Xproofs: New Aggregatable and Maintainable Matrix Commitment with Optimal Proof Size

Xinwei Yong, Jiaojiao Wu, and Jianfeng Wang(✉)

School of Cyber Engineering, Xidian University, Xi'an, China
{xwyong,jiaojiaowujj}@stu.xidian.edu.cn
{jfwang}@xidian.edu.cn

Abstract. Vector Commitment (VC) enables one to commit to a vector, and then the element at a specific position can be opened, with proof of consistency to the initial commitment. VC is a powerful primitive with various applications, including stateless cryptocurrency. Recently, matrix commitment Matproofs (Liu and Zhang CCS 2022), as an extension of VC, has been proposed to reduce the communication and computation complexity of VC-based cryptocurrency. However, Matproofs requires linear-sized public parameters, and the aggregated proof size may also increase linearly with the number of individual proofs aggregated. Additionally, the proof updating process involves the third party, known as Proof-Serving Nodes (PSNs), which leads to extra storage and communication overhead. In this paper, we first propose a multi-dimensional variant of matrix commitment and construct a new matrix commitment scheme for two-dimensional matrix, called 2D-Xproofs, which achieves optimal aggregated proof size without using PSNs. Furthermore, we present a highly maintainable three-dimensional scheme, 3D-Xproofs, which updates all proofs within time sublinear in the size of the committed matrix without PSNs' assistance. More generally, we could further increase the matrix dimensionality to achieve more efficient proof updates. Finally, we demonstrate the security of our schemes, showing that both schemes are position binding. We also implement both schemes, and the results indicate that our schemes enjoy constant-sized aggregated proofs and sublinear-sized public parameters, and the proof update time in 3D-Xproofs is 2.5× faster than Matproofs.

Keywords: Vector commitment · Matrix commitment · Stateless cryptocurrency.

1 Introduction

Vector Commitment (VC) [13,8] allows one to compute a commitment C for a vector $\mathbf{m} = (\mathrm{m}_1,\mathrm{m}_2,\ldots,\mathrm{m}_n)$ and later can generate the opening proofs $\boldsymbol{\pi}_1,\boldsymbol{\pi}_2,\ldots,\boldsymbol{\pi}_n$ for all positions. With the commitment C and a proof $\boldsymbol{\pi}_i$, any verifier can check the correctness of the element m_i positioned at i. VC serves as a foundational tool for various applications, including stateless cryptocurrency. In the VC-based stateless cryptocurrency, instead of storing the entire ledger state (the information of all accounts and historical transactions), the validators only need to store the commitment of a vector representing the account balances from all cryptocurrency users. Specifically, when initiating a coin transfer, the payer submits a transaction, showing its account balance and proving it is adequate. Upon receiving the transaction, the validator checks the correctness of the balance by using the corresponding balance proof and the account commitment. If the verification succeeds and the transfer coins are less than the payer's balance, this transaction will be stored in a new block. Finally, all accounts synchronize to update their local proofs based on the new block information.

To achieve a better balance between storage, communication and computation, numerous VC-based schemes [1,4,9,11,14,17,19,20] have been proposed to meet various properties, such as conciseness, aggregatable, easily updatable, and maintainable. In particular, Matrix Commitment (MC) and its instantiated scheme Matproofs, proposed by Liu et al [14], stand out as the only VC construction that satisfies all properties. However, the size of the aggregated proof is not constant in Matproofs, either increasing linearly with the number of individual proofs aggregated or sublinearly with respect to the size of the matrix, i.e. $O(\min\{b,\sqrt{n}\})$, where b is the number of aggregated proofs and n is the number of committed elements. Recently, BalanceProofs [20], based on polynomial commitments, achieves a constant size for both individual proofs and aggregated

Table 1. Comparisons with the existing easily updatable VCs

Scheme	pp size	Individual proof size	Aggregated proof size	Aggregate proofs time	Verify aggregated proof time	Open all proofs time	Update all proofs time
Edrax[9]	$O(n)$	$O(\log n)$	×	×	×	$O(n)$	$O(n\log n)$
aSVC[19]	$O(n)$	$O(1)$	$O(1)$	$O(b\log^2 b)$	$O(b\log^2 b)$	$O(n)$	$O(n)$
Pointproofs[11]	$O(n)$	$O(1)$	$O(1)$	$O(b)$	$O(b)$	$O(n\log n)$	$O(n)$
Hyperproofs[17]	$O(n)$	$O(\log n)$	$O(\log(b\log n))$	$O(b\log n)$	$O(b\log n)$	$O(n\log n)$	$O(\log n)^*$
Matproofs[14]	$O(n)$	$O(1)$	$O(\min\{b,\sqrt{n}\})$	$O(b+\min\{\sqrt{n},b\})$	$O(b+\min\{\sqrt{n},b\})$	$O(n\log n)$	$O(\sqrt{n})^*$
BalanceProofs[20]	$O(n)$	$O(1)$	$O(1)$	$O(b\log^2 b)$	$O(b\log^2 b)$	$O(n\log n)$	$O(\sqrt{n}\log n)^*$
Two-layer[20]	$O(n)$	$O(1)$	$O(\min\{b,\sqrt{n}\})$	$O(b\log^2 b)$	$O(b\log^2 b)$	$O(n\log n)$	$O(n^{1/4}\log n)^*$
2D-Xproofs	$O(\sqrt{n})$	$O(1)$	$O(1)$	$O(b+\min\{\sqrt{n},b\})$	$O(b+\min\{\sqrt{n},b\})$	$O(n\log\sqrt{n})$	$O(\sqrt{n})$
3D-Xproofs	$O(n^{2/3})$	$O(1)$	$O(1)$	$O(b+\min\{n^{1/3},b\})$	$O(b+\min\{n^{1/3},b\})$	$O(n^{2/3}\log n^{1/3})$	$O(n^{1/3})$

Note: The number of committed elements is n, and the number of aggregated individual proofs is b. The symbol "$*$" indicates that the process of updating all proofs relies on the assistance of PSNs. The proof size and time complexity are evaluated by group elements and group exponentiations/field operations, respectively.

proofs. However, in its two-layer varaints scheme, the size of the aggregate proof is still not constant. Moreover, all existing schemes [9,11,14,17,19,20] require public parameters of linear size, that is, $O(n)$.

In particular, during the process of updating proofs, two important properties are of concern: being easily updatable and maintainable. We know that a VC scheme is *easily updatable* if the update process can be executed for new elements based on the old commitment and proofs, without needing involvement with the old elements. To alleviate computational burden on cryptocurrency user, many schemes [14,17,20] utilize the third party Proof-Serving Nodes (PSNs) to update all proofs. If PSNs enable efficient updates of all proofs within sublinear time, we refer to such VC schemes as *maintainable*. However, the use of third-party PSNs may introduce additional storage and communication overhead, as well as potential security risks, to the stateless cryptocurrency system. Therefore, making a VC scheme highly maintainable, by eliminating PSNs and improving proof update efficiency, is highly meaningful.

In this paper, we further explore the design of matrix commitment schemes, aiming to address the following question:

How to design an easily updatable and highly maintainable commitment scheme without relying on PSNs, while achieving constant-sized aggregated proofs and sublinear-sized public parameters?

1.1 Our Contributions

In this paper, we provide a positive solution to the above question. We propose new multi-dimensional matrix commitment schemes, 2D-Xproofs and 3D-Xproofs, which have constant-sized aggregated proofs and sublinear-sized public parameters. Additionally, both schemes allow cryptocurrency users to efficiently update all proofs only within sublinear time in the matrix size, without third-party PSNs, when Xproofs serves as the basis for a stateless cryptocurrency. A brief comparison with previous works is shown in Table 1. In more detail, our main contributions are summarized as follows:

- We propose a new matrix commitment scheme 2D-Xproofs with optimal proof size and minimal public parameter size. Specifically, the individual proofs and aggregated proofs are of constant size $O(1)$, which consist of only two group elements. The size of public parameters is sublinear in the size of the committed matrix. In addition, the efficiency of proof updates in 2D-Xproofs without using PSNs closely rivals that of Matproofs [14] when PSNs are employed.
- We further obtain a three-dimensional scheme 3D-Xproofs, which also enjoys constant-sized aggregated proofs and sublinear-sized public parameters. Furthermore, in the 3D-Xproofs scheme, the update speed for all proofs is 2.5× faster than Matproofs [14]. In particular, we could increase the dimensionality further to achieve even more efficient proof updates.
- We provide a security analysis for our schemes, demonstrating that they satisfy position binding. Additionally, we implement 2D-Xproofs, 3D-Xproofs, and Matproofs [14] and perform a comprehensive evaluation and comparison. The result demonstrates that our schemes have optimal communication overhead and improved storage and computation costs compared to the-state-of-the-art Matproofs [14].

1.2 Related Work

The vector commitment schemes based on the Merkle tree [15] and k-ary Verkle tree [12] store messages within its leaf nodes. In the event of a message update, the cost of updating the proof incurs only $O(\log n)$ computational overhead, rendering the scheme maintainable. However, updating the commitment requires the old information of the updated message, making it not easily updatable. Furthermore, proofs from binary or multi-way Merkle trees are challenging to aggregate, and the proof size is directly linked to the tree's height.

Vector commitment schemes based on RSA [4,1] of this kind support batch openings for multiple positions within the vector and can directly generate batch proof for a set of positions. Both commitment and proof sizes remain constant. However, while proofs generated by this scheme do support aggregation, the efficiency of aggregation is relatively low and typically necessitates the utilization of Shamir's Trick technique. Furthermore, the time complexity for updating all proofs after a change in a single entry is $O(n)$, thus it is not maintainable. The proofs of the Pointproofs [11] and the VC of [19] schemes are of constant size and support aggregation. Additionally, both of these schemes are easily updatable but not maintainable.

