Hadamard Product Arguments and Their Applications

Kyeongtae Lee[1][0009-0007-7969-9289], Donghwan Oh[1][0009-0001-2728-9460], Hankyung Ko[2][0000-0001-9430-1768], Jihye Kim[3][0000-0003-2953-7883], and Hyunok Oh[1][0000-0002-9044-7441]

[1] Department of Information Systems, Hanyang University, Seoul 04763, South Korea
{rsias9049,donghwanoh,hoh}@hanyang.ac.kr
[2] Radius, Seoul 06232, South Korea
hankyungko@hanyang.ac.kr
[3] Electronics and Information System Engineering Major, Kookmin University, Seoul 02707, South Korea
jihyek@kookmin.ac.kr

Abstract. This paper introduces transparent and efficient arguments for Hadamard products between committed vectors from two source groups. For vectors of length n, the proofs consist of $\mathcal{O}_\lambda(\log n)$ target group elements and $\mathcal{O}(1)$ additional elements. The verifier's workload is dominated by $\mathcal{O}_\lambda(\log n)$ multi-exponentiations in the target group and $\mathcal{O}(1)$ pairings. We prove our security under the standard SXDH assumption. Additionally, we propose an aggregator for Groth16 pairing-based zk-SNARKs and a proof aggregation technique for the general case of the KZG pairing-based polynomial commitment scheme using our Hadamard product arguments. Both applications support logarithmic-sized aggregated proofs without requiring an additional trusted setup, significantly reducing the verifier's pairing operations to $\mathcal{O}(1)$.

Keywords: Zero-knowledge proof, Pairing-based cryptography, SNARK, Privacy, Blockchain application

1 Introduction

The role of inner product and Hadamard product, also known as entry-wise products, is fundamental in modern cryptography. These operations are crucial for supporting a wide array of cryptographic primitives, including zero-knowledge proofs and commitment schemes [5, 7, 8, 25].

In Groth's work [14], an extensive framework for zero-knowledge arguments is introduced, primarily emphasizing the reduction of Hadamard products to inner product relations. This scheme employs a modified form of Pedersen commitments [23] and integrates randomization and batch-verification techniques to streamline diverse linear algebra relations into inner product relations. The randomization technique is succinctly explained as follows: consider a randomly chosen scalar $r \in \mathbb{F}$. Introduce a binary operator $*$, which maps from $\mathbb{F}^n \times \mathbb{F}^n$ to $\mathbb{F}$ and is defined as $\mathbf{a} * \mathbf{b} = \mathbf{a}(\mathbf{b} \circ \mathbf{r})^T$, where $\mathbf{r} = (1, r, r^2, \cdots, r^{n-1})$. An interesting property emerges: if $\mathbf{a} * \mathbf{b} = \mathbf{c} * \mathbf{d}$, it implies $\mathbf{a} \circ \mathbf{b} = \mathbf{c} \circ \mathbf{d}$, with an error bound of at most $(n-1)/|\mathbb{F}|$. Incorporating this randomization technique enables the demonstration of Hadamard Product Arguments(HPA) from Inner Product Arguments(IPA), expressed as $\langle \mathbf{a_r}, \mathbf{b} \rangle = \langle \mathbf{c_r}, \mathbf{d} \rangle$. Consequently, leveraging efficient IPA, such as [7,8] allows the establishment of efficient HPA protocols. The Bulletproofs [7] achieves transparency but requires a linear verification cost for the verifier. Conversely, Bünz et al. [8], achieves sublinear verification times but necessitates a trusted setup. Both approaches face limitations due to the verifier computing the commitment keys: unstructured keys, while transparent, impose a linear computation cost on the verifier, whereas structured keys reduce the verifier's computation on commitment keys but require an additional trusted setup.

Dory [20], a recent advancement in the field, introduces efficient inner product arguments, ensuring transparency, and providing sublinear verification time. Leveraging the properties of Type III pairing $(\mathbb{F}, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, h)$, and with precomputation of commitment keys performed during an offline phase, Dory effectively delegates the implicit computation of challenge-dependent commitment keys to the prover. The application of Groth's randomization technique to Dory theoretically enables the design of a transparent HPA protocol with sublinear verification time. Unfortunately, Dory currently faces challenges in accommodating Groth's randomization, primarily because the verifier lacks direct control over the keys during the online phase of the protocol, making the verification of randomized values infeasible.

Algorithm 1 Dory-Reduce [20]

```
**Precompute:** $\chi = \langle \mathbf{ck_1}, \mathbf{ck_2} \rangle$,
$\varDelta_{1L} = \langle \mathbf{ck_1}_L, \mathbf{ck_2'} \rangle$, $\varDelta_{1R} = \langle \mathbf{ck_1}_R, \mathbf{ck_2'} \rangle$,
$\varDelta_{2L} = \langle \mathbf{ck_1'}, \mathbf{ck_2}_L \rangle$, $\varDelta_{2R} = \langle \mathbf{ck_1'}, \mathbf{ck_2}_R \rangle$
**Input:** $(C, D_1, D_2 \ ; \ \mathbf{a}, \mathbf{b}) \in \mathcal{R}_{2^m, \mathbf{ck_1}, \mathbf{ck_2}}$
$\mathcal{P} \to \mathcal{V} : D_{1L} = \langle \mathbf{a}_L, \mathbf{ck_2'} \rangle, \ D_{1R} = \langle \mathbf{a}_R, \mathbf{ck_2'} \rangle$,
          $D_{2L} = \langle \mathbf{ck_1'}, \mathbf{b}_L \rangle, \ D_{2R} = \langle \mathbf{ck_1'}, \mathbf{b}_R \rangle$
$\mathcal{V} \to \mathcal{P} : \beta \leftarrow\!\!\$\ \mathbb{F}$
$\mathcal{P} : \mathbf{a}^* \leftarrow \mathbf{a} + \beta \mathbf{ck_1}, \ \ \mathbf{b}^* \leftarrow \mathbf{b} + \beta^{-1} \mathbf{ck_2}$
$\mathcal{P} \to \mathcal{V} : C_+ = \langle \mathbf{a}_L, \mathbf{b}_R \rangle, \ \ C_- = \langle \mathbf{a}_R, \mathbf{b}_L \rangle$
$\mathcal{V} \to \mathcal{P} : \alpha \leftarrow\!\!\$\ \mathbb{F}$
$\mathcal{P} : \mathbf{a}' \leftarrow \alpha \mathbf{a}^*{}_L + \mathbf{a}^*{}_R, \ \ \mathbf{b}' \leftarrow \alpha^{-1} \mathbf{b}^*{}_L + \mathbf{b}^*{}_R$
$\mathcal{V} : C' \leftarrow C + \chi + \beta D_2 + \beta^{-1} D_1 + \alpha C_+ + \alpha^{-1} C_-$
    $D_1' \leftarrow \alpha D_{1L} + D_{1R} + \alpha\beta \varDelta_{1L} + \beta \varDelta_{1R}$
    $D_2' \leftarrow \alpha^{-1} D_{2L} + D_{2R} + \alpha^{-1}\beta^{-1} \varDelta_{2L} + \beta^{-1} \varDelta_{2R}$
**Output:** $(C', D_1', D_2') \ ; \ \mathbf{a}', \mathbf{b}') \in \mathcal{R}_{2^{m-1}, \mathbf{ck_1'}, \mathbf{ck_2'}}$
```

In particular, Dory leverages the benefits of symmetry within both the key and message spaces by employing the AFGHO commitment scheme [1]. When the message resides in $\mathbb{G}_1^n$, the commitment key resides in $\mathbb{G}_2^n$ (and vice versa). Consequently, messages and commitment keys can be merged using a random challenge β, yielding $\mathbf{a}^* = \mathbf{a} + \beta\mathbf{ck_1} \in \mathbb{G}_1^n$ and $\mathbf{b}^* = \mathbf{b} + \beta^{-1}\mathbf{ck_2} \in \mathbb{G}_2^n$ (refer to **Algorithm 1**). This approach delegates the computation of the key to the prover, thus reducing the verification time for the verifier. However, when applying Groth's randomization technique, $\mathbf{b} \circ \mathbf{r}$ is replaced with $\mathbf{b}$, leading to $\mathbf{b}^* = \mathbf{b} \circ \mathbf{r} + \beta^{-1}\mathbf{ck_2}$. The resulting reduced-length (folded) vector $\mathbf{b}'$ is then expressed as follows:

$$\alpha^{-1}(\mathbf{b}_L \circ \mathbf{r}_L) + (\mathbf{b}_R \circ \mathbf{r}_R) + \alpha^{-1}\beta^{-1}\mathbf{ck_2}_L + \beta^{-1}\mathbf{ck_2}_R \in \mathbb{G}_2^{\frac{n}{2}}$$

(refer to **Algorithm 1**). Consequently, the i-th entry of vector $\mathbf{b}'$ now includes r^i and $r^{\frac{n}{2}+i-1}$. Thus, the verifier cannot compute $D_2' = \langle \mathbf{ck_1'}, \mathbf{b}' \rangle$ using commitment values D_{2L} and D_{2R}, and furthermore, cannot verify whether the prover has correctly computed $\mathbf{b} \circ \mathbf{r}$ and folded commitment key $\mathbf{ck}'$.

In our research, we tackle this limitation by introducing an elimination folding technique. This technique maintains sublinear computation for the verifier, ensuring transparency, and achieves proper randomization of vectors through folding, all while preserving the Hadamard product $\mathbf{a} \circ \mathbf{b} = \mathbf{u} \in \mathbb{G}_T^n$.

To provide further detail, given witness vectors $\mathbf{a}$ and $\mathbf{a_r}$, along with a random challenge $r \leftarrow \mathbb{F}$, the verifier aims to verify that $\mathbf{a_r}$ and $(1, r, r^2, \cdots, r^{n-1}) \circ \mathbf{a}$ are equal. To achieve this, we employ an iterative halving process for the degree of r through a folding technique, ultimately deriving scalar a and a_r. Verification of their equality ensures that $\mathbf{a_r}$ is indeed equal to $\mathbf{r} \circ \mathbf{a}$ (Further details about the elimination folding technique, refer to section 1.3).

1.1 Our Contributions

Hadamard Product Arguments. We propose an efficient Hadamard product argument, named Π_{HP}. The Π_{HP} maintains transparency and ensures a verification time of $\mathcal{O}_\lambda(\log n)$, sublinear proof size, and $\mathcal{O}_\lambda(n)$ proving time. To the best of our knowledge, this work represents the first sublinear-size HPA with a transparent setup. Moreover, we establish the security of our scheme under the SXDH assumption. To achieve this, we propose the elimination folding technique. This novel method enables the effective folding of the vector and the elimination of half of the exponents from the random vector $\mathbf{r}$. Consequently, both the resulting vectors $\mathbf{a_r}$ and $\mathbf{r} \circ \mathbf{a}$ simplify to scalars, ensuring identical values due to the removal of randomness in r. This advancement empowers efficient verification, confirming whether $\mathbf{a_r}$ equals $\mathbf{r} \circ \mathbf{a}$, and stands as a crucial element in the development of our efficient Hadamard product arguments protocol.

Efficient aggregation of pairing equations using Hadamard product arguments. In this work, we introduce an efficient method for aggregating in

Table 1: Comparison of asymptotic performance of Hadamard Product Arguments. [6,26,27] are interactive protocols where the parties have access to (polynomial) oracles. To instanciate oracles, [26] uses Lagrange-based KZG commitment, [6] uses tensor code.

Protocol	Transparency	$\mathcal{P}$'s time	Proof size	$\mathcal{V}$'s time	Security assumption
Our $\mathit{\Pi}_{\mathsf{HP}}$	Yes	$\mathcal{O}_\lambda(n)$	$\mathcal{O}_\lambda(\log n)$	$\mathcal{O}_\lambda(\log n)$	SXDH
Bulletproofs [7]	Yes	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(n)$	DL
VOproof [27] + KZG	No	$\mathcal{O}(n \log n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	-
Xie et al. [26] + lagrange KZG	No	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	-
Bünz et al. [8]	No	$\mathcal{O}_\lambda(n)$	$\mathcal{O}_\lambda(\log n)$	$\mathcal{O}_\lambda(\log n)$	ASDBP, SDH, GGM
Poppins [19]	No	$\mathcal{O}_\lambda(n)$	$\mathcal{O}_\lambda(\log n)$	$\mathcal{O}_\lambda(\log n)$	BSDH, EPKE
Daza et al. [10]	No	$\mathcal{O}_\lambda(n)$	$\mathcal{O}_\lambda(\log n)$	$\mathcal{O}_\lambda(\log n)$	q-A-DLOG
Lipmaa et al. [22]	No	$\mathcal{O}_\lambda(n \log n)$	$\mathcal{O}(1)$	$\mathcal{O}_\lambda(1)$	AGM, PDL

the verification of pairing equations using Hadamard product arguments. Our key contributions are as follows:

- We address the most general case where all a_i, b_i, c_i, d_i are distinct. Here, a_i, b_i, c_i, and d_i are elements of the groups involved in the pairing operation. Traditional methods of randomized aggregation (as [4,18]) become inefficient in this setting. We propose a novel approach using Hadamard product arguments, which reduces the number of pairing operations required for verification. This makes the verification process more efficient.
- Our technique can be applied to various cryptographic protocols that rely on pairing equations, such as digital signatures, zero-knowledge proofs, and commitment schemes. By improving the efficiency of these schemes, our method facilitates faster and more practical implementations.
- We demonstrate that our approach significantly enhances scalability and performance in the verification of pairing equations. By reducing the computational burden on the verifier, our method ensures faster verification processes, which is critical for large-scale cryptographic applications. The reduction in pairing operations leads to substantial improvements in both time and resource efficiency.

Through these contributions, we provide a framework that not only advances the state-of-the-art in proof aggregation but also offers practical benefits for a wide range of cryptographic protocols and applications.

Proof Aggregation. We propose an aggregator for Groth16 [16] pairing-based zk-SNARKs and a proof aggregation technique for the most general case of the KZG [18] pairing-based polynomial commitment scheme using our $\mathit{\Pi}_{\mathsf{HP}}$. Both applications support logarithmic-sized aggregated proof without requiring an additional trusted setup and significantly reduce the verifier's $\mathcal{O}(n)$ pairing operations to $\mathcal{O}(1)$ pairing operations.

Implementation. To demonstrate the efficiency of our proposed Π_{HP}, we implement and optimize it using batch proving techniques. The evaluation results reveal that Π_{HP} employs a logarithmic verifier, a notable improvement over the linear verifier associated with Bulletproofs [7].

1.2 Applications

This subsection introduces applications where our Hadamard product arguments can be applied. Consider a scheme with a verification equation $e(a, b) = e(c, d)$. Verifying n pairing equations, i.e., $e(a_i, b_i) = e(c_i, d_i)$ for $i \in [n]$, requires the verifier to perform $2n$ pairing operations. However, if in the aforementioned pairing equations, all b_i ($b := b_i$) are the same (as in BLS multi-signature [4]), the prover can aggregate a_i into a single proof using a challenge $r \leftarrow\$ \mathbb{F}$. Consequently, the system of pairing equations reduces to $e(\prod a_i^{r^{i-1}}, b) = \prod_{i=1}^{n} e(c_i, d_i)^{r^{i-1}}$, thus reducing the verifier's pairing operations from $2n$ to $n+1$. Additionally, if all d_i ($d := d_i$) are also the same (as in the case of the same crs and point for KZG commitment [18]), we obtain the pairing equation $e(\prod a_i^{r^{i-1}}, b) = e(\prod c_i^{r^{i-1}}, d)$. This allows the verifier to verify all n verification equations with just 2 pairing operations.

We consider the most general case where all a_i, b_i, c_i, d_i are distinct. That is, we address the case of having n verification equations of the form $e(a_i, b_i) = e(c_i, d_i)$. In this case, randomized proof aggregation is no longer efficient, but verifier-efficient proof aggregation is achievable using our Hadamard product arguments.

Aggregation for Groth16 Groth16 [16] presents an efficient zk-SNARK based on pairings, where the verifier verifies the equation $e(A, B) = e(C, D) \cdot e(S, H)$ using the proof $\pi = (A, B, C)$ (for detailed explanation, refer to **Appendix A**). Therefore, to aggregate n instances of Groth16, the verifier needs to verify the pairing equations $e(A_i, B_i) = e(C_i, D_i) \cdot e(S_i, H_i)$ for all $i \in [n]$. We consider each proof as a vector and contemplate their Hadamard product: $\mathbf{A} \circ \mathbf{B} = \mathbf{C} \circ \mathbf{D} + \mathbf{S} \circ \mathbf{H}$. Our approach reduces the verifier's required pairings from $\mathcal{O}(n)$ to $\mathcal{O}(1)$, efficiently improving the verification time for aggregation (for detailed explanation, refer to **Section 5**).

KZG commitment Aggregation for distinct crs KZG commitment [18] is a widely used polynomial commitment. Its verification involves pairing equations, where the verifier checks $e(C - g^a, h) = e(\pi, h^{z-s})$ (for detailed explanation, refer to **Appendix B**). As mentioned earlier, in the most general case where crs, polynomials, and points are all different, the above equation manifests into n pairing equations as follows: for all $i \in [n]$, $e(C_i - g^{a_i}, h) = e(\pi_i, h^{z_i - s_i})$. Similarly, we can consider this as vectors and apply Hadamard product arguments: $\mathbf{C} \circ \mathbf{h} = \boldsymbol{\pi} \circ \mathbf{H}$ (for detailed explanation, refer to **Section 6**).

1.3 Technical Overview

Let $\mathbf{a}, \mathbf{c} \in \mathbb{G}_1^n$ and $\mathbf{b}, \mathbf{d} \in \mathbb{G}_2^n$ be vectors. Define $\mathbf{r} := (1, r, r^2, \cdots, r^{n-1})$ for a random challenge $r \in \mathbb{F}$. The verifier aims to confirm the equation $\mathbf{a} \circ \mathbf{b} = \mathbf{c} \circ \mathbf{d}$. Traditional methods involve verifying n concurrent pairing equations $e(a_i, b_i) = e(c_i, d_i)$, which, however, entail the disadvantage of requiring n pairing operations for the verifier. To reduce verification costs, we transform both sides of the equation into an inner product form incorporating the random challenge r. We restructure the provided vectors as follows:

$$\mathbf{a_r} \leftarrow \mathbf{r} \circ \mathbf{a}, \quad \mathbf{c_r} \leftarrow \mathbf{r} \circ \mathbf{c}.$$

Now, we employ inner products for comparison. Let $X = \langle \mathbf{a_r}, \mathbf{b} \rangle \in \mathbb{G}_T$ be the inner pairing product (i.e., an inner product performed in a bilinear pairing group; a detailed description is provided in 2) Similarly, $X' := \langle \mathbf{c_r}, \mathbf{d} \rangle \in \mathbb{G}_T$. Then verifying $X = X'$ ensures $\sum r^{i-1} e(a_i, b_i) = \sum r^{i-1} e(c_i, d_i)$, implying $\mathbf{a} \circ \mathbf{b} = \mathbf{c} \circ \mathbf{d}$ with high probability. Therefore, to complete the argument of knowledge for the Hadamard product, we need to verify the following three equations:

$$(1)\ X = X', \quad (2)\ \mathbf{a_r} = \mathbf{r} \circ \mathbf{a}, \quad (3)\ \mathbf{c_r} = \mathbf{r} \circ \mathbf{c}.$$

The first equation (1) can be directly verified by the verifier using the given instances X and X'. The remaining equations (2) and (3) (as the same method is applied due to the symmetry of our Hadamard product equations) are verified using a novel folding technique called elimination folding. Our folding technique folds the vectors while removing the random challenge r included in the vector $\mathbf{a_r}$. As a result, the fully folded vector $\mathbf{a_r}$ becomes a group element in $\mathbb{G}_1$ with all r removed, which equals the value of folding $\mathbf{a}$ entirely. Set the commitment key for vector $\mathbf{a} \in \mathbb{G}_1^n$ to $\mathbf{ck_2} \leftarrow\!\!\$\ \mathbb{G}_2^n$. Similarly, set the commitment key for vector $\mathbf{b} \in \mathbb{G}_2^n$ to $\mathbf{ck_1} \leftarrow\!\!\$\ \mathbb{G}_1^n$.

