EURASIAN JOURNAL OF MATHEMATICAL AND COMPUTER APPLICATIONS
ISSN 2306–6172
Volume 13, Issue 4 (2025) 53 – 74

MCTPROOFS: A NEW MATRIX COMMITMENT SCHEME BASED ON HIERARCHICAL DATA STRUCTURES AND BILINEAR FORMS

Devisti H., Hadian M.

Abstract We present a novel matrix commitment scheme, MCTproofs, which allows a committer to commit to a matrix along with its submatrices and generate proofs for an arbitrary subset of matrix elements. The scheme achieves efficient proof generation and verification by employing tree-based structures in its algorithms. MCTproofs builds upon the foundational principles of Matproofs, offering a significant enhancement over the original scheme. Notably, MCTproofs is aggregatable, maintainable, and updatable, achieving an average performance improvement of 10 $\times$ in proof aggregation and 3 $\times$ in verification. Additionally, the size of aggregated proofs in MCTproofs is $O(1)$, compared to that of Matproofs, which has a complexity of $O(\min\{b, \sqrt{n}\})$. We further demonstrate the applicability of MCTproofs in payment-only stateless cryptocurrencies and compare its performance against Matproofs and Hyperproofs. Experimental results show that MCTproofs outperforms both in aggregation and verification while maintaining comparable efficiency in other operations.

Keywords: commitment scheme, bilinear form, matrix tree, maintainability, Matrix commitments, Bilinear cryptography, Authenticated data structures, Scalable verification, Stateless blockchains, Verifiable data integrity, Cryptographic commitments, Blockchain cryptography.

AMS Mathematics Subject Classification: 94A60, 11T71, 14G50.

DOI: 10.32523/2306-6172-2025-13-4-53-74

1 Introduction

A matrix commitment scheme is a specialized type of vector commitment scheme designed to compute a commitment C for a matrix $M = \{M_{ij}\}_{m\times n}$. These schemes enable the generation of proofs Ω_{ij} for individual elements M_{ij} at specified positions (i, j) within the matrix M. Using the commitment C and the corresponding proof Ω_{ij}, a third party can efficiently verify the authenticity of M_{ij} without needing access to the entire matrix M. The primary advantage of matrix commitment schemes lies in their computational efficiency, as they require storing only the commitment C and the proofs Ω_{ij} for the relevant positions (i, j), rather than the full matrix M. The competitiveness of a matrix commitment scheme is determined by three key properties: homomorphism, maintainability, and aggregability.

1. **Maintainability**: This property ensures that when certain elements of a matrix are updated, the proofs associated with the unchanged elements can also be updated efficiently with minimal computational overhead. Schemes such as Matproofs [1] and Hyperproofs [2] exhibit this capability.

2. **Aggregability**: Aggregability allows for the generation of a single proof for a subset of elements within a matrix M. This is achieved by aggregating the individual proofs of each element in the subset, reducing the overhead of managing multiple proofs. Schemes like Pointproofs [3] and Xproofs [4] demonstrate this property.

3. **Homomorphism**: Homomorphism enables the efficient computation of a commitment for the sum of two matrices M and M'. Instead of computing the commitment for $M + M'$ from scratch, it can be directly derived as the product of the commitments of M and M'. Examples of homomorphic schemes include those described in [5], [2], [6], and [7].

From a computational perspective, the primary objective is to minimize both the proof size and the time complexity. Consequently, developing schemes with constant-size outputs for their constituent algorithms is a central focus of research in this field. The computations involved in commitment schemes are performed on both the committer's and verifier's sides. However, no existing scheme can simultaneously minimize both the size and computational cost of the elements in a vector commitment scheme. Each side, based on its specific requirements, selects certain algorithms with constant-size outputs or algorithms for which the size depend on their input values. This paper's approach aims to increase the number of commitments on the committer's side and reduce the computational complexity on verifier's side, which will be explored further in subsequent discussions.

In this work, we propose a novel matrix commitment scheme based on a matrix tree structure. This scheme facilitates the generation of commitments and proofs for both individual elements and submatrices of an arbitrary matrix M, providing substantial flexibility and computational efficiency. We refer to our proposed scheme as MCTproofs, derived from the term *Matrix commitments tree*. The structure of this paper is as follows: At the end of this section, we review related works concerning MCTproofs. Sec. 2 introduces fundamental definitions and key concepts. In Sec. 3, we discuss the hierarchical data structures that underpin the proposed schemes. Sec. 4 presents the matrix commitment scheme and its core properties. Sec 5 introduces MCTproofs, a critical component of this paper. In Sec. 6, we conduct an analysis of MCTproofs and provide a formal proof of the claims. The practical application of MCTproofs in a real-world scenario is discussed in Sec. 7. Finally, Sec. 8 presents the experimental results, performance evaluation, and comparative analysis.

1.1 Contributions

Model Structure. In this paper, we propose a matrix commitment scheme for a square matrix M of dimension $n = 2^l$, integrating the concepts of Merkle trees and bilinear forms. To construct this scheme, we introduce key concepts such as matrix partitioning and indexing. These concepts are used to build the matrix tree $\mathcal{T}_A$. Following the construction of $\mathcal{T}_A$, we develop the matrix polynomial tree $\mathcal{T}_{f_A}$ and the parameter trees $\mathcal{T}_{pp_1}$ and $\mathcal{T}_{pp_2}$. With the resulting group $(\mathcal{T}_A, \mathcal{T}_{f_A}, \mathcal{T}_{pp_1}, \mathcal{T}_{pp_2})$, the committer can efficiently generate proofs for both individual elements and subsets of M, while the verifier can quickly validate or reject the proofs through a simple lookup. Our model presents two key advantages over MatProofs. The first advantage is that it enables the committer to commit to specific submatrices of matrix M, subsequently generating proofs for the elements of M based on these submatrices. This approach reduces the matrix dimensions, thereby saving computational time and decreasing complexity. The second advantage is that, through pre-computation in the initial stages of the commitment scheme, it allows the generation of individual proofs for each subset of elements in M, without the need to incorporate all rows and columns of M in the computations, as required by Matproofs. However, a limitation of this model is that it is applicable only to matrices with a dimension of $n = 2^l$, and it cannot be used with matrices of other dimensions.

Application. The adoption of blockchain technology and cryptocurrencies in financial transactions is expanding. As a result, significant research efforts have been dedicated to developing security infrastructures and cryptographic protocols to ensure user privacy and data security. Commitment schemes, which are two-party cryptographic protocols, play a crucial role in facilitating the secure transmission of transactions between senders and recipients. Among the studies conducted in this field, references [8, 1, 2, 9] are noteworthy. Commitment schemes, such as Hyperproofs and Matproofs, are commonly applied in the UTXO model within cryptocurrencies like Bitcoin and Ethereum. MCTproofs can also be utilized

in blockchain technology like these schemes. In blockchain terminology, a UTXO (Unspent Transaction Output) refers to a set of immutable coins, with a transaction being considered valid only when the coins associated with the UTXO are spent. In Sec. 7, we demonstrate the application of MCTproofs in the construction of payment-only stateless cryptocurrencies. In Sec. 8, we evaluate the performance of our proposed scheme in comparison to the two previously mentioned schemes. The results demonstrate significant improvements in proof aggregation and verification.

Security and Evaluation. Since the foundational elements of our scheme are derived from Matproofs, the security of our scheme inherits the security properties of Matproofs. The proposed scheme, MCTproofs, is designed to be maintainable, aggregatable, and readily updatable. The majority of the computational load in MCTproofs is concentrated in the algorithm Commit, which has a complexity of $O(n)$, in contrast to Matproofs's Commit, which operates at $O(1)$. However, this modification significantly reduces the computational cost and proof size in other algorithms. Furthermore, most of the computational burden is placed on the committer's side, thereby reducing the overhead for the verifier. We benchmark MCTproofs and compared it with Matproofs. Simulation results show that MCTproofs is 10 times faster in AggProof algorithm and 3 times faster in the Verify algorithm compared to the corresponding algorithms in Matproofs.

1.2 Related works

A vector commitment (VC) scheme employs advanced cryptographic techniques and algebraic tools to securely commit to vectors or matrices. Algebraic tools, such as bilinear forms, and cryptographic constructs, such as Merkle trees, form the foundation of many recently proposed commitment schemes. In this paper, we review the foundational works based on Merkle trees and bilinear forms that have influenced our approach and compare our proposed scheme, MCTproofs, with existing schemes, as summarized in Tab. 1.

Merkle pioneered the use of hash trees for authentication schemes [10]. Building on this concept, public key cryptosystems and digital signatures expanded the application of hash trees [11, 5]. Micali et al. [12] advanced this idea by introducing the Merkle tree as a means to commit to the elements of a finite set. In their model, the elements of a set S serve as the leaves of a binary tree, which is converted into a Merkle tree by applying a cryptographic hash function to its nodes. The authentication path for an element x provides proof of $x \in S$.

The polynomial commitment (PC) scheme, introduced by Kate et al. [13], enables the generation of a commitment C for a polynomial $\phi(x)$. This commitment C can be interpreted as a commitment to the vector of coefficients of $\phi(x)$, a feature that provides significant advantages. Many schemes, including our proposed scheme, uses this property, as will be discussed in detail.

Papamanthou et al. [14] proposed a scheme for verifying dynamic computations in cloud settings using multivariate polynomials. Expanding on this, Srinivasan et al. introduced the multilinear tree (MLE), constructed with the multilinear extension polynomial for each vector, where each node stores a sub-vector. Their Hyperproofs scheme, based on the MLE, leverages homomorphic properties for efficient, maintainable, and aggregatable proofs. Merkle grids [15] extend the functionality of Merkle trees to square matrices. In this approach, two separate Merkle trees are constructed for the rows and columns of a matrix, with the resulting hash values combined to generate a single signed hash for the target matrix. The primary structural distinction between this method and our proposed scheme lies in the generalization of the Merkle tree concept. While Merkle grids focus on hashing rows and columns, our scheme expands this functionality to include submatrices within the target matrix. Gurbanov

Table 1: Comparison with similar schemes

Scheme	Commitment size	Public parameters size	Individual proof size	Aggregated proof size	Aggregate proof time	Verify aggregated proof time
Merkle	$O(1)$	$O(1)$	$O(\log n)$	$O(1)$	×	×
Merkle SNARK	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(k \log n \log(k \log n))$	$O(n$
$k-array$ Verkle	$O(1)$	$O(1)$	$O(\log_k n)$	×	×	×
[17]	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(b \log n)$	$O(1)$
[18]	$O(1)$	$O(n)$	$O(1)$	$O(1)$	$O(b \log^2 b)$	$O(b \log^2 b)$
Pointproofs	$O(1)$	$O(n)$	$O(1)$	$O(1)$	$O(b)$	$O(b)$
Hyperproofs	$O(1)$	$O(n)$	$O(\log n)$	$O(\log(b \log n))$	$O(b \log n)$	$O(b \log n)$
Matproofs	$O(1)$	$O(n)$	$O(1)$	$O(\min\{b, \sqrt{n}\})$	$O(b + \min\{\sqrt{n}, b\})$	$O(b + \min\{\sqrt{n}, b\})$
BalanceProofs[19]	$O(1)$	$O(n)$	$O(1)$	$O(b \log^2 b)$	$O(b \log^2 b)$	$O(b \log n)$
2D-Xproofs	$O(1)$	$O(\sqrt{n})$	$O(1)$	$O(1)$	$O(b + \min\{\sqrt{n}, b\})$	$O(b \log \sqrt{n})$
MCTproofs	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(b)$	$O(\lvert\Delta S\rvert)$

et al. [3] developed the Pointproofs scheme, enabling the aggregation of proofs about vector positions in a cross-commitment setting. This scheme, secure within the generic group model, updates all proofs with a complexity of $O(n)$.