Edrax [9] is based on polynomial commitment, which is easily updatable. However the size of the proofs is not constant, and it is not aggregatable and maintainable. Hyperproofs [17] is based on the PST commitment [16], which is easily updatable and maintainable. However the size of its proofs is not constant, and it requires the use of the inner-product argument (IPA) [6] to achieve proofs aggregation. Balanceproofs [20] is easily updatable, and its basic scheme has a constant size of individual proof and aggregated proof. Although it achieves sub-linear updates for all proofs, it is not efficient enough. The two-layer scheme of Balanceproofs improves the efficiency of updates for all proofs, but the aggregated proofs cannot maintain a constant size. Matproofs [14] is easily updatable, aggregatable, and maintainable. However, the size of its aggregated proofs is not constant. Although these schemes [17,14,20] can update all proofs in sublinear time, they all require the introduction of PSNs. This is because when an entry is updated, it affects the proofs at all positions. By using PSNs, the shared components of all proofs are calculated only once and then distributed to all positions. Therefore, it is only by introducing PSNs and increasing certain communication costs that maintainability can be achieved.

2 Preliminaries

For any $n \in \mathbb{N}$, let $[n] = \{1, 2, ..., n\}$. Let λ denote the security parameter. Let $\boldsymbol{\alpha}$ denote a vector of length n. For any position $(i, j) \in [n] \times [n]$, we denote by $\mathbf{M}_{ij}$ the entry corresponding to the position (i, j) of $\mathbf{M}$. For any $S \subseteq [n] \times [n]$, we denote $\mathbf{M}[S] = \{\mathbf{M}_{ij}\}_{(i,j)\in S}$. For any $i \in [n]$ and $j \in [n]$, we denote by $\mathbf{M}_{i*}$ and $\mathbf{M}_{*j}$ the entries in the same row i and same column j respectively, and denote $\boldsymbol{\alpha}[-i] = \boldsymbol{\alpha}[[n]\backslash\{i\}]$.

2.1 Bilinear Pairings

A probabilistic polynomial-time (PPT) algorithm $\mathsf{BG}(1^\lambda)$ takes a security parameter λ as input and outputs a *bilinear context* $\mathbf{bg} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$, where $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ are multiplicative cyclic groups with prime order $p \approx 2^\lambda$, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a *bilinear pairing*, satisfying the following properties:

- ***Efficiently computable:*** There is a polynomial-time algorithm to compute $e(g_1, g_2)$ for all $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$.
- ***Bilinearity:*** For any $g_1 \in \mathbb{G}_1$, $g_2 \in \mathbb{G}_2$ and $\alpha, \beta \in \mathbb{Z}_p$: $e(g_1^\alpha, g_2^\beta) = e(g_1, g_2)^{\alpha\beta}$.
- ***Non-degeneracy:*** For g_1 and g_2 being generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively, holds that: $e(g_1, g_2) \neq 1$.

2.2 Security Assumptions

ℓ-wBDHE* problem [11]. Let $\mathbf{bg} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2) \leftarrow \mathsf{BG}(1^\lambda)$ be a bilinear context generated using a bilinear group generator. The ℓ-wBDHE* assumption [3,5,7] states that the probability of a probabilistic polynomial-time (PPT) adversary successfully solving a variant of

the weak bilinear Diffie-Hellman exponent problem is negligible. This assumption holds in the generic bilinear group model, which provides a reasonable level of security for cryptographic protocols relying on bilinear pairings:

$$\text{Given}: g_1^{\alpha}, g_1^{\alpha^2}, ..., g_1^{\alpha^{\ell}}, g_1^{\alpha^{\ell+2}}, ..., g_1^{\alpha^{3\ell}}, g_2^{\alpha}, g_2^{\alpha^2}, ..., g_2^{\alpha^{\ell}}, \text{ where } \alpha \xleftarrow{\$} \mathbb{Z}_p$$
$$\text{Compute}: g_1^{\alpha^{\ell+1}}$$

The algebraic group model(AGM)[11,14]. Our security proofs adopt an AGM+ROM model, which combines the algebraic group model (AGM) [10] and the random oracle model (ROM) [2]. In AGM, adversaries are limited to algebraic operations and have access to group elements, including their bit representations. However, they can only produce new group elements by applying group operations to the received elements. Specifically, when given a set of group elements $I_1, ..., I_n \in \mathbb{G}$, the adversary must output the corresponding exponents $e_1, ..., e_n \in \mathbb{Z}p$ in order to construct a new group element $O = \prod j = 1^n I_j^{e_j}$.

In the ROM, cryptographic hash functions are modeled as truly random functions with an output space $\mathbb{Z}_p$. In our protocol, all parties can interact with these hash functions solely through Oracle queries.

2.3 Matrix Commitment

We extend the matrix commitment cryptographic primitive proposed by Liu et al.[14], which allows one to commit to a two-dimensional or three-dimensional matrix and generate the opening proofs for any specific position $(i, j) \in [n] \times [n]$ or $(i, j, k) \in [n] \times [n] \times [n]$. A matrix commitment scheme consists of the following seven algorithms:

- $\mathsf{Setup}(1^\lambda, 1^n)$: It takes a security parameter λ and the upper limit n for every coordinate in the two-dimensional or three-dimensional matrix as inputs and then outputs a set of public parameters, which can be accessed by all subsequent algorithms.
- $\mathsf{Commit}(\mathbf{M})$: It takes an $n \times n$ two-dimensional or $n \times n \times n$ three-dimensional matrix $\mathbf{M}$ as inputs and outputs the commitment C.
- $\mathsf{UpdateCommit}(\mathsf{pos}, \delta, C)$: It takes an increment δ for updating the message at any position $\mathsf{pos} \to (i, j) \in [n] \times [n]$ or $\mathsf{pos} \to (i, j, k) \in [n] \times [n] \times [n]$ and current commitment C as inputs, then outputs the updated commitment C'.
- $\mathsf{Prove}(\mathsf{pos}, \mathbf{M})$: It takes any position pos and the messages $\mathbf{M}$ as inputs, and outputs a proof π_{pos} for the entry $\mathbf{M}_{\mathsf{pos}}$.
- $\mathsf{UpdateProof}(\mathsf{pos}_1, \mathsf{pos}_2, \delta, \pi_{\mathsf{pos}_2})$: It takes an increment δ for updating the message $\mathbf{M}_{\mathsf{pos}_1}$ at position pos_1, as well as another position pos_2 with its corresponding old proof π_{pos_2} as inputs, and outputs the updated proof π'_{pos_2} for position pos_2.
- $\mathsf{AggregateProof}(C, S, \mathbf{M}[S], \{\pi_{\mathsf{pos}}\}_{\mathsf{pos} \in S})$: It takes commitment C, a set of positions $S \subseteq [n] \times [n]$ or $S \subseteq [n] \times [n] \times [n]$ and their corresponding messages $\mathbf{M}[S]$ and proofs $\{\pi_{\mathsf{pos}}\}_{\mathsf{pos} \in S}$ as inputs, then outputs aggregated proof $\hat{\Omega}$ for $\mathbf{M}[S]$.
- $\mathsf{Verify}(C, S, \mathbf{M}[S], \hat{\Omega})$: It takes commitment C, any positions set S with its corresponding messages $\mathbf{M}[S]$ and their aggregated proof $\hat{\Omega}$ as inputs, and verifies the correctness of the proof $\hat{\Omega}$ concerning the commitment C.

Definition 1 (Correctness of Opening). *For all λ, $n > 0$, matrix $\mathbf{M}$, and any position* pos*,*

$$\Pr\left[\begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^n), \\ C \leftarrow \mathsf{Commit}(\mathbf{M}), \\ \pi_{\mathsf{pos}} \leftarrow \mathsf{Prove}(\mathsf{pos}, \mathbf{M}) : \\ \mathsf{Verify}(C, \{\mathsf{pos}\}, \mathbf{M}_{\mathsf{pos}}, \pi_{\mathsf{pos}}) = 1 \end{array}\right] = 1.$$

Definition 2 (Correctness of Commitment Update). *For all λ, $n > 0$, matrix $\mathbf{M}$, any position* pos*, and incremental updating δ. The value of $\mathbf{M}_{\mathsf{pos}}$ is updated to $\mathbf{M}'_{\mathsf{pos}} = \mathbf{M}_{\mathsf{pos}} + \delta$,*

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

Definition 3 (Correctness of Proof Update). *For all λ, $n > 0$, matrix $\mathbf{M}$, any position pos_1, pos_2 and incremental updating δ. The value of $\mathbf{M}_{\mathsf{pos}_1}$ is updated to $\mathbf{M}'_{\mathsf{pos}_1} = \mathbf{M}_{\mathsf{pos}_1} + \delta$, Then*

$$\Pr\left[\begin{array}{l}\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^n),\\ \pi_{\mathsf{pos}_2} \leftarrow \mathsf{Prove}(\mathsf{pos}_2, \mathbf{M}),\\ \pi'_{\mathsf{pos}_2} \leftarrow \mathsf{UpdateProof}(\mathsf{pos}_1, \mathsf{pos}_2, \delta, \pi_{\mathsf{pos}_2}),\\ \pi''_{\mathsf{pos}_2} \leftarrow \mathsf{Prove}(\mathsf{pos}_2, \mathbf{M}') :\\ \pi'_{\mathsf{pos}_2} = \pi''_{\mathsf{pos}_2}\end{array}\right] = 1.$$