For the witness $\mathbf{a}$ and $\mathbf{b}$, our argument of knowledge for the Hadamard product consists of the following steps:

- (reformatting) Upon receiving a random challenge r from the verifier, the prover computes $\mathbf{a_r} := \mathbf{r} \circ \mathbf{a} \in \mathbb{G}_1^n$ and calculates $C = \langle \mathbf{a}, \mathbf{b} \rangle$, $X = \langle \mathbf{a_r}, \mathbf{b} \rangle$, and $K = \langle \mathbf{k}, \mathbf{b} \rangle$, along with the commitments $D_1 = \langle \mathbf{a}, \mathbf{ck_2} \rangle$, $D_2 = \langle \mathbf{ck_1}, \mathbf{b} \rangle$, $D_3 = \langle \mathbf{a_r}, \mathbf{ck_2} \rangle$, and $D_4 = \langle \mathbf{k}, \mathbf{ck_2} \rangle$. Here, a folding key $\mathbf{k}$ is a newly introduced vector used to reduce the length of the key $\mathbf{ck_1}$.
- (elimination folding) The folding process in our protocol serves two main purposes: reducing the length of vectors (messages and keys) and halving the degree of the random challenge r. Since the left-half vector $\mathbf{a}_L = \mathbf{r}_L \circ \mathbf{a}_L$ and the right-half vector $\mathbf{a_r}_R = \mathbf{r}_R \circ \mathbf{a}_R$ (for a vector of even length, we use subscript L and R to represent its left and right halves, respectively), we can fold the vectors using random challenges α and r as follows: $\mathbf{a}' \leftarrow \alpha \mathbf{a}_L + \mathbf{a}_R$, $\mathbf{a}'_\mathbf{r} \leftarrow \alpha \mathbf{a_r}_L + r^{-\frac{n}{2}} \mathbf{a_r}_R$ $(= \mathbf{r}_L \circ \mathbf{a}')$. Thus, through the folding process, we reduce the length of vectors and the degree of the random r in $\mathbf{a_r}$ by half. Repeating this folding $m := \log n$ times eliminates all the r values that were

initially present in $\mathbf{a_r}$. Our protocol delegates the folding of key $\mathbf{ck_2}$ to the prover by including it in the message $\mathbf{b}$. However, since message $\mathbf{a}$ has already been reformatted into $\mathbf{a_r}$, we cannot add key $\mathbf{ck_1}$ to it, and therefore, we cannot delegate the folding of $\mathbf{ck_1}$ to the prover. To tackle this issue, we use a vector $\mathbf{k} \in \mathbb{G}_1^n$ for folding $\mathbf{ck_1}$. As a result, we separate the folding operations of the message $\mathbf{a}$ and the key $\mathbf{ck_1}$, enabling us to delegate them individually to the prover.

(i) (combine the message and the key)
$\mathbf{b}^* \leftarrow \mathbf{b} + \beta^{-1}\mathbf{ck_2}$,
$\mathbf{k}^* \leftarrow \mathbf{k} + \beta\mathbf{ck_1}$.

(ii) (r-elimination folding)
$\mathbf{a}' \leftarrow \alpha\mathbf{a}_L + \mathbf{a}_R$,
$\mathbf{b}' \leftarrow \alpha^{-1}\mathbf{b}^*{}_L + \mathbf{b}^*{}_R$,
$\mathbf{a}'_\mathbf{r} \leftarrow \alpha\mathbf{a}'_{\mathbf{r}L} + r^{-\frac{n}{2}}\mathbf{a}'_{\mathbf{r}R}$,
$\mathbf{k}' \leftarrow \alpha\mathbf{k}^*{}_L + \mathbf{k}^*{}_R$.

– (pairing check) After completing m rounds of the reduction process, the vectors $\mathbf{a}$, $\mathbf{b}$, $\mathbf{a_r}$, $\mathbf{k}$, and the keys $\mathbf{ck_1}$, $\mathbf{ck_2}$ are transformed into group elements $a, a_r, k, ck_1 \in \mathbb{G}_1$, and $b, ck_2 \in \mathbb{G}_2$, respectively. Notably, as a consequence of the r reduction in our folding protocol, a becomes equivalent to a_r. As a result, the verifier can ensure the validity of $\mathbf{a_r} = \mathbf{r} \circ \mathbf{a}$ by verifying $C = X$ and $D_1 = D_3$.

For each witness pair $(\mathbf{a}, \mathbf{b})$ and $(\mathbf{c}, \mathbf{d})$, the verifier, after reformatting to obtain X and X', checks if $X = X'$. Next, the verifier and prover execute a reduction and finalcheck to verify $\mathbf{a_r} = \mathbf{r} \circ \mathbf{a}$ and $\mathbf{c_r} = \mathbf{r} \circ \mathbf{c}$. In conclusion, we establish $\mathbf{a} \circ \mathbf{b} = \mathbf{c} \circ \mathbf{d}$.

1.4 Related Works

Hadamard Product Arguments. Groth introduces the first transparent argument system for various linear algebraic relations, including the Hadamard product, in [14]. The proving time is $\mathcal{O}(n^2)$, and the verification time is $\mathcal{O}(n)$ with $\mathcal{O}(n)$ communication, where n denotes the vector length. Bayer and Groth extend this argument system to multi-Hadamard product arguments in [2], achieving efficiency of $\mathcal{O}(mn)$ for the prover, $\mathcal{O}(n+m)$ for the verifier, and $\mathcal{O}(n+m)$ for communication cost, where m vectors have a length of n.

Another approach to constructing efficient Hadamard product arguments is based on a trusted setup, which generates a Common Reference String (CRS) [9,11,15,19,21]. Groth in [15] presents Hadamard product arguments with $\Theta(n^2)$ proving time, $\Theta(1)$ verifying time, and communication cost, but with a CRS size of $\Theta(n^2)$. Subsequently, [21] improves the CRS size to $\Theta(n)$ without sacrificing other performance metrics, and [11] further enhances the efficiency of proving time to $\Theta(n \log n)$. In LegoSNARKs [9], Hadamard product arguments with a Commit and Prove paradigm (which combines two relations with shared inputs into one) are presented. Their Hadamard product argument has $\mathcal{O}(n)$ proving time and $\mathcal{O}(\log n)$ verifying time. In Poppins [19], a multi-Hadamard product argument is introduced, improving upon the one presented by Bayer [2]. Poppins's variant of the multi-Hadamard product argument outsources the verifier's final check by leveraging the inner product argument of [8], achieving $\mathcal{O}(nm^2)$ prover complexity and $\mathcal{O}(\log n + m)$ verifier complexity. While gaining benefits from verification efficiency, the precondition for a trusted setup arises, which was not needed in Bayer and Groth's original multi-Hadamard product argument [2].

[26,27] employs Interactive Oracle Proofs (IOP) approaches into the Hadamard product relation. [26] constructs a Hadamard product protocol from an inner product protocol, achieving $\mathcal{O}(n)$ proving time and $\mathcal{O}(\log n)$ verifying time. However, this approach inherits the trusted setup of the underlying KZG polynomial commitment scheme. [27] introduces vector oracle proofs for the Hadamard relation and compiles them into a polynomial IOP, achieving prover performance of $\mathcal{O}(n \log n)$ and verifier performance of $\mathcal{O}(1)$. However, this approach also requires a trusted setup for the KZG polynomial commitment scheme.

Groth16 Proofs Aggregation. Bünz et al. [8] and Snarkpack [13] introduce proof aggregation schemes enabling the aggregation of n Groth16 [16] zk-SNARKs proofs with $\mathcal{O}(\log n)$ proof size and verifier time. Both approaches, by Bünz et al. and Snarkpack, necessitate a specific trusted setup to construct the structured reference string required for verifying such aggregated proofs. They share the conceptual premise that pairing-based SNARKs like Groth16 can be proven and verified solely using algebraic operations [8]. This implies that they can consolidate n Groth16 verification equations into one randomized verification equation using inner product arguments.

KZG commitments Aggregation Plonk [12] and Boneh et al. [3] delve into aggregation techniques using KZG commitments. Plonk introduces a method to aggregate KZG commitments for the same point across different polynomials. This enables efficient combination when multiple polynomials have evaluation values at the same point. In this case, the proof size scales with the number of polynomials, yet the verifier only requires 2 pairing operations. Conversely, Boneh et al. investigate the aggregation of KZG commitments for different points across different polynomials. This presents a technique to efficiently combine commitments when each polynomial has evaluation values at different points. Similar to Plonk, the verifier utilizes only 2 pairing operations, and the proof size is reduced to a constant.

Building upon these foundational works, we extend the aggregation capabilities to handle the most general case where each polynomial has evaluation values at different points, and additionally, each polynomial may be associated with different common reference strings (crs). This method allows the verifier to still require only 2 pairing operations while supporting a logarithmic proof size relative to the number of polynomials and points involved.

2 Preliminaries

2.1 Notations

Let $[n] = \{1, 2, \dots, n\}$. We denote vectors using lowercase boldface letters, such as $\mathbf{a}$, and their components are expressed as $\mathbf{a} = (a_1, a_2, \dots, a_n)$. The inner product of vectors $\mathbf{a}$ and $\mathbf{b}$ is denoted as $\langle \mathbf{a}, \mathbf{b} \rangle$, and their Hadamard product is represented as $\mathbf{a} \circ \mathbf{b}$. Concatenation of vectors $\mathbf{a}$ and $\mathbf{b}$ is denoted as $(\mathbf{a} \| \mathbf{b})$. For a vector $\mathbf{a}$ of even length, we use $\mathbf{a}_L$ and $\mathbf{a}_R$ to represent its left and right halves, respectively. If S is a set, "$x \leftarrow\!\!\$\, S$" indicates sampling x uniformly at random from S. We utilize Type III pairing $(\mathbb{F}_p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, h)$ with the following properties:

1. $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are groups of prime order p with generators $g \in \mathbb{G}_1$ and $h \in \mathbb{G}_2$,
2. The pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a non-degenerate bilinear map,
3. The value $e(g, h)$ generates $\mathbb{G}_T$.

In Type III pairings, we use the following inner pairing product operation [8]:

$$\langle \cdot, \cdot \rangle : \mathbb{G}_1^n \times \mathbb{G}_2^n \to \mathbb{G}_T, \quad \langle \mathbf{a}, \mathbf{b} \rangle = \prod_{i=0}^{n} e(a_i, b_i)$$

2.2 Assumptions

Definition 1 (Symmetric external Diffie-Hellman (SXDH) [1]). *The **Symmetric XDH** (or **SXDH**) assumption holds for a Type III pairing $(\mathbb{F}_p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, h)$ if the decisional Diffie–Hellman (DDH) assumption holds for $(\mathbb{F}_p, \mathbb{G}_1, g)$ and $(\mathbb{F}_p, \mathbb{G}_2, h)$.*

Lemma 1. *Let $(\mathbb{F}_p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, h)$ be a Type III pairing satisfying SXDH assumption and $n = \mathsf{poly}(\lambda)$. Then, given vector $\mathbf{b} \leftarrow\!\!\$\, \mathbb{G}^n_{i \in \{1,2,T\}}$ there is no non-uniform polynomial-time adversary $\mathcal{A}$ which can compute a non-trivial $\mathbf{a} \in \mathbb{F}^n$ such that $\langle \mathbf{a}, \mathbf{b} \rangle = 0$.*

Lemma 2. *Let $(\mathbb{F}_p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, h)$ be a Type III pairing satisfying SXDH assumption. Then, given $a_1, a_2 \leftarrow\!\!\$\, \mathbb{G}_1$ there is no non-uniform polynomial-time adversary $\mathcal{A}$ which can compute non-trivial $b_1, b_2 \in \mathbb{G}_2$ such that $e(a_1, b_1) + e(a_2, b_2) = 0$. In general, for a given vector $\mathbf{a} \leftarrow\!\!\$\, \mathbb{G}_1^n$, there is no adversary which can compute non-trivial vector $\mathbf{b} \in \mathbb{G}_2^n$ such that $\langle \mathbf{a}, \mathbf{b} \rangle = 0$. Similarly, there exists no non-uniform polynomial-time adversary which can compute non-trivial vector $\mathbf{a} \in \mathbb{G}_1^n$ such that $\langle \mathbf{a}, \mathbf{b} \rangle = 0$ for $\mathbf{b} \leftarrow\!\!\$\, \mathbb{G}_2^n$.*

2.3 Commitment scheme

Definition 2 (AFGHO commitment [1]). *The AFGHO vector commitment scheme is a structure preserving commitment to group elements. For some message $\mathbf{m} \in \mathbb{G}_1^n$, the triple of PPT algorithms (KeyGen, Commit, Open) are defined as follows:*

- $\mathbf{pp} \leftarrow \mathit{KeyGen}(1^\lambda) = \{\mathbf{ck} \leftarrow\!\!\$\ \mathbb{G}_2^n, H_i \leftarrow\!\!\$\ \mathbb{G}_i\}$
- $(C, o) \leftarrow \mathit{Commit}(\mathbf{pp}; \mathbf{m}) = \{r \leftarrow\!\!\$\ \mathbb{F}; (\langle \mathbf{m}, \mathbf{ck}\rangle + r \cdot e(H_1, H_2), r)\}$
- $1/\bot \leftarrow \mathit{Open}(\mathbf{pp}; C, \mathbf{m}, o) = \{\langle \mathbf{m}, \mathbf{ck}\rangle + o \cdot e(H_1, H_2) \overset{?}{=} C\}$

Definition 3 (Computational binding). *A triple of three algorithms (KeyGen,Commit,Open) provides computational binding property if for any PPT adversary $\mathcal{A}$ having knowledge of $\mathbf{ck}$:*

$$\Pr\left[\begin{array}{l|l} \mathit{Open}(\mathbf{ck}, C, \mathbf{m}, o) & \\ \wedge\ \mathit{Open}(\mathbf{ck}, C, \mathbf{m}', o') & (C, \mathbf{m}, o, \mathbf{m}', o') \\ \wedge\ \mathbf{m} \neq \mathbf{m}' & \leftarrow \mathcal{A}(\mathbf{ck}) \end{array}\right] \leq \mathit{negl}(1^\lambda).$$

Definition 4 (Perfect hiding). *A triple of three algorithms (KeyGen,Commit,Open) provides perfect hiding property if for all unbounded adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$:*

$$\Pr\left[\ b = b' \left|\ \begin{array}{l} \mathbf{ck} \leftarrow \mathit{KeyGen}(1^\lambda) \\ \wedge\ (\mathbf{a}_0, \mathbf{a}_1, st) = \mathcal{A}_0(\mathbf{ck}) \\ \wedge\ b \leftarrow\!\!\$\ \{0,1\} \\ \wedge\ (C, o) \leftarrow \mathit{Commit}(\mathbf{ck}, \mathbf{a}_b) \\ \wedge\ b' \leftarrow \mathcal{A}_1(C, st) \end{array}\right.\right] = \frac{1}{2}.$$

Definition 5 (Doubly homomorphism). *A triple of three algorithms (KeyGen,Commit,Open) provides doubly homomorphism if it satisfies the following conditions:*

- *The sets $(\mathcal{K}, +)$, $(\mathcal{M}, +)$, and (Image(CM),+) form abelian groups.*
- *For any $\mathbf{ck}$, $\mathbf{ck}' \in \mathcal{K}$ and $\mathbf{m}, \mathbf{m}' \in \mathcal{M}$, it holds that:*
  - $\mathit{Commit}(\mathbf{ck}; \mathbf{m}) + \mathit{Commit}(\mathbf{ck}; \mathbf{m}')$
    $= \mathit{Commit}(\mathbf{ck}; \mathbf{m} + \mathbf{m}')$,
  - $\mathit{Commit}(\mathbf{ck}; \mathbf{m}) + \mathit{Commit}(\mathbf{ck}'; \mathbf{m})$
    $= \mathit{Commit}(\mathbf{ck} + \mathbf{ck}'; \mathbf{m})$.

Note that, AFGHO commitment is perfectly hiding, computationally binding, and doubly homomorphic.

2.4 Zero-Knowledge Succinct interactive arguments of knowledge

We follow the presentation in [20, 24]. Let $(\mathcal{P}, \mathcal{V})$ be a pair of interactive PPT algorithms. In zero-knowledge succinct arguments of knowledge, the prover $\mathcal{P}$ wants to convince $\mathcal{V}$ that some statement holds and she/he has the key knowledge of why the statement holds without leaking any information of the knowledge.

Definition 6 (Public coin succinct interactive arguments of knowledge). *A succinct interactive argument of knowledge scheme for a NP language $\mathcal{R}$ is a protocol between $(\mathcal{P}, \mathcal{V})$ with following properties:*

- ***Completeness** : An argument is complete if given true statement $x \in \mathcal{R}$, for any witness w and $r \in \{0,1\}^*$,*
  $\Pr[\langle \mathcal{P}(pp, w), \mathcal{V}(pp, r)\rangle(x) = 1] = 1.$

– ***Soundness*** : *An argument is sound if for $\mathsf{x} \notin \mathcal{R}$, any PPT Prover $\mathcal{P}^*$, and for all $r \in \{0,1\}^*$,*
$\Pr[\langle \mathcal{P}^*(pp), \mathcal{V}(pp, r)\rangle(\mathsf{x}) = 1] \leq \textsf{negl}(\lambda)$.
– ***Knowledge soundness*** : *An argument is knowledge sound if for any PPT adversary $\mathcal{A}$, there exists a PPT extractor $\mathcal{E}$ such that $\forall \mathsf{x} \in \mathcal{R}, \forall r \in \{0,1\}^*$, if $\Pr[\langle \mathcal{A}(pp), \mathcal{V}(pp, r)\rangle(\mathsf{x}) = 1] \geq \textsf{negl}(\lambda)$, then $\Pr[\mathtt{Sat}_{\mathcal{R}}(x, \mathcal{E}^{\mathcal{A}}(pp, x)) = 1] \geq \textsf{negl}(\lambda)$.*
– ***Succinctness*** : *An argument is succinct if the communication between $\mathcal{P}$ and $\mathcal{V}$ is sublinear in $|w|$.*
– ***Public coin*** : *An argument is public coin if each $\mathcal{V}$'s message $\mathcal{M} \leftarrow\!\!\$\ \mathcal{C}$, for $\mathcal{C}$ is some fixed set.*

Definition 7. *An interactive argument $(\textsf{Gen}, \mathcal{P}, \mathcal{V})$ for $\mathcal{R}$ is* ***honest-verifier statistical zero-knowledge (HVSZK)*** *if there exists a PPT algorithm $S(\mathbb{x}, z)$ called the simulator, running in time polynomial in $|\mathbb{x}|$, such that for every $\mathbb{x} \in \mathcal{R}$, $w \in \mathcal{R}_{\mathbb{x}}$, and $z \in \{0,1\}^*$, the statistical distance between the distributions $tr\langle \mathcal{P}(w), \mathcal{V}(z)\rangle(\mathbb{x})$ and $S(\mathbb{x}, z)$ is $negl(\lambda)$.*

Definition 8 (Witness-extended emulation [17, 20, 25]). *An public coin interactive argument $(\textsf{Gen}, \mathcal{P}, \mathcal{V})$ for $\mathcal{R}$ has witness-extended emulation if for all deterministic polynomial time programs $\mathcal{P}^*$ there exists an expected polynomial time emulator $\mathcal{E}$ such that for all non-uniform polynomial time adversaries $\mathcal{A}$ and all $z_{\mathcal{V}} \in \{0,1\}^*$, the following probabilities differ by at most $\textsf{negl}(\lambda)$:*

$$\mathbb{P}\left[\mathcal{A}(t, \mathbb{x}) = 1 \;\middle|\; \begin{array}{l} pp \leftarrow \textsf{Gen}(1^\lambda) \\ \wedge\ (\mathbb{x}, z_{\mathcal{P}}) \leftarrow \mathcal{A}(pp) \\ \wedge\ t \leftarrow tr\langle \mathcal{P}^*(z_{\mathcal{P}}), \mathcal{V}(z_{\mathcal{V}})\rangle(\mathbb{x}) \end{array}\right]$$

and

$$\mathbb{P}\left[\begin{array}{l} \mathcal{A}(t, \mathbb{x}) = 1 \wedge \\ (\textsf{Accept}(t) = 1 \\ \Rightarrow \textsf{Sat}_{\mathcal{R}}(\mathbb{x}, w) = 1) \end{array} \;\middle|\; \begin{array}{l} pp \leftarrow \textsf{Gen}(1^\lambda) \\ \wedge\ (\mathbb{x}, z_{\mathcal{P}}) \leftarrow \mathcal{A}(pp) \\ \wedge\ (t, w) \leftarrow \mathcal{E}^{\mathcal{P}^*(z_{\mathcal{P}})}(\mathbb{x}) \end{array}\right]$$

The witness-extended emulation implies the soundness and knowledge soundness.

Definition 9 (Tree extractability (arguments) [20]). *A $(2\mu+1)$-move interactive protocol $(\mathcal{P}, \mathcal{V})$ with Verifier message space $\mathcal{C}$ is (W, ϵ)-**tree extractable** if there exists a PPT algorithm extracting a witness from $(w_1, \cdots, w_\mu)$-tree of accepting transcripts with failure probability $\leq \epsilon$, $\prod w_i \leq W$ and $\max_i(w_i) \leq \epsilon|\mathcal{C}|$.*

Definition 10 (Tree extractability (reductions) [20]). *We say an interactive protocol reducing $\mathbb{x} \in \mathcal{R}$ to $\mathbb{x}' \in \mathcal{L}'$ is (W, ϵ)-**tree extractable** if the composition of this arguments with a final $\mathcal{P}$ message revealing a witness x' for $\mathbb{x}'$ is a (W, ϵ)-tree extractable arguments for $\mathcal{R}$.*

3 Hadamard Product Arguments

In this section, we propose an efficient and transparent Hadamard product argument, referred to as Π_{HP}. For easy of understanding, we omit the expressions only for zero-knowledge and hiding in commitments here. However, our protocol can be easily extended to zero-knowledge arguments, and these extensions can be found in Appendix D.

