Curdleproofs: A Shuffle Argument Protocol

The Ethereum Foundation Cryptography Research Team

September 14, 2022

Abstract

Curdleproofs is a zero-knowledge shuffle argument which is inspired by the work of Bayer and Groth [BG12]. Curdleproofs has applications to secret leader elections which prevents DDOS attacks on the Ethereum Proof of Stake consensus layer. Curdleproofs runs over a public coin setup in any group where the DDH assumption holds.
Curdleproofs is built from well established inner product arguments and does not need a trusted setup. The prover and verifier both run in linear time asymptotically with small constants because there is no reduction to NP constraints. Their concrete run time is highly practical: shuffling 252 elements requires 0.5 seconds for the prover and 25 milliseconds for the verifier on an Intel i7-8550U CPU at 1.80GHz over the BLS12-381 curve. The proof size is logarithmic size (dominated by $10\log(\ell)$ for ℓ the number of elements).

Contents

Chapter 1

Introduction

In Ethereum proof of stake, single secret leader elections (SSLE) [BEHG20, AC21] have been proposed as a privacy-preserving method for electing block proposers on the Ethereum beacon chain. The beacon chain currently elects the next 32 block proposers at the beginning of each epoch. The results of this election are public and everyone gets to learn the identity of those future block proposers. This information leak enables attackers to launch DoS attacks against each proposer sequentially in an attempt to disable Ethereum.

The proposal Whisk is a practical SSLE scheme that uses a shuffle argument as a backend. In Whisk the beacon chain first randomly picks a set of election candidates. Then and for an entire day, block proposers continuously shuffle that candidate list thousands of times. After the shuffling is finished, we use the final order of the candidate list to determine the future block proposers for the following day. Due to all that shuffling, only the candidates themselves know which position they have in the final shuffled set. For the shuffling process to work, Whisk does not shuffle the validators themselves, but cryptographic randomizable commitments that correspond to them. Election winners can open their commitments to prove that they won the elections.

Verifiable shuffling has been a research topic for decades due to its application in mixnets and hence to online anonymity and digital election schemes [Cha03, DK00]. Already since twenty years ago, zero-knowledge proofs based on randomizable El-Gamal ciphertexts have been proposed, and in recent years we've seen proofs based on pairings [CFG21, FLSZ17] as well as post-quantum proofs based on lattices [CMM19, HMS21, ABG+21].

In Whisk we use shuffling ZKPs that solely rely on the discrete logarithm assumption and don't require a trusted setup while still maintaining decent proving and verification performance. In this document we specify the zero-knowledge proving system Curdleproofs used by Whisk. The scheme is a modernisation of the Bayer-Groth shuffle argument [BG12] that makes use of inner product arguments as a backend.

1.1 Preliminaries

To denote a relation R_{rel} where a public instance ϕ and a private witness w is in R_{rel} if and only if certain properties hold, we write

$$R_{\mathsf{rel}} = \{\ \phi;\quad w \mid \text{ properties that } \phi \text{ and } w \text{ satisfy } \}.$$

Proving algorithms $\mathsf{Prove}_{\mathsf{rel}}$ take as input $(\mathsf{crs}_{\mathsf{rel}}, \phi, w)$ where $\mathsf{crs}_{\mathsf{rel}}$ is a common reference string.

They return a proof π_{rel}. Verification algorithms $\mathsf{Verify}_{\mathsf{rel}}$ take as input $(\mathsf{crs}_{\mathsf{rel}}, \phi, \pi_{\mathsf{rel}})$ where $\mathsf{crs}_{\mathsf{rel}}$ is a common reference string, ϕ is an instance the prover is claiming to be in the language, and π_{rel} is a proof. They return a bit 1 to indicate acceptance and 0 to indicate rejection.

We use bold font $\mathbf{a}$ to denote a vector with coordinates $a_1, a_2, \ldots, a_n$ and $\times$ is the dot product of vectors. We write $k\boldsymbol{a}$ to denote the vector $(ka_1, ka_2, \ldots, ka_n)$. We write $(\boldsymbol{a} \mid\mid \boldsymbol{b})$ to denote the concatenated vector $(a_1, \ldots, a_n, b_1, \ldots, b_n)$. We write $\boldsymbol{k} \circ \boldsymbol{a}$ to denote the vector $(k_1 a_1, \ldots, k_n a_n)$.