Definition 4 (Correctness of Proof Aggregation). *For all λ, $n > 0$, matrix $\mathbf{M}$, and any set of positions $S \subseteq [n] \times [n]$ or $S \subseteq [n] \times [n] \times [n]$,*

$$\Pr\left[\begin{array}{l}\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^n),\\ C \leftarrow \mathsf{Commit}(\mathbf{M}),\\ \pi_{\mathsf{pos}} \leftarrow \mathsf{Prove}(\mathsf{pos}, \mathbf{M}),\\ \hat{\Omega} \leftarrow \mathsf{AggregateProof}(C, S, \mathbf{M}[S], \{\pi_{\mathsf{pos}}\}_{\mathsf{pos}\in S}) :\\ \mathsf{Verify}(C, \{\mathsf{pos}\}_{\mathsf{pos}\in S}, \mathbf{M}[S], \hat{\Omega}) = 1\end{array}\right] = 1.$$

Definition 5 (Position Binding). *For all λ, $n > 0$, and any PPT adversary $\mathcal{A}$,*

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

3 Two-Dimensional Matrix Commitment Scheme

3.1 Overview

The Xproofs scheme commits to a multi-dimensional matrix consisting of messages, and we will describe the scheme using a two-dimensional matrix as an example. In the two-dimensional matrix commitment scheme 2D-Xproofs, a matrix of size N is committed, where $N = n \times n$ and $n \in \mathbb{N}$ is an integer value. During the committing phase, commitments are generated for every row and every column of the matrix, resulting in n row commitments $C_{\mathtt{row}} = \{c_{\mathtt{row}_i}\}_{i\in[n]}$ and n column commitments $C_{\mathtt{col}} = \{c_{\mathtt{col}_j}\}_{j\in[n]}$. These $2n$ values from bilinear group $\mathbb{G}_1$ constitute the commitment $C = (C_{\mathtt{row}}, C_{\mathtt{col}})$.

In the 2D-Xproofs, each position is uniquely determined by row and column coordinates. In other words, the proof for each position is composed of positions that intersect with this position in the two-dimensional matrix. Therefore, to open the commitment at the position $(i, j) \in [n]\times[n]$, a two-dimensional individual proof consisting of two parts, namely a row proof $\pi_{\mathtt{row}_j}$ and a column proof $\pi_{\mathtt{col}_i}$, is computed, which respectively determines that the entry exists in the specific row and column commitments (see Figure 1. for an example). Specifically, in the verification of the two-dimensional individual proof $\pi_{ij} = (\pi_{\mathtt{row}_j}, \pi_{\mathtt{col}_i})$, the row proof $\pi_{\mathtt{row}_j}$ is used to verify the correctness of the entry with respect to the ith-row commitment $c_{\mathtt{row}_i}$, and the column proof $\pi_{\mathtt{col}_i}$ is used to verify the correctness with respect to the jth-column commitment $c_{\mathtt{col}_j}$.

The proofs $\{\pi_{\mathtt{row}_j}, \pi_{\mathtt{col}_i}\}_{(i,j)\in S}$ for a set of position entries in the two-dimensional matrix can be aggregated into a single proof of the form $(\Omega_{\mathtt{row}}, \Omega_{\mathtt{col}})$, at this point, all the proofs are aggregated into just two elements from a bilinear group $\mathbb{G}_1$, and the set of $|S|$ proofs can be verified together. In our construction, these two elements in the aggregated proof can be further compressed into a single element from $\mathbb{G}_1$, resulting in an even smaller size of the aggregated proof. We will discuss this in more detail as follows.

The individual proof of each entry $(\pi_{\mathtt{row}_j}, \pi_{\mathtt{col}_i})$ in the two-dimensional matrix depends solely on the row and column in which the entry is located, i.e., $\{\mathrm{M}_{k\ell} : k = i\}$ and $\{\mathrm{M}_{k\ell} : \ell = j\}$. When an entry is updated, only the entries in the row and column need to update their proofs, while the entries at other positions in the matrix are not affected. Our design minimizes the changes to the proofs of other entries as much as possible, enabling all proofs to update efficiently.

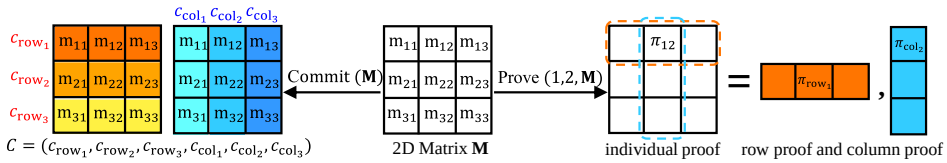


Fig. 1. 2D-Xproofs for 3×3 matrix ($n = 3$)

3.2 Construction

The details of our scheme 2D-Xproofs are described as follows:

- $\mathsf{Setup}(1^\lambda, 1^n)$: It takes a security parameter λ and an integer n as inputs. Generate a bilinear group context $\mathsf{bg} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$ using $\mathsf{BG}(1^\lambda)$. Sample $\alpha, \beta \leftarrow \mathbb{Z}_p$, and output the public parameters $\mathsf{pp} = (\mathsf{pp}_1, \mathsf{pp}_2)$

$$\mathsf{pp}_1 = \{g_1^{\boldsymbol{\alpha}}, g_1^{\alpha^n \boldsymbol{\alpha}[-1]}, g_1^{\boldsymbol{\beta}}, g_1^{\beta^n \boldsymbol{\beta}[-1]}\}, \mathsf{pp}_2 = \{g_2, g_2^{\boldsymbol{\alpha}}, g_2^{\boldsymbol{\beta}}\},$$

  where $\boldsymbol{\alpha} = (\alpha, \alpha^2, ..., \alpha^n)$, $\boldsymbol{\beta} = (\beta, \beta^2, ..., \beta^n)$. Note that α and β are private keys, and never be revealed to the public. After the Setup phase is completed, $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ can be destroyed.
- $\mathsf{Commit}(\mathbf{M})$: It takes the message $\mathbf{M} \in \mathbb{Z}_p^{n \times n}$ as inputs, computes the row commitment and column commitment

$$C_{\mathtt{row}} = \{c_{\mathtt{row}_i}\}_{i \in [n]} = \{g_1^{\sum_{j \in [n]} \mathbf{M}_{ij} \alpha^j}\}_{i \in [n]},$$

$$C_{\mathtt{col}} = \{c_{\mathtt{col}_j}\}_{j \in [n]} = \{g_1^{\sum_{i \in [n]} \mathbf{M}_{ij} \beta^i}\}_{j \in [n]},$$

  and outputs the commitment $C = (C_{\mathtt{row}}, C_{\mathtt{col}})$.
- $\mathsf{UpdateCommit}(i, j, \delta, C)$: It parses the commitment as $C = (\{c_{\mathtt{row}_i}\}_{i \in [n]}, \{c_{\mathtt{col}_j}\}_{j \in [n]})$. For any $(i, j) \in [n] \times [n]$ and $\delta \in \mathbb{Z}_p$, update the ith-row commitment $c'_{\mathtt{row}_i} = c_{\mathtt{row}_i} \cdot g_1^{\delta \alpha^j}$ and jth-column commitment $c'_{\mathtt{col}_j} = c_{\mathtt{col}_j} \cdot g_1^{\delta \beta^i}$ respectively.
- $\mathsf{Prove}(i, j, \mathbf{M})$: It takes a position $(i, j) \in [n] \times [n]$ and message matrix $\mathbf{M}$ as inputs, and computes an individual proof $\pi_{ij} = (\pi_{\mathtt{row}_j}, \pi_{\mathtt{col}_i})$, which consists of a row proof $\pi_{\mathtt{row}_j}$ and a column proof $\pi_{\mathtt{col}_i}$. The row proof $\pi_{\mathtt{row}_j} = g_1^{\sum_{q \in [n]/\{j\}} \mathbf{M}_{iq} \alpha^{n+1-j+q}} = g_1^{\alpha^{n+1-j} \mathbf{M}_{i*}[-ij] \boldsymbol{\alpha}[-j]}$ depends on $\mathbf{M}_{i*} \setminus \mathbf{M}_{ij}$ (i.e., all entries in ith-row except for the (i, j) entry). The column proof $\pi_{\mathtt{col}_i} = g_1^{\sum_{p \in [n]/\{i\}} \mathbf{M}_{pj} \beta^{n+1-i+q}} = g_1^{\beta^{n+1-i} \mathbf{M}_{*j}[-ij] \boldsymbol{\beta}[-i]}$ depends on $\mathbf{M}_{*j} \setminus \mathbf{M}_{ij}$ (i.e., all entries in jth-column except for the (i, j) entry.
- $\mathsf{UpdateProof}(i, j, k, \ell, \delta, \pi_{k\ell})$: It takes an updated position $(i, j) \in [n] \times [n]$ and a position $(k, \ell) \in [n] \times [n]$ with its individual proof as inputs, and parses the individual proof as $\pi_{k\ell} = (\pi_{\mathtt{row}_\ell}, \pi_{\mathtt{col}_k})$. When $\mathbf{M}_{ij}$ is updated to $\mathbf{M}_{ij} + \delta$, the updates to the individual proof can be discussed in the following cases:
  - When $i = k$ and $j \neq \ell$, the entry (i, j) being updated is in the same row as entry (k, ℓ). Therefore, the update to M_{ij} only affects the row proof $\pi_{\mathtt{row}_\ell}$ of the entry (k, ℓ). The row proof is updated to $\pi'_{\mathtt{row}_\ell} = \pi_{\mathtt{row}_\ell} \cdot g_1^{\delta \alpha^{n+1-\ell+j}}$. In this case, the algorithm outputs $\pi'_{k\ell} = (\pi'_{\mathtt{row}_\ell}, \pi_{\mathtt{col}_k})$.
  - When $j = \ell$ and $i \neq k$, the entry (i, j) is in the same column as entry (k, ℓ). Therefore, the update to M_{ij} only affects the column proof $\pi_{\mathtt{col}_k}$ of the entry (k, ℓ). Similarly, the column proof is updated to $\pi'_{\mathtt{col}_k} = \pi_{\mathtt{col}_k} \cdot g_1^{\delta \beta^{n+1-k+i}}$. In this case, the algorithm outputs $\pi'_{k\ell} = (\pi_{\mathtt{row}_\ell}, \pi'_{\mathtt{col}_k})$.
  - In other cases, the individual proof does not need to be updated.
- $\mathsf{AggregateProof}(C, S, \mathbf{M}[S], \{\pi_{ij}\}_{(i,j) \in S})$: It takes a commitment $C = (\{c_{\mathtt{row}_i}\}_{i \in [n]}, \{c_{\mathtt{col}_j}\}_{j \in [n]})$, a set of individual proofs $\{\pi_{ij}\}_{(i,j) \in S}$ as inputs. To prevent the proofs from being forged by adversaries, the individual proofs should be aggregated with two Hash Functions H and H'. The aggregation of the proofs is carried out in the following steps:
  - Firstly, the individual proofs located in the same row or the same column are aggregated based on the row or column respectively. The proofs of every row are aggregated as $\hat{\pi}_{\mathtt{row}_j} = \prod_{j \in S_j} \pi_{\mathtt{row}_j}^{r_{j,i}}$, and the proofs of every column are aggregated as $\hat{\pi}_{\mathtt{col}_i} = \prod_{i \in S_i} \pi_{\mathtt{col}_i}^{r_{i,j}}$. Where for all $i \in S_i$ and $j \in S_j$, the hash values $r_{j,i}$ and $r_{i,j}$ are computed as $r_{j,i} = H((j, i), c_{\mathtt{row}_i}, S_i)$ and $r_{i,j} = H((i, j), c_{\mathtt{col}_j}, S_j)$ respectively.