Definition 11 (Relation for Hadamard Prodcut Arguments). *For a given commitment keys* $\mathbf{ck_1} \leftarrow\$ \mathbb{G}_1^n$, $\mathbf{ck_2} \leftarrow\$ \mathbb{G}_2^n$, *the relation* $\mathcal{R}^{HP}_{n,\mathbf{ck_1},\mathbf{ck_2}}$ *is defined by*

$$\left\{ \begin{array}{l|l} (D_1, D_2, D'_1, D'_2 \in \mathbb{G}_T \ ; & D_1 = \langle \mathbf{a}, \mathbf{ck_2} \rangle, \\ \mathbf{a}, \mathbf{c} \in \mathbb{G}_1^n, \mathbf{b}, \mathbf{d} \in \mathbb{G}_2^n) & D_2 = \langle \mathbf{ck_1}, \mathbf{b} \rangle, \\ & D'_1 = \langle \mathbf{c}, \mathbf{ck_2} \rangle, \\ & D'_2 = \langle \mathbf{ck_1}, \mathbf{d} \rangle, \\ & \mathbf{a} \circ \mathbf{b} = \mathbf{c} \circ \mathbf{d} \end{array} \right\}$$

3.1 Reformatting Algorithm

To verify the Hadamard product relation $\mathbf{a} \circ \mathbf{b} = \mathbf{c} \circ \mathbf{d}$, we employ a polynomial-based approach. This method utilizes two polynomials, $X = \sum_{i=1}^{n} r^{i-1} e(a_i, b_i)$ and $X' = \sum_{i=1}^{n} r^{i-1} e(c_i, d_i)$, constructed with a randomly chosen $r \leftarrow\$ \mathbb{F}$. Comparing the coefficients of these polynomials confirms the equality of all entries. In simple terms, the prover and verifier follow the steps below.

1. The prover transmits the inner pairing products $\langle \mathbf{a}, \mathbf{b} \rangle$,$\langle \mathbf{c}, \mathbf{d} \rangle$ and D_1, D_2, D'_1 and D'_2 which are commitments of $\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}$.
2. The verifier selects a random challenge r and sends it to the prover.
3. The prover calculates the vector $\mathbf{r} = (1, r, r^2, \cdots, r^{n-1})$ and computes $\mathbf{a_r} = \mathbf{r} \circ \mathbf{a}$ and $\mathbf{c_r}$. Additionally, the prover sets $\mathbf{k}$ and $\mathbf{k}'$ as the identity vectors in $\mathbb{G}_1^n$.
4. The prover sends the inner pairing product $\langle \mathbf{a_r}, \mathbf{b} \rangle$, $\langle \mathbf{k}, \mathbf{b} \rangle$, $\langle \mathbf{c_r}, \mathbf{d} \rangle$, $\langle \mathbf{k}', \mathbf{d} \rangle$, and AFGHO commitments to $\mathbf{a_r}$ and $\mathbf{k}$, as well as $\mathbf{c_r}$ and $\mathbf{k}'$, to the verifier.

If not stated otherwise, in this paper, we denote D_1, D_2, D'_1 and D'_2 as AFGHO commitments of vectors $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$, and $\mathbf{d}$ with commitment keys $\mathbf{ck_1}$ and $\mathbf{ck_2}$. Then, our reformatting algorithm Σ_{ref} is designed as follows.

In Σ_{ref}, the prover obtains new vectors $\mathbf{a_r}$ and $\mathbf{c_r}$ as new witnesses, incorporating a random challenge r. Additionally, key vectors $\mathbf{k}$ and $\mathbf{k}'$ are introduced to the witness for elimination folding. Our scheme involves appending commitment keys $\mathbf{ck_2}$ to the witness vector $\mathbf{b}$ and performing folding operations (equivalently, reducing for vector length). This is possible because both vectors belong to the same space $\mathbb{G}_2^n$, enabling the prover to delegate the folding of commitment keys to the verifier and reduce verification costs through precomputation. However, the new witness vector $\mathbf{a_r}$, already multiplied by the random challenge r in each component, cannot have commitment keys $\mathbf{ck_1}$ added and folded (which complicates efficient verification by the verifier). Therefore, we introduce a new vector

Algorithm 2 Σ_{ref} : Reformatting Algorithm

```
**Input:** $(D_1, D_2, D'_1, D'_2; \mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d})$
$\mathcal{P} \to \mathcal{V} : C = \langle \mathbf{a}, \mathbf{b} \rangle, \quad C' = \langle \mathbf{c}, \mathbf{d} \rangle$
$\mathcal{V} \to \mathcal{P} : r \leftarrow\$ \mathbb{F}$
$\mathcal{P} : \mathbf{r} \leftarrow (1, r, r^2, \ldots, r^{n-1}), \quad \mathbf{a_r} \leftarrow \mathbf{r} \circ \mathbf{a}, \quad \mathbf{c_r} \leftarrow \mathbf{r} \circ \mathbf{c},$
        $\mathbf{k} \leftarrow \mathbf{1}, \quad \mathbf{k}' \leftarrow \mathbf{1} \in \mathbb{G}_1^n$
$\mathcal{P} \to \mathcal{V} : X = \langle \mathbf{a_r}, \mathbf{b} \rangle, \quad K = \langle \mathbf{k}, \mathbf{b} \rangle,$
        $D_3 = \langle \mathbf{a_r}, \mathbf{ck_2} \rangle, \quad D_4 = \langle \mathbf{k}, \mathbf{ck_2} \rangle,$
        $X' = \langle \mathbf{c_r}, \mathbf{b} \rangle, \quad K' = \langle \mathbf{k}', \mathbf{b} \rangle,$
        $D'_3 = \langle \mathbf{c_r}, \mathbf{ck_2} \rangle, \quad D'_4 = \langle \mathbf{k}', \mathbf{ck_2} \rangle$
**Output:** $(C, X, K, D_{t\in[4]}, C', X', K', D'_{t\in[4]}\ ;$
        $\mathbf{a}, \mathbf{b}, \mathbf{a_r}, \mathbf{k}, \mathbf{c}, \mathbf{d}, \mathbf{c_r}, \mathbf{k}')$
```

$\mathbf{k}$ (and $\mathbf{k}'$) to efficiently fold the key $\mathbf{ck_1}$. $\mathbf{k}$ is initially set as the identity vector in group $\mathbb{G}_1^n$ at the beginning of the protocol and undergoes addition and folding with the key $\mathbf{ck_1}$. The verifier can verify the folding operation solely through the commitments, and if necessary, complete calculation of the i-th folding vector $\mathbf{k}$.

The verifier checks $X = X'$ from instances $(C, X, K, D_{t\in[4]})$ and $(C', X', K', D'_{t\in[4]})$. Now, the remaining task is to verify whether the two witness vectors $\mathbf{a_r}$ and $\mathbf{c_r}$ are correctly constructed from r.

3.2 Elimination Folding Algorithm

Specifically, the verifier needs to confirm that $\mathbf{a_r} = \mathbf{r} \circ \mathbf{a}$ and $\mathbf{c_r} = \mathbf{r} \circ \mathbf{c}$. As these equations are symmetric, the same verification scheme can be applied to each equation independently. Therefore, in this section, we devise an algorithm to verify $\mathbf{a_r} = \mathbf{r} \circ \mathbf{a}$ for $(C, X, K, D_{t\in[4]}; \mathbf{a}, \mathbf{b}, \mathbf{a_r}, \mathbf{k})$.

Our algorithm folds the witness $(\mathbf{a}, \mathbf{b}, \mathbf{a_r}, \mathbf{k})$ to halve the length of the vectors. Unlike other folding schemes, ours eliminates the factor r multiplied to $\mathbf{a_r}$ during folding (referred to as the elimination folding technique). This ensures that the folded $\mathbf{a}$ and $\mathbf{a_r}$ are identical, confirming that $\mathbf{a_r}$ correctly underwent Hadamard product with $\mathbf{a}$ using $\mathbf{r}$. We denote this elimination folding algorithm as Σ_{elim}. The Σ_{elim} relation is defined as follows:

Definition 12 (Relation for Elimination Folding). *Let $i = 1, 2$. For a given commitment keys $\mathbf{ck_i} \leftarrow\$ \mathbb{G}_i^n$ and a random challenge $r \in \mathbb{F}$, the relation $\mathcal{R}^{Elim}_{r,n,\mathbf{ck_1},\mathbf{ck_2}}$ is defined as follows:*

$$\left\{ \begin{array}{c|l} (C, X, K, D_{t\in[4]} \in \mathbb{G}_T\ ; & \mathbf{a_r} = (1, r, r^2, \cdots, r^{n-1}) \circ \mathbf{a}, \\ \quad \mathbf{a}, \mathbf{a_r}, \mathbf{k} \in \mathbb{G}_1^n, \mathbf{b} \in \mathbb{G}_2^n) & C = \langle \mathbf{a}, \mathbf{b} \rangle, \\ & X = \langle \mathbf{a_r}, \mathbf{b} \rangle, \quad K = \langle \mathbf{k}, \mathbf{b} \rangle, \\ & D_1 = \langle \mathbf{a}, \mathbf{ck_2} \rangle, \quad D_2 = \langle \mathbf{ck_1}, \mathbf{b} \rangle, \\ & D_3 = \langle \mathbf{a_r}, \mathbf{ck_2} \rangle, \quad D_4 = \langle \mathbf{k}, \mathbf{ck_2} \rangle \end{array} \right\}$$

After the reformatting, $(C, X, K, D_{t\in[4]})$ are computed. In Σ_{elim}, instead of the verifier, the prover combines the keys $\mathbf{ck_1}$ and $\mathbf{ck_2}$ with $\mathbf{k}$ and the message $\mathbf{b}$, respectively, to fold the keys. This process involves interactive computation for verifying statements within the relation and simultaneously reduces the witness $\mathbf{a}$, $\mathbf{b}$, $\mathbf{a_r}$, and $\mathbf{k}$ by half in size, utilizing a random challenge supplied by the verifier.

1. The prover calculates the inner pairing product values X_L, X_R, D_{3L}, D_{3R}, $D'_{t,L}$, $D'_{t,R}$ of halves for all $t \in [4]$ and sends these values to the verifier.
2. The verifier chooses a random challenge $\beta \leftarrow\$ \mathbb{F}$ and sends it to the prover.
3. Using β, the prover combines the witness $\mathbf{b}$ with the commitment key $\mathbf{ck_1}$ to create a new vector $\mathbf{b}^*$. The prover also combines the key $\mathbf{ck_1}$ with $\mathbf{k}$ to create a new vector $\mathbf{k}^*$.
4. The prover calculates the cross-terms $C_\pm, X_\pm, K_\pm$ of halves and sends them to the verifier.
5. The verifier chooses another random challenge $\alpha \leftarrow\$ \mathbb{F}$ and sends it to the prover.
6. The prover folds the vectors $\mathbf{a}$, $\mathbf{b}^*$, $\mathbf{a_r}$, and $\mathbf{k}^*$ using α and r.
7. The verifier verifies that the sum of values received in step 1 matches the previous round's instance and computes $C', X', K', D'_{t\in[4]}$ using α, β, and r.

Theorem 1. *Let $i = 1, 2$. For $\mathbf{ck'_i} \leftarrow\$ \mathbb{G}_i^{n/2}$, Σ_{elim} is a HVSZK, public-coin, succinct interactive argument of knowledge for $\mathcal{R}^{Elim}_{r,n/2,\mathbf{ck_1},\mathbf{ck_2}}$ with* $(9, 12/|\mathbb{F}|)$*-tree extractability under SXDH.*

Proof. First and foremost, succinctness and the public coin property are immediate. Refer to **Theorem 6** in the **Appendix** for the validity of HVSZK.
Completeness : Using the properties of AFGHO commitment, completeness can be easily verified as follows:

$$\begin{aligned}
C' &= \langle \mathbf{a}', \mathbf{b}' \rangle = \langle \alpha \mathbf{a}_L + \mathbf{a}_R, \alpha^{-1}\mathbf{b}^*_L + \mathbf{b}^*_R \rangle \\
&= C + \beta^{-1} D_1 + \alpha C_+ + \alpha^{-1} C_-, \\
X' &= \langle \mathbf{a_r}', \mathbf{b}' \rangle = \langle \alpha \mathbf{a_r}_L + r^{-\frac{n}{2}} \mathbf{a_r}_R, \alpha^{-1}\mathbf{b}^*_L + \mathbf{b}^*_R \rangle \\
&= X_L + r^{-\frac{n}{2}} X_R + \beta^{-1}(D_{3L} + r^{-\frac{n}{2}} D_{3R}) \\
&\quad + \alpha X_+ + \alpha^{-1} r^{-\frac{n}{2}} X_-, \\
K' &= \langle \mathbf{k}', \mathbf{b}' \rangle = \langle \alpha \mathbf{k}^*_L + \mathbf{k}^*_R, \alpha^{-1}\mathbf{b}^*_L + \mathbf{b}^*_R \rangle \\
&= K + \chi + \beta D_2 + \beta^{-1} D_4 + \alpha K_+ + \alpha^{-1} K_-, \\
D'_1 &= \langle \mathbf{a}', \mathbf{ck'_2} \rangle = \langle \alpha \mathbf{a}_L + \mathbf{a}_R, \mathbf{ck'_2} \rangle = \alpha D'_{1L} + D'_{1R}, \\
D'_2 &= \langle \mathbf{ck'_1}, \mathbf{b}' \rangle = \langle \mathbf{ck'_1}, \alpha^{-1}\mathbf{b}^*_L + \mathbf{b}^*_R \rangle \\
&= \alpha^{-1} D'_{2L} + D'_{2R} + \alpha^{-1}\beta^{-1} \varDelta_{2L} + \beta^{-1} \varDelta_{2R}, \\
D'_3 &= \langle \mathbf{a_r}', \mathbf{ck'_2} \rangle = \langle \alpha \mathbf{a_r}_L + r^{-\frac{n}{2}} \mathbf{a_r}_R, \mathbf{ck'_2} \rangle \\
&= \alpha D'_{3L} + r^{-\frac{n}{2}} D'_{3R}, \\
D'_4 &= \langle \mathbf{k}', \mathbf{ck'_2} \rangle = \langle \alpha \mathbf{k}^*_L + \mathbf{k}^*_R, \mathbf{ck'_2} \rangle \\
&= \alpha D'_{4L} + D'_{4R} + \alpha\beta \varDelta_{1L} + \beta \varDelta_{1R}.
\end{aligned}$$

Algorithm 3 Σ_{elim} : Elimination Folding Algorithm

```
  **Precompute:** $\mathbf{ck_1'} \leftarrow \mathbf{ck_1}_L,\ \mathbf{ck_2'} \leftarrow \mathbf{ck_2}_L,\ \chi = \langle \mathbf{ck_1}, \mathbf{ck_2} \rangle,$
              for $I \in \{L, R\},\ \Delta_{1I} = \langle \mathbf{ck_1}_I, \mathbf{ck_2'} \rangle,\ \Delta_{2I} = \langle \mathbf{ck_1'}, \mathbf{ck_2}_I \rangle$
  **Input:** $(C, X, K, D_{t\in[4]}; \mathbf{a}, \mathbf{b}, \mathbf{a_r}, \mathbf{k}) \in \mathcal{R}^{Elim}_{r,n,\mathbf{ck_1},\mathbf{ck_2}}$
  $\mathcal{P} \to \mathcal{V}$ : for $I \in \{L, R\}$,
          $X_I = \langle \mathbf{a_r}_I, \mathbf{b}_I \rangle,\ D_{3I} = \langle \mathbf{a_r}_I, \mathbf{ck_2}_I \rangle,$
          $D'_{1I} = \langle \mathbf{a}_I, \mathbf{ck_2'} \rangle,\ D'_{2I} = \langle \mathbf{ck_1'}, \mathbf{b}_I \rangle,$
          $D'_{3I} = \langle \mathbf{a_r}_I, \mathbf{ck_2'} \rangle,\ D'_{4I} = \langle \mathbf{k}_I, \mathbf{ck_2'} \rangle$
  $\mathcal{V} \to \mathcal{P}$ : $\beta \leftarrow\!\$\ \mathbb{F}$
  $\mathcal{P}$ : $\mathbf{b}^* \leftarrow \mathbf{b} + \beta^{-1}\mathbf{ck_2},\ \mathbf{k}^* \leftarrow \mathbf{k} + \beta\mathbf{ck_1}$
  $\mathcal{P} \to \mathcal{V}$ : $C_+ = \langle \mathbf{a}_L, \mathbf{b}^*_R \rangle,\ C_- = \langle \mathbf{a}_R, \mathbf{b}^*_L \rangle,$
          $X_+ = \langle \mathbf{a_r}_L, \mathbf{b}^*_R \rangle,\ X_- = \langle \mathbf{a_r}_R, \mathbf{b}^*_L \rangle,$
          $K_+ = \langle \mathbf{k}^*_L, \mathbf{b}^*_R \rangle,\ K_- = \langle \mathbf{k}^*_R, \mathbf{b}^*_L \rangle$
  $\mathcal{V} \to \mathcal{P}$ : $\alpha \leftarrow\!\$\ \mathbb{F}$
  $\mathcal{P}$ : $\mathbf{a}' \leftarrow \alpha\mathbf{a}_L + \mathbf{a}_R,$          $\mathbf{b}' \leftarrow \alpha^{-1}\mathbf{b}^*_L + \mathbf{b}^*_R,$
      $\mathbf{a}'_\mathbf{r} \leftarrow \alpha\mathbf{a_r}_L + r^{-\frac{n}{2}}\mathbf{a_r}_R,$   $\mathbf{k}' \leftarrow \alpha\mathbf{k}^*_L + \mathbf{k}^*_R$
  $\mathcal{V}$ : Check if $X = X_L + X_R,\quad D_3 = D_{3L} + D_{3R}$
      $C' \leftarrow C + \beta^{-1}D_1 + \alpha C_+ + \alpha^{-1}C_-$
      $X' \leftarrow X_L + r^{-\frac{n}{2}}X_R + \beta^{-1}(D_{3L} + r^{-\frac{n}{2}}D_{3R})$
            $+\alpha X_+ + \alpha^{-1}r^{-\frac{n}{2}}X_-$
      $K' \leftarrow K + \chi + \beta D_2 + \beta^{-1}D_4 + \alpha K_+ + \alpha^{-1}K_-$
      $D'_1 \leftarrow \alpha D'_{1L} + D'_{1R},$
      $D'_2 \leftarrow \alpha^{-1}D'_{2L} + D'_{2R} + \alpha^{-1}\beta^{-1}\Delta_{2L} + \beta^{-1}\Delta_{2R}$
      $D'_3 \leftarrow \alpha D'_{3L} + r^{-\frac{n}{2}}D'_{3R},$
      $D'_4 \leftarrow \alpha D'_{4L} + D'_{4R} + \alpha\beta\Delta_{1L} + \beta\Delta_{1R}$
  **Output:** $(C', X', K', D'_{t\in[4]}; \mathbf{a}', \mathbf{b}', \mathbf{a_r}', \mathbf{k}') \in \mathcal{R}^{Elim}_{r,\frac{n}{2},\mathbf{ck_1'},\mathbf{ck_2'}}$
```

Tree extractability : From **Definition 9**, we start with $\mu = 2$ and set $w_1 = w_2 = 3$. Thus, we have a tree of accepting transcripts for 3 values of β, and each β has 3 corresponding values of α. The failure scenario arises if any of these challenges are 0, which happens with a probability of $\leq 9/|\mathbb{F}|$. Next, we interpolate $C_\pm$, $X_\pm$, and $K_\pm$ as Laurent polynomials in $\mathbb{G}_T[\beta, \beta^{-1}]$ of degree 1 and order 1. Moreover, we perform interpolation for $\mathbf{a}'$ and $\mathbf{a_r}'$ as polynomials in $\mathbb{G}_T[\alpha]$ of degree 1, and $\mathbf{b}'$ and $\mathbf{k}'$ as Laurent polynomials in $\mathbb{G}_T[\alpha, \beta]$ of degree 1 and order 1. Let $s = r^{-\frac{n}{2}}$. For each leaf, considering $D'_3 = \alpha D'_{3L} + sD'_{3R} = \langle \mathbf{a_r}'(\alpha), \mathbf{ck_2'} \rangle$, we deduce that $\mathbf{a_r}'(\alpha) = \alpha\mathbf{a_r}_L + s\mathbf{a_r}_R$ by comparing the coefficients corresponding to α. Additionally, deriving from the equation $D'_4 = \alpha D'_{4L} + D'_{4R} + \alpha\beta\Delta_{1L} + \beta\Delta_{1R} = \langle \mathbf{k}', \mathbf{ck_2'} \rangle$, we can deduce that $\mathbf{k}'(\alpha, \beta) = \alpha\mathbf{k}_L + \mathbf{k}_R + \alpha\beta\mathbf{ck_1}_L + \beta\mathbf{ck_1}_R$. It's worth highlighting that from D'_1 and D'_2, the following relations emerge: $\mathbf{a}'(\alpha) = \alpha\mathbf{a}_L + \mathbf{a}_R$ and $\mathbf{b}'(\alpha, \beta) = \alpha^{-1}\mathbf{b}_L + \mathbf{b}_R + \alpha^{-1}\beta^{-1}\mathbf{ck_2}_L + \beta^{-1}\mathbf{ck_2}_R$.