A significant focus in reducing the complexity of VC schemes is minimizing the exchange of elements between the committing party and the verifier. For instance, Bulletproofs [16] reduced the elements exchanged to $2 \log n$ for commitments to the inner product of two vectors. The proposed scheme, MCTproofs, adopts a similar principle by minimizing the elements required on the verifier's side, albeit at the expense of increasing the computational load on the committer's side.

Matproofs [1] is an idealized scheme achieving conciseness, maintainability, and updatability under the (n_1, n_2)-bBDHE and l-wBDHE assumptions. Our scheme extensively leverages Matproofs as a security foundation. However, the algorithms Aggproof and Verify in MCTproofs are, on average, 10 times and 3 times faster than those in Matproofs, respectively.

The recently introduced Xproofs scheme [4] addresses weaknesses in Matproofs, such as reliance on pairing-sensitive networks (PSNs). Xproofs employs a row-column commitment structure, $C = (C_{\text{row}}, C_{\text{col}})$, with proofs consisting of paired elements in $\mathbb{G}_1$. Similar to Xproofs, our scheme maintains a constant size for AggProof while enhancing storage efficiency and reducing computational costs in the AggProof and Verify algorithms. In summary, MCTproofs scheme offers significant computational improvements while addressing key challenges in existing schemes, though some trade-offs in conciseness persist. A comparative analysis of various commitment schemes is presented in Tab. 1. To ensure consistency and facilitate alignment with other schemes, we define n as the number of matrix elements, and b as the number of S elements. The parameter ΔS is defined in Sec. 2. As demonstrated, the MCTproofs within the Aggproof exhibit conciseness compared to Matproofs. Additionally, the computational cost is lower, scaling as $O(b)$, in contrast to $O(b + \min\{\sqrt{n}, b\})$ for Matproofs.

2 Preliminaries

Notation. Let $n = 2^l$ and $[n] = \{1, \ldots, n\}$. For any $n \times n$ matrix M, denote the (i, j)-th entry of M by M_{ij}. Define $\{(i, j) \mid i, j \in [n]\}$ as the set of all positions of M, and for any subset $S \subseteq [n] \times [n]$, let $M[S] = \{M_{ij} \mid (i, j) \in S\}$. For $\alpha, \beta \in \mathbb{Z}_p$, define the vectors $\boldsymbol{\alpha} = [\alpha, \ldots, \alpha^n]$ and $\boldsymbol{\beta} = [\beta, \ldots, \beta^n]$. Additionally, for $k \in [l]$, define $\boldsymbol{\alpha}^k = [\alpha, \ldots, \alpha^{2^{l-k}}]$ and $\boldsymbol{\beta}^k = [\beta, \ldots, \beta^{2^{l-k}}]$.

2.1 Pairing

Let λ be a security parameter, and let $BilGen(1^\lambda)$ be a probabilistic polynomial-time (PPT) algorithm that takes λ as input and outputs a context $(p, \mathbb{G}_1, \mathbb{G}_2, e, g_1, g_2)$. Here, $\mathbb{G}_1 = \langle g_1 \rangle$

and $\mathbb{G}_2 = \langle g_2 \rangle$ are cyclic groups, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a bilinear pairing satisfying the following properties:

1. $e(u^a, w^b) = e(u, w)^{ab}$ for all $u \in \mathbb{G}_1$, $w \in \mathbb{G}_2$, and $a, b \in \mathbb{Z}_p$.
2. $e(u, h)e(v, h) = e(uv, h)$ for all $u, v \in \mathbb{G}_1$ and $h \in \mathbb{G}_2$.

In this work, the pair $(\mathbb{G}_1, \mathbb{G}_2)$ is assumed to be a bilinear group of $Type - 3$, meaning that there is no efficiently computable homomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$.

2.2 Indices and Partitions

To construct our commitment scheme, we introduce the concept of a matrix tree for a square matrix M of dimension $n = 2^l$. While hierarchical data structures such as (k, d)-trees [20] and quadtrees [21] are commonly used for similar purposes, they are not suitable for our scheme. Specifically, quadtrees are designed primarily for sparse matrices, whereas our approach imposes no restrictions on the elements of the matrix. Nevertheless, the idea of the matrix tree is inspired by the quadtree structure. The matrix tree for M is constructed using partitions and indices, which will be formally defined later. For clarity, we represent matrices in a hierarchical, grid forms. Initially, we introduce the foundational definitions, including various matrix partitions, and then describe how these partitions are formulated using indices. Let M be a square matrix of dimension $n = 2^l$. M can be divided into four disjoint submatrices, each of dimension 2^{l-1}. This set of submatrices forms a partition of the elements of M. Each resulting submatrix can then be further subdivided into four smaller submatrices, equivalent to dividing M into 16 submatrices of dimension 2^{l-2}. For an example with $n = 8$, see Fig. 1 in Example 1. Each collection of submatrices obtained through this hierarchical subdivision is referred to as a window partition of M. A formal definition of window partitions is provided below.

Definition 1. *A window partition of degree k, denoted by $\mathcal{P}_M^k$, is a set of non-overlapping submatrices of a matrix M, where each submatrix has order 2^{l-k}, and each element of M belongs to exactly one submatrix. Formally, it is defined as $\mathcal{P}_M^k = \{M_{uv}^k \,|\, u, v \in \{0, \ldots, 2^k - 1\}\}$, where M_{uv}^k are the submatrices of M with dimension 2^{l-k}.*

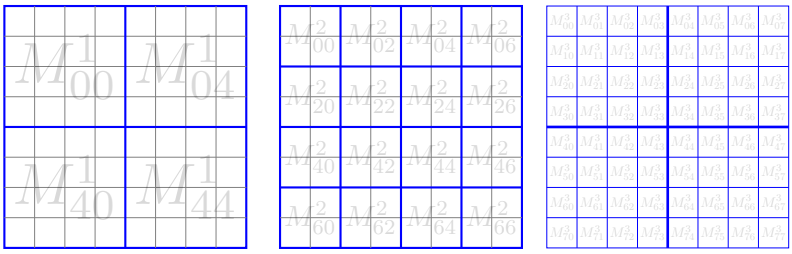
Definition 2. *A perfect nested window partition for a matrix M, denoted by $\mathcal{P}_M$, is defined as follows $\mathcal{P}_M = \{\mathcal{P}_M^k \mid k \in [l]\}$.*

In this section, we introduce the concepts of direct index and null index for the elements and submatrices of matrix M, which serve as the foundation for defining the matrix tree of M. The position of an element M_{ij} in matrix M is specified by its coordinates (i, j). However, we are interested in determining the position of M_{ij} within the submatrices of M. The concept of an index indicates which member of $\mathcal{P}_M$ contains the element M_{ij}, without specifying its exact position within that submatrix. We now proceed directly to the formal definitions of these concepts.

Definition 3. *Let M be an n-dimensional matrix. Then we define:*

- *$\mathcal{A}_M^k$: Direct index of M under the $\mathcal{P}_M^k$:* $\mathcal{A}_M^k = \left\{ (i_k, j_k) \middle| \begin{array}{l} i_k = S_k 2^{\ell-k} \\ j_k = S'_k 2^{\ell-k} \end{array} , 0 \le i_k, j_k < n \right\}$
- *$\mathcal{A}_M$: Direct index of M under the $\mathcal{P}_M$:* $\mathcal{A}_M = \bigcup_{k \in [l]} \mathcal{A}_M^k$
- *$\mathcal{A}_M(i, j)$: Direct index of position $(i, j) \in [n] \times [n]$ under the $\mathcal{P}_M$:*

$$\mathcal{A}_M(i,j) = \left\{ (i_k, j_k) \middle| \begin{array}{llll} i - 1 = S_k 2^{\ell-k} + r_k, & i_k = S_k 2^{\ell-k}, & 0 \le i_k, j_k < n \\ j - 1 = S'_k 2^{\ell-k} + r'_k, & j_k = S'_k 2^{\ell-k}, & k \in [l] \end{array} \right\}$$



$\mathcal{P}_M^1$ $\mathcal{P}_M^2$ $\mathcal{P}_M^3$

Figure 1: Partitions of matrix M

- $\mathcal{A}_M^c$ *Null index of position* $(i,j) \in S$ *under* $\mathcal{P}_M$*:*

$$\mathcal{A}_M^c(i,j) = \left\{ (i_k^{'}, j_k^{'}) \middle| i_k^{'}, j_k^{'} \in \{i_k, j_k\} \text{ where } (i_k, j_k) \in \mathcal{A}_M^k(i,j) \wedge i_k^{'} \neq i_k, j_k^{'} \neq j_k \right\}$$

- *Let* S *is defined as specified. We define* ΔS *as follows:* $\Delta S = \bigcup_{(i,j)\in S} \mathcal{A}_M^c(i,j) \setminus \bigcup_{(i,j)\in S} \mathcal{A}_M(i,j)$.

The purpose of defining $\mathcal{A}_M^k$ is to represent the matrix M as a block matrix, where the position of each submatrix $B \in \mathcal{P}_M^k$ is determined by these indices. Specifically, for each $B \in \mathcal{P}_M^k$, there exists a pair of indices $(i_k, j_k) \in \mathcal{A}_M^k$ such that $\forall B \in \mathcal{P}_M^k$, $\exists (i_k, j_k) \in \mathcal{A}_M^k$ such that $B = M_{i_k j_k}^k$. Therefore, $\mathcal{P}_M^k$ can be rewritten as follows, based on the previous explanation $\mathcal{P}_M^k = \left\{ M_{i_k j_k}^k \mid (i_k, j_k) \in \mathcal{A}_M^k \right\}$. The set $\mathcal{P}_M$ is defined in a similar manner. Using $\mathcal{A}_M(i,j)$, we can identify all the submatrices that contain the element (i,j). By applying the partition method described above, the matrix M is transformed into a block matrix, where the arrays are the submatrices of M. Additionally, $\mathcal{A}_M^c(i,j)$ represents a special subset of $\mathcal{A}_M$ that satisfies the following two conditions:

1 For every $(i_k', j_k') \in \mathcal{A}_M$, the submatrix $M_{i_k' j_k'}^k$ does not contain the element (i,j).

2 If $M_{i_k' j_k'}^k$ and $M_{i_r' j_r'}^r$ do not contain the element (i,j), and $M_{i_r' j_r'}^r$ is a submatrix of $M_{i_k' j_k'}^k$, then we consider $M_{i_k' j_k'}^k$.

Condition 2 establishes that priority is given to larger matrices. A "larger matrix" refers to a matrix with a higher dimension, and this term can similarly be applied to matrices with smaller dimensions, depending on the context. The definition of ΔS will be used in the construction of *AggProof* in the MCTproofs scheme.

Example 1. *For an arbitrary matrix* M *of dimension* 8*, the partitions* $\mathcal{P}_M^1$*,* $\mathcal{P}_M^2$*, and* $\mathcal{P}_M^3$ *are presented in Fig. 1, where according to Def. 3, each of these partitions can be written as a set. Specially, for* $\mathcal{P}_M^1$ *we have* $\mathcal{P}_M^1 = \{M_{00}^1, M_{04}^1, M_{40}^1, M_{44}^1\}$

3 Tree structure

In this section, we aim to establish a correspondence between a tree and a matrix M of dimension 4, subsequently generalizing this relationship to matrices of arbitrary dimensions. The construction of matrix tree is inspired by the approach presented in [12]. The matrix tree associated with M is denoted by $\mathcal{T}_M$. The tree is constructed from top to bottom based on partitions and indices introduced in Sec.

Step 1 (Root Initialization): We set $e = M_{00} = M$, considering M as the root of the tree.

Step 2 (First Level): The elements of the partition $\mathcal{P}_M^1 = \{M_{00}^1, M_{04}^1, M_{40}^1, M_{44}^1\}$ are arranged from left to right below the root matrix M.

Steps i (Rest Level): For each $i \geq 2$, the members of $\mathcal{P}_M^i$, representing submatrices of the matrices in $\mathcal{P}_M^{i-1}$, are arranged in a similar manner. This hierarchical process continues until the final step, where the elements of matrix M are reached.