– Then, the proofs that have been aggregated from different rows can be further aggregated into one proof $\Omega_{\mathtt{row}}$, specifically, $\Omega_{\mathtt{row}} = \prod_{i\in[p]} \hat{\pi}_{\mathtt{row}_j}^{r_1'}$. Similarly, the proofs that have been aggregated from different columns can also be further aggregated into one proof $\Omega_{\mathtt{col}}$, specifically, $\Omega_{\mathtt{col}} = \prod_{j\in[q]} \hat{\pi}_{\mathtt{col}_i}^{r_2'}$. Where $[p]$ and $[q]$ represent the sets of rows and columns of the matrix involved in the AggregateProof algorithm. For all $i \in [p]$ and $j \in [q]$, the hash values r_1' and r_2' are computed as $r_1' = H'(i, \{c_{\mathtt{row}_i}, S_i, \mathbf{M}[S_i], \hat{\pi}_{\mathtt{row}_j}\}_{i\in[p]})$ and $r_2' = H'(j, \{c_{\mathtt{col}_j}, S_j, \mathbf{M}[S_j], \hat{\pi}_{\mathtt{col}_i}\}_{j\in[q]})$ respectively.

Finally, this algorithm outputs the aggregated proof $\hat{\Omega} = (\Omega_{\mathtt{row}}, \Omega_{\mathtt{col}})$.

- Verify$(C, S, \mathbf{M}[S], \hat{\Omega})$: During the verification phase, we need to consider two cases:
  – When $|S| = 1$, only the proof $\hat{\Omega} = \pi_{ij} = (\pi_{\mathtt{row}_j}, \pi_{\mathtt{col}_i})$ for the entry in position (i, j) needs to be verified. This algorithm checks two equations:

$$e(c_{\mathtt{row}_i}, g_2^{\alpha^{n+1-j}}) \stackrel{?}{=} e(\pi_{\mathtt{row}_j}, g_2) \cdot g_T^{\alpha^{n+1}\mathbf{M}_{ij}},$$

$$e(c_{\mathtt{col}_j}, g_2^{\beta^{n+1-i}}) \stackrel{?}{=} e(\pi_{\mathtt{col}_i}, g_2) \cdot g_T^{\beta^{n+1}\mathbf{M}_{ij}}.$$

  – When $|S| > 1$, this proof $\hat{\Omega} = (\Omega_{\mathtt{row}}, \Omega_{\mathtt{col}})$ is used to verify all entries in a set of positions $(i, j) \in S$. Also, this algorithm checks two equations: This equation is verified for all row proofs:

$$\prod_{i=1}^{p} e(c_{\mathtt{row}_i}, g_2^{\sum_{j\in S_j} \alpha^{n+1-j} r_{j,i}})^{r_1'} \stackrel{?}{=} e(\Omega_{\mathtt{row}}, g_2) \cdot g_T^{\alpha^{n+1} \sum_{i=1}^{p} \sum_{j\in S_j} \mathbf{M}_{ij} r_{j,i} r_1'},$$

  and this equation is verified for all column proofs:

$$\prod_{j=1}^{q} e(c_{\mathtt{col}_j}, g_2^{\sum_{i\in S_i} \beta^{n+1-i} r_{i,j}})^{r_2'} \stackrel{?}{=} e(\Omega_{\mathtt{col}}, g_2) \cdot g_T^{\beta^{n+1} \sum_{j=1}^{q} \sum_{i\in S_i} \mathbf{M}_{ij} r_{i,j} r_2'}.$$

In either case, if both equations hold, output 1; otherwise, output 0.

3.3 Minimizing the Size of 2D-Xproofs

Using this 2D-Xproofs construction, the individual proof for a single entry and the aggregated proof for multiple entries in the matrix only contains two elements from a bilinear group $\mathbb{G}_1$. Even more interestingly, these two elements in the proof $\pi_{ij} = (\pi_{\mathtt{row}_j}, \pi_{\mathtt{col}_i})$ output by the Prove$(i, j, \mathbf{M})$ algorithm can be further aggregated into one element from a bilinear group $\mathbb{G}_1$. This aggregated proof can be computed as $\pi_{ij} = \pi_{\mathtt{row}_j}^{r_1''} \cdot \pi_{\mathtt{col}_i}^{r_2''}$. At this point, the correctness of the aggregated proof π_{ij} to the commitment C can be verified as

$$e(c_{\mathtt{row}_i}, g_2^{\alpha^{n+1-j}})^{r_1''} \cdot e(c_{\mathtt{col}_j}, g_2^{\beta^{n+1-i}})^{r_2''} \stackrel{?}{=} e(\pi_{ij}, g_2) \cdot g_T^{\alpha^{n+1}\mathbf{M}_{ij} r_1'' + \beta^{n+1}\mathbf{M}_{ij} r_2''}.$$

Where the hash values r_1'' and r_2'' are computed as $r_1'' = H''(\{j, c_{\mathtt{row}_i}, S_i, \mathbf{M}[S_i], \hat{\pi}_{\mathtt{row}_j}\}_{i\in[p]})$ and $r_2'' = H''(\{i, c_{\mathtt{col}_j}, S_j, \mathbf{M}[S_j], \hat{\pi}_{\mathtt{col}_i}\}_{j\in[q]})$ respectively. Where, $|p|, |q| = 1$.

In addition, the aggregated proof $\hat{\Omega} = (\Omega_{\mathtt{row}}, \Omega_{\mathtt{col}})$ output by the AggregatePr oof$(C, S, \mathbf{M}[S], \{\pi_{ij}\}_{(i,j)\in S})$ algorithm also has similar property. The two elements in the proof can be further aggregated as $\hat{\Omega} = \Omega_{\mathtt{row}}^{r_1''} \cdot \Omega_{\mathtt{col}}^{r_2''}$ using the hash function H''. The correctness of the aggregated proof $\hat{\Omega}$ to the commitment C can be verified as

$$\begin{aligned}&(\prod_{i=1}^{p} e(c_{\mathtt{row}_i}, g_2^{\sum_j \alpha^{n+1-j} r_{j,i}})^{r_1'})^{r_1''} \cdot (\prod_{i=1}^{q} e(c_{\mathtt{col}_j}, g_2^{\sum_j \beta^{n+1-i} r_{i,j}})^{r_2'})^{r_2''} \\ &\stackrel{?}{=} e(\hat{\Omega}, g_2) \cdot g_T^{\alpha^{n+1} \sum_{i=1}^{p} \sum_j \mathbf{M}_{ij} r_{j,i} r_1' r_1'' + \beta^{n+1} \sum_{j=1}^{q} \sum_i \mathbf{M}_{ij} r_{i,j} r_2' r_2''}\end{aligned}$$

Where the hash values r_1'' and r_2'' are computed as $r_1'' = H''(\{i, c_{\mathtt{row}_i}, S_i, \mathbf{M}[S_i], \hat{\pi}_{\mathtt{row}_j}\}_{i\in[p]})$ and $r_2'' = H''(\{j, c_{\mathtt{col}_j}, S_j, \mathbf{M}[S_j], \hat{\pi}_{\mathtt{col}_i}\}_{j\in[q]})$ respectively.

Using this construction, our scheme minimizes the size of the aggregated proof to just one element from a group $\mathbb{G}_1$.

3.4 Security Analysis

Theorem 1 (Position Binding). *Under the $\ell-$ wBDHE* assumption, no PPT adversary in the AGM+ROM model can open 2D-Xproofs to two different values at the same positions, except with a negligible probability. Therefore, the above scheme 2D-Xproofs is position binding.*

Proof. In our 2D-Xproofs scheme, the aggregation of proofs is performed in two steps: 1. aggregating the proofs within the same row/column separately; 2. aggregating the proofs between different rows and between different columns. The process of aggregating proofs in our scheme is similar to the Pointproofs scheme. The two steps in the aggregation process are the aggregation within the same commitment and the aggregation across different commitments.