Finally, we proceed to substitute these expressions into C', X', and K':

$$\begin{aligned}C' &= C + \beta^{-1}D_1 + \alpha C_+ + \alpha^{-1}C_- = \langle \mathbf{a}'(\alpha), \mathbf{b}'(\alpha,\beta)\rangle \\ &= \langle \mathbf{a}, \mathbf{b}^*\rangle + \alpha\langle \mathbf{a}_L, \mathbf{b}^*_R\rangle + \alpha^{-1}\langle \mathbf{a}_R, \mathbf{b}^*_L\rangle \\ &= \langle \mathbf{a}, \mathbf{b}\rangle + \beta^{-1}\langle \mathbf{a}, \mathbf{ck_2}\rangle + \alpha\langle \mathbf{a}_L, \mathbf{b}^*_R\rangle + \alpha^{-1}\langle \mathbf{a}_R, \mathbf{b}^*_L\rangle,\end{aligned}$$

$$\begin{aligned}X' &= X_L + sX_R + \beta^{-1}(D_{3L} + sD_{3R}) + \alpha X_+ + \alpha^{-1}sX_- \\ &= \langle \mathbf{a_r}'(\alpha), \mathbf{b}'(\alpha,\beta)\rangle \\ &= \langle \alpha\mathbf{a_r}_L + s\mathbf{a_r}_R, \alpha^{-1}\mathbf{b}^*_L + \mathbf{b}^*_R\rangle \\ &= \langle \mathbf{a_r}_L, \mathbf{b}_L\rangle + s\langle \mathbf{a_r}_R, \mathbf{b}_R\rangle + \beta^{-1}(\langle \mathbf{a_r}_L, \mathbf{ck_2}_L\rangle \\ &\quad + s\langle \mathbf{a_r}_R, \mathbf{ck_2}_R\rangle) + \alpha\langle \mathbf{a_r}_L, \mathbf{b}^*_R\rangle + \alpha^{-1}s\langle \mathbf{a_r}_R, \mathbf{b}^*_L\rangle, \\ K' &= K + \chi + \beta D_2 + \beta^{-1}D_4 + \alpha K_+ + \alpha^{-1}K_- \\ &= \langle \mathbf{k}'(\alpha,\beta), \mathbf{b}'(\alpha,\beta)\rangle = \langle \alpha\mathbf{k}^*_L + \mathbf{k}^*_R, \alpha^{-1}\mathbf{b}^*_L + \mathbf{b}^*_R\rangle \\ &= \langle \mathbf{k}, \mathbf{b}\rangle + \chi + \beta\langle \mathbf{ck_1}, \mathbf{b}\rangle + \beta^{-1}\langle \mathbf{k}, \mathbf{ck_2}\rangle \\ &\quad + \alpha\langle \mathbf{k}^*_L, \mathbf{b}^*_R\rangle + \alpha^{-1}\langle \mathbf{k}^*_R, \mathbf{b}^*_L\rangle.\end{aligned}$$

Upon comparing the coefficients for $1, \alpha, \alpha^{-1}, \beta, \beta^{-1}$ (notably, s is a known value and easily divisible), we arrive at the following expressions:

$$\begin{aligned}C &= \langle \mathbf{a}, \mathbf{b}\rangle,\ K = \langle \mathbf{k}, \mathbf{b}\rangle, \\ D_1 &= \langle \mathbf{a}, \mathbf{ck_2}\rangle,\ D_2 = \langle \mathbf{ck_1}, \mathbf{b}\rangle,\ D_4 = \langle \mathbf{k}, \mathbf{ck_2}\rangle, \\ X &= X_L + X_R = \langle \mathbf{a_r}_L, \mathbf{b}_L\rangle + \langle \mathbf{a_r}_R, \mathbf{b}_R\rangle = \langle \mathbf{a_r}, \mathbf{b}\rangle, \\ D_3 &= D_{3L} + D_{3R} = \langle \mathbf{a_r}_L, \mathbf{ck_2}_L\rangle + \langle \mathbf{a_r}_R, \mathbf{ck_2}_R\rangle = \langle \mathbf{a_r}, \mathbf{ck_2}\rangle\end{aligned}$$

The final concept we're going to establish is $\mathbf{a_r} = \mathbf{r} \circ \mathbf{a}$. We are already aware of the fact that $\mathbf{a_r}' = \alpha\mathbf{a_r}_L + s\mathbf{a_r}_R$ and $\mathbf{a}' = \alpha\mathbf{a}_L + \mathbf{a}_R$. Based on the assumption, $\mathbf{a_r}' = \mathbf{r}' \circ \mathbf{a}'$, where $\mathbf{r}' = \mathbf{r}_L = (r^0, r^1, \cdots, r^{\frac{n}{2}-1})$. This yields:

$$\alpha\mathbf{a_r}_L + s\mathbf{a_r}_R = \mathbf{r}' \circ (\alpha\mathbf{a}_L + \mathbf{a}_R) = \mathbf{r}' \circ \alpha\mathbf{a}_L + \mathbf{r}' \circ \mathbf{a}_R.$$

By comparing coefficients involving α, we conclude that $\mathbf{a_r}_L = \mathbf{r}' \circ \mathbf{a}_L$ and $s\mathbf{a_r}_R = \mathbf{r}' \circ \mathbf{a}_R$. Additionally, $\mathbf{a_r}_R = s^{-1}(\mathbf{r}' \circ \mathbf{a}_R) = (r^{\frac{n}{2}}, r^{\frac{n}{2}+1}, \cdots, r^{n-1}) \circ \mathbf{a}_R$. Hence, we establish that $\mathbf{a_r} = \mathbf{r} \circ \mathbf{a}$. Therefore, the provided witness $(\mathbf{a}, \mathbf{b}, \mathbf{a_r}, \mathbf{k})$ satisfies the desired properties for $(C, X, K, D_{t\in[4]})$.

3.3 Pairing Check Algorithm

After completing the reduction $m := \log n$ times, the prover obtains a witness a, b, a_r, k. The folded witnesses belong to the groups $\mathbb{G}_1$, $\mathbb{G}_2$, and the verifier can verify $\mathbf{a_r} = \mathbf{r} \circ \mathbf{a}$ through only one pairing operation. Our verification algorithm, Σ_{check}, utilizes the instance $C = e(a, b)$, $X = e(a_r, b)$, $D_1 = e(a, ck_2)$, $D_3 = e(a_r, ck_2)$ for the verifier to confirm the pairing equations $e(a, b) = e(a_r, b)$ and $e(a, ck_2) = e(a_r, ck_2)$.

1. The verifier sends a random challenge $c \leftarrow\!\!\$\ \mathbb{F}$ to the prover.
2. Utilizing the random challenge c, the prover randomizes a, b, a_r, k, and sends the resulting values $E_1 \in \mathbb{G}_1$ and $E_2 \in \mathbb{G}_2$ to the verifier. Here, particularly, E_1 is defined as $ca + c^3a + c^5a_r + c^7k$. Separating the same a with challenges c and c^3, the verifier substitutes C and X into the two C terms obtained from expanding the pairing equations to check if C and X are equal (similarly for $D_1 = D_3$).
3. To simultaneously verify the correctness of witness and key folding, the verifier utilizes a random $d \leftarrow\!\!\$\ \mathbb{F}$ to perform pairing operations by adding E_1 and $d \cdot ck_1$, and similarly E_2 and $d^{-1} \cdot ck_2$.
4. The term $c^4e(a, b)$ that appears when expanding the pairing equation should be c^4C if our reduction algorithm operates correctly, but this value should also be $c^4e(a_r, b)$, i.e., c^4X. Hence, the verifier substitutes C with X, and D_1 with D_3 respectively, into the verification equation. If this pairing equation holds, it indicates that $a_r = a$, ensuring the correct application of elimination folding, i.e., ensuring that the vector $\mathbf{a_r}$ equals $\mathbf{r} \circ \mathbf{a}$.

Algorithm 4 Σ_{check} : Pairing Check Algorithm

```
**Precompute:** $\chi = e(ck_1, ck_2)$
**Input:** $(C, X, K, D_{t\in[4]}; a, b, a_r, k) \in \mathcal{R}^{Elim}_{r,1,ck_i}$
$\mathcal{V} \to \mathcal{P} : c \leftarrow\!\!\$\ \mathbb{F}$
$\mathcal{P} \to \mathcal{V} : E_1 \leftarrow ca + c^3a + c^5a_r + c^7k, \quad E_2 \leftarrow cb,$
$\mathcal{V} : d \leftarrow\!\!\$\ \mathbb{F}$, accept if :
        $e(E_1 + d \cdot ck_1, E_2 + d^{-1} \cdot ck_2)$
          $= \chi + c^2C + c^4(1+c^2)X + c^8K + cd^{-1}D_1 + c^3d^{-1}(1+c^2)D_3 + c^7d^{-1}D_4$
```

Theorem 2. *Let $i = 1, 2$. For $ck_i \leftarrow\!\!\$\ \mathbb{G}_i$, Σ_{check} is a HVSZK, public-coin, succinct interactive argument of knowledge for $\mathcal{R}^{Elim}_{r,1,ck_1,ck_2}$ with $(27, 27/|\mathbb{F}|)$-tree extractability under SXDH.*

Proof. Succinctness and the public coin property are evident. Refer to **Theorem 7** in the **Appendix** for the validity of HVSZK.
Completeness : Given that $\mathbf{r} = (r^0)$ in $\mathcal{R}^{Elim}_{r,1,ck_1,ck_2}$, we have $a = a_r$. This implies $X = e(a_r, b) = e(a, b) = C$ and $D_1 = e(a, ck_2) = e(a_r, ck_2) = D_3$ in $\mathcal{R}^{Elim}_{r,1,ck_1,ck_2}$. Hence, completeness is demonstrated as follows:

$$\begin{aligned}
&e(E_1 + d \cdot ck_1, E_2 + d^{-1} \cdot ck_2) = e(ca + c^3a + c^5a_r + c^7k + d \cdot ck_1, cb + d^{-1} \cdot ck_2)\\
&= \chi + c^2e(a, b) + c^4e(a, b) + c^6e(a_r, b) + c^8e(k, b)\\
&\quad + cde(ck_1, b) + cd^{-1}e(a, ck_2) + c^3d^{-1}e(a, ck_2)\\
&\quad + c^5d^{-1}e(a_r, ck_2) + c^7d^{-1}e(k, ck_2)\\
&= \chi + c^2(C + c^2X + c^4X + c^6K) + cdD_2 + cd^{-1}(D_1 + c^2D_3 + c^4D_3 + c^6D_4)
\end{aligned}$$

Tree extractability : In our scenario, we set $\mu = 2$ and designate $w_1 = 9$ and $w_2 = 3$. Consequently, we find ourselves with a structure comprising accepting transcripts for 9 distinct values of c, and within each c value, there exists 3 corresponding accepting values of d. The event of failure arises if any of these 27 d values turns out to be 0. This unfortunate outcome transpires with a probability that is $\leq 27/|\mathbb{F}|$. Across all the presented transcripts, it remains constant that C, X, K, and $D_{t\in[4]}$ hold fixed. Furthermore, it is possible to interpolate E_1 and E_2 as polynomials of degree 8 concerning c. This interpolation is expressed as: $E_1(c) = d_1 + ca + c^3a + c^5a_r + c^7k + \sum_{i=1}^{4} c^{2i}U_i$ and $E_2(c) = d_2 + cb + \sum_{i=1}^{7} c^{j+1}U_j$. For each distinct c value, the conclusive verification encompasses terms reliant solely on d in the forms of $d, 1, d^{-1}$. This verification takes the shape of an equality check involving Laurent polynomials marked by degrees and orders of 1. The distinctive aspect here is that this discrepancy disappears for three distinct selections of d, thereby compelling the coefficients for each degree to be equal independently. Thus, for each of the 9 challenge values of c:

$$e(E_1(c), ck_2) = P_1 + cD_1 + (c^3 + c^5)D_3 + c^7D_4 \tag{1}$$

$$e(ck_1, E_2(c)) = P_2 + cD_2 \tag{2}$$

$$e(E_1(c), E_2(c)) = \sum_{i=1}^{3} c^{2i-1}Q_i + c^7P_3 + c^2C + (c^4 + c^6)X + c^8K \tag{3}$$

Equation (1) represents an equality of polynomials in $\mathbb{G}_T[c]$ of degree 8, which is valid at 9 points. Therefore, the coefficients on both the left-hand side and the right-hand side must be equal. Let's expand the coefficients to reveal their equivalence:

$$\begin{aligned}
&e(E_1(c), ck_2)\\
&\quad = e\Big(d_1 + ca + c^3a + c^5a_r + c^7k + \sum c^{2i}U_i, ck_2\Big)\\
&\quad = e(d_1, ck_2) + ce(a, ck_2) + c^3e(a, ck_2) + c^5e(x, ck_2)\\
&\qquad + c^7e(k, ck_2) + \sum c^{2i}e(U_i, ck_2)\\
&\quad = P_1 + cD_1 + c^3D_3 + c^5D_3 + c^7D_4 + \sum c^{2i}e(U_i, ck_2)
\end{aligned}$$

Thus, by comparing coefficients, we deduce that $D_1 = e(a, ck_2),\ D_3 = e(a_r, ck_2) = e(a, ck_2),\ D_4 = e(k, ck_2)$, and $e(U_i, ck_2) = 0$ for all $i \in [4]$. Similarly, comparing the coefficients of $e(ck_1, E_2(c))$ and $P_2 + cD_2$ leads us to $D_2 = e(ck_1, b)$ and $e(ck_1, V_j) = 0$ for all $j \in [7]$. As a result, we obtain the relationships: $a = a_r$, $U_i = V_j = r_{U_i} = r_{V_j} = 0$, and thus $E_1(c) = d_1 + ca + c^3a + c^5a_r + c^7k$ and $E_2(c) = d_2 + cb$. This allows us to identify tuple (a, b, a_r, k) that adhere to our constraints for (D_1, D_2, D_3, D_4). Substituting the expressions for E_1 and E_2 into

Equation (3), we get:

$$\begin{aligned}
&e(E_1(c), E_2(c)) \\
&= e(d_1 + ca + c^3 a + c^5 a_r + c^7 k, d_2 + cb) \\
&= e(d_1, d_2) + c(e(d_1, b) + e(a, d_2)) + c^3 e(a, d_2) + c^5 e(a_r, d_2) \\
&\quad + c^7 e(k, d_2) + c^2 e(a, b) + c^4 e(a, b) + c^6 e(a_r, b) + c^8 e(k, d_2) \\
&= R + cQ_1 + c^3 Q_2 + c^5 Q_3 + c^7 P_3 + c^2 C + c^4 X + c^6 X + c^8 K
\end{aligned}$$

Since this equality holds in $\mathbb{G}_T[c]$ at 9 distinct values, we can equate the coefficients of the powers of c on both sides. So, we obtain the following equations: $C = e(a, b),\ X = e(a, b) = e(a_r, b),\ K = e(k, b)$. From these equations, we have (a, b), (a_r, b), and (k, b) satisfying our constraints on C, X, K. Additionally, since $a = a_r$, a_r fulfills the requirement $a_r = r^0 \circ a$. We conclude that (a, b, a_r, k) forms a valid witness for $(C, X, K, D_{t\in[4]}) \in \mathcal{R}^{Elim}_{r,1,ck_1,ck_2}$.

3.4 Hadamard Product Arguments

In this section, we introduce an efficient Hadamard product argument Π_{HP} characterized by transparency and sublinear verification time. Our Π_{HP} determines whether the Hadamard product $\mathbf{a} \circ \mathbf{b} = \mathbf{c} \circ \mathbf{d} \in \mathbb{G}_T^n$ holds solely based on commitments of vectors. Leveraging doubly homomorphic commitments via the AFGHO commitment [1], our scheme guarantees security under the SXDH assumption.

In this protocol, since the commitment keys $\mathbf{ck_1} \leftarrow\!\$\ \mathbb{G}_1^n$ and $\mathbf{ck_2} \leftarrow\!\$\ \mathbb{G}_2^n$ are public, the verifier can precompute all the inner pairing products $\{\Delta_{1L}, \Delta_{1R}, \Delta_{2L}, \Delta_{2R}, \chi\}$ with respect to these keys before executing the elimination folding algorithm Σ_{elim}. This proactive step significantly reduces the verification time of the verifier to logarithmic size. Our Π_{HP} conducts precomputation of all inner pairing products required in the reduction algorithm before the reformatting step. This approach streamlines the verification process, enhancing efficiency. Using the provided commitments in the instance, the prover and verifier engage in Σ_{ref} to compute new witnesses and commitments, integrating the challenge r. Subsequently, the verifier verifies if the resulting X and X' are identical. In essence, $X = X'$ implies $\sum_{i=1}^{n} r^{i-1} e(a_i, b_i) = \sum_{i=1}^{n} r^{i-1} e(c_i, d_i)$. Consequently, with an error probability of $(n-1)/|\mathbb{F}|$, it ensures that $e(a_i, b_i) = e(c_i, d_i)$ for all i.

Subsequently, the prover and verifier iteratively apply Σ_{elim} $m = \log n$ times to the left and right relations $(C, X, K, D_{t\in[4]})$, $(C', X', K', D'_{t\in[4]})$ respectively. This iterative application facilitates elimination folding to each vector, effectively removing all random challenges r. Finally, through the execution of Σ_{check}, the verifier verifies the pairing equations, ensuring that the prover correctly computed the witnesses $\mathbf{a_r}$ and $\mathbf{c_r}$.

