land \text{Verify}(x^*, \text{wit}'_{x^*}, \text{Acc}_{X^*}) = \text{non-mem}] \leq \text{negl}(\lambda). \tag{1}
$$

where the probability $P[\cdot]$ is computed over the randomness of the algorithms.
3 A Universal Accumulator Scheme via Subset Difference Method

We now present our scheme. The parameters of our scheme involves:

- a security parameter $\lambda \in \mathbb{N}$;
- a message set $M = \{x_0, \ldots, x_{N-1}\}$ (we assume $N = 2^d$ for some $d \in \mathbb{N}$);
- a family $\mathcal{H}_\lambda = \{H_k : \{0,1\}^* \rightarrow \{0,1\}^\lambda\}$ of $\lambda$-secure collision-resistant hash functions;

The scheme is described as follows:

- **SetUp($1^\lambda$):** Given $\lambda$ and $M = \{x_0, \ldots, x_{N-1}\}$ as inputs, it proceeds as follows:
  - Sample a $\lambda$-secure collision-resistant hash function $H = H_k : \{0,1\}^* \rightarrow \{0,1\}^\lambda \in \mathcal{H}_\lambda$.
  - Build a binary tree $\mathbf{BT}_{d,M}$ modelling $M$ under $H$ (see Definition 2).
  - Turn $\mathbf{BT}_{d,M}$ into Merkle tree $\mathbf{MT}_{d,M}$ (see Definition 3).
  - The algorithm sets auxiliary information $\text{aux}_M$ to $\mathbf{MT}_{d,M}$ and $H$. The $\text{aux}_M$ will be an implicit input to both Accumulate and WitGen algorithms.

- **Accumulate($X \subseteq M, \text{aux}_M$):** For an arbitrary set $X \subseteq M$, its accumulation value $\text{Acc}_X$ is generated as follows:
  - Let $X = \{x_{j_0}, \ldots, x_{j_t-1}\}$ for some $j_0, \ldots, j_{t-1} \in \{0, \ldots, N-1\}$. Using $\text{aux}$, find $i_k$ in $2^d \leq i_k \leq 2^{d+1}-1$ ($k = 0, \ldots, t-1$) such that $v_{i_k} = H(x_{j_k}) \in \text{LN}(\mathbf{BT}_{d,M})$.
  - Run subset cover algorithm ($\S$ 2.2.1) $\text{SD.Cover}((\mathbf{BT}_{d,M}, R')$ on $R'$ to obtain $\mathbf{CV}_{R'}$.
  - For each $(i_k, j_k) \in \text{Index}(\mathbf{CV}_{R'})$, get $v_{i_k}, v_{j_k}$ (with the associated values) from the Merkle tree $\mathbf{MT}_{d,M}$.
  - Finally, the accumulation of $X$ is set to:
    \[ \text{Acc}_X = \{(\text{Mem} = (v_{i_1}, \ldots, v_{i_r}), \text{Non-Mem} = (v_{j_1}, \ldots, v_{j_r})\}. \]

- **WitGen($x \in M, X \subseteq M, \text{Acc}_X, \text{aux}_M$):** On input $x \in M$, it computes a witness as follows.
  - (Case 1) $x \in X$: In this case, a membership witness is issued as follows. As $\mathbf{CV}_{R'}$ partitions $X'$, there exists a unique $(i, j) \in \text{Index}(\mathbf{CV}_{R'})$ for which $S_{i,j}$ has a leaf node $v_\nu = H(x)$, where $2^d \leq \nu < 2^{d+1}$. The membership witness is set to the pseudo path from $v_\nu$ to $v_i$ as follows:

$^1$The auxiliary information is to be stored by the accumulator manager who issues accumulator values and generates membership witnesses. The aux is not used to verify correctness of accumulation values nor to check the validity of membership witnesses.
For all domain sets $3.1$ Security, there exists a negligible function $\text{negl}$. A toy example describing an instance of our scheme is discussed in Figure 3.

\[
\text{wit}_x = PP_{v_0 \rightarrow v_1} = \left( (\text{sibl}(v_0), (-1)^{\text{lbl}(v_0)}), (\text{sibl}(\text{prnt}(v_0)), (-1)^{\text{lbl}(\text{prnt}(v_0))}), \right.

\left. (\text{sibl}(\text{prnt}(\text{prnt}(v_0))), (-1)^{\text{lbl}(\text{prnt}(\text{prnt}(v_0))))}, \ldots, (\text{nc}_{v_0}(v_i), (-1)^{\text{lbl}(\text{nc}_{v_0}(v_i)))}) \right),
\]

where $\text{nc}_{v_0}(v_i)$ is the children of $v_i$ which is not on the exact path from $v_i$ to $v_0$, and $\text{lbl}_v$ denotes the label of the node $v$.