Analysis of same-commitment aggregation. Based on the Gorbunov et al.[11], the formal analysis of aggregation within the same commitment involves the following two steps:

Step 1: "H-lucky" query. Considering an adversary who generates any query $(*, C, S, \mathbf{m}[S])$ based on the hash function H acting as a random oracle, it must output $\mathbf{z} \in \mathbb{Z}_p^n$, $\mathbf{y} \in \mathbb{Z}_p^{n-1}$ such that

$$C = g_1^{\mathbf{z}^\top \mathbf{a} + \alpha^n \mathbf{y}^\top \mathbf{a}[-1]} = g_1^{\sum_{i\in[n]} z_i \alpha^i + \sum_{j\in[n-1]} y_j \alpha^{n+1+j}}.$$

When $\mathbf{m}[S] \not\equiv_p \mathbf{z}[S]$ and $(\mathbf{m}[S] - \mathbf{z}[S])^\top \mathbf{t} \equiv_p 0$ holds, it is referred to as a "H-lucky" query, where $\mathbf{t} = (H(i, C, S, \mathbf{m}[S]) : i \in S)$. The probability of a query being "H-lucky" is at most $1/p$. Here we use the fact that the query to H fixes $(S, \mathbf{m}[S], \mathbf{z}[S])$.

The probability of a successful "H-lucky" query for the adversary is at most q_H/p, where q_H represents the number of queries. Next, let's assume that this scenario never happens.

Step 2: computing $g_1^{\alpha^{n+1}}$. In this step, we demonstrate that an algebraic adversary cannot break the $\ell-$ wBDHE* to compute $g_1^{\alpha^{n+1}}$.

The successful adversary outputs $C, \{S^b, \mathbf{m}^b[S^b], \hat{\pi}^b\}_{b=0,1}$, together with $\mathbf{z}$, $\mathbf{y}$ such that $C = g_1^{\mathbf{z}^\top \mathbf{a} + \alpha^n \mathbf{y}^\top \mathbf{a}[-1]}$. Since $\mathbf{m}^0[S^0 \cap S^1] \neq \mathbf{m}^1[S^0 \cap S^1]$, it implies that $\mathbf{m}^0[S_0] \neq \mathbf{z}[S_0]$ or $\mathbf{m}^1[S_1] \neq \mathbf{z}[S_1]$. Let $(S^*, \mathbf{m}^*, \hat{\pi}^*)$ be such that

$$\mathbf{m}^*[S^*] \neq \mathbf{z}[S^*] \text{ and } \mathsf{Verify}(C, S^*, \mathbf{m}^*[S^*], \hat{\pi}^*) = 1.$$

Since $\hat{\pi}^*$ is a valid proof, it satisfies the verification equation:

$$e(C, g_2^{\sum_{i\in S^*} \alpha^{n+1-i} t_i}) = e(\hat{\pi}^*, g_2) \cdot g_T^{\alpha^{n+1} \mathbf{m}^*[S^*]^\top \mathbf{t}},$$

where $t_i = H(i, C, S^*, \mathbf{m}^*[S^*])$. This indicates

$$C^{\sum_{i\in S^*} \alpha^{n+1-i} t_i} = \hat{\pi}^* \cdot g_1^{\alpha^{n+1} \mathbf{m}^*[S^*]^\top \mathbf{t}}.$$

The left side of the equation can be split into two parts: one that involves $g_1^{\alpha^{n+1}}$ and another that does not. The left side of the equation can be written as:

$$(g_1^{\alpha^{n+1} \mathbf{z}[S^*]^\top \mathbf{t}}) \cdot (g_1^{\sum_{i\in S^*} \alpha^{n+1-i} \mathbf{z}[-i]^\top \mathbf{a}[-i] t_i}) \cdot (g_1^{\alpha^n \mathbf{y}^\top \mathbf{a}[-1] \sum_{i\in S^*} \alpha^{n+1-i} t_i}).$$

Move the part that involves $g_1^{\alpha^{n+1}}$ to the right side of the equation, resulting in:

$$(g_1^{\sum_{i\in S^*} \alpha^{n+1-i} \mathbf{z}[-i]^\top \mathbf{a}[-i] t_i}) \cdot (g_1^{\alpha^n \mathbf{y}^\top \mathbf{a}[-1] \sum_{i\in S^*} \alpha^{n+1-i} t_i}) \cdot (\hat{\pi}^*)^{-1} = g_1^{\alpha^{n+1} (\mathbf{m}^*[S^*] - \mathbf{z}[S^*])^\top \mathbf{t}}.$$

Since $\mathbf{m}^*[S^*] \neq \mathbf{z}[S^*]$ and we assume that there are no H-lucky queries, it must hold that $(\mathbf{m}^*[S^*] - \mathbf{z}[S^*])^\top \mathbf{t} \not\equiv_p 0$. Therefore, we can compute the inverse r modulo p and obtain $g_1^{\alpha^{N+1}}$ on the right side of the equation. Since the left side of the equation can be computed based on the adversary's output and $g_1^{\mathbf{a}}$, $g_1^{\alpha^N \mathbf{a}[-1]}$, $g_1^{\alpha^{2n} \mathbf{a}}$, we can compute $g_1^{\alpha^{n+1}}$.

Analysis of cross-commitment aggregation. Our scheme also satisfies the proof binding for cross-commitment aggregation. As referenced by Gorbunov et al.[11], the formal analysis of proof binding for cross-commitment aggregation can be divided into the following three steps:

Step 1: "H-lucky" query. The analysis in this step is the same as that of same-commitment aggregation.

Step 2:"H'-lucky" query. Considering an adversary who generates any query $(*, \{C_j, S_j, \mathbf{m}_j[S_j]\}_{j\in[\ell]})$ based on the hash function H' acting as a random oracle, it must output $\{\mathbf{z}_j, \mathbf{y}_j\}_{j\in[\ell]}$, such that$C_j = g_1^{\mathbf{z}_j^\top \mathbf{a} + \alpha^n \mathbf{y}_j^\top \mathbf{a}[-1]}$. When

$$\exists j : (\mathbf{m}_j[S_j] - \mathbf{z}_j[S_j])^\top \mathbf{t}_j \not\equiv_p 0 \text{ and } \sum_{j=1}^{\ell} (\mathbf{m}_j[S_j] - \mathbf{z}_j[S_j])^\top \mathbf{t}_j t'_j \equiv_P 0$$

holds, it is referred to as a "H'-lucky" query, where $\mathbf{t}_j = (H(i, C_j, S_j, \mathbf{m}[S_j]) : i \in S_j)$ and $t'_j = (H'(j, \{C_j, S_j, \mathbf{m}_j[S_j]) : j \in [\ell])$. A query is "$H'$-lucky" with probability at most $1/p$. Here we use the fact that the query to H' fixes $\{(S_j, \mathbf{m}_j[S_j], \mathbf{z}_j[S_j])\}_{j\in[\ell]}$.

Step 3: computing $g_1^{\alpha^{n+1}}$. The outputs of winning adversary contain $\{C_j^b\}_{j\in[\ell^b]}$ for $b = 0, 1$, $\mathbf{z}_j^b$, and $\mathbf{y}_j^b$, such that $C_j^b = g_1^{{\mathbf{z}_j^b}^\top \mathbf{a} + \alpha^n {\mathbf{y}_j^b}^\top \mathbf{a}[-1]}$. The winning conditions specifies j^0 and j^1 such that $C_{j^0}^0 = C_{j^1}^1$. Regardless of what the adversary outputs, we will set $\mathbf{z}_{j^1}^1 = \mathbf{z}_{j^0}^0$ and $\mathbf{y}_{j^1}^1 = \mathbf{y}_{j^0}^0$. We can argue in the same way as we do with same-commitment aggregation. Let $* = 0$ or 1, such that

$$\mathbf{m}_{j^*}^*[S_{j^*}^*] \neq \mathbf{z}_{j^*}^*[S_{j^*}^*] \text{ and } \mathsf{Verify}(\{C_j^*, S_j^*, \mathbf{m}_j^*[S_j^*]\}_{j\in[\ell^*]}, \pi*) = 1.$$

When $\ell^* = 1$, we can demonstrate using the same approach as same-commitment aggregation. If the verification passes, we can obtain a similar equation

$$(g_1^{\sum_{j=1}^{\ell^*} \sum_{i\in S_j^*} \alpha^{n+1-i} \mathbf{z}_j^*[-i]^\top \mathbf{a}[-i] t_{j,i} t'_j}) \cdot (g_1^{\sum_{j=1}^{\ell^*} \alpha^n \mathbf{y}_j^\top \mathbf{a}[-1] \sum_{i\in S^*} \alpha^{n+1-i} t_{j,i} t'_j}) \cdot (\hat{\pi}^*)^{-1} = g_1^{\alpha^{n+1} \sum_{j=1}^{\ell^*} (\mathbf{m}_j^*[S_j^*] - \mathbf{z}_j^*[S_j^*])^\top \mathbf{t}_j t'_j}.$$

Since both "H-lucky" queries and "H'-lucky" queries do not exist, we must have

$$(\mathbf{m}_{j^*}^*[S_{j^*}^*] - \mathbf{z}_{j^*}^*[S_{j^*}^*])^\top \mathbf{t}_{j^*} \not\equiv_p 0 \text{ and } \sum_{j=1}^{\ell^*} (\mathbf{m}_j^*[S_j^*] - \mathbf{z}_j^*[S_j^*])^\top \mathbf{t}_j t'_j \not\equiv_p 0.$$

Then, we can compute $g_1^{\alpha^{n+1}}$ using the same method as same-commitment aggregation.