Theorem 3. *Let $n = 2^m$ and $i = 1, 2$. For $\mathbf{ck_{i,0}} \leftarrow\!\$\ \mathbb{G}_i^n$, the Π_{HP} is a HVSZK, public-coin, succinct interactive argument of knowledge for $\mathcal{R}^{HP}_{n,\mathbf{ck_{1,0}},\mathbf{ck_{2,0}}}$ with $(6 \cdot 9^{m+1}, 57 \cdot 9^m/|\mathbb{F}|)$-tree extractability under SXDH.*

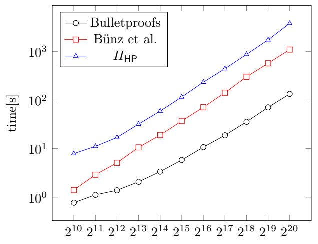
Algorithm 5 Π_{HP} : Hadamard Product Arguments

```
**Precompute:** for all $0 \le j \le m-1$, compute
        $\mathbf{ck_{1,j+1}} = (\mathbf{ck_{1,j}})_L,\ \ \mathbf{ck_{2,j+1}} = (\mathbf{ck_{2,j}})_L,$
for all $0 \le j \le m$, compute
        $\chi_j = \langle \mathbf{ck_{1,j}}, \mathbf{ck_{2,j}} \rangle,$
and for all $0 \le j \le m-1$, compute
        $\Delta_{1L,j} = \langle (\mathbf{ck_{1,j}})_L, \mathbf{ck_{2,j+1}} \rangle,$
        $\Delta_{2L,j} = \langle \mathbf{ck_{1,j+1}}, (\mathbf{ck_{2,j}})_L \rangle,$
        $\Delta_{1R,j} = \langle (\mathbf{ck_{1,j}})_R, \mathbf{ck_{2,j+1}} \rangle,$
        $\Delta_{2R,j} = \langle \mathbf{ck_{1,j+1}}, (\mathbf{ck_{2,j}})_R \rangle$
**Input:** $(D_1, D_2, D'_1, D'_2; \mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}) \in \mathcal{R}^{HP}_{2^m, \mathbf{ck_{1,0}}, \mathbf{ck_{2,0}}}$
$\mathcal{P}, \mathcal{V}$ : $\Sigma_{\mathsf{ref}}(D_1, D_2, D'_1, D'_2; \mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d})$
    $\to (C, X, K, D_{t\in[4]}; \mathbf{a}, \mathbf{b}, \mathbf{a_r}, \mathbf{k}) \in \mathcal{R}^{Elim}_{r, 2^m, \mathbf{ck_{1,0}}, \mathbf{ck_{2,0}}}$
and $(C', X', K', D'_{t\in[4]}; \mathbf{a}', \mathbf{b}', \mathbf{a}'_{\mathbf{r}}, \mathbf{k}') \in \mathcal{R}^{Elim}_{r, 2^m, \mathbf{ck_{1,0}}, \mathbf{ck_{2,0}}}$
$\mathcal{V}$ : Check if $X = X'$
For $1 \le j \le m$ :
$\mathcal{P}, \mathcal{V}$ : $\Sigma_{\mathsf{elim}}(C, X, K, D_{t\in[4]})$
        $\to (C, X, K, D_{t\in[4]}) \in \mathcal{R}^{Elim}_{r, 2^{m-j}, \mathbf{ck_{1,j}}, \mathbf{ck_{2,j}}}$
$\mathcal{P}, \mathcal{V}$ : $\Sigma_{\mathsf{elim}}(C', X', K', D'_{t\in[4]})$
        $\to (C', X', K', D'_{t\in[4]}) \in \mathcal{R}^{Elim}_{r, 2^{m-j}, \mathbf{ck_{1,j}}, \mathbf{ck_{2,j}}}$
$\mathcal{P}, \mathcal{V}$ : $\Sigma_{\mathsf{check}}(C, X, K, D_{t\in[4]}) \to result_1$
$\mathcal{P}, \mathcal{V}$ : $\Sigma_{\mathsf{check}}(C', X', K', D'_{t\in[4]}) \to result_2$
$\mathcal{V}$ : Accept if $(result_1 = 1 \ \wedge \ result_2 = 1)$
```

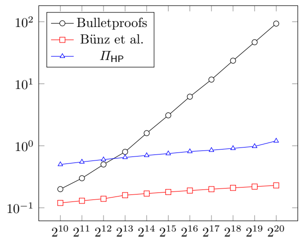
Proof. This follows directly from **Theorem 1** and **Theorem 2**. In this protocol, we have $(W, \epsilon) = (9, 12)$ for $m = \log n$ rounds of Σ_{elim} and $(27, 27)$ for 1 round of Σ_{check}. Therefore, W equals $27 \cdot 9^m$, while the error ϵ amounts to $12(1 + 9 + \cdots + 9^{m-1}) + 27 \cdot 9^m \approx 28.5 \cdot 9^m$, as outlined in Lemma 4 of [20]. We use Σ_{elim} and Σ_{check} on the left and right sides, respectively, so ultimately, $(W, \epsilon) = 2 \cdot (27 \cdot 9^m, 28.5 \cdot 9^m) = (6 \cdot 9^{m+1}, 57 \cdot 9^m)$.

3.5 Analyis

In Figure 1, we benchmark the efficacy of our proposed Π_{HP} against both Bulletproofs [7] and Bünz et al. [8]. While all three HPAs exhibit linear prover complexities, they differ in their verifier complexities. Specifically, the Bulletproofs operates with a linear verifier complexity, whereas both the Bünz et al. and our Π_{HP} utilize a logarithmic verifier complexity. However, the Bünz et al. requires a trusted setup. For vector sizes exceeding 2^{13}, our Π_{HP} demonstrates superior verification efficiency compared to the Bulletproofs. In scenarios where prover performance is the primary concern, the Bulletproofs proves to be the most performant. Conversely, for applications where both proof generation and verification efficiency are critical, our Π_{HP} stands out as the optimal choice.



(a) Prover's time



(b) Verifer's time

Fig. 1: Performance of Hadamard product arguments varying vector length

4 Batching Scheme for Elimination Folding

In this section, we propose a scheme for batching our elimination algoritm Σ_{elim}. Specifically, we suggest a process to combine two given Σ_{elim} relations into one relation. Since Π_{HP} executes two Σ_{elim} protocols, our batching scheme can double the verification speed of the Hadamard product arguments we propose.

Suppose $(C, X, K, D_{t\in[4]}; \mathbf{a}, \mathbf{b}, \mathbf{a_r}, \mathbf{k})$, and $(C', X', K', D'_{t\in[4]}; \mathbf{a}', \mathbf{b}', \mathbf{a}'_\mathbf{r}, \mathbf{k}')$ represent elimination folding relations within $\mathcal{R}^{Elim}_{r,n,\mathbf{ck_1},\mathbf{ck_2}}$. Then, we can derive the batching interactive arguments:

Algorithm 6 Batch-Σ_{elim}

```
**Input:** $(C, X, K, D_{t\in[4]}, C', X', K', D'_{t\in[4]}; \mathbf{a}, \mathbf{b}, \mathbf{a_r}, \mathbf{k}, \mathbf{a}', \mathbf{b}', \mathbf{a}'_\mathbf{r}, \mathbf{k}')$
$\mathcal{P} \to \mathcal{V}$ : $Z_C = \langle \mathbf{a}, \mathbf{b}' \rangle + \langle \mathbf{a}', \mathbf{b} \rangle$
            $Z_X = \langle \mathbf{a_r}, \mathbf{b}' \rangle + \langle \mathbf{a}'_\mathbf{r}, \mathbf{b} \rangle$
            $Z_K = \langle \mathbf{k}, \mathbf{b}' \rangle + \langle \mathbf{k}', \mathbf{b} \rangle$
$\mathcal{V} \to \mathcal{P}$ : $\delta \leftarrow\!\!\$\ \mathbb{F}$
$\mathcal{P}$ : $\mathbf{a}'' \leftarrow \delta\mathbf{a} + \mathbf{a}'$,     $\mathbf{b}'' \leftarrow \delta\mathbf{b} + \mathbf{b}'$
    $\mathbf{a}''_\mathbf{r} \leftarrow \delta\mathbf{a_r} + \mathbf{a}'_\mathbf{r}$,     $\mathbf{k}'' \leftarrow \delta\mathbf{k} + \mathbf{k}'$
$\mathcal{V}$ : $C'' \leftarrow \delta^2 C + \delta Z_C + C'$
    $X'' \leftarrow \delta^2 X + \delta Z_X + X'$
    $K'' \leftarrow \delta^2 K + \delta Z_K + K'$
    $D''_{t\in[4]} \leftarrow \delta D_{t\in[4]} + D'_{t\in[4]}$
**Output:** $(C'', X'', K'', D''_{t\in[4]}; \mathbf{a}'', \mathbf{b}'', \mathbf{a}''_\mathbf{r}, \mathbf{k}'') \in \mathcal{R}^{Elim}_{r,n,\mathbf{ck_1},\mathbf{ck_2}}$
```

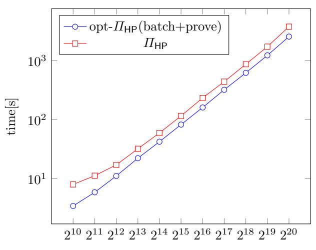
Theorem 4. *Let $i = 1, 2$. For $\mathbf{ck_i} \leftarrow\!\!\$\ \mathbb{G}_i^n$, Batch-$\Sigma_{\mathsf{elim}}$ is a HVSZK, public-coin, succinct interactive argument of knowledge for $(\mathcal{R}^{Elim}_{r,n,\mathbf{ck_1},\mathbf{ck_2}})^2$ with $(3, 3/|\mathbb{F}|)$-tree extractability under SXDH.*

Proof. Our proof can be found in Appendix D.6.

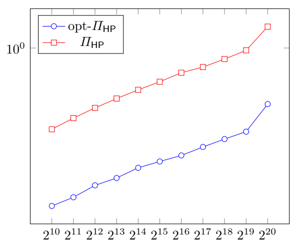
By batching two Σ_{elim} relations into one, we can design Hadamard product arguments by executing only half of Σ_{elim} and Σ_{check}. The optimized version of our Π_{HP} is as follows.

Algorithm 7 Optimized Hadamard Product Arguments

```
**Precompute:** for all $0 \le j \le m-1$, compute
        $\mathbf{ck_{1,j+1}} = (\mathbf{ck_{1,j}})_L$,  $\mathbf{ck_{2,j+1}} = (\mathbf{ck_{2,j}})_L$,
            for all $0 \le j \le m$, compute
              $\chi_j = \langle \mathbf{ck_{1,j}}, \mathbf{ck_{2,j}} \rangle$,
            and for all $0 \le j \le m-1$, compute
          $\Delta_{1L,j} = \langle (\mathbf{ck_{1,j}})_L, \mathbf{ck_{2,j+1}} \rangle$,
          $\Delta_{2L,j} = \langle \mathbf{ck_{1,j+1}}, (\mathbf{ck_{2,j}})_L \rangle$,
          $\Delta_{1R,j} = \langle (\mathbf{ck_{1,j}})_R, \mathbf{ck_{2,j+1}} \rangle$,
          $\Delta_{2R,j} = \langle \mathbf{ck_{1,j+1}}, (\mathbf{ck_{2,j}})_R \rangle$
**Input:** $(D_1, D_2, D'_1, D'_2 \;;\; \mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}) \in \mathcal{R}^{HP}_{2^m, \mathbf{ck_{1,0}}, \mathbf{ck_{2,0}}}$
$\mathcal{P}, \mathcal{V}$ : $\Sigma_{\mathsf{ref}}(D_1, D_2, D'_1, D'_2; \mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d})$
        $\to (C, X, K, D_{t\in[4]}; \mathbf{a}, \mathbf{b}, \mathbf{a_r}, \mathbf{k})$ and
    $(C', X', K', D'_{t\in[4]}; \mathbf{a}', \mathbf{b}', \mathbf{a}'_{\mathbf{r}}, \mathbf{k}') \in \mathcal{R}^{elim}_{r, 2^m, \mathbf{ck_{1,0}}, \mathbf{ck_{2,0}}}$
$\mathcal{V}$ : Check if $X = X'$
$\mathcal{P}, \mathcal{V}$ : Batch-$\Sigma_{\mathsf{elim}}((C, X, K, D_{t\in[4]}; \mathbf{a}, \mathbf{b}, \mathbf{a_r}, \mathbf{k}),$
            $(C', X', K', D'_{t\in[4]}; \mathbf{a}', \mathbf{b}', \mathbf{a}'_{\mathbf{r}}, \mathbf{k}'))$
$\to (C'', X'', K'', D''_{t\in[4]}; \mathbf{a}'', \mathbf{b}'', \mathbf{a}''_{\mathbf{r}}, \mathbf{k}'') \in \mathcal{R}^{elim}_{r, 2^m, \mathbf{ck_{1,0}}, \mathbf{ck_{2,0}}}$
**For** $1 \le j \le m$ :
$\mathcal{P}, \mathcal{V}$ : $\Sigma_{\mathsf{elim}}(C'', X'', K'', D''_{t\in[4]})$
        $\to (C'', X'', K'', D''_{t\in[4]}) \in \mathcal{R}^{elim}_{r, 2^{m-j}, \mathbf{ck_{1,j}}, \mathbf{ck_{2,j}}}$
$\mathcal{P}, \mathcal{V}$ : $\Sigma_{\mathsf{check}}(C, X, K, D_{t\in[4]}) \to result$
$\mathcal{V}$ : Accept if $(result = 1)$
```



(a) Prover's time



(b) Verifer's time

Fig. 2: Performance comparison of opt-Π_{HP} and Π_{HP}

5 Groth16 Proof Aggregation

In this section, we aggregate n proofs into a $\mathcal{O}(\log n)$-size proof using the Π_{HP} protocol without any additional trusted setup. Our aggregation protocol, $\mathsf{Gro16.Agg}$, supports proof aggregation for pairing-based SNARKs. Specifically, we present algorithms supporting proofs of Groth16 zk-SNARKs in this paper.

In Groth16 [16], the verifier uses the proof $\pi = (A, B, C) \in \mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{G}_1$ received from the prover to verify a single equation:

$$e(A,B) = e(g^\alpha, h^\beta) + e\left(\sum_{j=0}^{\ell} S_j^{a_j}, h^\gamma\right) + e(C, h^\delta)$$

where $S_j = g^{\frac{\beta u_j(z)+\alpha v_j(z)+w_j(z)}{\gamma}}$. (For detailed information on Groth16 Construction, please refer to Appendix A). Let $\pi_i = (A_i, B_i, C_i)$, $i \in [n]$, be n Groth16 proofs. For each Groth16 proof, the verifier must compute the following pairing equation using the verification key $\mathbf{vk}_i = (g^\alpha, h^\beta, \{S_{i,j}\}, h^\gamma, h^\delta)$:

$$\forall i \in [n],\ e(A_i, B_i) = e(g^\alpha, h^\beta) + e\left(\sum_{j=0}^{\ell} S_{i,j}^{a_{i,j}}, h^\gamma\right) + e(C_i, h^\delta).$$

This can be defined as a relation as follows:

Definition 13 (Relation for n Groth16). *The relation for n-length Groth16 proofs π_i with respect to a verification key $\mathsf{vk} = (g^\alpha,\ h^\beta,\ \{S_j\},\ h^\gamma,\ h^\delta)$: For all $i \in [n]$,*

$$\left\{ \begin{array}{c} \Big(\{a_{i,j}, \mathsf{CM}(A_i), \mathsf{CM}(B_i), \mathsf{CM}(C_i)\}; \{\pi_i\}\Big) : \\ e(A_i, B_i) = e(g^\alpha, h^\beta) + e\left(\sum_{j=0}^{\ell} S_{i,j}^{a_{i,j}}, h^\gamma\right) + e(C_i, h^\delta) \end{array} \right\}$$

Typically, due to the different values of all B_i (or A_i), it is impossible to aggregate them by forming a linear combination using a random challenge. We consider each component as a vector and transform n equations into Hadamard pairing product form to simultaneously prove n equations.

We form a vector composed of n Groth16 proofs $\pi_i = (A_i, B_i, C_i)$ as follows: $\mathbf{A} = (A_1, A_2, \cdots, A_n) \in \mathbb{G}_1^n$, $\mathbf{B} = (B_1, B_2, \cdots, B_n) \in \mathbb{G}_2^n$, $\mathbf{C} = (C_1, C_2, \cdots, C_n) \in \mathbb{G}_1^n$. Thus, the n individual proofs can be combined into a single proof denoted as $\pi = (\mathbf{A}, \mathbf{B}, \mathbf{C})$. We define $\mathbf{P} \in \mathbb{G}_1^n$ as the vector obtained by repeating g^α n times. Similarly, $\mathbf{Q} \in \mathbb{G}_2^n$ is the vector obtained by repeating h^β n times, $\mathbf{H} \in \mathbb{G}_2^n$ is the vector obtained by repeating h^γ n times, and $\mathbf{D} \in \mathbb{G}_2^n$ is the vector obtained by repeating h^δ n times. Let $\mathbf{S}$ be a verification vector defined as

$$\left(\sum_{j=0}^{\ell} S_{i,j}^{a_{1,j}}, \cdots, \sum_{j=0}^{\ell} S_{i,j}^{a_{n,j}}\right) \in \mathbb{G}_1^n.$$

Then, these n pairing equations are equivalent to a single Hadamard pairing product equation:

$$\mathbf{A} \circ \mathbf{B} = \mathbf{P} \circ \mathbf{Q} + \mathbf{S} \circ \mathbf{H} + \mathbf{C} \circ \mathbf{D}.$$

Similar to the Hadamard pairing product argument, the above equation can be proven using a random challenge $r \leftarrow\$ \mathbb{F}$, demonstrating that

$$\langle \mathbf{A_r}, \mathbf{B} \rangle = \langle \mathbf{P_r}, \mathbf{Q} \rangle + \langle \mathbf{S_r}, \mathbf{H} \rangle + \langle \mathbf{C_r}, \mathbf{D} \rangle.$$

where

$$\begin{aligned}
\mathbf{A_r} &= (1, r, r^2, \cdots, r^{n-1}) \circ \mathbf{A}, \\
\mathbf{P_r} &= (1, r, r^2, \cdots, r^{n-1}) \circ \mathbf{P}, \\
\mathbf{S_r} &= (1, r, r^2, \cdots, r^{n-1}) \circ \mathbf{S}, \\
\mathbf{C_r} &= (1, r, r^2, \cdots, r^{n-1}) \circ \mathbf{C}.
\end{aligned}$$

In this equation, by utilizing $\mathbf{vk}$, the verifier can compute the term $\langle \mathbf{P_r}, \mathbf{Q} \rangle + \langle \mathbf{S_r}, \mathbf{H} \rangle$. Therefore, similar to the Hadamard pairing product relation, we define the relation in $\mathsf{Gro16.Agg}$ as follows:

Definition 14 (Relation for Gro16.Agg). *The relation for* $\mathsf{Gro16.Agg}$ *is defined as follows: Let*

- $\mathbf{U} := \{a_{i,j}\}_{0 \leq j \leq \ell}^{1 \leq i \leq n}$ *be an instance matrix.*
- $\mathsf{vk} := (g^\alpha, h^\beta, S_j, h^\gamma, h^\delta)$ *be a verification key.*

Then $\mathcal{R}^{gra}n, \mathbf{ck_1}, \mathbf{ck_2}$ *with* vk *is a*

$$\left\{ \begin{array}{l|l}
\begin{array}{l} (\mathbf{U}, D_1, D_2, D_1' \ ; \\ \mathbf{A}, \mathbf{B}, \mathbf{C}) \end{array} &
\begin{array}{l}
D_1 = \langle \mathbf{A}, \mathbf{ck_2} \rangle, \\
D_2 = \langle \mathbf{ck_1}, \mathbf{B} \rangle, \\
D_1' = \langle \mathbf{C}, \mathbf{ck_2} \rangle, \\
\mathbf{P} = (g^\alpha, g^\alpha, \cdots, g^\alpha), \\
\mathbf{Q} = (h^\beta, h^\beta, \cdots, h^\beta), \\
\mathbf{H} = (h^\gamma, h^\gamma, \cdots, h^\gamma), \\
\mathbf{D} = (h^\delta, h^\delta, \cdots, h^\delta), \\
\mathbf{S} = \left(\sum_{j=0}^{\ell} S_{i,j}^{a_{1,j}}, \cdots, \sum_{j=0}^{\ell} S_{i,j}^{a_{n,j}} \right), \\
\mathbf{A} \circ \mathbf{B} = \mathbf{P} \circ \mathbf{Q} + \mathbf{S} \circ \mathbf{H} + \mathbf{C} \circ \mathbf{D}
\end{array}
\end{array} \right\}.$$

To verify the above relation, similar to the Hadamard pairing product, reformatting is performed, and additionally, the verifier must compute $\varPhi$. To achieve this, the prover generates a vector $\mathbf{D}$ and computes its commitment D_2', which is then sent to the verifier. The prover and verifier execute the reformatting algorithm Σ_{ref} using $(D_1, D_2, D_1', D_2'; \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D})$. Additionally, the verifier computes $\varPhi = \langle \mathbf{r} \circ \mathbf{P}, \mathbf{Q} \rangle + \langle \mathbf{r} \circ \mathbf{S}, \mathbf{H} \rangle$ using the verification key vk.

After the reformatting process, the verifier obtains the instance ($\varPhi$, C, X, K, $D_{t\in[4]}$, C', X', K', $D'_{t\in[4]}$). Similar to $\varPi_{\mathsf{HP}}$, the verifier checks $X = X' + \varPhi$. This

Algorithm 8 Gro16.Ref : Reformatting for Gro16.Agg

```
Input: $(\mathbf{U}, D_1, D_2, D_1'; \mathbf{A}, \mathbf{B}, \mathbf{C})$
$\mathcal{P}$ : $\mathbf{D} \leftarrow (h^\delta, h^\delta, \cdots, h^\delta)$
$\mathcal{P} \to \mathcal{V}$ : $D_2' = \langle \mathbf{ck_1}, \mathbf{D} \rangle$
$\mathcal{P}, \mathcal{V}$ : $\Sigma_{\mathsf{ref}}(D_1, D_2, D_1', D_2'; \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D})$
        $\to (C, X, K, D_{t\in[4]}, C', X', K', D'_{t\in[4]})$
$\mathcal{V}$ : Compute $\mathbf{P} = (g^\alpha, g^\alpha, \cdots, g^\alpha)$,
        $\mathbf{Q} = (h^\beta, h^\beta, \cdots, h^\beta)$
        $\mathbf{H} = (h^\gamma, h^\gamma, \cdots, h^\gamma)$,
        $\mathbf{S} = \left( \sum_{j=0}^{\ell} S_{i,j}^{a_{1,j}}, \cdots, \sum_{j=0}^{\ell} S_{i,j}^{a_{n,j}} \right)$,
        $\Phi = \langle (1, r, r^2, \cdots, r^{n-1}) \circ \mathbf{P}, \mathbf{Q} \rangle$
            $+ \langle (1, r, r^2, \cdots, r^{n-1}) \circ \mathbf{S}, \mathbf{H} \rangle$
Output: $(\Phi, C, X, K, D_{t\in[4]}, C', X', K', D'_{t\in[4]})$
```

ensures $\mathbf{A} \circ \mathbf{B} = \mathbf{P} \circ \mathbf{Q} + \mathbf{S} \circ \mathbf{H} + \mathbf{C} \circ \mathbf{D}$ with an error probability of $(n-1)/|\mathbb{F}|$. Finally, the verifier verifies if the witness $\mathbf{A_r}$ and $\mathbf{C_r}$ are computed correctly by executing Σ_{elim} and Σ_{check}. With this, we verify all n Groth16 verification equations. Our proof aggregation scheme Gro16.Agg does not require a trusted setup and has a verification time of $\mathcal{O}_\lambda(\log n)$. Additionally, it has proof size $\mathcal{O}(\log n)$ and proof generation time $\mathcal{O}_\lambda(n)$, demonstrating its efficiency.