The other components of the tree $\mathcal{T}_A$ are as follows:

- *Nodes*: Each submatrix $M_{uv}^i \in \mathcal{P}_M^i$ represents a node at level i, and the total number of nodes at this level is 4^i.
- *Children*: Every node at level i has four children, which are members of $\mathcal{P}_M^{i+1}$.
- *Parent*: Every node, except the root, has a parent defined as parent $(M_{uv}^i) = M_{uv}^{i-1} \in \mathcal{P}_M^{i-1}$.
- *Leaves*: The leaves of $\mathcal{T}_M$ correspond to the elements of matrix M.

The path from the root M to a leaf $M_{uv}^l = M_{ij}$ is a sequence of matrices $\{M_{uv}\}_{i=1}^l \subseteq \mathcal{P}_M$, all of which contain M_{ij}.

Let $S = \{(i, j) \mid i, j \in [n] \times [n]\}$ and $M[S]$ be a set of elements M. We define:

- $TREE(S)$: The subtree of $\mathcal{T}_M$ consisting of the union of all paths from the root to the leaves in S. If S is empty, $TREE(S) = e$.
- $FRONTIER(S)$: $= \{M_{uv}^i \mid M_{uv}^i \notin TREE(S)$, parent$(M_{uv}^i) \in TREE(S)\}$.

Example 2. *Suppose the matrix M is as follows:*

$$M = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 3 & 2 & 1 & 2 \\ 5 & 1 & 2 & 3 \\ 4 & 1 & 5 & 1 \end{bmatrix}$$

Also, for a set $S = \{(1, 1), (1, 3), (2, 4)\}$, the subtree containing S is as follows:

$$Tree(S) = \{M, M_{00}^1, M_{00}^2, M_{02}^1, M_{02}^2, M_{13}^2\} \tag{1}$$

Given the steps mentioned above, we have (see Fig. 2) which, after labeling the nodes, we have (see Fig. 3).

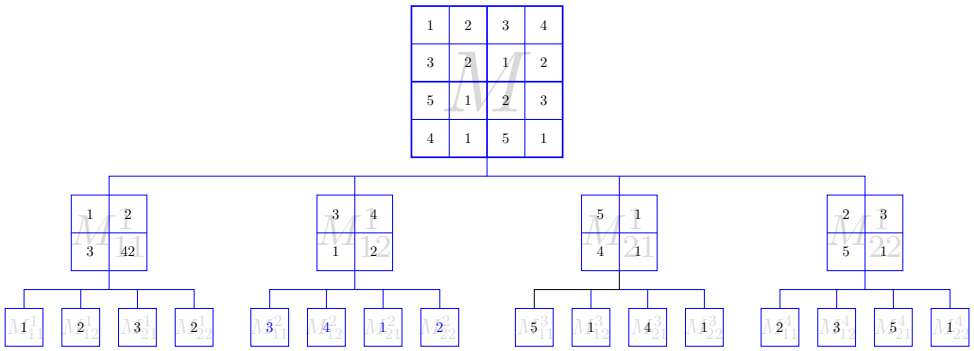


Figure 2: Nested window partition $\mathcal{P}_M$

3.1 Bilinear forms

Throughout this paper we focus on the use of bilinear forms. These are a specific class of functions defined on vector spaces, characterized by their bilinearity, which can be advantageous in various contexts. For instance, bilinearity is a crucial property in the design of homomorphic commitment schemes.

Definition 4. *Let V be a vector space on the field $\mathbb{F}_p$. A bilinear form on V is a function like f that assigns a scalar $f(\alpha,\beta)$ to every pair $(\alpha,\beta)\in V\times V$ and satisfy in the following conditions: i) $f(c\alpha_1+\alpha_2,\beta)=cf(\alpha_1,\beta)+f(\alpha_2,\beta)$, ii) $f(\alpha,c\beta_1+\beta_2)=cf(\alpha,\beta_1)+f(\alpha,\beta_2)$.*

Now,two important theorems concerning bilinear forms over $\mathbb{Z}_p$ are presented, which frequently used in subsequent discussions.

Theorem 3.1. *Let $\mathbb{Z}_p^n$ be a vector space over $\mathbb{Z}_p$, and let M be an $n\times n$ matrix with elements in $\mathbb{Z}_p$. Define f as a bilinear form given by $f_M(X,Y)=\boldsymbol{Y}^T M\boldsymbol{X}$, where $\boldsymbol{X}=[X,\ldots,X^n]^T$ and $\boldsymbol{Y}=[Y,\ldots,Y^n]$, with $X,Y\in\mathbb{Z}_p$.*

Proof. To show that f_M satisfies conditions (i) and (ii) of Def. 4, suppose $c\in\mathbb{Z}_p$ and $\boldsymbol{X}_1,\boldsymbol{X}_2,\boldsymbol{Y}\in\mathbb{Z}_p^n$. By condition (i) of Def. 4, we obtain

$$f_M(c\,\boldsymbol{X}_1+\boldsymbol{X}_2,\boldsymbol{Y})=\boldsymbol{Y}^T M(c\boldsymbol{X}_1+\boldsymbol{X}_2)=c\boldsymbol{Y}^T M\boldsymbol{X}_1+\boldsymbol{Y}^T M\boldsymbol{X}_2=cf_M(\boldsymbol{X}_1,\boldsymbol{Y})+f_M(\boldsymbol{X}_2,\boldsymbol{Y}),$$

which establishes condition (i).

For condition (ii), let $c\in\mathbb{Z}_p$ and $\boldsymbol{X},\boldsymbol{Y}_1,\boldsymbol{Y}_2\in\mathbb{Z}_p^n$. Then

$$f_M(\boldsymbol{X},c\boldsymbol{Y}_1+\boldsymbol{Y}_2)=(c\boldsymbol{Y}_1+\boldsymbol{Y}_2)^T M\boldsymbol{X}=c\boldsymbol{Y}_1^T M\boldsymbol{X}+\boldsymbol{Y}_2^T M\boldsymbol{X}=cf_M(\boldsymbol{X},\boldsymbol{Y}_1)+f_M(\boldsymbol{X},\boldsymbol{Y}_2).$$

Hence, f_M satisfies condition (ii) as well and is therefore a bilinear form. □

Theorem 3.2. *Let $n=2^l$ and M be a squared matrix of dimension n, let $\mathcal{P}_M^1=\{M_{00}^1,M_{o2^{l-1}}^1,M_{2^{l-1}0}^1,M_{2^{l-1}2^{l-1}}^1\}$ be a partition of degree 1 for M. Then the above function f satisfy in the following equation:*

$$f_M(X,Y)=f_{M_{00}^1}(X,Y)+X^{2^{l-1}}f_{M_{o2^{l-1}}^1}(X,Y)+Y^{2^{l-1}}f_{M_{2^{l-1}0}^1}(X,Y)+(XY)^{2^{l-1}}f_{M_{2^{l-1}2^{l-1}}^1}(X,Y)$$

Proof. Let M be a square matrix of dimension $n=2^l$. By Def. 1, the function $f(X,Y)$ can be written as

$$f(X,Y)=\begin{bmatrix}\boldsymbol{Y}^1\\ \boldsymbol{Y}^2\end{bmatrix}^T\begin{bmatrix}M_{00}^1 & M_{04}^1\\ M_{40}^1 & M_{44}^1\end{bmatrix}\begin{bmatrix}\boldsymbol{X}^1\\ \boldsymbol{X}^2\end{bmatrix},\tag{2}$$

where $\boldsymbol{X}^1=[X,\ldots,X^{n/2}]^T$, $\boldsymbol{X}^2=[X^{n/2+1},\ldots,X^n]^T$ and $\boldsymbol{Y}^1=[Y,\ldots,Y^{n/2}]$, $\boldsymbol{Y}^2=[Y^{n/2+1},\ldots,Y^n]$, with $X,Y\in\mathbb{Z}_p$. The remainder of the proof proceeds via straightforward matrix multiplication arguments. In conjunction with expression 2 and Theorem 3.1, these computations complete the proof. □

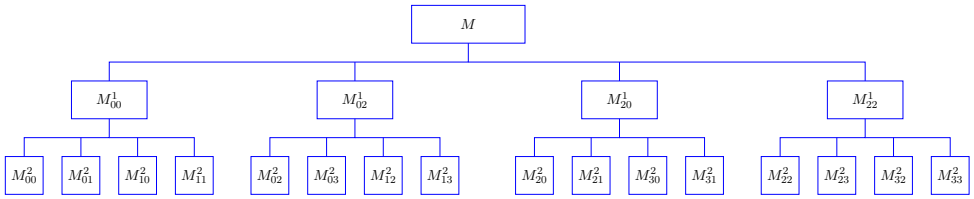


Figure 3: Matrix tree $\mathcal{T}_M$

An important application of Theorems 3.1 and 3.2 lies in the construction of a polynomial tree, which will be discussed in detail below. To illustrate this, we first provide an example of Theorem 3.2 applied to an 4-dimensional matrix, followed by a detailed discussion.

Example 3. *Let the matrix M be as given in Example 1, and let $\alpha, \beta \in \mathbb{Z}_p$. Then, according to Def. 3 and Theorem 3.2, we obtain:*

$$\begin{aligned} f(\alpha,\beta) = \boldsymbol{\beta}^T M \boldsymbol{\alpha} &= \begin{bmatrix} \boldsymbol{\beta}^1 \\ \hdashline \boldsymbol{\beta}^2 \end{bmatrix}^T \begin{bmatrix} M^1_{00} & M^1_{04} \\ \hdashline M^1_{40} & M^1_{44} \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha}^1 \\ \hdashline \boldsymbol{\alpha}^2 \end{bmatrix} \\ &= \underbrace{f^1_{00}(\alpha,\beta)}_{\substack{\alpha\beta+2\alpha\beta^2+ \\ 3\alpha^2\beta+2\alpha^2\beta^2}} + \underbrace{\alpha^2 f^1_{02}(\alpha,\beta)}_{\substack{3\alpha\beta^3+4\alpha\beta^4+ \\ \alpha^2\beta^3+2\alpha^2\beta^4}} + \underbrace{\beta^2 f^1_{20}(\alpha,\beta)}_{\substack{5\alpha^3\beta+\alpha^3\beta^2+ \\ 4\alpha^2\beta+\alpha^4\beta}} + \underbrace{\alpha^2\beta^2 f^1_{22}(\alpha,\beta)}_{\substack{2\alpha^3\beta^3+3\alpha^3\beta^4+ \\ 5\alpha^4\beta^3+\alpha^4\beta^4}} \end{aligned}$$

3.2 Martrix polynomial tree

After constructing the tree $\mathcal{T}_M$ of the matrix M, the polynomial tree of M, denoted as $\mathcal{T}_{f_M}$, is subsequently derived. To construct $\mathcal{T}_{f_M}$, it suffices to compute the corresponding two-variable polynomials(bilinear form) for each node in $\mathcal{T}_M$. $\mathcal{T}_{f_M}$ is defined as follows $\mathcal{T}_{f_M} = \{f^k_{i_k j_k}(X,Y) \mid (i_k, j_k) \in \mathcal{A}^k_M\}$ where $k \in [l]$ indicates the level of the nodes. The polynomial at each node is given by $f^k_{i_k j_k}(X,Y) = \boldsymbol{Y}^{k^T} M^k_{i_k j_k} \boldsymbol{X}^k$ where $M^k_{i_k j_k} \in \mathcal{P}_M$. Therefore, for $M^l_{i_l j_l} = M_{ij}$, i.e., for the leaves of $\mathcal{T}_{f_M}$, the corresponding polynomial is $f^l_{i_l j_l}(X,Y) = X^{j-1}Y^{i-1}$ for $(i,j) \in [n]\times[n]$.