Therefore, in both cases of same-commitment aggregation and cross-commitment aggregation, the $\ell -$ wBDHE* assumption cannot be broken. Our 2D-Xproofs satisfies *position binding*.

3.5 Efficiency Analysis

We let the total number of positions be $N = n^2$. The space complexity and computational complexity analysis of the 2D-Xproofs scheme are as follows.

Public parameters size. We incorporate the commitment as a part of the public parameters in our proposal. Consequently, the public parameters consist of two components: 1. the public parameters required for committing, opening, updating, and verifying; 2. the commitment values. Therefore, the public parameters are consist of $6\sqrt{N} - 2$ elements in $\mathbb{G}_1$ and $2\sqrt{N} + 1$ elements in $\mathbb{G}_2$.

Proof size. In 2D-Xproofs, each proof $\pi_{ij} = (\pi_{\mathtt{row}_j}, \pi_{\mathtt{col}_i})$ consists of 2 elements. Hence, the proofs are of constant size. For any $S \subseteq [\sqrt{N}] \times [\sqrt{N}]$, the aggregated proof still contains only 2 elements in $\mathbb{G}_1$. By employing our further optimization approach, it is possible to reduce the aggregated proof to just a single element in $\mathbb{G}_1$.

Computational complexity. Next, the time complexity is analyzed, focusing on the theoretical analysis of the number of multiplications, exponentiations, and pairing operations involved in each process.

- **Commitment generation and update.** The cost of $\mathsf{Commit}(\mathbf{M})$ is $2\sqrt{N}$ product of $\sqrt{N}$ exponentiations in $\mathbb{G}_1$, and the cost of $\mathsf{UpdateCommit}(i, j, \delta, C)$ is two exponentiations in $\mathbb{G}_1$.
- **Proofs generation.** For any $(i, j) \in [\sqrt{N}] \times [\sqrt{N}]$, $\mathsf{Prove}(i, j, \mathbf{M})$ outputs a individual proof $\pi_{ij} = (\pi_{\mathtt{row}_j}, \pi_{\mathtt{col}_i})$. The cost of computing $\pi_{\mathtt{row}_j}$ and $\pi_{\mathtt{col}_i}$ are two product of $\sqrt{N}$ exponentiations in $\mathbb{G}_1$. When computing all individual proofs at once, we need to compute all row proofs and column proofs. Fortunately, Tomescu [18] shows that, for any $i \in [\sqrt{N}]$, computing all proofs in time $O(\sqrt{N}\mathtt{log}\sqrt{N})$. Therefore, the time required to compute all proofs is in $O(N\mathtt{log}\sqrt{N})$.
- **Proofs aggregation.** For any $S \subseteq [\sqrt{N}] \times [\sqrt{N}]$, their $|S|$ row proofs and $|S|$ column proofs are aggregated respectively. Therefore, it requires $2(|S| + \min\{|S|, \sqrt{N}\})$ exponentiations in $\mathbb{G}_1$.

- **Proofs verification.** Verifying an individual proof requires 2×3 pairings and 2×1 exponentiations in $\mathbb{G}_2$, and verifying an aggregated proof requires $2(|S|+\min\{|S|,\sqrt{N}\})$ exponentiations and $2(|S|+\min\{|S|,\sqrt{N}\}+2)$ pairings.
- **Proofs update.** Updating an individual proof requires executing the $\mathsf{Update\ Proof}(i,j,k,\ell,\delta,\pi_{k\ell})$ algorithm, which involves 2 exponentiations in $\mathbb{G}_1$ to update row proofs and column proofs respectively. When an entry in the matrix is updated, it only affects the proofs at $2\sqrt{N}-2$ positions. Therefore, updating all the proofs requires a total of $2\sqrt{N}-2$ exponentiations in $\mathbb{G}_1$.

4 Three-Dimensional Matrix Commitment Scheme

Our three-dimensional matrix commitment scheme 3D-Xproofs is obtained by committing to a three-dimensional matrix, achieving even better update efficiency. This transformation comes at a certain cost, as the total size of public parameters and commitment will increase, but it is still sublinear. Additionally, the aggregation and verification time for proofs will be slightly slower than the 3D-Xproofs scheme, but it remains efficient. Similar to the 2D-Xproofs scheme, in the 3D-Xproofs scheme, the proof for each position is composed of positions that intersect with this position in the three-dimensional matrix. Therefore, when an update occurs at a certain position, it only affects $3N^{1/3}-3$ positions. We denote the three directions in the three-dimensional matrix as x, y, and z, respectively. The algorithm description of the 3D-Xproofs is as follows.

- $\mathsf{Setup}(1^\lambda,1^n)$: It takes a security parameter λ and an integer n as inputs. Generate a bilinear group context $\mathrm{bg}=(p,\mathbb{G}_1,\mathbb{G}_2,\mathbb{G}_T,e,g_1,g_2)$ using $\mathsf{BG}(1^\lambda)$. Sample $\alpha,\beta,\gamma\leftarrow\mathbb{Z}_p$, and output the public parameters $\mathsf{pp}=(\mathsf{pp}_1,\mathsf{pp}_2)$

$$\mathsf{pp}_1=\{g_1^{\boldsymbol{\alpha}},g_1^{\alpha^n\boldsymbol{\alpha}[-1]},g_1^{\boldsymbol{\beta}},g_1^{\beta^n\boldsymbol{\beta}[-1]},g_1^{\boldsymbol{\gamma}},g_1^{\gamma^n\boldsymbol{\gamma}[-1]},\},\mathsf{pp}_2=\{g_2,g_2^{\boldsymbol{\alpha}},g_2^{\boldsymbol{\beta}},g_2^{\boldsymbol{\gamma}}\},$$

  where $\boldsymbol{\alpha}=(\alpha,\alpha^2,...,\alpha^n)$, $\boldsymbol{\beta}=(\beta,\beta^2,...,\beta^n)$, and $\boldsymbol{\gamma}=(\gamma,\gamma^2,...,\gamma^n)$.
- $\mathsf{Commit}(\mathbf{M})$: It takes the message matrix $\mathbf{M}\in\mathbb{Z}_p^{n\times n\times n}$ as inputs, computes the x commitment $C_{\mathtt{x}}=\{\{g_1^{\sum_{i\in[n]}\mathbf{M}_{ijk}\alpha^i}\}_{j\in[n]}\}_{k\in[n]}$, the computation method for y commitment and z commitment are similar to that of $C_{\mathtt{x}}$. Outputs the commitment $C=(C_{\mathtt{x}},C_{\mathtt{y}},C_{\mathtt{z}})$.
- $\mathsf{UpdateCommit}(i,j,k,\delta,C)$: It parses the commitment as $C=(C_{\mathtt{x}},C_{\mathtt{y}},C_{\mathtt{z}})$. For any $(i,j,k)\in[n]\times[n]\times[n]$ and $\delta\in\mathbb{Z}_p$, update the x commitment, y commitment, and z commitment respectively.
- $\mathsf{Prove}(i,j,k,\mathbf{M})$: It takes a position $(i,j,k)\in[n]\times[n]\times[n]$ and message m as inputs, and computes an individual proof $\pi_{ijk}=(\pi_{\mathtt{x}},\pi_{\mathtt{y}},\pi_{\mathtt{z}})$, which consists of a x proof, a y proof and a z proof. The x proof $\pi_{\mathtt{x}}=g_1^{\alpha^{n+1-i}\mathbf{M}_{*jk}[-ijk]\boldsymbol{\alpha}[-i]}$ depends on $\mathbf{M}_{*jk}\setminus\mathbf{M}_{ijk}$ (i.e., all entries in $\{(*,j,k)\}_{*\in[n]}$ except for the (i,j,k) entry). The computation method for $\pi_{\mathtt{y}}$ and $\pi_{\mathtt{z}}$ are similar to that of $\pi_{\mathtt{x}}$.
- $\mathsf{UpdateProof}(i,j,k,\ell,m,n,\delta,\pi_{\ell mn})$: It takes a position $(i,j,k)\in[n]\times[n]\times[n]$ and an individual proof as inputs, and parses proof as $\pi_{\ell mn}=(\pi_{\mathtt{x}},\pi_{\mathtt{y}},\pi_{\mathtt{z}})$. Similar to the 2D-Xproofs scheme, when $\mathbf{M}_{ijk}$ is updated to $\mathbf{M}_{ijk}+\delta$, it is only necessary to update the proofs at positions (i,j,k) and (ℓ,m,n) that are located on the same coordinate axis.
- $\mathsf{AggregateProof}(C,S,\mathbf{M}[S],\{\pi_{ijk}\}_{(i,j,k)\in S})$: It takes the commitment $C=(C_{\mathtt{x}},C_{\mathtt{y}},C_{\mathtt{z}})$, a set of individual proofs $\{\pi_{ijk}\}_{(i,j,k)\in S}$ as inputs. Similarly, the proofs should be aggregated with two Hash Functions H and H'. The aggregation of these proofs follows a similar approach to the 2D-Xproofs scheme and consists of the following two steps:
  - Firstly, the proofs located in the same coordinates are aggregated based on the x, y, and z coordinates respectively. Obtain the values $\hat{\pi}_{\mathtt{x}}$, $\hat{\pi}_{\mathtt{y}}$, and $\hat{\pi}_{\mathtt{z}}$ using the hash function H.
  - Then, the proofs aggregated from different coordinates can be further aggregated into one proof. Obtain the values $\Omega_{\mathtt{x}}$, $\Omega_{\mathtt{y}}$, and $\Omega_{\mathtt{z}}$ using the hash function H'.