Algorithm 9 Gro16.Agg: Groth16 proof aggregation

```
Input: $(\mathbf{U}, D_1, D_2, D_1'; \mathbf{A}, \mathbf{B}, \mathbf{C})$
$\mathcal{P}, \mathcal{V}$ : Run Gro16.Ref$(\mathbf{U}, D_1, D_2, D_1'; \mathbf{A}, \mathbf{B}, \mathbf{C})$
        $\to (\Phi, C, X, K, D_{t\in[4]}, C', X', K', D'_{t\in[4]})$
$\mathcal{V}$ : Check if $X = X' + \Phi$
For $i = 1, 2$ :
$\mathcal{P}, \mathcal{V}$ : Run $\Pi_{\mathsf{HP}}(D_1, D_2, D_1', D_2') \to result$
$\mathcal{V}$ : Accept if $result = 1$
```

Theorem 5. *Let $n = 2^m$ and $i = 1, 2$. For $\mathbf{ck_i} \leftarrow\$ \mathbb{G}_i^n$, the Gro16.Agg is a HVSZK, public-coin, succinct interactive argument of knowledge for $\mathcal{R}^{AGG}_{n,\boldsymbol{vk},\mathbf{ck_i}}$ with $(6 \cdot 9^{m+1}, 57 \cdot 9^m/|\mathbb{F}|)$tree extractability under SXDH.*

Proof. The proof is straightforward by Theorem 3.

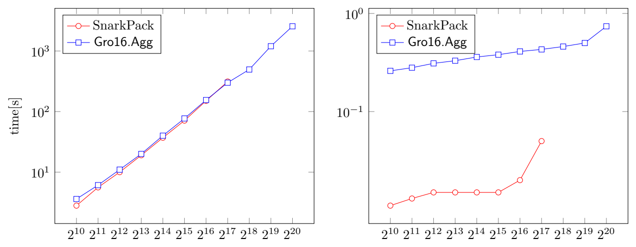
5.1 Analysis

In this section, we compare our aggregation scheme, Gro16.Agg, with SnarkPack [13]. SnarkPack is designed based on TIPP (target inner pairing product) and

MIPP (multiexponentiation inner product) [8], requiring an additional trusted setup. In contrast, our scheme, influenced by Dory [20], achieves Hadamard product arguments while maintaining transparency.

Both protocols have the same asymptotic complexity ($\mathcal{O}_\lambda(n)$ proving time, $\mathcal{O}_\lambda(\log n)$ verification time, $\mathcal{O}(\log n)$ proof size), but our scheme offers greater security due to its transparency. Table 2 illustrates the asymptotic performance comparison between [13] and our work. Here, n represents the number of Groth16 proofs. Notably, our scheme does not require additional trusted setups for proof aggregation, yet the $\mathcal{P}$'s time, $\mathcal{V}$'s time, and proof size align with those of SnarkPack. Both schemes are designed based on pairings. SnarkPack proposes the ASDGH (Auxiliary Structured Double Group Pairing) as a security assumption, which requires additional GGM (Generic Group Model). However, our Gro16.Agg guarantees soundness solely based on the SXDH assumption without requiring GGM.

Figure 3 compares the specific performance of our aggregation scheme against SnarkPack. The figure demonstrates consistent equivalence in proving time for both schemes. While evaluations for proof numbers exceeding 2^{18} were not conducted on a 32GB RAM M1 Pro MacBook, extrapolation from observation trends suggests they would likely exhibit similar performance. In terms of verification efficiency, SnarkPack consistently achieves faster results. However, our approach also demonstrates commendable practical efficiency. Thus, in applications where both security and practical speed are crucial, our scheme proves to be more beneficial.



(a) Prover's time (b) Verifer's time

Fig. 3: Performance of Groth16 aggregation varying number of proof

Table 2: Comparison of asymptotic performance of Groth16 aggregation

Protocol	additional trusted setup	$\mathcal{P}$'s time	Proof size	$\mathcal{V}$'s time	CRS size
SnarkPack [13]	✓	$\mathcal{O}_\lambda(n)$	$\mathcal{O}(\log n)$	$\mathcal{O}_\lambda(\log n)$	$\mathcal{O}(n)$
Gro16.Agg	✗	$\mathcal{O}_\lambda(n)$	$\mathcal{O}(\log n)$	$\mathcal{O}_\lambda(\log n)$	$\mathcal{O}(n)$

Table 3: Comparison of opening complexity of KZG aggregation

	[12]	[3]	Our $\mathsf{KZG.Agg}$
$\mathcal{P}$'s work	$\mathcal{O}(dn)\mathbb{G}_1$	$\mathcal{O}(d)\mathbb{G}_1$	$\mathcal{O}(n)P$, $\mathcal{O}(dn)\mathbb{G}_1$
Proof size	$\mathcal{O}(n)\mathbb{G}_1$	$\mathcal{O}(1)\mathbb{G}_1$	$\mathcal{O}(\log n)\mathbb{G}_T$
$\mathcal{V}$'s work	$\mathcal{O}(n)\mathbb{G}_1$, $2P$	$\mathcal{O}(n)\mathbb{G}_1$, $2P$	$\mathcal{O}(\log n)\mathbb{G}_T$, $3P$
Support for distinct polynomials	✓	✓	✓
Support for distinct points	✗	✓	✓
Support for distinct srs	✗	✗	✓
srs size (distinct srs case)	$\mathcal{O}(d)\mathbb{G}_1$ (✗)	$\mathcal{O}(d)\mathbb{G}_1$ (✗)	$\mathcal{O}(d)\mathbb{G}_1$ ($\mathcal{O}(dn)\mathbb{G}_1$)

6 KZG Proof Aggregation

In this section, we introduce the $\mathsf{KZG.Agg}$ for aggregating KZG commitments using our ZKP Π_{HP}. Our scheme $\mathsf{KZG.Agg}$ aims to minimize verification costs by reducing the pairing operations performed by the Verifier to a constant size.

In the KZG commitment scheme, the verifier computes the pairing equation for verification:

$$e(C - g^e, h) = e(\pi, h^{z-s})$$

In this section, we explore the generation of KZG proofs π_i with random challenges $s_i \leftarrow\!\!\$\ \mathbb{F}$ in the general KZG commitment scheme. Additionally, we consider the scenario where all polynomials are committed using different commitment keys (the same s_i, $f_i(x)$, or commitment keys' usage can be considered for specific cases). Consequently, in our setting, the verifier needs to compute the following n pairing equations:

$$\begin{aligned} e(C_1 - g^{e_1}, h) &= e(\pi_1, h^{z_1 - s_1}), \\ e(C_2 - g^{e_2}, h) &= e(\pi_2, h^{z_2 - s_2}), \\ &\vdots \\ e(C_n - g^{e_n}, h) &= e(\pi_n, h^{z_n - s_n}). \end{aligned}$$

(In this section, we assume that all polynomials have the same degree.) Since the left side uses the same h, we can compress the single pairing equation by computing a linear combination $C = \sum_{i=1}^{n} r^{i-1}(C_i - g^{e_i})$ using challenge $r \leftarrow\!\!\$\ \mathbb{F}$:

$$e(C, h) = e(\pi_1, h^{z_1 - s_1}) \cdot e(\pi_2, h^{z_2 - s_2})^r \cdots e(\pi_n, h^{z_n - s_n})^{r^{n-1}}$$

However, this still requires $\mathcal{O}(n)$ pairing operations. Therefore, to efficiently prove n KZG commitments, we employ a Hadamard product argument Π_{HP}. In our setting, the n pairing equations are represented in a Hadamard product form $\mathbf{a} \circ \mathbf{b} = \mathbf{c} \circ \mathbf{d}$, utilizing only the commitments of each vector, allowing verification of all KZG commitments with just one pairing operation where $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$, $\mathbf{d}$ are defined as follows.

$$\begin{aligned}
\mathbf{a} &:= (C_1 - g^{e_1}, C_2 - g^{e_2}, \cdots, C_n - g^{e_n}) \in \mathbb{G}_1^n \\
\mathbf{b} &:= (h, h, \cdots, h) \in \mathbb{G}_2^n \\
\mathbf{c} &:= (\pi_1, \pi_2, \cdots, \pi_n) \in \mathbb{G}_1^n \\
\mathbf{d} &:= \left(h^{z_1 - s_1}, h^{z_2 - s_2}, \cdots, h^{z_n - s_n}\right) \in \mathbb{G}_2^n
\end{aligned}$$

Our goal is to verifier-efficiently compute commitments for each vector $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$, $\mathbf{d}$. Each vector is committed using commitment keys $\mathbf{ck_1}$ and $\mathbf{ck_2}$ along with AFGHO commitments. Specifically:

$$\begin{aligned}
&\mathsf{CM}(\mathbf{ck_2}; \mathbf{a}) = \langle \mathbf{a}, \mathbf{ck_2} \rangle,\ \mathsf{CM}(\mathbf{ck_1}; \mathbf{b}) = \langle \mathbf{ck_1}, \mathbf{b} \rangle, \\
&\mathsf{CM}(\mathbf{ck_2}; \mathbf{c}) = \langle \mathbf{c}, \mathbf{ck_2} \rangle,\ \mathsf{CM}(\mathbf{ck_1}; \mathbf{d}) = \langle \mathbf{ck_1}, \mathbf{d} \rangle
\end{aligned}$$

If the verifier computes commitments directly, each commitment requires n pairing operations. To reduce this, we utilize properties of pairings and the doubly homomorphic property of AFGHO commitments.

1. For $\mathsf{CM}(\mathbf{ck_2}; \mathbf{a})$, we have the reduced equation
$$\langle (C_1, C_2, \cdots, C_n), \mathbf{ck_2} \rangle - e(g, \prod ck_{2j}^{e_j}).$$
Thus, the verifier can compute $e(g, \prod ck_{2j}^{e_j})$ using 1 pairing operation and n group exponentiation operations, and then use the received $X = \langle (C_1, C_2, \cdots, C_n), \mathbf{ck_2} \rangle$ from the prover to compute $\mathsf{CM}(\mathbf{ck_2}; \mathbf{a})$.
2. Since h is public, the verifier can precompute the commitment $\mathsf{CM}(\mathbf{ck_1}; \mathbf{b}) = \langle \mathbf{ck_1}, (h, h, \cdots, h) \rangle$.
3. The commitment $\mathsf{CM}(\mathbf{ck_2}; \mathbf{c}) = \langle (\pi_1, \pi_2, \cdots, \pi_n), \mathbf{ck_2} \rangle$ serves as the commitment for the proof, which the verifier can obtain from the prover without any additional computation.
4. Lastly, $\mathsf{CM}(\mathbf{ck_1}; \mathbf{d})$ follows a similar reduced equation as $\mathsf{CM}(\mathbf{ck_2}; \mathbf{a})$:
$$\langle \mathbf{ck_1}, (h^{z_1}, h^{z_2}, \cdots, h^{z_n}) \rangle - e\left(\prod ck_{1j}^{s_j}, h\right)$$
In this equation, the first term can be precomputed by the verifier, and the second term can be computed with just one pairing operation. Therefore, the verifier can efficiently compute $\mathsf{CM}(\mathbf{ck_1}; \mathbf{d})$.

Prover and verifier execute the KZG Aggregator to obtain the relation $(D_1, D_2, D'_1, D'_2; \mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}) \in \mathcal{R}^{HP}_{n, \mathbf{ck_1}, \mathbf{ck_2}}$. This enables the use of Π_{HP}, and our KZG proof aggregation scheme $\mathsf{KZG.Agg}$ is completed by executing KZG Aggregator and Π_{HP}.

Algorithm 10 KZG Aggregator

```
Precompute: $D_2 = \langle \mathbf{ck_1}, (h, h, \cdots, h) \rangle$,
            $Y = \langle \mathbf{ck_1}, (h^{z_1}, h^{z_2}, \cdots, h^{z_n}) \rangle$

$\mathcal{P}$ : For all $i \in [n]$,
        $C_i \leftarrow \mathsf{cm}(f_i(x); \mathbf{pk_i})$
        and $\mathbf{cm} \leftarrow (C_1, C_2, \cdots, C_n)$
$\mathcal{P} \rightarrow \mathcal{V}$ : $X = \langle \mathbf{cm}, \mathbf{ck_2} \rangle$
$\mathcal{V} \rightarrow \mathcal{P}$ : $s_1, \cdots, s_n \leftarrow\!\!\$\ \mathbb{F}$
$\mathcal{P} \rightarrow \mathcal{V}$ : For all $i \in [n]$, $e_i \leftarrow f_i(s_i)$
$\mathcal{P}$ : For all $i \in [n]$,
        $\pi_i \leftarrow g^{\frac{f_i(z_i) - e_i}{z_i - s_i}}$ and $\boldsymbol{\pi} \leftarrow (\pi_1, \pi_2, \cdots, \pi_n)$
$\mathcal{P} \rightarrow \mathcal{V}$ : $D'_1 = \langle \boldsymbol{\pi}, \mathbf{ck_2} \rangle$
$\mathcal{V}$ : $\mathbf{e} \leftarrow (e_1, e_2, \cdots, e_n)$,  $\mathbf{s} \leftarrow (s_1, s_2, \cdots, s_n)$
$\mathcal{V}$ : $D_1 = X + e(g, \mathbf{ck_2}^{\mathbf{e}})$
    $D'_2 = Y + e(\mathbf{ck_1}^{\mathbf{s}}, h)$
Output: $(D_1, D_2, D'_1, D'_2; \mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d})$
```

Algorithm 11 KZG.Agg : KZG Proof Aggregation

```
$\mathcal{P}, \mathcal{V}$ : KZG Aggregator($\mathsf{srs}; \{f_i(x)\}$)
        $\rightarrow (D_1, D_2, D'_1, D'_2; \mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d})$
$\mathcal{P}, \mathcal{V}$ : $\Pi_{\mathsf{HP}}(D_1, D_2, D'_1, D'_2; \mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d})$
```

6.1 Analysis

Generally, n KZG aggregations require the verifier to perform $2n$ pairing operations, resulting in high verification costs. In this subsection, we compare and analyze the performance of KZG commitment aggregation schemes [12], [3], and our KZG.Agg to address this drawback.

Table 3 shows the opening performance of each protocol for n distinct polynomials $f_1(x), f_2(x), \cdots, f_n(x)$ and points $s_1, s_2, \cdots, s_n$ (we consider only cases where all polynomials have the same degree d). [12] proposed a batch KZG scheme that allows efficient verification for n polynomials $f_i(x)$. However, it is efficient only when the openings are for the same point s, and the verification cost increases with the number of distinct points s_i. In this case, the proof size is $\mathcal{O}(dn)$, and the verifier needs $\mathcal{O}(n)$ group multiplications. [3] improved on this by enabling efficient aggregation of openings for f_i and n distinct points s_i. [3] reduced the proof size to a constant and decreased the prover's cost from $\mathcal{O}(dn)$ to $\mathcal{O}(d)$, making it independent of the number of openings. Our aggregation scheme considers a more general case, supporting KZG opening aggregation where the polynomials, points, and srs are all distinct. Our KZG.Agg has a proof size of $\mathcal{O}(\log n)$ for the general case and can be verified with $\mathcal{O}(\log n)\mathbb{G}_T$ operations and only 3 pairing operations. Compared to the two aggregation schemes above, our results reduce the verifier's cost from $\mathcal{O}(n)\mathbb{G}_1$ to $\mathcal{O}(\log n)\mathbb{G}_T$. This means it is more efficient in terms of verification, requiring only 3 pairing operations, which is comparable to the operations needed by the previous schemes. Additionally, our scheme effectively supports KZG aggregation even when all srs are distinct, in which case the size of the srs increases proportionally to n as $\mathcal{O}(dn)$.

References

1. Abe, M., Fuchsbauer, G., Groth, J., Haralambiev, K., Ohkubo, M.: Structure-preserving signatures and commitments to group elements. In: Annual Cryptology Conference. pp. 209–236. Springer (2010)
2. Bayer, S., Groth, J.: Efficient zero-knowledge argument for correctness of a shuffle. In: Advances in Cryptology–EUROCRYPT 2012: 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings 31. pp. 263–280. Springer (2012)
3. Boneh, D., Drake, J., Fisch, B., Gabizon, A.: Efficient polynomial commitment schemes for multiple points and polynomials. Cryptology ePrint Archive (2020)
4. Boneh, D., Drijvers, M., Neven, G.: Compact multi-signatures for smaller blockchains. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 435–464. Springer (2018)
5. Bootle, J., Cerulli, A., Chaidos, P., Groth, J., Petit, C.: Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 327–357. Springer (2016)
6. Bootle, J., Chiesa, A., Groth, J.: Linear-time arguments with sublinear verification from tensor codes. In: Theory of Cryptography: 18th International Conference, TCC 2020, Durham, NC, USA, November 16–19, 2020. Springer (2020)
7. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: 2018 IEEE symposium on security and privacy (SP). pp. 315–334. IEEE (2018)
8. Bünz, B., Maller, M., Mishra, P., Tyagi, N., Vesely, P.: Proofs for inner pairing products and applications. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 65–97. Springer (2021)
9. Campanelli, M., Fiore, D., Querol, A.: Legosnark: Modular design and composition of succinct zero-knowledge proofs. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. pp. 2075–2092 (2019)
10. Daza, V., Ràfols, C., Zacharakis, A.: Updateable inner product argument with logarithmic verifier and applications. In: Public-Key Cryptography–PKC 2020: 23rd IACR International Conference on Practice and Theory of Public-Key Cryptography, Edinburgh, UK, May 4–7, 2020, Proceedings, Part I 23. pp. 527–557. Springer (2020)
11. Fauzi, P., Lipmaa, H., Zhang, B.: Efficient modular nizk arguments from shift and product. In: Cryptology and Network Security: 12th International Conference, CANS 2013, Paraty, Brazil. Springer (2013)
12. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive (2019)
13. Gailly, N., Maller, M., Nitulescu, A.: Snarkpack: practical snark aggregation. In: Financial Cryptography and Data Security: 26th International Conference, FC 2022, Grenada, May 2–6, 2022, Revised Selected Papers. Springer (2022)

14. Groth, J.: Linear algebra with sub-linear zero-knowledge arguments. In: Advances in Cryptology-CRYPTO 2009: 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings. Springer (2009)
15. Groth, J.: Short pairing-based non-interactive zero-knowledge arguments. In: Asiacrypt. vol. 6477, pp. 321–340. Springer (2010)
16. Groth, J.: On the size of pairing-based non-interactive arguments. In: Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35. pp. 305–326. Springer (2016)
17. Groth, J., Ishai, Y.: Sub-linear zero-knowledge argument for correctness of a shuffle. In: Advances in Cryptology–EUROCRYPT 2008: 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings 27. pp. 379–396. Springer (2008)
18. 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. pp. 177–194. Springer (2010)
19. Kothapalli, A., Masserova, E., Parno, B.: Poppins: a direct construction for asymptotically optimal zksnarks. Cryptology ePrint Archive (2020)
20. Lee, J.: Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. In: TCC 2021. Springer (2021)
21. Lipmaa, H.: Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In: Theory of Cryptography: 9th Theory of Cryptography Conference, TCC 2012, Taormina, Sicily, Italy, March 19-21, 2012. Springer (2012)
22. Lipmaa, H., Siim, J., Zajac, M.: Counting vampires: from univariate sumcheck to updatable zk-snark. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 249–278. Springer (2022)
23. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Annual international cryptology conference. Springer (1992)
24. Setty, S.: Spartan: Efficient and general-purpose zksnarks without trusted setup. In: Annual International Cryptology Conference. pp. 704–737. Springer (2020)
25. Wahby, R.S., Tzialla, I., Shelat, A., Thaler, J., Walfish, M.: Doubly-efficient zksnarks without trusted setup. In: 2018 IEEE Symposium on Security and Privacy (SP). pp. 926–943. IEEE (2018)
26. Xie, J., Hu, Y., Yu, Y.: Hadamard product argument from lagrange-based univariate polynomials. Cryptology ePrint Archive (2024)
27. Zhang, Y., Szepeniec, A., Zhang, R., Sun, S.F., Wang, G., Gu, D.: Voproof: efficient zksnarks from vector oracle compilers. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. pp. 3195–3208 (2022)