3.3 Public parameters tree

We define the public parameters tree $\mathcal{T}_{pp} = (\mathcal{T}_{pp_1}, \mathcal{T}_{pp_2})$ for the matrix M, which will be utilized in the subsequent section. It suffices to construct only $\mathcal{T}_{pp_1}$, as the construction of $\mathcal{T}_{pp_2}$ follows a process analogous to that of $\mathcal{T}_{pp_1}$. The tree $\mathcal{T}_{pp_1}$ is constructed from the set $\mathcal{A}_M$, with the arrangement of its nodes and leaves resembling that of $\mathcal{T}_A$ and $\mathcal{T}_{f_A}$. In summary, the construction of this tree is straightforward and proceeds in the following three steps:

- Set g_1 as the root of the tree.
- Select $\alpha, \beta \in \mathbb{Z}_p$ randomly.
- For $k \in \{1, 2, \cdots, l\}$ the nodes of level i are $g_1^{\alpha^{j_k}\beta^{i_k}}$, $(i_k, j_k) \in \mathcal{A}^k_M$
- The arrangement of nodes in the $\mathcal{T}_{pp_1}$ tree follows a left-to-right order, similar to that of $\mathcal{T}_M$.

Example 4. *For the matrix M presented in Example 3, the structure of $\mathcal{T}_{pp_1}$ is as follows:*

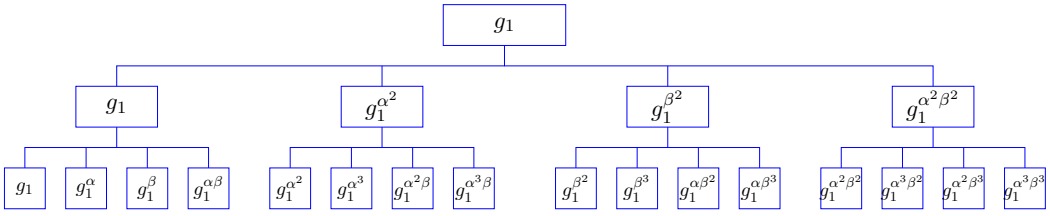


Figure 4: Public parameters tree $\mathcal{T}_{pp_1}$

3.4 Commitment polynomial trees

We have constructed four trees, denoted as $\mathcal{T}_M$, $\mathcal{T}_{f_M}$, $\mathcal{T}_{pp_1}$, and $\mathcal{T}_{pp_1}$ corresponding to the desired matrix M of dimension $n = 2^l$. In the final step, the commitment tree, which is derived from $\mathcal{T}_{f_M}$, is constructed. Appendix 8.2 provides a step-by-step explanation of the process for constructing the commitments tree. This process is straightforward, as it involves computing elements in $\mathbb{G}_1$. Specifically, we compute commitments associated with the matrix $M^k_{i_k j_k} \in \mathcal{P}_M$ based on the Matproofs framework. The resulting matrix polynomial tree is denoted by $\mathcal{T}_C$, and it is defined as follows: $\mathcal{T}_C = \{C^k_{i_k j_k} \mid (i_k, j_k) \in \mathcal{A}^k_M, k \in [l]\}$, where $C^k_{i_k j_k}$ represents the commitment associated with $M^k_{i_k j_k}$. These commitments are computed as: $C^k_{i_k j_k} = g_1^{f^k_{i_k j_k}(\alpha,\beta)}$ $(\alpha, \beta \in \mathbb{Z}_p)$.

4 Matrix commitment scheme

To ensure consistency with prior research, the proposed matrix commitment scheme adheres to the standard conventions employed in vector commitment schemes. A matrix commitment scheme is defined by seven PPT algorithms, as described below.

1. $\mathsf{Setup}(1^\lambda, 1^n)$: This algorithm, given the security parameter λ and an integer n, produces a set of public parameters pp. These parameters are organized hierarchically and can be represented as a public parameter tree.
2. $\mathsf{Commit}(M, \mathcal{A}_M, \mathcal{P}_M)$: This algorithm takes as input a matrix M of dimensions $n \times n$ and corresponding $\mathcal{A}_M$ and $\mathcal{P}_M$. It produces as output a commitment C, along with commitments to the individual elements of $\mathcal{P}_M$, organized in the form of a hierarchical commitment tree.
3. $\mathsf{UpdateCommit}((i,j), \delta, C, \{C^k_{uv}\}_{(u,v)\in\mathcal{A}^c_M(i,j)})$: This algorithm is designed to update all submatrices containing the arbitrary position (i, j). It takes as input the position (i, j), the update parameter δ, and the set of commitments $\{C^k_{uv}\}$ associated with the elements of $\mathcal{A}^c_M(i, j)$.
4. Prove $((i,j), \mathcal{A}^c_M(i,j), M)$: For any $(i, j) \in [n] \times [n]$, this algorithm generates a proof Ω_{ij} corresponding to the matrix entry M_{ij}.
5. $\mathsf{UpdProof}((i,j), (i',j'), \delta, \Omega_{i'j'})$: This algorithm updates the proofs $\Omega_{i'j'}$ for all entries $(i', j') \in [n] \times [n]$ in response to an update at the matrix entry (i, j).
6. $\mathsf{Aggproof}(C, S, \{C^k_{i_k j_k}\}_{(i_k,j_k)\in\Delta S}, \{\Omega_{ij}\}_{(i,j)\in S})$: This algorithm takes as input a set of positions $S \subseteq [n] \times [n]$, their corresponding entries $M[S]$, a set of commitments $\{C^k_{i_k j_k}\}_{(i_k,j_k)\in\Delta S}$, and proofs $\{\Omega_{ij}\}_{(i,j)\in S}$. It outputs a triple $(\hat{\omega}, C_S, \hat{\Omega})$.
7. $\mathsf{Verify}(C, S, \Delta S, \{C^k_{i_k j_k}\}_{(i_k,j_k)\in\Delta S})$: This algorithm takes as input a set of positions $S \subseteq [n] \times [n]$, their corresponding entries $M[S]$ a set of commitments $\{C^k_{i_k j_k}\}_{(i_k,j_k)\in\Delta S}$, and proof triple $(\hat{\omega}, C_S, \hat{\Omega})$. It verifies the correctness of the entries $M[S]$ against the commitment C.

A matrix commitment scheme must satisfy to the principles of correctness and positional binding. The fundamental definitions of these properties in the context of our scheme include: opening correctness, aggregation correctness, commitment update correctness, proof update correctness, and position binding. These properties are defined as follows:

Definition 5 (Opening Correctness)**.** *For all λ, $n = 2^l$, $k \in [l]$, matrix $M^k_{i_k j_k}$ of dimension*

$n_k = 2^{l-k}$, *and any* $(i,j) \in [n] \times [n]$ *and* $(i_k, j_k) \in \mathcal{A}_M(i,j)$, *the following holds:*

$$\Pr\left[\begin{array}{l} pp \leftarrow \mathsf{Setup}(1^\lambda, 1^{n_k}), \\ C^k_{i_k j_k} \leftarrow \mathsf{Commit}(M^k_{i_k j_k}), \\ \Omega_{i_k j_k} \leftarrow \mathsf{Prove}((i_k, j_k), M^k_{i_k j_k}) : \\ \mathsf{Verify}(C^k_{i_k j_k}, \{(i_k, j_k)\}, M^k_{i_k j_k}, \Omega_{i_k j_k}) = 1 \end{array}\right] = 1.$$

This relation guarantees the correctness of the commitment and proof verification process for our scheme.

Definition 6 (Commitment update correctness)**.** *For all* λ, $k \in [l]$, *all integers* $n, n_k > 0$, *any* $(i,j) \in [n] \times [n]$ *with its corresponding direct index* $\mathcal{A}^c_M(i,j)$, *any matrix* $M^k_{i_k j_k}$ *of dimension* $n_k = 2^{l-k}$, *and any* δ, *when the value of* $M_{i_k j_k}$ *is updated to* $M'_{i_k j_k} = M_{i_k j_k} + \delta$, *the following property holds:*

$$\Pr\left[\begin{array}{l} \mathrm{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^{n_k}), \\ C^k_{i_k j_k} \leftarrow \mathsf{Commit}(M^k_{i_k j_k}), \\ {C^k_{i_k j_k}}' \leftarrow \mathsf{UpdateCommit}((i_k, j_k), \delta, C^k_{i_k j_k}), \\ {C^k_{i_k j_k}}'' \leftarrow \mathsf{Commit}({M^k_{i_k j_k}}') : \\ {C^k_{i_k j_k}}' = {C^k_{i_k j_k}}'' \end{array}\right] = 1.$$

This relation ensures that the outputs of the algorithms Commit and $\mathsf{UpdateCommit}$ are consistent for the matrix $M^k_{i_k j_k}$ and its updated version ${M^k_{i_k j_k}}'$.

Definition 7. *(Proof Update Correctness) For all* λ, $k \in [l]$, *integers* $n, n_k > 0$, *a position* $(i,j) \in [n] \times [n]$ *with its corresponding direct index* $\mathcal{A}_M(i,j)$, *a matrix* $M^k_{i_k j_k}$ *of dimension* n_k, *and any* δ, *when the value of* $M_{i_k j_k}$ *is updated to* $M'_{i_k j_k} = M_{i_k j_k} + \delta$, *then:*

$$Pr\left[\begin{array}{l} \mathrm{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^{n_k}), \\ \Omega_{i'_k j'_k} \leftarrow \mathsf{Prove}((i'_k j'_k), M^k_{i_k j_k}), \\ \Omega'_{i'_k j'_k} \leftarrow \mathsf{UpdateProof}((i_k, j_k), (i'_k, j'_k), \delta, \Omega_{i'_k j'_k}), \\ \Omega''_{i'_k j'_k} \leftarrow \mathsf{Prove}((i'_k j'_k), {M^k_{i_k j_k}}') : \\ \Omega'_{i'_k j'_k} = \Omega''_{i'_k j'_k} \end{array}\right] = 1.$$

This relationship establishes that for all $(i,j) \in [n] \times [n]$ and $(i_k, j_k) \in \mathcal{A}_M(i,j)$, when M_{ij} is updated to $M_{ij} + \delta$, the outputs of the algorithms $\mathsf{Prove}((i'_k, j'_k), {M^k_{i_k j_k}}')$ and $\mathsf{UpdateProof}((i_k, j_k), (i'_k, j'_k), \delta, \Omega_{i'_k j'_k})$ remain identical.

Definition 8 (Aggregation Correctness)**.** *For all* λ, $k \in [l]$, *integers* $n, n_k > 0$, *a matrix* M, *and any set of positions* $S^k_{i_k j_k} \subseteq [n_k] \times [n_k]$, *the following holds:*

$$\Pr\left[\begin{array}{l} \mathrm{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^{n_k}), \\ C^k_{i_k j_k} \leftarrow \mathsf{Commit}(M^k_{i_k j_k}), \\ \Omega_{i_k j_k} \leftarrow \mathsf{Prove}((i_k, j_k), M^k_{i_k j_k}), \\ \hat{\Omega} \leftarrow \mathsf{AggregateProof}(C^k_{i_k j_k}, S^k_{i_k j_k}, M^k_{i_k j_k}[S^k_{i_k j_k}], \{\Omega_{i_k j_k}\}_{(i_k, j_k) \in S^k_{i_k j_k}}) : \\ \mathsf{Verify}(C^k_{i_k j_k}, \{(i_k, j_k)\}_{(i_k, j_k) \in S^k_{i_k j_k}}, M^k_{i_k j_k}[S^k_{i_k j_k}], \hat{\Omega}) = 1 \end{array}\right] = 1.$$

This relation demonstrates that for any arbitrary subset $S^k_{i_k j_k}$, a single aggregated proof can be constructed to verify the correctness of the associated positions.