  Finally, this algorithm outputs the aggregated proof $\hat{\Omega}=(\Omega_{\mathtt{x}},\Omega_{\mathtt{y}},\Omega_{\mathtt{z}})$.
- $\mathsf{Verify}(C,S,\mathbf{M}[S],\hat{\Omega})$: During the verification phase, we also need to consider two cases:
  - When $|S|=1$, only the proof $\hat{\Omega}=\pi_{ijk}=(\pi_{\mathtt{x}},\pi_{\mathtt{y}},\pi_{\mathtt{z}})$ for one entry in position (i,j,k) needs to be verified. Similar to the 2D-Xproofs scheme, this verification process requires three equations.
  - When $|S|>1$, this proof $\hat{\Omega}=(\Omega_{\mathtt{x}},\Omega_{\mathtt{y}},\Omega_{\mathtt{z}})$ is used to verify all entries in a set of positions $(i,j,k)\in S$. Correspondingly, the verification process also requires three equations.

4.1 Analysis

3D-Xproofs scheme still satisfies the correctness of opening, aggregation, proof update, and commitment update, which can be directly derived. Additionally, the security analysis of the 3D-Xproofs is similar to 2D-Xproofs, and it also satisfies *position binding* under both same-commitment aggregation and cross-commitment aggregation. We let the total number of positions be $N = n^3$. The space complexity and computational complexity analysis of the 3D-Xproofs scheme are as follows:

- **Public parameters size.** The public parameters still consist of two parts: the public parameters and the commitments. The public key consists of $6N^{1/3} - 3$ elements in $\mathbb{G}_1$ and $3N^{1/3} + 1$ elements in $\mathbb{G}_2$, while the commitment consists of $3N^{2/3}$ elements in $\mathbb{G}_1$.
- **Poof size.** The proof and the aggregated proof are still of constant size, with each proof containing 3 elements in $\mathbb{G}_1$.
- **Proofs aggregation and verification.** The aggregation of $|S|$ proofs requires $3(|S| + \min\{|S|, N^{1/3}\})$ exponentiations in $\mathbb{G}_1$. Verifying an individual proof requires 3×3 pairings and 3×1 exponentiations in $\mathbb{G}_2$, and verifying an aggregated proof requires $3(|S| + \min\{|S|, N^{1/3}\})$ exponentiations and $3(|S| + \min\{|S|, N^{1/3}\} + 2)$ pairings.
- **Proofs and commitments update.** When a certain position is updated, it affects the proofs of $3N^{1/3} - 3$ positions. Therefore, only $3N^{1/3} - 3$ exponentiations are required to complete the updates for all proofs. The commitments can be updated by performing only 3 exponentiations in $\mathbb{G}_1$.

Note that we could further increase the matrix dimensionality to achieve more efficient proof updates. Here, we omit the details of further expansion.

5 Applications to Blockchain

In blockchain networks such as Ethereum, which are based on account architecture, the initiator of a transfer creates a transaction (TX) when initiating a transfer request. At this point, the TX includes the account balance of the sender as well as the amount being transferred. All these transactions, known as TXs, undergo validation by block proposers and are subsequently packaged into a new block. Once this new block receives consensus from the majority of block validators, it is formally added to the blockchain. To ascertain the validity of transactions, both block proposers and validators traditionally necessitate the storage of the complete account balances across the entire blockchain — a representation of the state of the blockchain. Nevertheless, in stateless cryptocurrencies, the block proposers and validators no longer retain the entire state of the blockchain, resulting in a significant reduction in storage overhead. We will elucidate the utilization of Xproofs in realizing stateless cryptocurrencies.

Based on the implementation scheme of Pointproofs [11], efficient aggregation and batch verification of proofs from different TXs can be achieved. However, the time required for all users to locally synchronize the proofs is $O(N)$. The solution implemented based on Edrax [9] and Hyperproofs [17] reduces the time for updating all proofs to $O(\log N)$. However, the implementation based on Hyperproofs requires IPA [6] to achieve proofs aggregation, and the size of the proofs is not constant but rather dependent on the number of accounts. The implementation based on Matproofs [14] and Balanceproofs[20] support efficient aggregation and verification. However, the size of the aggregated proof is not constant, and the efficient updating of proofs relies on PSNs. Without PSNs, all accounts would need to perform local computations for proof updates.

5.1 Xproofs-based Solution

In our Xproofs approach, all account balances are structured into a matrix. The Setup algorithm generates public parameters for the entire stateless blockchain system, followed by employing the Commit algorithm to commit to the matrix, and subsequently store the commitment values and public parameters within the blockchain system. Subsequently, the Prove algorithm generates proofs for the balances of all users individually. Users retain and manage their proofs locally. When users initiate a transaction (TX), the transaction includes both TX information and proofs corresponding to account balances. Block proposers utilize the Verify algorithm to validate such

proofs. These proofs can be aggregated into one proof of constant size using the AggregateProof algorithm and verified all at once. Subsequently, the UpdateCommit algorithm is employed to update the commitments. Once verified TXs are packaged into a new block, upon receiving validation and consensus from the majority of validators, all users utilize the UpdateProof procedure to synchronize their local balances and proofs.

5.2 Evaluation and Comparisons

Public parameters size. In the implementation based on Xproofs, commitments are a part of public parameters. The size of the public parameters is sub-linearly related to the number of accounts, reducing the storage overhead.
Proofs size. In our implementation, the size of the proof for each account is constant. Furthermore, when processing multiple transactions, it is possible to aggregate the proofs of multiple accounts. The size of the aggregated proof is constant, consisting of only 2 or 3 elements in $\mathbb{G}_1$.
Aggregation and verification time. The aggregation and verification of our implementation do not require the use of additional argument systems. Although the aggregation and verification time of our scheme are slightly higher than the Matproofs [14] scheme, it is still efficient.
Update of commitment and proofs. In our scheme, commitment updates are efficient, requiring only 2 or 3 exponentiations in $\mathbb{G}_1$. Proof updates do not require the use of PSNs, avoiding the introduction of additional nodes and reducing communication overhead. The updates for all account proofs only require performing $4\sqrt{N}-4$ or $9N^{1/3}-9$ exponentiations in $\mathbb{G}_1$, respectively in the 2D-Xproofs and 3D-Xproofs schemes.

6 Performance Evaluation

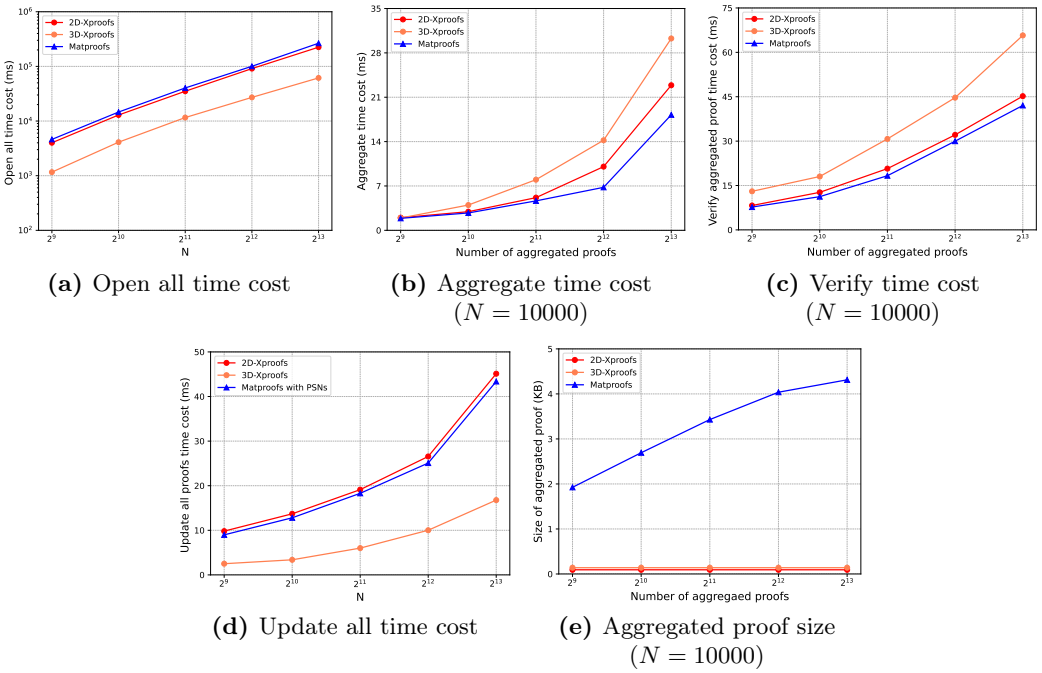
In this section, the performance of Xproofs in stateless blockchain applications is evaluated. Specifically, a comparative analysis is conducted among 2D-Xproofs, 3D-Xproofs, and Matproofs. The implementation of 2D-Xproofs, 3D-Xproofs, and Matproofs is carried out using the Python programming language and the pypbc library. We use the PBC-0.5.14 library with a type A elliptic curve for pairing-based primitives to achieve 128-bit security and the SHA-256 hash function. We deploy our experiments on the machine with Intel(R) Core(TM) i5-11500 @ 2.70 GHz RAM 4 GB and Ubuntu 18.04.