A Groth16: Non-interactive linear proofs for quadratic arithmetic programs

In this section, we introduce Groth16 [16], a pairing-based Non-Interactive Zero-Knowledge (NIZK) argument. Groth16 is a zk-SNARK for quadratic arithmetic programs, characterized by constant-sized proofs and proofs being group elements. A quadratic arithmetic program defines the subsequent binary relation, setting $a_0 = 1$:

$$\mathcal{R} = \left\{ (\mathbb{x}, \mathbb{w}) \;\middle|\; \begin{array}{l} \mathbb{x} = (a_1, a_2, \cdots, a_\ell) \\ \wedge\ \mathbb{w} = (a_{\ell+1}, a_{\ell+2}, \cdots, a_m) \\ \wedge\ \sum_{i=0}^{m} a_i u_i(x) \cdot \sum_{i=0}^{m} a_i v_i(x) \\ \quad \equiv \sum_{i=0}^{m} a_i w_i(x) \bmod t(x) \end{array} \right\}$$

```
Groth.Setup$(1^\lambda, \mathcal{R})$

$\alpha, \beta, \gamma, \delta \leftarrow\$ \mathbb{Z}_p^*, z \leftarrow\$ \mathbb{Z}_p^*,$
$\sigma_1 = \left(g^\alpha, g^\beta, g^\delta, \left\{g^{z^i}\right\}_{i=0}^{n-1}, \left\{g^{\frac{\beta u_i(z)+\alpha v_i(z)+w_i(z)}{\gamma}}\right\}_{i=0}^{\ell},\right.$
        $\left.\left\{g^{\frac{\beta u_i(z)+\alpha v_i(z)+w_i(z)}{\delta}}\right\}_{i=\ell+1}^{m}, \left\{g^{\frac{z^i t(z)}{\delta}}\right\}_{i=0}^{n-2}\right)$
$\sigma_2 = \left(h^\beta, h^\gamma, h^\delta, \left\{h^{z^i}\right\}_{i=0}^{n-1}\right)$
$\mathsf{crs} = (\mathsf{QAP}, (\sigma_1, \sigma_2))$
$\mathsf{vk} = \left(P = g^\alpha, Q = h^\beta, \left\{S_i = g^{\frac{\beta u_i(z)+\alpha v_i(z)+w_i(z)}{\gamma}}\right\}_{i=0}^{\ell}, H = h^\gamma, D = h^\delta\right)$
$\mathsf{td} = (z, \alpha, \beta, \gamma, \delta)$
return $(\mathsf{crs}, \mathsf{td})$
```

```
Groth.Prove$(\mathsf{crs}, \mathbb{x}, \mathbb{w})$

$\mathbb{x} = (a_1, \cdots, a_\ell), a_0 = 1$
$\mathbb{w} = (a_{\ell+1}, \cdots, a_m)$
$u(x) = \sum_{i=0}^{m} a_i u_i(x)$
$u_{\mathbb{w}}(x) = \sum_{i=\ell+1}^{m} a_i u_i(x)$
$v(x) = \sum_{i=0}^{m} a_i v_i(x)$
$v_{\mathbb{w}}(x) = \sum_{i=\ell+1}^{m} a_i v_i(x)$
$w(x) = \sum_{i=0}^{m} a_i w_i(x)$
$w_{\mathbb{w}}(x) = \sum_{i=\ell+1}^{m} a_i w_i(x)$
$h(x) = \dfrac{u(x)v(x) - w(x)}{t(x)}$
$r, s \leftarrow\$ \mathbb{Z}_p^*$
$f_{\mathbb{w}} = \dfrac{\beta u_{\mathbb{w}}(z) + \alpha v_{\mathbb{w}}(z) + w_{\mathbb{w}}(z)}{\delta}$
$a = \alpha + u(z) + r\delta$
$b = \beta + v(z) + s\delta$
$c = f_{\mathbb{w}} + \dfrac{t(z)h(z)}{\delta} + sa + rb - sr\delta$
$A = g^a, B = h^b, C = g^c$
return $\pi = (A, B, C)$
```

```
Groth.Verify$(\mathsf{vk}, \mathbb{x}, \pi)$

$\pi = (A, B, C)$
$u_{\mathbb{x}}(x) = \sum_{i=0}^{\ell} a_i u_i(x)$
$v_{\mathbb{x}}(x) = \sum_{i=0}^{\ell} a_i v_i(x)$
$w_{\mathbb{x}}(x) = \sum_{i=0}^{\ell} a_i w_i(x)$
$f_{\mathbb{x}} = \dfrac{\beta u_{\mathbb{x}}(z) + \alpha v_{\mathbb{x}}(z) + w_{\mathbb{x}}(z)}{\gamma}$
check: $e(A, B) = e(P, Q) \cdot e(g^{f_{\mathbb{x}}}, H) \cdot e(C, D)$
```

```
Groth.Sim$(\mathsf{td}, \mathbb{x})$

$a, b \leftarrow\$ \mathbb{Z}_p^*$
$A = g^a, B = h^b$
$c = \dfrac{ab - \alpha\beta - \beta u_{\mathbb{x}}(z) - \alpha v_{\mathbb{x}}(z) - w_{\mathbb{x}}(z)}{\delta}$
$C = g^c$
return $\pi = (A, B, C)$
```

Fig. 4: Groth16: a zk-SNARK for QAP

B KZG Polynomial Commitment Scheme

Let $\mathsf{para} = \big(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, h, z\big)$ be a system parameters.

```
$\mathsf{KZG.Setup}(\mathsf{para})$
    $\mathsf{srs} = (g^{z^i}, h^{z^i})$
    return $\mathsf{srs}$

$\mathsf{KZG.Commit}(\mathsf{srs}, f(x))$
    $C = g^{f(z)}$
    return $C$

$\mathsf{KZG.Open}(\mathsf{srs}, f(x), s)$
    $q(x) = \frac{f(x) - f(s)}{x - s}$
    $a = f(s)$
    $\pi = g^{q(z)}$
    return $(a, \pi)$

$\mathsf{KZG.Verify}(\mathsf{srs}, C, s, a, \pi)$
    check: $e(C - g^a, h) = e(\pi, h^{z-s})$
    return $1/0$
```

Fig. 5: KZG Polynomial Commitment Scheme

C Dory: An inner product argument with a logarithmic Verifier

In this section, we introduce Dory [20], an inner product arguments with sub-linear verification time. Dory exploits the properties of AFGHO commitments to achieve efficient inner product arguments with verification time reduced to $\mathcal{O}_\lambda(\log n)$.

Dory consists of two main sub-protocols. The first sub-protocol, the Reduce protocol, reduces the length of the given witness vectors $\mathbf{a} \in \mathbb{G}_1^n$ and $\mathbf{b} \in \mathbb{G}_2^n$ by half. The prover and verifier interactively reduce the length of the vectors using commitments. After a logarithmic number of iterations, the vector size becomes 1, and the Reduce step terminates. Subsequently, in the second sub-protocol, the Scalar-Product protocol, the verifier confirms correct computation through pairing operations.

We introduce Dory's Language, Reduce protocol, and Scalar-Product protocol sequentially.

C.1 Language for Dory

Definition 15. *The Language for Dory defined as follows:*

$$\begin{aligned}
&(C, D_1, D_2) \in \mathcal{L}_{n,\mathbf{ck_1},\mathbf{ck_2},H_1,H_2} \subset \mathbb{G}_T^3 \\
&\iff \exists(\mathbf{a} \in \mathbb{G}_1^n, \mathbf{b} \in \mathbb{G}_2^n, r_C, r_{D_1}, r_{D_2} \in \mathbb{F}) : \\
&\quad C = \langle \mathbf{a}, \mathbf{b} \rangle + r_C \cdot e(H_1, H_2), \\
&\quad D_1 = \langle \mathbf{a}, \mathbf{ck_2} \rangle + r_{D_1} \cdot e(H_1, H_2), \\
&\quad D_2 = \langle \mathbf{ck_1}, \mathbf{b} \rangle + r_{D_2} \cdot e(H_1, H_2)
\end{aligned}$$

C.2 Dory-Reduce

Algorithm 12 Dory-Reduce

```
**Precompute:** $H_T = e(H_1, H_2)$,  $\chi = \langle \mathbf{ck_1}, \mathbf{ck_2} \rangle$
        $\Delta_{1L} = \langle \mathbf{ck_1}_L, \mathbf{ck_2'} \rangle$, $\Delta_{1R} = \langle \mathbf{ck_1}_R, \mathbf{ck_2'} \rangle$,
        $\Delta_{2L} = \langle \mathbf{ck_1'}, \mathbf{ck_2}_L \rangle$, $\Delta_{2R} = \langle \mathbf{ck_1'}, \mathbf{ck_2}_R \rangle$,
**witness:** $(\mathbf{a}, \mathbf{b}, r_C, r_{D_1}, r_{D_2}) \in \mathcal{L}_{2^m,\mathbf{ck_1},\mathbf{ck_2},H_1,H_2}$
$\mathcal{P} : r_{D_{1L}}, r_{D_{1R}}, r_{D_{2L}}, r_{D_{2R}} \leftarrow\$ \mathbb{F}$
$\mathcal{P} \to \mathcal{V} : D_{1L} = \langle \mathbf{a}_L, \mathbf{ck_2'} \rangle + r_{D_{1L}} H_T$,
        $D_{1R} = \langle \mathbf{a}_R, \mathbf{ck_2'} \rangle + r_{D_{1R}} H_T$,
        $D_{2L} = \langle \mathbf{ck_1'}, \mathbf{b}_L \rangle + r_{D_{2L}} H_T$,
        $D_{2R} = \langle \mathbf{ck_1'}, \mathbf{b}_R \rangle + r_{D_{2R}} H_T$
$\mathcal{V} \to \mathcal{P} : \beta \leftarrow\$ \mathbb{F}$
$\mathcal{P} : \mathbf{a}^* \leftarrow \mathbf{a} + \beta \mathbf{ck_1}$,  $\mathbf{b}^* \leftarrow \mathbf{b} + \beta^{-1} \mathbf{ck_2}$,
    $r_C \leftarrow r_C + \beta r_{D_2} + \beta^{-1} r_{D_1}$
$\mathcal{P} : r_{C_+}, r_{C_-} \leftarrow\$ \mathbb{F}$
$\mathcal{P} \to \mathcal{V} : C_+ = \langle \mathbf{a}_L, \mathbf{b}_R \rangle + r_{C_+} H_T$,
        $C_- = \langle \mathbf{a}_R, \mathbf{b}_L \rangle + r_{C_-} H_T$
$\mathcal{V} \to \mathcal{P} : \alpha \leftarrow\$ \mathbb{F}$
$\mathcal{P} : \mathbf{a}' \leftarrow \alpha \mathbf{a}^*{}_L + \mathbf{a}^*{}_R$,  $\mathbf{b}' \leftarrow \alpha^{-1} \mathbf{b}^*{}_L + \mathbf{b}^*{}_R$,
    $r'_{D_1} \leftarrow \alpha r_{D_{1L}} + r_{D_{1R}}$,  $r'_{D_2} \leftarrow \alpha^{-1} r_{D_{2L}} + r_{D_{2R}}$,
    $r'_C \leftarrow r_C + \alpha r_{C_+} + \alpha^{-1} r_{C_-}$
$\mathcal{V} : C' \leftarrow C + \chi + \beta D_2 + \beta^{-1} D_1 + \alpha C_+ + \alpha^{-1} C_-$
    $D_1' \leftarrow \alpha D_{1L} + D_{1R} + \alpha\beta \Delta_{1L} + \beta \Delta_{1R}$
    $D_2' \leftarrow \alpha^{-1} D_{2L} + D_{2R} + \alpha^{-1}\beta^{-1} \Delta_{2L} + \beta^{-1} \Delta_{2R}$
$\mathcal{V}$ : Accept if $(C', D_1', D_2') \in \mathcal{L}_{2^{m-1},\mathbf{ck_1'},\mathbf{ck_2'},H_1,H_2}$
**witness:** $(\mathbf{a}', \mathbf{b}', r'_C, r'_{D_1}, r'_{D_2})$
```

C.3 Scalar-Product

Algorithm 13 Scalar-Product

```
**Precompute:** $H_T = e(H_1, H_2)$, $\chi = \langle \mathbf{ck_1}, \mathbf{ck_2} \rangle$
**witness:** $(a, b, r_C, r_{D_1}, r_{D_2}) \in \mathcal{L}_{1, ck_1, ck_2, H_1, H_2}$
$\mathcal{P} : r_{P_1}, r_{P_2}, r_Q, r_R \leftarrow\$ \mathbb{F}, d_1 \leftarrow\$ \mathbb{G}_1, d_2 \leftarrow\$ \mathbb{G}_2$
$\mathcal{P} \to \mathcal{V} : P_1 = e(d_1, ck_2) + r_{P_1} H_T,\ P_2 = e(ck_1, d_2) + r_{P_2} H_T,$
        $Q = e(d_1, b) + e(a, d_2) + r_Q H_T,$
        $R = e(d_1, d_2) + r_R H_T$
$\mathcal{V} \to \mathcal{P} : c \leftarrow\$ \mathbb{F}$
$\mathcal{P} \to \mathcal{V} : E_1 \leftarrow d_1 + ca,\ E_2 \leftarrow d_2 + cb,$
        $r_1 \leftarrow r_{P_1} + c r_{D_1},\ r_2 \leftarrow r_{P_2} + c r_{D_2},$
        $r_3 \leftarrow r_R + c r_Q + c^2 r_C$
$\mathcal{V} : d \leftarrow\$ \mathbb{F}$, accept if:
    $e(E_1 + d \cdot ck_1, E_2 + d^{-1} \cdot ck_2)$
        $= \chi + R + cQ + c^2 C$
        $+ dP_2 + cdD_2 + d^{-1} P_1 + cd^{-1} D_1$
        $- (r_3 + dr_2 + d^{-1} r_1) H_T$
```

D Zero-Knowledge Arguments

In this section, we introduce the zero-knowledge arguments of our protocols omitted in the main text. The blue color indicates the zero-knowledge. These include the relation for Half-Hadamard pairing product arguments, reformatting algorithm, final check algorithm, and batching scheme, presented in sequence. The remaining protocol, Π_{HP}, can be expressed as combinations of the zero-knowledge arguments introduced above, so they are omitted.

D.1 Relation for Hadamard Product Arguments with zero-knowledge

Definition 16. *Let $i = 1, 2$. For a given $\mathbf{ck_i} \leftarrow\$ \mathbb{G}_i^n$, $r \leftarrow\$ \mathbb{F}$, $H_i \leftarrow\$ \mathbb{G}_i$, the relation $\mathcal{R}^{Elim}_{r,n,\mathbf{ck_i},H_i}$ is defined by*

$$\left\{ \begin{array}{l|l} (D_1, D_2, D_1', D_2' \in \mathbb{G}_T \ ; & D_1 = \langle \mathbf{a}, \mathbf{ck_2} \rangle + r_{D_1} \cdot e(H_1, H_2), \\ \ \mathbf{a}, \mathbf{c} \in \mathbb{G}_1^n, \mathbf{b}, \mathbf{d} \in \mathbb{G}_2^n, & D_2 = \langle \mathbf{ck_1}, \mathbf{b} \rangle + r_{D_2} \cdot e(H_1, H_2), \\ \ r_{D_1}, r_{D_2}, r_{D_1'}, r_{D_2'} \in \mathbb{F}) & D_1' = \langle \mathbf{c}, \mathbf{ck_2} \rangle + r_{D_1'} \cdot e(H_1, H_2), \\ & D_2' = \langle \mathbf{ck_1}, \mathbf{d} \rangle + r_{D_2'} \cdot e(H_1, H_2) \\ & \mathbf{a} \circ \mathbf{b} = \mathbf{c} \circ \mathbf{d} \end{array} \right\}$$

D.2 Reformatting for Hadamard Product Arguments

Algorithm 14 Σ_{ref} with zero-knowledge

```
Input: $(D_1, D_2, D'_1, D'_2\ ;\ \mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d})$
$\mathcal{P} : r_C, r_X, r_K, r_{D_{t\in[4]}}, r_{C'}, r_{K'}, r_{D'_{t\in[4]}} \leftarrow\$\ \mathbb{F}$
$\mathcal{P} \to \mathcal{V} : C = \langle \mathbf{a}, \mathbf{b} \rangle + r_C \cdot e(H_1, H_2),$
            $C' = \langle \mathbf{c}, \mathbf{d} \rangle + r_{C'} \cdot e(H_1, H_2)$
$\mathcal{V} \to \mathcal{P} : r \leftarrow\$\ \mathbb{F}$
$\mathcal{P} : \mathbf{a_r} \leftarrow (1, r, r^2, \cdots, r^{n-1}) \circ \mathbf{a}, \quad \mathbf{k} \leftarrow \mathbf{1} \in \mathbb{G}_1^n,$
     $\mathbf{c_r} \leftarrow (1, r, r^2, \cdots, r^{n-1}) \circ \mathbf{c}, \quad \mathbf{k}' \leftarrow \mathbf{1} \in \mathbb{G}_1^n$
$\mathcal{P} \to \mathcal{V} : X = \langle \mathbf{a_r}, \mathbf{b} \rangle + r_X \cdot e(H_1, H_2),$
            $K = \langle \mathbf{k}, \mathbf{b} \rangle + r_K \cdot e(H_1, H_2),$
            $D_3 = \langle \mathbf{a_r}, \mathbf{ck_2} \rangle + r_{D_3} \cdot e(H_1, H_2),$
            $D_4 = \langle \mathbf{k}, \mathbf{ck_2} \rangle + r_{D_4} \cdot e(H_1, H_2),$
            $X' = \langle \mathbf{c_r}, \mathbf{d} \rangle + r_X \cdot e(H_1, H_2),$
            $K' = \langle \mathbf{k}', \mathbf{d} \rangle + r_{K'} \cdot e(H_1, H_2),$
            $D'_3 = \langle \mathbf{c_r}, \mathbf{ck_2} \rangle + r_{D'_3} \cdot e(H_1, H_2),$
            $D'_4 = \langle \mathbf{k}', \mathbf{ck_2} \rangle + r_{D'_4} \cdot e(H_1, H_2)$
Output: $(C, X, K, D_{t\in[4]}; \mathbf{a}, \mathbf{b}, \mathbf{a_r}, \mathbf{k}, r_C, r_X, r_K, r_{D_{t\in[4]}})$
and $(C', X', K', D'_{t\in[4]}; \mathbf{c}, \mathbf{d}, \mathbf{c_r}, \mathbf{k}', r_{C'}, r_X, r_{K'}, r_{D'_{t\in[4]}})$
```

D.3 Relation for Elimination Folding with zero-knowledge

Definition 17. *Let* $i = 1, 2$. *For a given* $\mathbf{ck_i} \leftarrow\$\ \mathbb{G}_i^n$, $r \leftarrow\$\ \mathbb{F}$, $H_i \leftarrow\$\ \mathbb{G}_i$, *the relation* $\mathcal{R}^{Elim}_{r,n,\mathbf{ck_i},H_i}$ *is defined by*

$$\left\{ \begin{array}{l|l} \begin{array}{c} (C, X, K, D_{t\in[4]} \in \mathbb{G}_T\ ; \\ \mathbf{a}, \mathbf{a_r}, \mathbf{k} \in \mathbb{G}_1^n, \mathbf{b} \in \mathbb{G}_2^n, \\ r_C, r_X, r_K, r_{D_{t\in[4]}} \in \mathbb{F}) \end{array} & \begin{array}{l} \mathbf{a_r} = (1, r, r^2, \cdots, r^{n-1}) \circ \mathbf{a}, \\ C = \langle \mathbf{a}, \mathbf{b} \rangle + r_C \cdot e(H_1, H_2), \\ X = \langle \mathbf{a_r}, \mathbf{b} \rangle + r_X \cdot e(H_1, H_2), \\ K = \langle \mathbf{k}, \mathbf{b} \rangle + r_K \cdot e(H_1, H_2), \\ D_1 = \langle \mathbf{a}, \mathbf{ck_2} \rangle + r_{D_1} \cdot e(H_1, H_2), \\ D_2 = \langle \mathbf{ck_1}, \mathbf{b} \rangle + r_{D_2} \cdot e(H_1, H_2), \\ D_3 = \langle \mathbf{a_r}, \mathbf{ck_2} \rangle + r_{D_3} \cdot e(H_1, H_2), \\ D_4 = \langle \mathbf{k}, \mathbf{ck_2} \rangle + r_{D_4} \cdot e(H_1, H_2) \end{array} \end{array} \right\}$$