Definition 9 (Position Binding)**.** *For all λ, $k \in [l]$, $n, n_k > 0$, and any PPT adversary $\mathcal{A}$, the following holds:*

$$\Pr\left[\begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^{n_k}), \\ (C^k_{i_kj_k}, ({\Omega^k_{i_kj_k}}^b, {S^k_{i_kj_k}}^b, {M^k_{i_kj_k}}^b[{S^k_{i_kj_k}}^b])_{b=0,1}) \leftarrow \mathcal{A}(\mathsf{pp}) : \\ (\mathsf{Verify}(C^k_{i_kj_k}, {S^k_{i_kj_k}}^b, {M^k_{i_kj_k}}^b[{S^k_{i_kj_k}}^b], {\Omega^k_{i_kj_k}}^b) = 1)_{b=0,1} \\ \wedge {M^k_{i_kj_k}}^0[{S^k_{i_kj_k}}^0 \cap {S^k_{i_kj_k}}^1] \neq {M^k_{i_kj_k}}^1[{S^k_{i_kj_k}}^0 \cap {S^k_{i_kj_k}}^1] \end{array}\right] \neq \mathrm{negl}(\lambda).$$

This property, known as position binding, ensures that for all $M^k_{i_kj_k} \in \mathcal{P}_M$, no PPT adversary can open the same matrix commitment $C^k_{i_kj_k}$ to produce different values at the same positions.

4.1 Security requirements

The proposed matrix commitment scheme achieves security by leveraging the ℓ-wBDHE* assumption [6] and the (n_1, n_2)-bBDHE assumption [1] within the Generic Group Model (GGM). The ℓ-wBDHE* assumption is rooted in the infeasibility of solving the weak bilinear Diffie-Hellman exponent problem by any probabilistic polynomial-time (PPT) adversary. Its formal definition is as follows:

Definition 10. *Let $(\mathbb{G}_1, \mathbb{G}_2)$ be generic bilinear groups of Type-3, and let $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$ define a bilinear context generated by $BG(1^\lambda)$. The ℓ-wBDHE* assumption states that no PPT adversary can solve the following variant of the weak bilinear Diffie-Hellman exponent problem except with negligible probability: Given $(g_1^{P(\alpha)}, g_2^{Q(\alpha)})$, where $P(X) = (X, X^2, \ldots, X^\ell, X^{\ell+2}, \ldots, X^{3\ell})$ and $Q(X) = (X, X^2, \ldots, X^\ell)$, and $\alpha \leftarrow \mathbb{Z}_p$, compute $g_1^{\alpha^{\ell+1}}$. This assumption underpins the security guarantees of the scheme by ensuring that solving this computational problem is infeasible for any PPT adversary.*

The second assumption presented by Liu and Zhang [1] is based on a extention of ℓ-wBDHE* to two-variable polynomials and is as follows:

Definition 11 ((n_1, n_2)-bBDHE)**.** *Let $(\mathbb{G}_1, \mathbb{G}_2)$ be generic bilinear groups of Type-3, and let $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$ define a bilinear context generated by $BG(1^\lambda)$. For $n_1, n_2 > 0$, the (n_1, n_2)-bivariate Bilinear Diffie Hellman Exponent (bBDHE) assumption assumes that, except with a negligible probability, no PPT adversary can solve the following problem: given $(g_1^{P(\alpha,\beta)}, g_2^{Q(\alpha,\beta)})$ for the (P, Q, R) in the ring of polynomials $\mathcal{F}_{p,c}$ and $\alpha, \beta \leftarrow \mathbb{Z}_p$, output $(e, g_1^{e^\top \alpha \beta^{n_1+1}})$ fore $\in \mathbb{Z}_p^{n_2} \backslash \{0\}$, where $\boldsymbol{\alpha} = (\alpha, \alpha^2, \ldots, \alpha^{n_2})$.*

The algebraic group model(AGM) The *AGM + ROM* model plays a critical role in establishing the security of our scheme. This model combines the Algebraic Group Model (AGM) and the Random Oracle Model (ROM), effectively limiting adversaries from exploiting computational complexity to compromise the scheme. In the AGM, adversaries are restricted to algebraic operations and are granted access to the bit-level representations of group elements. However, they can only generate new group elements by applying group operations to the elements they already possess. Specifically, for a given set of group elements $I_1, \ldots, I_n \in G$, the adversary must produce the corresponding coefficients $e_1, \ldots, e_n \in \mathbb{Z}_p$ to construct a new group element $O = \prod_{j=1}^n I_j^{e_j}$. In the ROM, cryptographic hash functions are treated as ideal random functions with an output space in $\mathbb{Z}_p$. All parties in the protocol interact with these hash functions exclusively through oracle queries, ensuring consistent and unpredictable hash outputs, which are essential for security.

5 Scheme structure

A matrix commitment scheme consists of two fundamental components: the commitment process and the verification process. Our approach integrates concepts from three existing schemes [12, 2, 1] to develop a novel scheme based on Merkle trees. The proposed scheme, MCTproofs, is aggregatable, concise, and optimized for efficient updates and maintenance. Let $n = 2^l$, and let $M = (M_{ij})_{n\times n}$ represent a square matrix of dimension n, where $M_{ij} \in \mathbb{Z}_p$. Additionally, let $X = (X, \ldots, X^n)$ and $Y = (Y, \ldots, Y^n)$ denote two vectors, and let (α, β) be random elements in $\mathbb{Z}_p \times \mathbb{Z}_p$. The MCTproofs scheme provides an efficient mechanism for committing to and verifying a single element M_{ij} from the matrix M. As outlined in Sec. 3, the scheme begins with the generation of the tree structure, followed by subsequent procedures. This section introduces the MCTproofs scheme.

5.1 MCTproofs scheme

We provide an explanation of how the MCTproofs operates for an element M_{ij} of the matrix M. The MCTproofs scheme comprises seven PPT algorithms, which are outlined below. These algorithms are executed on a square matrix of dimension $n = 2^l$.

- $\mathsf{Setup}(1^\lambda, 1^n)$: This algorithm takes as input a security parameter $\lambda \in \mathbb{N}$ and generates a bilinear group model $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$ by running the bilinear group generator $BG(1^\lambda)$. Next, it selects $\alpha, \beta \in \mathbb{Z}_p$ uniformly at random and outputs the trees $\mathcal{T}_{pp_1}$ and $\mathcal{T}_{pp_2}$, defined as follows: $\mathcal{T}_{pp_1} = \{g_1^{\alpha^{j_k}\beta^{i_k}} \,|\, (i_k, j_k) \in \mathcal{A}_M\}$, $\mathcal{T}_{pp_2} = \{g_2^{\alpha^{j_k}\beta^{i_k}} \,|\, (i_k, j_k) \in \mathcal{A}_M\}$, where $\mathcal{A}_M^k$ denotes the direct index associated with the matrix M.
- $\mathsf{Commit}(M, \mathcal{A}_M, \mathcal{P}_M)$: This algorithm takes as input a matrix M, the direct index set $\mathcal{A}_M$, and the nested perfect partition $\mathcal{P}_M$. It outputs the tree $\mathcal{T}_C$, defined as follows: $\mathcal{T}_C = \{C^k_{i_kj_k} \,|\, (i_k, j_k) \in \mathcal{A}_M\}$. Here, $\mathcal{T}_C$ is constructed from $\mathcal{T}_{f_M}$ as described in Sec. 3.
- $\mathsf{UpdCommit}((i, j), \delta, C, C^k_{i_kj_k})$: This algorithm updates the commitments C and $C^k_{i_kj_k}$ when the element M_{ij} of the matrix M changes to $M_{ij} + \delta$. The updated commitments C' and $C^k_{i_kj_k}{}'$ are computed as follows: $C' = C \cdot g_1^{\delta\alpha^j\beta^i}$, $C^k_{i_kj_k}{}' = C^k_{i_kj_k} \cdot g_1^{\delta\alpha^{j_k}\beta^{i_k}}$ for $(i_k, j_k) \in \mathcal{A}_M(i, j)$, where C and $C^k_{i_kj_k}$ are the commitments corresponding to the matrix M and the submatrices $M^k_{i_kj_k}$ containing the element (i, j). As demonstrated, when M_{ij} is updated, the commitments of all submatrices that include M_{ij} are also updated accordingly.
- $\mathsf{Prove}((i, j), \mathcal{A}^c_M(i, j), M)$: This algorithm generates a proof for the matrix element M_{ij}. The proof is computed as: $\Omega_{ij} = \prod_{(i_k,j_k)\in\mathcal{A}^c_M(i,j)} \left[C^k_{i_kj_k}\right]^{\alpha^{j_k}\beta^{i_k}}$, where Ω^k_{ij} can be efficiently derived using the pair of $(\mathcal{T}_C, \mathcal{T}_{pp_1})$.
- $\mathsf{UpdProof}((i, j), (i', j'), \Omega_{i'j'})$: This algorithm updates the proofs of other elements in M when the element M_{ij} is modified. Two cases are considered:

  1. $Case1$: When M_{ij} is updated to $M_{ij} + \delta$ and $(i', j') = (i, j)$, the proof $\Omega_{i'j'}$ remains unchanged. However, in subsequent computations, the updated commitment C' should be used instead of C.

  2. $Case2$: When M_{ij} is updated to $M_{ij} + \delta$ and $(i', j') \neq (i, j)$, there exists $(i'_r, j'_r) \in \mathcal{A}_M(i', j')$ such that $M^r_{i'_rj'_r}$ includes M_{ij}. Consequently, one term $C^k_{i'_rj'_r}$ in the proof $\Omega'_{i'j'}$ corresponds to the commitment for $M^k_{i'_rj'_r}$. Thus, the updated proof $\Omega'_{i'j'}$ is given by:

$$\Omega'_{i'j'} = \prod_{(i'_k,j'_k)\in\mathcal{A}_M(i',j')} [C^k_{i'_k,j'_k}]^{\alpha^{j'_k}\beta^{i'_k}} \times [C^r_{i'_r,j'_r}]^{\alpha^{j'_r}\beta^{i'_r}} g^{\alpha^{j_r{}'+s}\beta^{i_r{}'+t}} = \Omega_{i'j'} g^{\delta\alpha^j\beta^i},$$

where (s,t) is the position (i,j) in matrix $M^k_{i'_r j'_r}$.

- $\mathsf{Aggproof}(C, S, \{C^k_{i_k j_k}\}_{(i_k,j_k)\in\Delta S}, \{\Omega^k_{ij}\}_{(i,j)\in S})$: This algorithm enables the aggregation of proofs for elements of M where $(i,j) \in S$. The output of the algorithm consists of the following components:

  1. Aggregated Proof: $\hat{\omega} = \prod_{(i_k,j_k)\in\Delta S} \left[C^k_{i_k j_k}\right]^{\alpha^{j_k}\beta^{i_k}}$.
  2. Aggregated Commitment: $C_S = g_1^{\sum_{(i,j)\in S} M_{ij}\alpha^j\beta^i}$.
  3. Product of Individual Proofs: $\Omega = \prod_{(i,j)\in S} \Omega_{ij}^{h_{ij}}$, where the weight h_{ij} is computed as $h_{ij} = H((i,j), C, S, M[S])$, and H denotes a hash function used to ensure uniqueness and integrity in the aggregation process.

- $\mathsf{Verify}(C, S, \{C^k_{i_k j_k}\}_{(i_k,j_k)\in\Delta S}, \{\Omega_{ij}\}_{(i,j)\in S})$: The verification process differs depending on the size of the set S:

  1. Case 1: $|S| = 1$ In this case, verification is required for a single element (i,j). The verifier must confirm the following equation: $e\left(C/g^{M_{ij}\alpha^j\beta^i}, g_2\right) = \prod_{\mathcal{A}^c_M(i,j)} e\left(C^k_{i_k j_k}, g_2^{\alpha^{j_k}\beta^{i_k}}\right)$, where $e(\cdot,\cdot)$ denotes the bilinear pairing, and $\mathcal{A}^c_M(i,j)$ is direct index associated of (i,j).

  2. Case 2: $|S| > 1$ When verifying multiple elements, the verifier must check the following two equations:

  - Eq. 1: $e\left(C/C_S, g_2\right) = \prod_{(i_k,j_k)\in\Delta S} e\left(C^k_{i_k j_k}, g_2^{\alpha^{j_k}\beta^{i_k}}\right)$, where C_S represents the aggregated commitment.