1.1.1 Public Coin Setup

Curdleproofs is built over a cryptographic group $\mathbb{G}$ in which both the discrete logarithm problem and the decisional Diffie-Hellman problem is hard. Curdleproofs requires a common reference string (crs) consisting of ℓ random group elements $\boldsymbol{g}$. This allows us to generate Pedersen commitments V to vectors of scalars $\boldsymbol{v}$ of the form

$$V = \mathsf{Commit}(\mathsf{crs};\ \boldsymbol{v}) = \boldsymbol{v} \times \boldsymbol{g}$$

We are shuffling ℓ-tuples of group elements. In order to mask the resulting positions of these group elements we require that our Pedersen commitments use blinders. Specifically we need up to $n_{\mathsf{bl}} = 4$ blinders in each commitment. We thus include n_{bl} additional random group elements $\boldsymbol{h}$ such that blinded commitments V have the form

$$V = \mathsf{Commit}(\mathsf{crs};\ \boldsymbol{v};\ \boldsymbol{r}_V) = \boldsymbol{v} \times \boldsymbol{g} + \boldsymbol{r}_V \times \boldsymbol{h}$$

with random $\boldsymbol{r}_V$. We additionally require a random group elements G_T, G_U, H that are used for committing a group element

$$\mathsf{GroupCommit}((G_T, H);\ T;\ r_T) = \mathsf{cm}_T = (\mathsf{cm}_{T,1}, \mathsf{cm}_{T,2}) = (r_T G_T,\ T + r_T H)$$

This group commitment scheme is statistically binding and hiding under the DDH assumption. It is also equipped with a homomorphism such that

$$\begin{aligned}\mathsf{GroupCommit}((G_T, H);\ A;\ r_A) + \mathsf{GroupCommit}((G_T, H);\ B;\ r_B)& \\ = \mathsf{GroupCommit}((G_T, H);\ A + B;\ r_A + r_B)& \\ = ((r_A + r_B)G_T,\ (A + B) + (r_A + r_B)H)&\end{aligned}$$

It is based of the El-Gamal encryption scheme.

Chapter 2

Full Curdleproofs Construction

2.1 Problem Statement

The aim of Curdleproofs is to build a shuffle argument that preserves discrete logarithm relations between pairs of group elements. More precisely, given a public set of 2ℓ group elements

$$\boldsymbol{R} = (R_1, \ldots, R_\ell) \text{ and } \boldsymbol{S} = (S_1, \ldots, S_\ell)$$

a shuffler computes a second set of group elements

$$\boldsymbol{T} = (T_1, \ldots, T_\ell) \text{ and } \boldsymbol{U} = (U_1, \ldots, U_\ell)$$

and proves in zero knowledge that there exists a permutation

$$\sigma() : [1, \ell] \mapsto [1, \ell]$$

and a field element $k \in \mathbb{F}$ such that for all $1 \leqslant i \leqslant \ell$

$$T_i = kR_{\sigma(i)} \ \wedge \ U_i = kS_{\sigma(i)} \ .$$

We use the same scalar k for each i. The permutation $\sigma()$ is committed to in $M \in \mathbb{G}$ under some randomness $\boldsymbol{r}_M \in \mathbb{F}^{n_{\mathsf{bl}}}$

$$M = \sigma([1, \ell]) \times \boldsymbol{g} + \boldsymbol{r}_M \times \boldsymbol{h}$$

Note that by the ℓ-ddh assumption (Assumption 7.0.1) it is difficult to distinguish the randomised ciphertexts from truly random values.

In other words we define a zero-knowledge proof for the relation

$$R_{\mathsf{shuffle}} = \left\{ \begin{array}{lll|l} (\boldsymbol{R}, \boldsymbol{S}, \boldsymbol{T}, \boldsymbol{U}, M); & \sigma() & & \boldsymbol{T} = \sigma(k\boldsymbol{R}) \\ & k \in \mathbb{F}/\{0\} & & \boldsymbol{U} = \sigma(k\boldsymbol{S}) \\ & \boldsymbol{r}_M \in \mathbb{F}^{n_{\mathsf{bl}}} & & M = \sigma(1, \ldots, \ell) \times \boldsymbol{g} + \boldsymbol{r}_M \times \boldsymbol{h} \\ & & & \sigma() \in \text{permutations over } [1, \ldots, \ell] \end{array} \right\}$$

To do this we make use of a permutation argument by Bayer and Groth [BG12] which we modify to make use of more recent work on inner product arguments. All modifications are formally justified. If any mistakes are spotted please file an issue on the github repo.

2.2 Curdleproofs Construction

We begin by giving a full overview of the construction. For an informal overview see Figure 2.1 and for the formal construction see Figures 2.2 and 2.3. The security arguments are deferred to Theorems 2.2.1 and 2.2.2. The construction makes use of three subprotocols: a SameScalar, SameMultiScalar, and SamePerm arguments. We specify the relations for these subprotocols in Section 2.2.2 below but we defer discussing their proving and verifying algorithms to Chapter 3, 4, & 5.