D.4 Elimination Folding Algorithm with zero-knowledge

Algorithm 15 Σ_{elim} with zero-knowledge

```
Input: $(C, X, K, D_{t\in[4]}; \mathbf{a}, \mathbf{b}, \mathbf{a_r}, \mathbf{k} \ , r_C, r_X, r_K, r_{D_{t\in[4]}})$
Precompute : $H = e(H_1, H_2),\ \ \chi = \langle \mathbf{ck_1}, \mathbf{ck_2} \rangle$
             $\Delta_{1L} = \langle \mathbf{ck_1}_L, \mathbf{ck_2'} \rangle,\ \ \Delta_{2L} = \langle \mathbf{ck_1'}, \mathbf{ck_2}_L \rangle$
             $\Delta_{1R} = \langle \mathbf{ck_1}_R, \mathbf{ck_2'} \rangle,\ \ \Delta_{2R} = \langle \mathbf{ck_1'}, \mathbf{ck_2}_R \rangle$
$\mathcal{P} : r_{X_L}, r_{X_R}, r_{D_{3L}}, r_{D_{3R}}, r_{D'_{1L}}, r_{D'_{1R}},$
    $r_{D'_{2L}}, r_{D'_{2R}}, r_{D'_{3L}}, r_{D'_{3R}}, r_{D'_{4L}}, r_{D'_{4R}} \leftarrow\!\$\ \mathbb{F}$
$\mathcal{P} \rightarrow \mathcal{V} : X_L = \langle \mathbf{a_r}_L, \mathbf{b}_L \rangle + r_{X_L} H,$
          $X_R = \langle \mathbf{a_r}_R, \mathbf{b}_R \rangle + r_{X_R} H$
          $D_{3L} = \langle \mathbf{a_r}_L, \mathbf{ck_2}_L \rangle + r_{D_{3L}} H,$
          $D_{3R} = \langle \mathbf{a_r}_R, \mathbf{ck_2}_R \rangle + r_{D_{3R}} H$
          $D'_{1L} = \langle \mathbf{a}_L, \mathbf{ck_2'} \rangle + r_{D'_{1L}} H,$
          $D'_{1R} = \langle \mathbf{a}_R, \mathbf{ck_2'} \rangle + r_{D'_{1R}} H$
          $D'_{2L} = \langle \mathbf{ck_1'}, \mathbf{b}_L \rangle + r_{D'_{2L}} H,$
          $D'_{2R} = \langle \mathbf{ck_1'}, \mathbf{b}_R \rangle + r_{D'_{2R}} H$
          $D'_{3L} = \langle \mathbf{a_r}_L, \mathbf{ck_2'} \rangle + r_{D'_{3L}} H,$
          $D'_{3R} = \langle \mathbf{a_r}_R, \mathbf{ck_2'} \rangle + r_{D'_{3R}} H$
          $D'_{4L} = \langle \mathbf{k}_L, \mathbf{ck_2'} \rangle + r_{D'_{4L}} H,$
          $D'_{4R} = \langle \mathbf{k}_R, \mathbf{ck_2'} \rangle + r_{D'_{4R}} H$
$\mathcal{V} \rightarrow \mathcal{P} : \beta \leftarrow\!\$\ \mathbb{F}$
$\mathcal{P} : \mathbf{b}^* \leftarrow \mathbf{b} + \beta^{-1}\mathbf{ck_2},\ \ \mathbf{k}^* \leftarrow \mathbf{k} + \beta\mathbf{ck_1},\ \ r_K^* \leftarrow r_K + \beta r_{D_2} + \beta^{-1} r_{D_4}$
$\mathcal{P} : r_{C_\pm}, r_{X_\pm}, r_{K_\pm} \leftarrow\!\$\ \mathbb{F}$
$\mathcal{P} \rightarrow \mathcal{V} : C_+ = \langle \mathbf{a}_L, \mathbf{b}^*_R \rangle + r_{C_+} H,$
         $C_- = \langle \mathbf{a}_R, \mathbf{b}^*_L \rangle + r_{C_-} H$
         $X_+ = \langle \mathbf{a_r}_L, \mathbf{b}^*_R \rangle + r_{X_+} H,$
         $X_- = \langle \mathbf{a_r}_R, \mathbf{b}^*_L \rangle + r_{X_-} H$
         $K_+ = \langle \mathbf{k^*_L}, \mathbf{b}^*_R \rangle + r_{K_+} H,$
         $K_- = \langle \mathbf{k^*_R}, \mathbf{b}^*_L \rangle + r_{K_-} H$
$\mathcal{V} \rightarrow \mathcal{P} : \alpha \leftarrow\!\$\ \mathbb{F}$
$\mathcal{P} : \mathbf{a}' \leftarrow \alpha\mathbf{a}_L + \mathbf{a}_R,\ \ \mathbf{b}' \leftarrow \alpha^{-1}\mathbf{b}^*_L + \mathbf{b}^*_R$
    $\mathbf{a}'_\mathbf{r} \leftarrow \alpha\mathbf{a_r}_L + r^{-\frac{n}{2}}\mathbf{a_r}_R,\ \ \mathbf{k}' \leftarrow \alpha\mathbf{k}^*_L + \mathbf{k}^*_R$
    $r_{C'} \leftarrow r_C + \beta^{-1} r_{D_1} + \alpha r_{C_+} + \alpha^{-1} r_{C_-}$
    $r_{X'} \leftarrow r_{X_L} + r^{-\frac{n}{2}} r_{X_R} + \beta^{-1}(r_{D_{3L}} + r^{-\frac{n}{2}} r_{D_{3R}})$
    $+ \alpha r_{X_+} + \alpha^{-1} r^{-\frac{n}{2}} r_{X_-}$
    $r_{K'} \leftarrow r_K^* + \alpha r_{K_+} + \alpha^{-1} r_{K_-}$
    $r_{D'_1} \leftarrow \alpha r_{D'_{1L}} + r_{D'_{1R}},\ \ r_{D'_2} \leftarrow \alpha^{-1} r_{D'_{2L}} + r_{D'_{2R}}$
    $r_{D'_3} \leftarrow \alpha r_{D'_{3L}} + r^{-\frac{n}{2}} r_{D'_{3R}},\ \ r_{D'_4} \leftarrow \alpha^{-1} r_{D'_{4L}} + r_{D'_{4R}}$
$\mathcal{V} : X \overset{?}{=} X_L + X_R,\ \ D_3 \overset{?}{=} D_{3L} + D_{3R}$
    $C' \leftarrow C + \beta^{-1} D_1 + \alpha C_+ + \alpha^{-1} C_-$
    $X' \leftarrow X_L + r^{-\frac{n}{2}} X_R + \beta^{-1}(D_{3L} + r^{-\frac{n}{2}} D_{3R})$
        $+ \alpha X_+ + \alpha^{-1} r^{-\frac{n}{2}} X_-$
    $K' \leftarrow K + \chi + \beta D_2 + \beta^{-1} D_4 + \alpha K_+ + \alpha^{-1} K_-$
    $D'_1 \leftarrow \alpha D'_{1L} + D'_{1R}$
    $D'_2 \leftarrow \alpha^{-1} D'_{2L} + D'_{2R} + \alpha^{-1}\beta^{-1}\Delta_{2L} + \beta^{-1}\Delta_{2R}$
    $D'_3 \leftarrow \alpha D'_{3L} + r^{-\frac{n}{2}} D'_{3R}$
    $D'_4 \leftarrow \alpha D'_{4L} + D'_{4R} + \alpha\beta\Delta_{1L} + \beta\Delta_{1R}$
Output: $(C', X', K', D'_{t\in[4]}\ ;\ \mathbf{a}', \mathbf{b}', \mathbf{a}'_\mathbf{r}, \mathbf{k}', r_{C'}, r_{X'}, r_{K'}, r_{D'_{t\in[4]}})$
```

Theorem 6. *Let $i = 1, 2$. For $\mathbf{ck_i'} \leftarrow\$ \mathbb{G}_i^{n/2}$, $H_i \leftarrow\$ \mathbb{G}_i$, Σ_{elim} is a HVSZK, public-coin, succinct interactive argument of knowledge for $\mathcal{R}^{Elim}_{r,n/2,\mathbf{ck_1},\mathbf{ck_2},H_i}$ with $(9, 12/|\mathbb{F}|)$-tree extractability under SXDH.*

Revised:

Proof (Proof of HVSZK). The validity of HVSZK holds since all messages except X_L, X_R, D_{3L}, and D_{3R} from $\mathcal{P}$ to $\mathcal{V}$ consist of uniformly random elements of $\mathbb{G}_T$, and thus can be trivially simulated. The variables X_L, X_R, D_{3L}, and D_{3R} can also be simulated by initially sampling X_L and D_{3L} from $\mathbb{G}_T$, followed by computing $X_R \leftarrow X - X_L$ and $D_{3L} \leftarrow D_3 - D_{3R}$.

D.5 Pairing Check Algorithm with zero-knowledge

Algorithm 16 Σ_{check} with zero-knowledge

```
  Input: $(C, X, K, D_{t\in[4]}; a, b, a_r, k\ , r_C, r_X, r_K, r_{D_{t\in[4]}})$
  Precompute : $H = e(H_1, H_2),\ \ \chi = e(ck_1, ck_2)$
  $\mathcal{P} : r_{P_1}, r_{P_2}, r_{P_3}, r_{Q_1}, r_{Q_2}, r_{Q_3}, r_R \leftarrow\$ \mathbb{F},$
          $d_1 \leftarrow\$ \mathbb{G}_1,\ d_2 \leftarrow\$ \mathbb{G}_2$
  $\mathcal{P} \to \mathcal{V} : P_1 = e(d_1, ck_2) + r_{P_1} H,$
          $P_2 = e(ck_1, d_2) + r_{P_2} H,$
          $P_3 = e(k, d_2) + r_{P_3} H,$
          $Q_1 = e(d_1, b) + e(a, d_2) + r_{Q_1} H,$
          $Q_2 = e(a, d_2) + r_{Q_2} H,$
          $Q_3 = e(a_r, d_2) + r_{Q_3} H,\ R = e(d_1, d_2) + r_R H$
  $\mathcal{V} \to \mathcal{P} : c \leftarrow\$ \mathbb{F}$
  $\mathcal{P} \to \mathcal{V} : E_1 \leftarrow d_1 + ca + c^3 a + c^5 a_r + c^7 k,$
          $E_2 \leftarrow d_2 + cb,$
          $r_1 \leftarrow c r_{D_1} + c^3(1 + c^2) r_{D_3} + c^7 r_{D_4} + r_{P_1},$
          $r_2 \leftarrow c r_{D_2} + r_{P_2},$
          $r_3 \leftarrow c^2 r_C + c^4(1 + c^2) r_X + c^8 r_K$
              $+ c(r_{Q_1} + c^2 r_{Q_2} + c^4 r_{Q_3} + c^6 r_{P_3}) + r_R$
  $\mathcal{V} : d \leftarrow\$ \mathbb{F}$, accept if :
        $e(E_1 + d \cdot ck_1, E_2 + d^{-1} \cdot ck_2)$
          $= \chi + c^2 C + c^4(1 + c^2) X + c^8 K + c d^{-1} D_1$
            $+ c^3 d^{-1}(1 + c^2) D_3 + c^7 d^{-1} D_4$
            $+ c(Q_1 + c^2 Q_2 + c^4 Q_3 + c^6 P_3)$
            $+ d^{-1} P_1 + c d D_2 + d P_2$
            $+ R - (d^{-1} r_1 + d r_2 + r_3) H$
```

Theorem 7. *Let $i = 1, 2$. For $ck_i \leftarrow\$ \mathbb{G}_i$, $H_i \leftarrow\$ \mathbb{G}_i$, Σ_{check} is a HVSZK, public-coin, succinct interactive argument of knowledge for $\mathcal{R}^{Elim}_{r,1,ck_1,ck_2,H_1,H_2}$ with $(27, 27/|\mathbb{F}|)$-tree extractability under SXDH.*

Proof (Proof of HVSZK). To construct a simulator, the following steps are undertaken:

1. Sample $Q_1, Q_2, Q_3, E_1, E_2, P_3$ from $\mathbb{G}_T$.
2. Compute the challenge c using $\mathcal{V}$'s coins.
3. Sample r_1, r_2, r_3 from $\mathbb{F}$.
4. Compute P_1, P_2, R as follows: $P_1 = e(E_1, ck_2) - c(D_1 + c^2 D_3 + c^4 D_3 + c^6 D_4)$, $P_2 = e(ck_1, E_2) - cD_2$, and $R = e(E_1, E_2) - c(Q_1 + c^2 Q_2 + c^4 Q_3 + c^6 P_3) - c^2(C + c^2 X + c^4 X + c^6 K)$.

D.6 Batching Scheme with zero-knowledge

Algorithm 17 Batch-Σ_{elim}

```
**Input:** $(C_i, X_i, K_i, D_{i,t\in[4]}; \mathbf{a}_i, \mathbf{b}_i, \mathbf{a_r}_i, \mathbf{k}_i, r_{C_i}, r_{X_i}, r_{K_i}, r_{D_{i,t\in[4]}})_{i\in[2]}$
**Precompute** : $H = e(H_1, H_2)$
$\mathcal{P} : r_{Z_C}, r_{Z_X}, r_{Z_K} \leftarrow\$ \mathbb{F}$
$\mathcal{P} \to \mathcal{V} : Z_C = \langle \mathbf{a}_1, \mathbf{b}_2 \rangle + \langle \mathbf{a}_2, \mathbf{b}_1 \rangle + r_{Z_C} H,$
        $Z_X = \langle \mathbf{a_r}_1, \mathbf{b}_2 \rangle + \langle \mathbf{a_r}_2, \mathbf{b}_1 \rangle + r_{Z_X} H,$
        $Z_K = \langle \mathbf{k}_1, \mathbf{b}_2 \rangle + \langle \mathbf{k}_2, \mathbf{b}_1 \rangle + r_{Z_K} H$
$\mathcal{V} \to \mathcal{P} : \delta \leftarrow\$ \mathbb{F}$
$\mathcal{P} : \mathbf{a} \leftarrow \delta \mathbf{a}_1 + \mathbf{a}_2, \ \ \mathbf{b} \leftarrow \delta \mathbf{b}_1 + \mathbf{b}_2,$
    $\mathbf{a_r} \leftarrow \delta \mathbf{a_r}_1 + \mathbf{a_r}_2, \ \ \mathbf{k} \leftarrow \delta \mathbf{k}_1 + \mathbf{k}_2$
    $r_C \leftarrow \delta^2 r_{C_1} + \delta r_{Z_C} + r_{C_2},$
    $r_X \leftarrow \delta^2 r_{X_1} + \delta r_{Z_X} + r_{X_2},$
    $r_K \leftarrow \delta^2 r_{K_1} + \delta r_{Z_K} + r_{K_2}$
    $r_{D_{t\in[4]}} \leftarrow \delta r_{D_{1,t}} + r_{D_{2,t}}$
$\mathcal{V} : C \leftarrow \delta^2 C_1 + \delta Z_C + C_2,$
    $X \leftarrow \delta^2 X_1 + \delta Z_X + X_2,$
    $K \leftarrow \delta^2 K_1 + \delta Z_K + K_2,$
    $D_{t\in[4]} \leftarrow \delta D_{i,t} + D_{2,t}$
**Output:** $(C, X, K, D_{t\in[4]}; \mathbf{a}, \mathbf{b}, \mathbf{a_r}, \mathbf{k}, r_C, r_X, r_K, r_{D_{t\in[4]}})$
```

Theorem 8. *Let $i = 1, 2$. For $\mathbf{ck_i} \leftarrow\$ \mathbb{G}_i^n$, $H_i \leftarrow\$ \mathbb{G}_i$, Batch-$\Sigma_{\mathsf{elim}}$ is a HVSZK, public-coin, succinct interactive argument of knowledge for $(\mathcal{R}^{Elim}_{r,n,\mathbf{ck_1},\mathbf{ck_2},H_i})^2$ with $(3, 3/|\mathbb{F}|)$-tree extractability under SXDH.*

Proof. Since Succinctness, the Public Coin property, Completeness, Soundness and HVSZK of this protocol are immediate, It is enough to show tree extractability. We have $\mu = 1$ and set $w_1 = 3$. We are given witnesses for 3 distinct challenges δ. We interpolate $\mathbf{a}$, $\mathbf{b}$, $\mathbf{a_r}$, and $\mathbf{k}$ as quadratics in δ. Then from Lemma 6 in [20], the contribution of the quadratic terms to $D_{t\in[4]} = \delta D_{1,t} + D_{2,t}$ is identically zero, and so from Lemma 3 in [20] there are no quadratic terms. Hence

$\mathbf{a} = \delta\mathbf{a}_1 + \mathbf{a}_2$, $\mathbf{b} = \delta\mathbf{b}_1 + \mathbf{b}_2$, $\mathbf{a}_\mathbf{r} = \delta\mathbf{a}_{\mathbf{r}1} + \mathbf{a}_{\mathbf{r}2}$ and $\mathbf{k} = \delta\mathbf{k}_1 + \mathbf{k}_2$ for some $\mathbf{a}_i$, $\mathbf{b}_i$, $\mathbf{a}_{\mathbf{r}i}$, and $\mathbf{k}_i$. We define the following interpolations:

$$\begin{aligned} r_C(\delta) &= r_{C_2} + \delta r_{Z_C} + \delta^2 r_{C_1}, \\ r_X(\delta) &= r_{X_2} + \delta r_{Z_X} + \delta^2 r_{X_1}, \\ r_K(\delta) &= r_{K_2} + \delta r_{Z_K} + \delta^2 r_{K_1} \end{aligned}$$

Substituting in $\mathbf{a}$, $\mathbf{b}$, $\mathbf{a}_\mathbf{r}$, $\mathbf{k}$:

$$\begin{aligned} C(\delta) &= \delta^2 C_1 + \delta Z_C + C_2 = \langle \mathbf{a}(\delta), \mathbf{b}(\delta)\rangle + r_C(\delta)H \\ &= \delta^2[\langle \mathbf{a}_1, \mathbf{b}_1\rangle + r_{C_1}H] + \delta[\langle \mathbf{a}_1, \mathbf{b}_2\rangle + \langle \mathbf{a}_2, \mathbf{b}_1\rangle + r_{Z_C}H] \\ &\quad + [\langle \mathbf{a}_2, \mathbf{b}_2\rangle + r_{C_2}H], \\ X(\delta) &= \delta^2 X_1 + \delta Z_X + X_2 = \langle \mathbf{a}_\mathbf{r}(\delta), \mathbf{b}(\delta)\rangle + r_X(\delta)H \\ &= \delta^2[\langle \mathbf{a}_\mathbf{r}, \mathbf{b}\rangle + r_{X_1}H] + \delta[\langle \mathbf{a}_{\mathbf{r}1}, \mathbf{b}_2\rangle + \langle \mathbf{a}_{\mathbf{r}2}, \mathbf{b}_1\rangle + r_{Z_X}H] \\ &\quad + [\langle \mathbf{a}_{\mathbf{r}2}, \mathbf{b}_2\rangle + r_{X_2}H], \\ K(\delta) &= \delta^2 K_1 + \delta Z_K + K_2 = \langle \mathbf{k}(\delta), \mathbf{b}(\delta)\rangle + r_K(\delta)H \\ &= \delta^2[\langle \mathbf{k}_1, \mathbf{b}_2\rangle + r_{K_1}H] + \delta[\langle \mathbf{k}_1, \mathbf{b}_2\rangle + \langle \mathbf{k}_2, \mathbf{b}_1\rangle + r_{Z_K}H] \\ &\quad + [\langle \mathbf{k}_2, \mathbf{b}_2\rangle + r_{K_2}H]. \end{aligned}$$

As this holds for three values of δ, Lemma 6 in [20] implies that the two polynomials have identical coefficients, yielding:

$$\begin{aligned} C_i &= \langle \mathbf{a}_i, \mathbf{b}_i\rangle + r_{C_i}H, \\ X_i &= \langle \mathbf{a}_{\mathbf{r}i}, \mathbf{b}_i\rangle + r_{X_i}H, \\ K_i &= \langle \mathbf{k}_i, \mathbf{b}_i\rangle + r_{K_i}H \end{aligned}$$

Since $\mathbf{a}_{\mathbf{r}i} = \mathbf{r} \circ \mathbf{a}_i$, we deduce:

$$\mathbf{a}_\mathbf{r} = \delta\mathbf{a}_{\mathbf{r}1} + \mathbf{a}_{\mathbf{r}2} = \delta\mathbf{r} \circ \mathbf{a}_1 + \mathbf{r} \circ \mathbf{a}_2 = \mathbf{r} \circ (\delta\mathbf{a}_1 + \mathbf{a}_2) = \mathbf{r} \circ \mathbf{a}.$$

Thus, we have extracted the required witnesses.