  - Eq. 2: $e\left(C, g_2^{\sum_{(i,j)\in S} h_{ij}}\right) = e(\hat{\Omega}, g_2) \cdot g_T^{\sum_{(i,j)\in S} M_{ij}h_{ij}}$, where $\hat{\Omega}$ is the product of individual proofs, h_{ij} represents the weight for each element, and g_T is a generator in the bilinear group $\mathbb{G}_T$. These conditions ensure the correctness and consistency of the commitments and aggregated proofs for the specified elements.

6 Analysis

In this section, we analyze the properties defined for our matrix commitment scheme. As demonstrated below, the MCTproofs satisfies the properties outlined in Sec. 4 and fully meets the security requirements of the scheme. This verification is performed specifically for the matrix M, but it is equally valid for other submatrices of M under the same framework.

Theorem 6.1. *The MCTproofs scheme is binding in the AGM+ROM model under the assumptions of* n-wBDHE* *and* (n,n)-*bBDHE.*

Proof. The proof is carried out in two steps:

step 1. Analysis of H-lucky queries:

Consider any query $(\mathcal{O}, C, S, M[S])$ issued by an algebraic adversary attempting to break the binding property of the MCTproofs scheme. To succeed, the adversary must extract matrices $A \in \mathbb{Z}_p^{n\times n}$ and $B \in \mathbb{Z}_p^{(n-1)\times(n-1)}$ such that the following condition is satisfied: $g_1^{\boldsymbol{\beta}^T A\boldsymbol{\alpha}+\beta^n\boldsymbol{\beta}^T[-1]B\boldsymbol{\alpha}[-1]\alpha^n}$. The query is defined as H-lucky if the following relation holds: $M[S] \equiv_p A[S]$ and $(M[S]-A[S])^T t \equiv_p 0$, where $\boldsymbol{h} = \big(H((i,j), C, S, M[S])) : (i,j) \in S\big)$

is a set derived from the cryptographic hash function H, applied to combinations of C, S, and $M[S]$. In the best-case scenario, the query $\mathcal{O}$ is H-lucky with a probability of $1/p$. For q_H queries to the function H, the probability that the adversary makes an H-lucky query is q_H/p.

Step 2. Infeasibility of extracting $g_1^{(\alpha\beta)^{n+1}}$: In this step, we demonstrate that the adversary $\mathcal{A}$ cannot extract $g_1^{(\alpha\beta)^{n+1}}$ without violating the (n,n)-bBDHE assumption. Suppose $\mathcal{A}$ outputs: $C, \{\{S^b, M^b[S^b], \hat{\Omega}^b\}\}_{b=0,1}$. Additionally, assume that $\mathcal{A}$ outputs matrices A and B such that C is computed as: $C = g_1^{\boldsymbol{\beta}^T A \boldsymbol{\alpha} + \beta^n \boldsymbol{\beta}^T [-1] B \boldsymbol{\alpha}[-1] \alpha^n}$. Under the given assumptions, we have: $M^0[S^0 \cap S^1] \neq M^1[S^0 \cap S^1] \implies M^0[S^0] \neq A[S^0] \vee M^1[S^1] \neq A[S^1]$. Let $(S^*, M^*, \hat{\pi}^*)$ be such that: $M^*[S^*] \neq A[S^*]$ and $\mathsf{Verify}(C, S^*, M^*[S^*], \hat{\pi}^*) = 1$. Since $\hat{\pi}^*$ is an accepting proof, the following relation holds:

$$e(C, g_2^{\sum_{(i,j)\in S^*} \alpha^{n+1-j}\beta^{n+1-i}h_{ij}}) = e(\hat{\pi}^*, g_2) \cdot g_T^{(\alpha\beta)^{n+1} M^*[S^*]^T \boldsymbol{h}}.$$

By eliminating g_2 from both sides using the properties of the pairing e, we obtain:

$$C^{\sum_{(i,j)\in S^*} \alpha^{n+1-j}\beta^{n+1-i}h_{ij}} = \hat{\pi}^* \cdot g_1^{(\alpha\beta)^{n+1} M^*[S^*]^T \boldsymbol{h}}.$$

Decomposing the left-hand side into terms involving $g_1^{(\alpha\beta)^{n+1}}$, we isolate dependencies:

$$C^{\sum_{(i,j)\in S^*} \alpha^{n+1-j}\beta^{n+1-i}h_{ij}} = \underbrace{(g_1^{\sum_{(i,j)\in S^*} \alpha^{n+1-j}\beta^{n+1-i}} \boldsymbol{\beta}^T[-i]A[-i][-j]\boldsymbol{\alpha}[-j]h_{ij})}_{\text{depeneds on } \quad g_1^{(\alpha\beta)^2},\ldots,g_1^{(\alpha\beta)^n},g_1^{(\alpha\beta)^{n+2}},\ldots,g_1^{(\alpha\beta)^{2n}}}$$
$$\times (g_1^{(\alpha\beta)^n A[S^*]\boldsymbol{h}}). \underbrace{(g_1^{(\alpha\beta)^n} \boldsymbol{\beta}^T[-i]B\boldsymbol{\alpha}[-j] \sum_{(i,j)\in S^*} \alpha^{n+1-j}\beta^{n+1-i}h_{ij})}_{\text{depeneds on } \quad g_1^{(\alpha\beta)^{n+3}},\ldots,g_1^{(\alpha\beta)^{3n}}}$$

The matrix $A[-i][-j]$ is obtained by removing the i-th row and j-th column from the matrix A. Next, multiply both sides of the equation by $\hat{\pi}^{*-1}$ on the LHS and $g_1^{(\alpha\beta)^{n+1}}$ on the RHS.

$$g_1^{\sum_{(i,j)\in S^*} \alpha^{n+1-j}\beta^{n+1-i}\boldsymbol{\beta}^T[-i]A[-i][-j]\boldsymbol{\alpha}[-j]h_{ij}} g_1^{(\alpha\beta)^n \boldsymbol{\beta}^T[-i]B\boldsymbol{\alpha}[-j]\sum_{(i,j)\in S^*} \alpha^{n+1-j}\beta^{n+1-i}h_{ij}} \hat{\pi^*}^{-1}$$
$$= g_1^{(\alpha\beta)^{n+1}(M^*[S^*]-A[S^*])^T \boldsymbol{h}}$$

Finally, given that $M^*[S^*] \neq A^*[S^*]$ and there are no H-lucky queries, we conclude that $(M[S] - A[S])^T t \equiv_p 0$. By computing the modular inverse r of t modulo p and raising both sides of the above equation to the power of r, we obtain $g_1^{\alpha^{n+1}}$ on the RHS. Since the LHS can be computed using the adversary's output and the known terms $g_1^{\boldsymbol{\alpha}\boldsymbol{\beta}^T}$, $g_1^{\alpha^n\beta^n\boldsymbol{\alpha}[-j]\boldsymbol{\beta}[-i]^T}$, and $g_1^{(\alpha\beta)^{2n}\boldsymbol{\alpha}\boldsymbol{\beta}^T}$, it follows that we can compute $g_1^{(\alpha\beta)^{n+1}}$. □

Theorem 6.2. *The MCTproofs satisfies the correctness of the opening property for any $(i,j) \in [n] \times [n]$ with respect to M_{ij}.*

Proof. To establish this property, the following equation must hold for any $(i,j) \in S \subseteq [n]\times[n]$ and $M_{ij} \in M[S]$:

$$e\left(C/g_1^{M_{ij}\alpha^j\beta^i}, g_2\right) = \prod_{\mathcal{A}_M^c(i,j)} e\left(C_{i_k j_k}^k, g_2^{\alpha^{j_k}\beta^{i_k}}\right). \tag{3}$$

First, the bilinear form of the matrix M is defined as follows: $f(\alpha,\beta)=\beta^T M\alpha=\sum_{i=1}^{n}\sum_{j=1}^{n}M_{ij}\alpha^j\beta^i$.

Using Th. 3.2 and rearranging the function f based on the $\mathcal{A}_M$, we obtain $f(\alpha,\beta)=\sum_{(i_k,j_k)\in\mathcal{A}_M^c(i,j)}\alpha^{j_k}\beta^{i_k}f^k_{i_kj_k}(\alpha,\beta)$. Subtracting $M_{ij}\alpha^j\beta^i$ from $f(\alpha,\beta)$ gives

$$f(\alpha,\beta)-M_{ij}\alpha^j\beta^i=\sum_{(i_k,j_k)\in\mathcal{A}_M^c(i,j)}\alpha^{j_k}\beta^{i_k}f^k_{i_kj_k}(\alpha,\beta).$$

Finally, by leveraging the properties of the bilinear pairing function e, we derive Eq. (3). □

Theorem 6.3. *The MCTproofs satisfies the commitment update correctness property when the matrix entry M_{ij} is updated to $M_{ij}+\delta$.*

Proof. According to Def. 7, when M_{ij} is updated to $M_{ij}+\delta$, the algorithms $\mathsf{Commit}(M')$ and $\mathsf{UpdCommit}(M)$ produce equivalent outcomes. Consequently, we have

$$C''=g_1^{\boldsymbol{\beta}^TM'\boldsymbol{\alpha}}=g_1^{\sum_{(k,l)\neq(i,j)}M_{kl}\alpha^l\beta^k+(M_{ij}+\delta)\alpha^j\beta^i}=g_1^{\sum M_{kl}\alpha^l\beta^k+\delta\alpha^j\beta^i}=Cg_1^{\delta\alpha^j\beta^i}=C'.$$ □

Theorem 6.4. *MCTproofs satisfy in proof update correctness for any $(i',j')\in S\subset[n]\times[n]$, when M_{ij} updated to $M_{ij}+\delta$*

Proof. Consider the case where $(i,j)\neq(i',j')$, and let $\mathsf{Prove}(i',j',M_{i'j'})$ denote the proof algorithm for the entry $M_{i'j'}$, which outputs $\Omega_{i'j'}$. Additionally, let $\mathsf{UpdProof}(i,j,i',j',\delta,\Omega'_{i'j'})$ represent the update algorithm introduced in 5.1.

Suppose M' is the updated matrix obtained after modifying M_{ij} to $M_{ij}+\delta$, and $\Omega'_{i'j'}$ and $\Omega''_{i'j'}$ are the proofs defined in Def. 7. Based on the structure of the proof $\Omega'_{i'j'}$ and the matrix M' under the index $\mathcal{A}^c_{M'}(i',j')$, there exists a submatrix $M^r_{i'_rj'_r}$ that includes M_{ij}, along with the corresponding commitment $C^r_{i'_rj'_r}$, where $C^r_{i'_rj'_r}=\mathsf{Commit}(M^r_{i'_rj'_r})$. Consequently, $\Omega'_{i'j'}$ can be reorganized as follows:

$$\Omega'_{i'j'}=\left[\prod_{\mathcal{A}_M(i',j')\setminus\{(i'_r,j'_r)\}}[C^k_{i'_kj'_k}]^{\alpha^{i'_k}\beta^{j'_k}}\right][C^r_{i'_rj'_r}]^{\alpha^{j'_r}\beta^{i'_r}}=\left[\prod_{\mathcal{A}_M(i',j')\setminus\{(i'_r,j'_r)\}}[C^k_{i'_kj'_k}]^{\alpha^{i'_k}\beta^{j'_k}}\right]g_1^{\alpha^{j'_r}\beta^{i'_r}f^r_{i'_rj'_r}(\alpha,\beta)}$$

$$=\left[\prod_{\mathcal{A}_M(i',j')\setminus\{(i'_r,j'_r)\}}[C^k_{i'_kj'_k}]^{\alpha^{i'_k}\beta^{j'_k}}\right]g_1^{\alpha^{j'_r}\beta^{i'_r}\left[\sum_{i=1}^{2^{l-r}}\sum_{j=1}^{2^{l-r}}M'_{ij}\alpha^j\beta^i\right]} \tag{4}$$

where $\sum_{i=1}^{2^{l-k}}\sum_{j=1}^{2^{l-k}}M'_{ij}\alpha^j\beta^i$ represents a bilinear form of the submatrix $M^k_{i'_rj'_r}$. By separating the above summation based on the updated value $M'_{ij}=M_{ij}+\delta$, we obtain $\sum_{i=1}^{2^{l-k}}\sum_{j=1}^{2^{l-k}}M_{ij}\alpha^j\beta^i+\delta\alpha^s\beta^t$.