- (Case 2) $x \in M \setminus X$: In this case a non-membership witness is issued as follows. There exists a unique $S_{i,j} \in \text{CV}_{RT}$ such that $T_j$ has the leaf node $v_\nu = H(x)$, where $2^d \leq \nu < 2^{d+1}$ and $\text{LN}(T_j) \cap X' = \emptyset$. The non-membership witness is set to the pseudo path from $v_\nu$ to $v_j$ as follows:

\[
\text{wit}_x = PP_{v_\nu \rightarrow v_j} = \left( (\text{sibl}(v_\nu), (-1)^{\text{lbl}(v_\nu)}), (\text{sibl}(\text{prnt}(v_\nu)), (-1)^{\text{lbl}(\text{prnt}(v_\nu))}), \right.

\left. (\text{sibl}(\text{prnt}(\text{prnt}(v_\nu))), (-1)^{\text{lbl}(\text{prnt}(\text{prnt}(v_\nu))))}, \ldots, (\text{nc}_{v_\nu}(v_j), (-1)^{\text{lbl}(\text{nc}_{v_\nu}(v_j)))}) \right),
\]

where $\text{nc}_{v_\nu}(v_j)$ is the children of $v_j$ which is not on the path from $v_j$ to $v_\nu$, and $\text{lbl}_v$ denotes the label of the node $v$.

- **Verify**($x, \text{wit}_x, \text{Acc}_X$): Suppose, $\text{wit}_x = \left( (v_{\eta_1}, \tau_1), (v_{\eta_{\ell - 1}}, \tau_{\ell - 1}), \ldots, (v_{\eta_1}, \tau_1) \right), \ell \leq d$. The verification algorithm proceeds as follows: Let $V_\ell = H(x)$. It computes the exact path from $V_\ell$ to a node in Mem/Non-Mem as follows. Recursively compute $V_i$'s, $i = \ell - 1, \ldots, 0$, the internal nodes on the exact path from $V_\ell$ to this node as follows:

\[
V_i = \left\{ \begin{array}{ll}
H(V_{i+1}, v_{\eta_{i+1}}) & \tau_{i+1} = -1 \\
H(v_{\eta_{i+1}}, V_{i+1}) & \tau_{i+1} = 1 
\end{array} \right.
\]

Thus, $\text{EP}_{V_\ell \rightarrow V_0} = (V_\ell, V_{\ell - 1}, \ldots, V_1, V_0)$. The algorithm finally outputs “mem”/“non-mem”/“⊥” as follows:

- Case 1: $V_0 = v_{i_k} \in \text{mem}$ for some $k$ in $1 \leq k \leq r$.

\[
\text{Output} = \left\{ \begin{array}{ll}
\bot, & \text{if } \exists \eta \text{ an } \eta \in 1 \leq \eta \leq \ell - 1 \text{ with } V_\eta \in \text{non-mem} \\
\text{mem}, & \text{otherwise}
\end{array} \right.
\]

(2)

- Case 2: $V_0 = v_{j_k} \in \text{non-mem}$ for some $k$ in $1 \leq k \leq r$.

\[
\text{Output} = \left\{ \begin{array}{ll}
\bot, & \text{if } \exists \eta \text{ an } \eta \in 1 \leq \eta \leq \ell - 1 \text{ with } V_\eta \in \text{mem} \\
\text{non-mem}, & \text{otherwise}
\end{array} \right.
\]

(3)

- Output $\bot$, otherwise.

A toy example describing an instance of our scheme is discussed in Figure 3.

### 3.1 Security

**Theorem 1** For all domain sets $M$, all $\lambda \in \mathbb{N}$, and for all polynomial time (in $\lambda$) adversaries $\mathcal{A}$, there exists a negligible function $\text{negl}(\lambda)$ such that:

\[
P[\text{aux}_M, \leftarrow \text{Setup}(1^\lambda); (X^* \subseteq M, x^* \in M, \text{wit}_{x^*}, \text{wit}'_{x^*}) \leftarrow \mathcal{A}(\text{aux}_M) \mid \text{Verify}(x^*, \text{wit}_{x^*}, \text{Acc}_{X^*}) = \text{mem} \land \text{Verify}(x^*, \text{wit}'_{x^*}, \text{Acc}_{X^*}) = \text{non-mem}] \leq \text{negl}(\lambda),
\]