6.1 Evaluation

The time cost of opening all proofs. In our two Xproofs schemes, the computational cost of each position's proof is uniform. On the other hand, in the Matproofs scheme, the computation cost of individual global proofs is higher. However, every $\sqrt{N}$ positions share the same individual global proof. Consequently, as observed from Figure 2(a), we set the total number of accounts $N = 2^9, 2^{10}, 2^{11}, 2^{12}, 2^{13}$, and open all proofs, with the vertical axis representing the time required to open all proofs. The total time required to open all positions and generate proofs in the 2D-Xproofs scheme is comparable to that of the Matproofs scheme. However, due to the lower time cost of generating each proof in the three-dimensional scenario, our 3D-Xproofs scheme exhibits significantly better performance, being approximately $3\times$ faster than other schemes.

The time cost of proof aggregation. In our 2D-Xproofs scheme, proofs need to be aggregated separately by rows and columns, resulting in a total of $2(|S| + \min\{|S|, \sqrt{N}\})$ exponentiations in the $\mathbb{G}_1$ group. Similarly, in the 3D-Xproofs scheme, $3(|S| + \min\{|S|, N^{1/3}\})$ exponentiations in the $\mathbb{G}_1$ group are required due to the need to aggregate proofs along three dimensions. However, in the Matproofs scheme, aggregating the $|S|$ individual proofs requires $|S| + \min\{|S|, \sqrt{N}\}$ exponentiations in the $\mathbb{G}_1$ group. Therefore, as observed from Figure 2(b), we set the total number of accounts $N = 10000$ and the number of proofs to be aggregated $|S| = 2^9, 2^{10}, 2^{11}, 2^{12}, 2^{13}$. The vertical axis represents the time required for proof aggregation. During the proof aggregation phase, the performance of the Matproofs scheme is approximately $1.5 - 2.5\times$ faster than the Xproofs scheme.

The time cost of proof verification. In the verification phase, whether it is verifying individual proofs or aggregated proofs, both our 2D-Xproofs and Matproofs schemes only require two equations and have similar computational complexity. However, the 3D-Xproofs scheme requires an



(a) Open all time cost

(b) Aggregate time cost ($N = 10000$)

(c) Verify time cost ($N = 10000$)

(d) Update all time cost

(e) Aggregated proof size ($N = 10000$)

Fig. 2. Performance comparison

additional verification equation. Therefore, as observed from Figure 2(c), we set the total number of accounts $N = 10000$ and the number of proofs to be aggregated $|S| = 2^9, 2^{10}, 2^{11}, 2^{12}, 2^{13}$. 2D-Xproofs and Matproofs exhibit comparable performance during the verification process. However, the 3D-Xproofs scheme incurs approximately a 0.5× increase in time overhead.

The time cost of proof update. When the balance of a specific account is updated, the proofs of all other accounts need to be updated as well. Therefore, achieving sublinear time for updating all proofs is necessary. In the Matproofs scheme, there are two cases: 1. Proofs in the same sub-vector as the updated position only require updating the individual local proofs, and these $\sqrt{N}$ proofs share the same local commitment, thus requiring only one update to the local commitment. 2. Proofs in different sub-vectors from the updated position do not require updating individual local proofs, and every sub-vector shares the same individual global proof, which only needs to be computed once every $\sqrt{N}$ individual proof. From this perspective, the time complexity for updating all proofs in the Matproofs scheme appears to be $O(\sqrt{N})$. However, in practical scenarios, even though reducing the complexity by allowing certain proofs to share the same elements, each user still needs to perform local computations for the shared values. To achieve sublinear time complexity in the Matproofs scheme, it is necessary to introduce PSNs to compute new proofs and share the updated proofs with other users. However, this approach introduces additional communication overhead and involves third-party PSNs. In our Xproofs scheme, the impact of updating one proof on the individual proofs of other users is minimized. In the two-dimensional case, it only affects $2\sqrt{N} - 2$ proofs, and in the three-dimensional case, it only affects $3N^{1/3} - 3$ proofs. Therefore, our scheme achieves sublinear time for updating all proofs without needing PSNs. As shown in Figure 2(d), we set the total number of accounts $N = 2^9, 2^{10}, 2^{11}, 2^{12}, 2^{13}$ and update all proofs. Without the assistance of PSNs, the update time in the Matproofs scheme remains linear, whereas, with the assistance of PSNs, its performance is comparable to that of the 2D-Xproofs scheme. However, our 3D-Xproofs scheme exhibits better efficiency in proof updates, approximately 2.5× faster than other schemes. Furthermore, with an increasing number of users, the performance of our 3D-Xproofs scheme will be even better.

Size of individual proof and aggregated proof. In the individual proofs, our 2D-Xproofs scheme comprises two elements in $\mathbb{G}_1$ per position, namely the row proof and the column proof. Similarly, in the 3D-Xproofs scheme, each proof consists of three elements in $\mathbb{G}_1$. Lastly, in the Matproofs scheme, each individual proof consists of three elements in $\mathbb{G}_1$. In the aggregated proofs, our Xproofs scheme achieves the same optimal result as the Pointproofs scheme by minimizing the proof size, resulting in only one element in $\mathbb{G}_1$ for the aggregated proof of a set. However, in the

Matproofs scheme, the local commitments in the proof do not support aggregation. Therefore, in the worst case, the number of elements in the aggregated proof can reach $\sqrt{N} + 2$. Therefore, as observed from Figure 2(e), we set the total number of accounts $N = 10000$ and the number of proofs to be aggregated $|S| = 2^9, 2^{10}, 2^{11}, 2^{12}, 2^{13}$. The size of the aggregated proofs in our 2D-Xproofs and 3D-Xproofs schemes is significantly smaller than that of the Matproofs scheme. In the worst case, the size of aggregated proofs in the Matproofs scheme is approximately 4.78KB, related to the total number of accounts N. On the other hand, the 2D-Xproofs scheme consistently maintains a size of 0.093KB, while the 3D-Xproofs scheme maintains a size of 0.148KB.

7 Conclusions

In this paper, we propose Xproofs, an easily updatable and highly maintainable commitment scheme without relying on PSNs. Xproofs scheme achieves the optimal proof size and more efficient proof update efficiency than Matproofs, while introducing a small amount of aggregation and verification overhead. Additionally, the commitment values in Xproofs are not constant, but we include the commitment as part of the public parameters, resulting in a sublinear overall size of the public parameters. How to reduce the commitment values to a constant size while maintaining the advantages of our scheme is a problem worth addressing.

References

1. Agrawal, S., Raghuraman, S.: Kvac: Key-value commitments for blockchains and beyond. In: IACR Cryptology ePrint Archive (2020), https://api.semanticscholar.org/CorpusID:222036449
2. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: CCS. pp. 62–73 (1993)
3. Boneh, D., Boyen, X., Goh, E.J.: Hierarchical identity based encryption with constant size ciphertext. In: EUROCRYPT. pp. 440–456. Springer (2005)
4. Boneh, D., Bünz, B., Fisch, B.: Batching techniques for accumulators with applications to iops and stateless blockchains. In: CRYPTO 2019. pp. 561–586. Springer (2019)
5. Boneh, D., Gentry, C., Waters, B.: Collusion resistant broadcast encryption with short ciphertexts and private keys. In: CRYPTO 2005. pp. 258–275. Springer (2005)
6. Bünz, B., Maller, M., Mishra, P., Tyagi, N., Vesely, P.: Proofs for inner pairing products and applications. In: ASIACRYPT 2021. pp. 65–97. Springer (2021)
7. Camenisch, J., Kohlweiss, M., Soriente, C.: An accumulator based on bilinear maps and efficient revocation for anonymous credentials. In: PKC. pp. 481–500. Springer (2009)
8. Catalano, D., Fiore, D.: Vector commitments and their applications. In: PKC (2013), https://api.semanticscholar.org/CorpusID:5476579
9. Chepurnoy, A., Papamanthou, C., Zhang, Y.: Edrax: A cryptocurrency with stateless transaction validation. IACR Cryptol. ePrint Arch. **2018**, 968 (2018), https://api.semanticscholar.org/CorpusID:53244182
10. Fuchsbauer, G., Kiltz, E., Loss, J.: The algebraic group model and its applications. In: CRYPTO 2018. pp. 33–62. Springer (2018)
11. Gorbunov, S., Reyzin, L., Wee, H., Zhang, Z.: Pointproofs: Aggregating proofs for multiple vector commitments. CCS (2020), https://api.semanticscholar.org/CorpusID:215797856
12. Kuszmaul, J.: Verkle trees
13. Libert, B., Yung, M.: Concise mercurial vector commitments and independent zero-knowledge sets with short proofs. In: TCC (2010), https://api.semanticscholar.org/CorpusID:14038646
14. Liu, J., Zhang, L.F.: Matproofs: Maintainable matrix commitment with efficient aggregation. CCS (2022), https://api.semanticscholar.org/CorpusID:253371251
15. Merkle, R.C.: A digital signature based on a conventional encryption function. In: CRYPTO 1987. pp. 369–378. Springer (1987)
16. Papamanthou, C., Shi, E., Tamassia, R.: Signatures of correct computation. In: TCC. pp. 222–242. Springer (2013)
17. Srinivasan, S., Chepurnoy, A., Papamanthou, C., Tomescu, A., Zhang, Y.: Hyperproofs: Aggregating and maintaining proofs in vector commitments. In: USENIX Security. pp. 3001–3018 (2022)
18. Tomescu, A.: How to compute all pointproofs. Cryptology ePrint Archive (2020)
19. Tomescu, A., Abraham, I., Buterin, V., Drake, J., Feist, D., Khovratovich, D.: Aggregatable subvector commitments for stateless cryptocurrencies. In: IACR Cryptology ePrint Archive (2020), https://api.semanticscholar.org/CorpusID:218531758
20. Wang, W., Ulichney, A., Papamanthou, C.: {BalanceProofs}: Maintainable vector commitments with fast aggregation. In: USENIX Security. pp. 4409–4426 (2023)