Thus, the commitment $C^r_{i'_rj'_r}$ can be rearranged as follows:

$$C^r_{i'_rj'_r}=g_1^{\sum_{i=1}^{2^{l-k}}\sum_{j=1}^{2^{l-k}}M_{ij}\alpha^j\beta^i+\delta\alpha^s\beta^t}=C^r_{i'_rj'_r}\cdot g_1^{\delta\alpha^s\beta^t}.$$

By combining equations (4) and (5), we obtain:

$$\Omega'_{i'j'}=\prod_{\mathcal{A}^c(i',j')}\left[C^k_{i'_kj'_k}\right]^{\alpha^{j'_k}\beta^{i'_k}}\cdot g_1^{\delta\alpha^{j'_r+s}\beta^{i'_r+t}}=\Omega_{i'j'}\cdot g_1^{\delta\alpha^j\beta^i}=\Omega''_{i'j'}.$$

Here, the right-hand side of (4) corresponds to the output of the $\mathsf{UpdProof}$ algorithm $((i,j),(i',j'),\Omega_{i'j'})$, and the equality holds true. □

7 Application

We propose the use of MCTproofs in constructing a payment-only stateless cryptocurrency, aimed at reducing storage costs in blockchain systems like Ethereum. In traditional systems, validators must maintain the full state of the system, including account balances, which leads to high storage requirements. By contrast, a stateless cryptocurrency, as proposed by Edrax, eliminates the need for validators to store balances [8]. In this model, account balances are represented as a vector, with a commitment generated through a verifiable commitment (VC) scheme. Each transfer transaction includes the payer's balance and a proof, which validators use for validation without needing to store account balances. This approach requires updating the vector commitment and the balance proofs after each transaction. While previous works have introduced aggregatable and maintainable VC schemes, they suffer from inefficiencies, particularly an information transmission problem. MCTproofs offer a solution by enabling a stateless cryptocurrency that is both aggregatable and maintainable, eliminates the information transmission problem, and allows more efficient aggregation than existing systems. This solution introduces a matrix-based structure for organizing account balances in a stateless cryptocurrency system. The system is initialized with a matrix M set to zero, and two algorithms, Commit and Prove, are executed to generate the initial commitment and individual proofs for each account. The matrix commitment is included in every block, while each account owner maintains their individual proof. When proposing a transaction (TX), the payer includes their balance and the individual proof in the TX. A block proposer validates transactions, aggregates the individual proofs, and updates the matrix commitment using the Verify, AggProof, and UpdCommit algorithms, respectively. The proposed block includes the valid TXs, the aggregated proof (instead of individual proofs), and the new commitment. Validators then verify the aggregated proof with the Verify algorithm and the new commitment by running UpdCommit. Once the block is confirmed by most validators, individual proofs are updated with the UpdProof algorithm. This method reduces storage requirements while ensuring secure and efficient transaction validation in the stateless system.

8 Complexity and Efficiency

The schemes presented in this paper primarily focus on square matrices of dimension $n = 2^l$. In this section, we evaluate the computational complexity and efficiency of the proposed scheme and compare it with Matproofs and Hyperproofs. In Matproofs, commitments are generated for a matrix $M_{m\times n}$, with proofs provided for an arbitrary position (i, j). In Hyperproofs, the commitment is generated for a vector of size n, with proofs available for an arbitrary position i. Our scheme specifically considers the case where $m = n$ and employs a novel approach. As a result, the number of positions in M is equal to n^2. As previously noted, $\mathcal{P}_M$ represents a perfect nested window partition for M, where the number of its elements, according to Def. 2, is computed as follows:

$$| \mathcal{P}_M | = | \mathcal{P}_M^0 | + \cdots + | \mathcal{P}_M^l | = 1 + \cdots + 4^l = \frac{4^l - 1}{3}.$$

Therefore, the number of elements in $\mathcal{A}_M$ is also $(4^l - 1)/3$. The time complexity and computational cost of the MCTproofs scheme are analyzed in the following discussion.

- **Pubic Parameters**: MCTproofs uses trees $\mathcal{T}_{pp_1}$ and $\mathcal{T}_{pp_2}$ to commit to matrix M and its elements and submatrices. Additionally, the generation of proofs, proving keys, and verification keys is based on these trees. To construct $\mathcal{T}_{pp_1}$ as shown in Example 4, and to generalize it for an arbitrary $n = 2^l$, the number of elements that must be computed in $\mathbb{G}_1$ is

$$| \mathcal{T}_{pp_1} | = n^2(1 + \frac{1}{4} + \cdots + \frac{1}{4^l}) \approx \frac{4}{3}n^2.$$

Thus, constructing $\mathcal{T}_{pp_2}$ requires $\frac{4}{3}n^2$ elements in $\mathbb{G}_2$. Consequently, both $\mathcal{T}_{pp_1}$ and $\mathcal{T}_{pp_2}$ are $O(n^2)$-time trees. If the matrix M is represented as a vector of size n^2, Hyperproofs necessitates the generation of $4n^2 - 1$ public parameters in $\mathbb{G}_1$. The total number of elements in the public parameter sets pp_1 and pp_2 is $2n^2 + n$ and $2n + 2$ in $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. From this perspective, our scheme demonstrates greater efficiency and competitiveness compared to Hyperproofs.

- **Proof Size**: For each position $(i, j) \in [n] \times [n]$, the Prove algorithm generates Ω_{ij}, consisting of a single element in $\mathbb{G}_1$. The Commit algorithm produces $\frac{4}{3}n^2$ elements in $\mathbb{G}_1$, which act as commitments to the elements of $\mathcal{P}_M$. Compared to MatProofs, the Prove algorithm in our scheme reduces the number of generated elements to 1. However, the number of elements produced by the Commit algorithm scales with n, which affects its conciseness. In MCTproofs, the Aggproof algorithm generates a fixed number of 3 elements in $\mathbb{G}_1$, whereas in Matproofs, the number of elements generated by Aggproof in $\mathbb{G}_1$ is $|\bar{S}| + 2$, depending on $|S|$. As a result, the *Aggproof* algorithm in our scheme has a constant size.
- **Time Complexity**: The MCTproofs scheme inherits certain features from the Matproofs scheme to compute the time complexity of its algorithms. In both schemes, the Commit algorithm performs n^2 exponentiations to commit a matrix M in $\mathbb{G}_1$. In the MCTproofs scheme, the Commit algorithm generates 4^k commitments, denoted as $C^k_{i_k j_k}$, for each pair $(i_k, j_k) \in \mathcal{A}_M$. Each commitment involves 4^{l-k} exponentiations, implying that the total computational cost of the Commit algorithm in MCTproofs amounts to $(l + 1)n^2$ exponentiations in $\mathbb{G}_1$. The UpdCommit algorithm in MCTproofs updates all submatrices containing the element M_{ij}, represented by $\{M^k_{i_k j_k} \mid (i_k, j_k) \in \mathcal{A}_M(i, j)\}$, when M_{ij} is updated to $M_{ij} + \delta$. This update requires l exponentiations in $\mathbb{G}_1$. The Prove algorithm generates proofs associated with a specific position (i, j) in the matrix M. The computation of $\Omega_{i,j}$ is based on the set $\{A^k_{i_k j_k} \mid (i_k, j_k) \in \mathcal{A}^c_M\}$, and the computational cost for generating $\Omega_{i,j}$ involves 3^l exponentiations in $\mathbb{G}_1$. The cost of generating an individual proof for position (i, j) is $4^l + 2^l - 1$ exponentiations. Overall, the MCTproofs scheme requires fewer computations compared to Matproofs. In the UpdProof algorithm of MCTproofs, when M_{ij} is updated to $M_{ij} + \delta$, only one element of Ω_{ij} is affected, requiring a single exponentiation in $\mathbb{G}_1$. In contrast, the Matproofs scheme may require updates to at most two elements in $\mathbb{G}_1$. Overall, both schemes support efficient updates. For any $S \subseteq [n] \times [n]$, Aggproof requires $|S| + 2$ exponentiations in $\mathbb{G}_1$ to aggregate $\{\Omega_{ij}\}_{(i,j)\in S}$. Specifically, $|S|$ exponentiations are needed for $\hat{\Omega}$, one exponentiation for C_S, and one for $\hat{\omega}$.

8.1 Performance and comparison

We evaluated the computational costs of the Aggproof and Verify algorithms, comparing their performance under identical parameter settings ($|S|$ and n) as those used in the Matproofs. This comparison was conducted based on the average performance of these algorithms over five independent experimental trials. The implementation of MCTproofs was carried out using Python version 3.1.1, leveraging libraries such as pandas, pypbc, and matplotlib for computation and visualization. The code was executed in a single-threaded configuration on an Intel(R) Xeon(R) E-2287G CPU operating at 4.0 GHz, featuring six cores and supported by 64 GB of memory.

Aggregation Proofs In Matproofs, Aggproof algorithm aggregates individual proofs for each subset $S \subseteq [n] \times [n]$ to generate the proof for $M[S]$. This process requires $|\bar{S}|$, $|S|$,

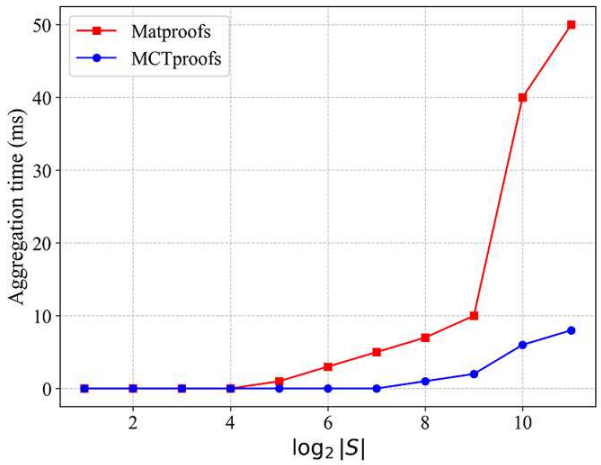


Figure 5: Aggregation time of $|S|$ proofs ($n = 2^{11}$)

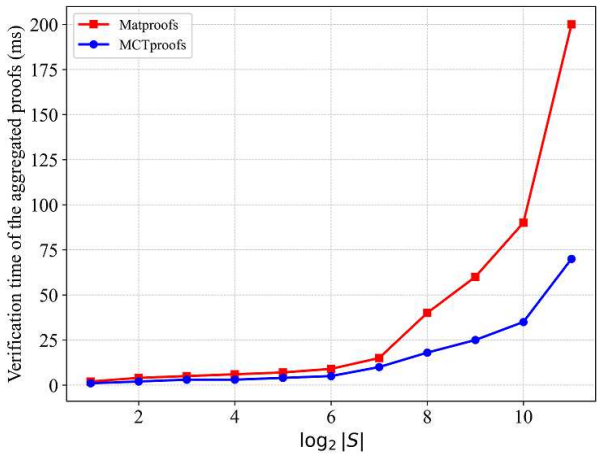


Figure 6: Verification time of $|S|$ proofs ($n = 2^{11}$)

and $|\bar{S}|$ exponentiations in $\mathbb{G}_1$ for $\bar{\omega}$, Ω_i, and $\hat{\Omega}$, respectively. In contrast, the Aggproof algorithm within the MCTproofs requires a total of $|S| + 1$ exponentiations in $\mathbb{G}_1$, with $|S|$ exponentiations for $\hat{\Omega}$ and one for $\hat{\omega}$. Our computational analysis indicates that, with assumed $|S|$ and n as used in Matproofs, the Aggproof algorithm achieves an average performance improvement of approximately 10 times over Matproofs in the best-case scenario (see Fig. 5).