where the probability $P[\cdot]$ is computed over randomness in the Setup algorithm.
Figure 3: A Toy Example: (1) **SetUp**: For $M = \{x_0, \ldots, x_7\}$, construct the Merkle tree $MT_{d,M}$ under $H$ as above. (2) **Accumulate**: Let $X = \{x_1, x_2, x_6, x_7\}$. Compute $R' = LN(BT_3) \setminus X' = \{v_8, v_{11}, v_{12}, v_{13}\}$, where $X' = \{H(x) \mid x \in X\}$. The color filled nodes denote the Steiner tree $ST_{R'}$ induced by $R'$. Run SD.Cover($BT_3, R'$) to obtain $CV_{R'} = \{S_{4,8}, S_{5,11}, S_{6,8}\}$. The accumulation of $X$ is set to $Acc_X = \{Mem = (v_4, v_5, v_3), Non-Mem = (v_8, v_{11}, v_6)\}$. (3) **WitGen**: Membership witnesses to $x_1, x_2, x_6$ and $x_7$ are given by $wit_{x_1} = (v_8, (-1)^8)$, $wit_{x_2} = (v_{11}, (-1)^{11})$, $wit_{x_6} = ((v_{15}, (-1)^{15}), (v_6, (-1)^6))$ and $wit_{x_7} = ((v_{14}, (-1)^{14}), (v_6, (-1)^6))$ respectively. Non-membership witness to $x_0, x_3, x_4, x_5$ are given by $wit_{x_0} = (v_8)$, $wit_{x_3} = (v_{11})$, $wit_{x_4} = (v_{13}, (-1)^{13})$ and $wit_{x_5} = (v_{12}, (-1)^{12})$ respectively. Note that, all witnesses are strictly smaller than the logarithmic height $d = 3$. (4) **Verify**: On input $(x_1, wit_{x_1} = (v_8), 1)$, the verification step outputs $mem$ by checking that $H(v_8, H(x_1)) = H(v_8, v_9) = v_4 \in mem$ and $EP_{v_9 \rightarrow v_4} = (v_9, v_4)$ has no internal nodes on it from $Non-Mem$. Similarly, $Verify(x_6, wit_{x_6} = ((v_{15}, -1), (v_6, 1)))$ outputs $mem$ as $H(v_6, H(x_6), v_{15}) = H(v_6, H(v_{14}, v_{15})) = H(v_6, v_7) = v_3 \in Mem$ and $EP_{v_{14} \rightarrow v_3} = (v_{14}, v_7, v_3)$ has no internal nodes coming from $Non-Mem$. Where as, $Verify(x_4, wit_{x_4} = (v_{13}, (-1)^{13}))$ outputs $non-mem$ as $H(H(x_4), v_{13}) = H(v_{12}, v_{13}) = v_6 \in Non-Mem$ and $EP_{v_{12} \rightarrow v_6} = (v_{12}, v_6)$ has no internal nodes on it from $Mem$. Similarly, $Verify(x_0, wit_{x_0} = (v_8))$ outputs $non-mem$ as $H(x_0) = v_8 \in Non-Mem$.

**Proof**: An immediate way to attack the scheme is when the adversary can find a pair of elements $x_1, x_2 \in M$ such that $H(x_1) = H(x_2)$. It can then choose an $X^* \subseteq M$ such that $x_1 \in X^*$ and $x_2 \in M \setminus X^*$. It finally sets $x^* = x_1$, membership witness $wit_{x^*}$ to the membership witness $wit_{x_1}$ for $x_1$, and non-membership witness $wit'_{x^*}$ to the non-membership witness $wit_{x_2}$ for $x_2$. Clearly, $Verify(x^*, wit_{x^*}, Acc_{X^*}) = Verify(x_1, wit_{x_1}, Acc_{X^*}) = mem$, and $Verify(x^*, wit'_{x^*}, Acc_{X^*}) = Verify(x_1, wit_{x_2}, Acc_{X^*}) = non-mem$. But the probability that adversary can find such a collision is negligible due to the collision resistant property of the underlying hash family. Therefore, except a negligible probability, we assume that for every $x_1, x_2 \in M$, $x_1 \neq x_2$ implies $H(x_1) \neq H(x_2)$.

We now show that, for any $x \in M$, $X \subseteq M$, an adversary cannot simultaneously produce both membership and non-membership witnesses. The proof below considers the following two cases. 

**Case1**: Assume $x \in X$. Thus $H(x) \in X'$. As $CV_{R'}$ partition $X'$, there exists a unique $S_{i,j} \in CV_{R'}$ such that $H(x) \in S_{i,j} = LN(T_i, T_j)$. Therefore a valid membership witness for $x$ exists and it is equal to $PP_{H(x) \rightarrow v_i}$, where $v_i$ is the root node for subtree $T_i$ (see Figure 4). We now show that a non-member witness for $x$ cannot be issued simultaneously. Assuming otherwise, let $wit'_{x^*}$ be such that $Verify(x, wit'_{x^*}, Acc_{X^*}) = non-mem$. This implies $wit'_{x^*} = PP_{H(x) \rightarrow v_k}$, where $v_k \in Non-Mem$ and $EP_{H(x) \rightarrow v_k}$ doesn’t have any internal node belonging to $Mem$. Clearly, $k > i$. Therefore $k \leq i$. But, this implies that the exact path $EP_{H(x) \rightarrow v_k}$ computed using $PP_{H(x) \rightarrow v_k}$, must contain $v_i$ as an internal node. This is true as $H(x)$ is a leaf node of $T_i$. This is a contradiction as $v_i \in Mem$ and therefore $Verify(x, wit'_{x^*}, Acc_{X^*})$ will output $\bot$.