Verification To validate an Aggproof for a subset S with $|S|$ elements in M, Matproofs requires $|\bar{S}| + 1$ exponentiations in $\mathbb{G}_1$, $|S| + |\bar{S}|$ exponentiations in $\mathbb{G}_2$, and $2|\bar{S}| + 3$ pairing computations. In contrast, our proposed scheme requires only 1 exponentiation in $\mathbb{G}_1$, 1 exponentiation in $\mathbb{G}_2$, and $|\Delta S| + 1$ pairings. To evaluate performance, we measured the computational cost of our scheme for $n = 2^{11}$ and $|S| \in \{2^2, 2^3, \ldots, 2^{11}\}$, comparing it with the Verify algorithm in Matproofs. Simulation results demonstrate that, on average, our scheme is 3 times faster than the Verify algorithm in Matproofs (see Fig. 6).

8.2 Performance in cryptocurrencies

We compare the performance of MCTproofs with Hyperproofs and Matproofs in the context of payment-only stateless cryptocurrencies. All three schemes including Hyperproofs, Matproofs,

and MCTproofs are synchronously aggregatable and maintainable. In addition to these properties, MCTproofs, like Matproofs, is also easily updatable. Unlike Hyperproofs, MCTproofs offers concise individual proofs, whose sizes remain independent of the committed matrices or vectors. We consider a matrix of dimension $n = 2^l$ in our computations and comparisons. We conducted a similar comparison with Hyperproofs and Matproofs for values of $|S| = 1024$ and $l \in \{11, 12, 13, 14, 15, 16\}$. The MCTproofs implementation in the UpdCommit algorithm exhibits slower performance compared to standard MCTproofs; however, the difference in speed is negligible and does not significantly impact overall functionality. The experimental results demonstrate the improved performance of MCTproofs in the Aggproofs and Verify algorithms, compared to Matproofs. Furthermore, in all other algorithms, MCTproofs outperforms Hyperproofs. Details of the comparison of the performance and the implementation results are given in Tab. 2.

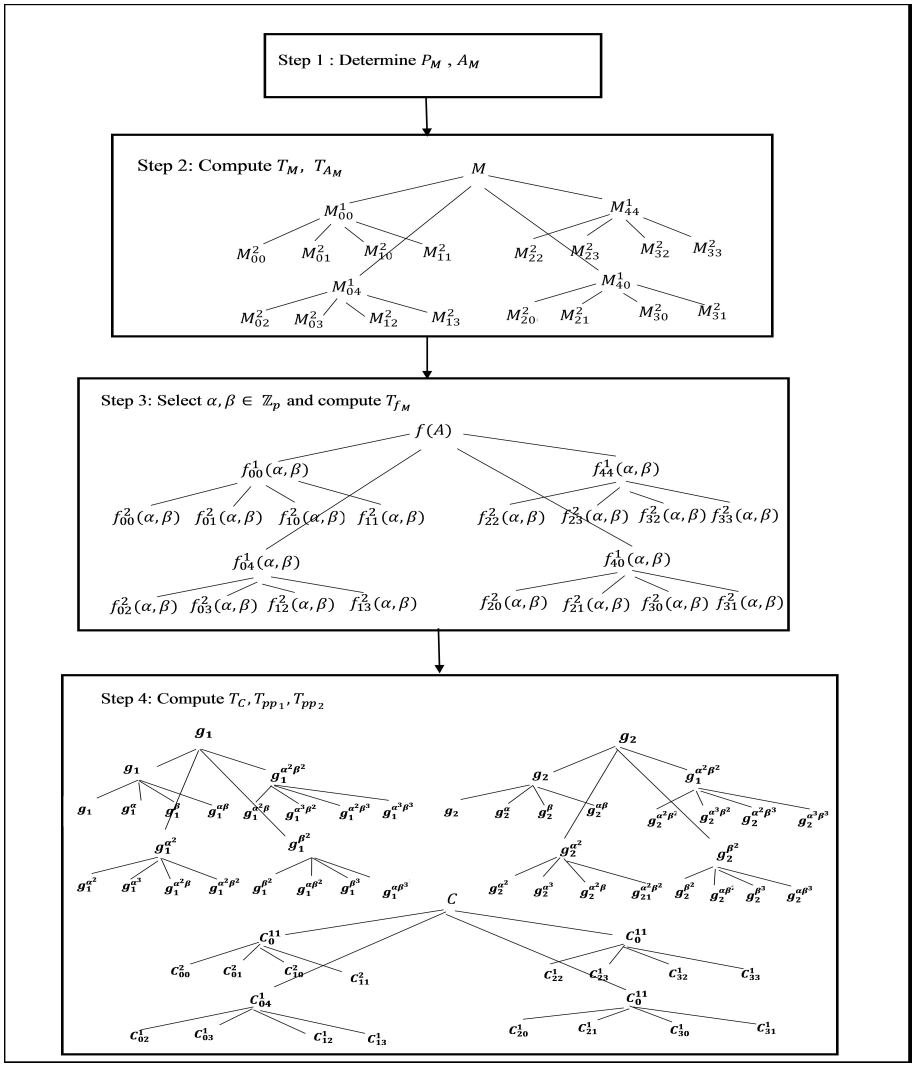
Table 2: Comparison of the performance between MCTproofs, Matproofs and Hyperproofs. The blue row indicates the performance improvement of MCTproofs, while the red row highlights the decrease in performance relative to the other two schemes.

$l = \log_2 n$		11	12	13	14	15	16
Individual proof size (KiB)	MCTproofs	0.11	0.13	0.14	0.15	0.18	0.19
	Matproofs	0.12	0.13	0.14	0.16	0.17	0.2
	Hyperproofs	1.03	1.13	1.22	1.31	1.41	1.62
Aggregated proof size (KiB)	MCTproofs	45.14	46.72	46.86	49.01	49.03	50.01
	Matproofs	46.12	47.51	48.09	49.06	49.09	50.03
	Hyperproofs	50.63	50.63	50.63	50.63	50.63	50.63
Verify an individual proof(ml)	MCTproofs	1.85	1.89	2.04	2.14	2.26	2.38
	Matproofs	1.82	1.91	2.07	2.13	2.27	2.35
	Hyperproofs	4.15	4.31	4.62	4.87	5.12	5.24
Aggregate proofs (s)	MCTproofs	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
	Matproofs	0.06	0.06	0.07	0.07	0.07	0.08
	Hyperproofs	53.51	53.88	56.95	58.26	59.59	60.23
Verify an aggregate proofs(s)	MCTproofs	0.17	0.18	0.19	0.22	0.23	0.24
	Matproofs	0.52	0.55	0.57	0.64	0.68	0.73
	Hyperproofs	6.30	6.60	7.29	7.74	8.22	8.86
Update all proofs(ms)	MCTproofs	249.23	513.31	1023.73	2161.36	4412.59	8448.08
	Matproofs	236.85	491.70	966.94	1950.87	3838.04	7709.81
	Hyperproofs	1.32	1.37	1.61	2.07	2.24	2.47

Conclusion

We introduced MCTproofs, a novel matrix commitment scheme based on the bilinear form decomposition of square matrices. This approach offers substantial improvements over existing schemes, notably achieving at least ten times faster proof aggregation and three times faster verification compared to Matproofs. By utilizing multiple tree structures, MCTproofs optimizes both proof generation and verification processes, highlighting its efficiency and innovative design. Experimental results demonstrate that MCTproofs not only competes effectively with Matproofs and Hyperproofs in payment-only stateless cryptocurrencies but also delivers superior performance, making it a promising advancement in the blockchain technology.

Appendix: The process of building trees in MCTproofs



References

[1] Liu J., Zhang L. F., *Matproofs: Maintainable matrix commitment with efficient aggregation,* in Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, 2022, 2041-2054.

[2] Srinivasan S., Chepurnoy A., Papamanthou C., Tomescu A., Zhang Y., *Hyperproofs: Aggregating and maintaining proofs in vector commitments,* in 31st USENIX Security Symposium (USENIX Security 22), 2022, 3001-3018.

[3] Gorbunov S., Reyzin L., Wee H., Zhang Z., *Pointproofs: Aggregating proofs for multiple vector commitments,* in Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, 2020, 2007-2023.

[4] Yong X., Wu J., Wang J., *Xproofs: new aggregatable and maintainable matrix commitment with optimal proof size,* in IFIP International Conference on ICT Systems Security and Privacy Protection, 2024, 465-480.

[5] Merkle R. C. *A digital signature based on a conventional encryption function,* in Conference on the theory and application of cryptographic techniques, 1987, 369-378.

[6] Gorbunov S., Reyzin L., Wee H., Zhang Z., *Pointproofs: Aggregating proofs for multiple vector commitments,* in Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, 2020, 2007-2023.

[7] Boyen X., *The uber-assumption family: A unified complexity framework for bilinear groups,* in International Conference on Pairing-Based Cryptography, 2008, 39-56.

[8] Chepurnoy A., Chattopadhyay S., Papamanthou C., Srinivasan S., Zhang Y., *A cryptocurrency with stateless transaction validation,* Cryptology ePrint Archive, 2018.

[9] Subhas B., Chattopadhyay S., Samanta D., Barman S., *A blockchain-based approach to secure electronic health records using fuzzy commitment scheme,* Security and Privacy, 5 (2022), 231-241.

[10] Boneh D., Boyen X., Goh E.-J., *Hierarchical identity based encryption with constant size ciphertext,* in Annual international conference on the theory and applications of cryptographic techniques, 2005, 440-456.

[11] Merkle R.C., *Protocols for public key cryptosystems,* in Secure communications and asymmetric cryptosystems, 2019, 73-104.

[12] Micali S., Rabin M., Kilian J., *Zero-knowledge sets,* in Proceedings of 44th Annual IEEE Symposium on Foundations of Computer Science, 2003, 80-91.

[13] Kate A., Zaverucha G.M., Goldberg I., *Constant-size commitments to polynomials and their applications,* in Advances in Cryptology-ASIACRYPT 2010: 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings 16, 177-194.

[14] Papamanthou C., Shi E., Tamassia R., Yi K., *Streaming authenticated data structures,* in Advances in Cryptology–EUROCRYPT 2013: 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings 32, 353-370.

[15] Pâris J.-F., Schwarz T., *Merkle hash grids instead of merkle trees,* in 2020 28th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), 2020, 1-8.

[16] Bünz B., Bootle J., Boneh D., Poelstra A., Poelstra A., Wuille P., Maxwell G., *Bulletproofs: Short proofs for confidential transactions and more,* in 2018 IEEE symposium on security and privacy (SP), 2018, 315-334.

[17] Boneh D., Bünz B., Fisch B. *Batching techniques for accumulators with applications to iops and stateless blockchains,* in Advances in Cryptology-CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part I 39, 561-586.

[18] Tomescu A., Abraham I., Buterin V., Drake J., Feist D., Khovratovich D., *Aggregatable subvector commitments for stateless cryptocurrencies,* in International Conference on Security and Cryptography for Networks, 2020, 45-64.

[19] Wang W., Ulichney A., Papamanthou C., {*BalanceProofs*}*: Maintainable vector commitments with fast aggregation,* in 32nd USENIX Security Symposium (USENIX Security 23), 2023, 4409-4426.

[20] Bentley J.L., *Multidimensional binary search trees used for associative searching,* Communications of the ACM, 18.9 (1975), 509-517.

[21] Samet H., *The quadtree and related hierarchical data structures,* ACM Computing Surveys (CSUR), 16.2 (1984), 187-260.

Devisti H. ,
School of Mathematics and Computer Science,
Iran University of Science,
Hengam, Tehran, Province, Iran,
Email: *h_devisti@mathdep.iust.ac.ir*

Hadian M.,
School of Mathematics and Computer Science, Iran
University of Science and Technology,
Hengam, Tehran, Tehran Province, Iran,
Email: *mhadian@iust.ac.ir*

Received 09.12.2024, Accepted 06.10.2025, Available online 31.12.2025.