**Case2**: Assume $x \in M \setminus X$. There exists a unique $S_{i,j} \in CV_{R'}$ such that $H(x)$ is the leaf node of $T_j$ and $LN(T_j) \cap X' = \emptyset$. Therefore a valid non-membership witness for $x$ exists and it is equal to $PP_{H(x) \rightarrow v_j}$, where $v_j$ is the root node for subtree $T_j$. We now show that a membership witness
for $x$ cannot be issued simultaneously. Assuming otherwise, let $\text{wit}_x$ be a membership witness for $x$. This implies $\text{wit}_x = \text{PP}_{H(x) \rightarrow v_k}$, where $v_k \in \text{Mem}$ and the exact path $\text{EP}_{H(x) \rightarrow v_k}$ doesn't have any internal node belonging to $\text{Non-Mem}$. Clearly, $k = i$ will not work. For any other choice, $k \geq j$ as $\text{LN}(T_j) \cap X' = \emptyset$. Therefore $k < j$. But, this implies that the exact path $\text{EP}_{H(x) \rightarrow v_k}$, computed using $\text{PP}_{H(x) \rightarrow v_k}$, must contain $v_j$ as an internal node. This is true as $H(x)$ is a leaf node of $T_j$. This is a contradiction as $v_j \in \text{Non-Mem}$ and therefore $\text{Verify}(x, \text{wit}_x, \text{Acc}_X)$ will output ⊥.

### 3.2 Efficiency

The primary motivation behind this work is to find a way that takes us beyond the logarithmic size bottleneck for witnesses, a typical for Merkle tree based accumulator schemes. First, unlike the existing Merkle tree based schemes, we have elements admitting different witness sizes. In particular, assuming $\text{Acc}_X = \{\text{Mem} = (v_1, \ldots, v_i), \text{Non-Mem} = (v_j, \ldots, v_r)\}$, all members in $X \cap S_{i,k,j}$ admit witnesses of size $d - r$, where $2^r \leq i_k \leq 2^{r+1} - 1$. Similarly, the witness size for non-members under $T_{i, j}$ is $d - s < d - r$, where $2^s \leq j_k \leq 2^{s+1} - 1$. Clearly, $\text{Max} = \max\{\text{size}(\text{wit}_x) \mid x \in M\} < d$. For $S_{i,k,j}$, closer the node $v_{i,k}$ is to the root, greater is the witness size for members in $S_{i,j}$. The node $v_{i,k}$ gets closer to the root only if the portion $X \cap S_{i,k,j}$ is contiguous. For randomly selected $X \subseteq M$, the probability is exponentially low for $X$ to have larger contiguous subsets. Smaller contiguous subsets $X \cap S_{i,k,j}$ lead to near constant witness sizes for members in $S_{i,k,j}$. However, the decrease in witness size is achieved at a cost that affects accumulation size. As noted in Theorem 2, for a random set $X$ of size $r$, the number of sets in $\text{CV}_{r'}$ is roughly $1.25r$, which implies $|\text{Acc}_X| = |\text{Mem}| + |\text{Non-Mem}| \approx 2.5r$. One might conclude if this is worse than that of a trivial solution for accumulators, i.e., for $X = \{x_1, \ldots, x_n\}$, set $\text{Acc}_X = \{H(x_1), \ldots, H(x_n)\}$; and identities themselves constitute witnesses. But, this trivial solution works for accumulators that only output membership witnesses. If this trivial solution is to be extended for a universal accumulator, the accumulation size will directly depend on $|M|$ $\text{Acc}_X = \{H(x) \mid x \in X\} \cup \{H(x) \mid x \in M \setminus X\}$ and not on $X$, and therefore not work. Also, in addition, one discards the trivial solution as it doesn’t go well in keeping basic privacy properties. For example, the trivial system reveals the size of $X$, and also $\text{Acc}_X$ is vulnerable to offline dictionary attacks as it holds $H(x)$’s as it is.
4 Conclusions

In this work, we have proposed a Merkle tree based static universal accumulator scheme assuming only collision-resistant hash functions exists. Our scheme achieves a tradeoff between accumulation size and witness size. Such a tradeoff for Merkle tree based accumulators was not known to exist earlier. The proposed scheme used a well known subset covering technique called the Subset Difference method to the setting of Merkle trees to achieve this tradeoff. We found that the problem of achieving a trade off between accumulation and witness size is not easy. We consider our proposed solution to be a stepping stone in this direction for a better tradeoff.

References