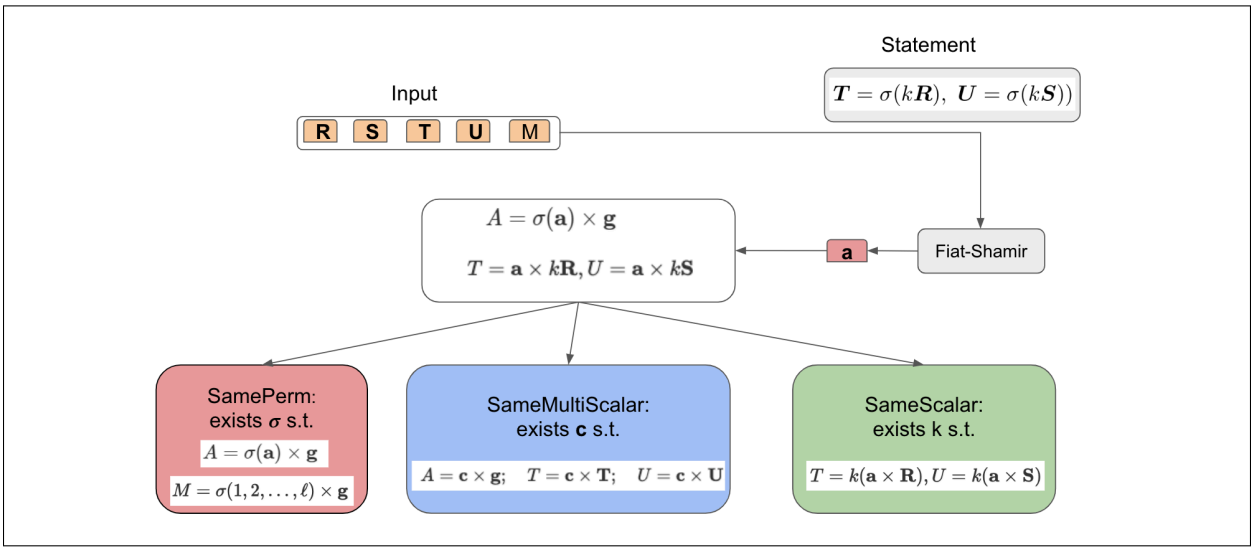

Figure 2.1: Overview of the shuffle argument. There are three subprotocols: SamePerm argument, SameMultiScalar argument, and SameScalar argument.

2.2.1 Informal Overview

Let $\ell > 1$. The prover will take as input the $\boldsymbol{R}, \boldsymbol{S}, \boldsymbol{T}, \boldsymbol{U}, M$ and aims to prove knowledge of $\sigma(), k$ such that:

- $M = \sigma(1, 2, \ldots, \ell) \times \boldsymbol{g}$ is a commitment to the permutation $\sigma()$
- $\boldsymbol{T} = \sigma(k\boldsymbol{R})$ is a randomised permutation of $\boldsymbol{R}$
- $\boldsymbol{U} = \sigma(k\boldsymbol{S})$ is a randomised permutation of $\boldsymbol{S}$

Initially all the public inputs are hashed to get a vector $\boldsymbol{a}$ of challenges. Then the prover computes values $A = \sigma(\boldsymbol{a}) \times \boldsymbol{g}$, $T = \boldsymbol{a} \times k\boldsymbol{R}$, and $U = \boldsymbol{a} \times k\boldsymbol{S}$ which it sends to the verifier. As part of our full construction we require zero-knowledge algorithms for proving and verifying three additional relations: a same permutation relation, a same scalar relation, and a same multiscalar relation.

The prover runs

- SamePerm argument to demonstrate that A is a commitment to $\sigma(\boldsymbol{a})$ for $\sigma()$ the permutation committed to with M.

- SameMultiScalar argument to show knowledge of some $\boldsymbol{x}$ such that $A = \boldsymbol{x} \times \boldsymbol{g}$, $T = \boldsymbol{x} \times \boldsymbol{T}$ and $U = \boldsymbol{x} \times \boldsymbol{U}$. Given that $A = \sigma(\boldsymbol{a}) \times \boldsymbol{g} = \boldsymbol{x} \times \boldsymbol{g}$ this implies that $T = \sigma(\boldsymbol{a}) \times \boldsymbol{T}$ and $U = \sigma(\boldsymbol{a}) \times \boldsymbol{U}$.
- SameScalar argument to show the existence of k such that $T = k(\boldsymbol{a} \times \boldsymbol{R})$ and $U = k(\boldsymbol{a} \times \boldsymbol{S})$.

Together this means that

$$T = k(\boldsymbol{a} \times \boldsymbol{R}) = \sigma(\boldsymbol{a}) \times \boldsymbol{T} \text{ and } U = \sigma(\boldsymbol{a}) \times \boldsymbol{U} = k(\boldsymbol{a} \times \boldsymbol{S})$$

Where $\boldsymbol{a}$ is random this means that $kR_{\sigma(i)} = T_i$ for all i except with negligible probability.

Note that the full protocol has some additional masking values that are included to ensure zero-knowledge. For simplicity we have ignored these terms in this overview.

2.2.2 Relations

SamePerm Relation

SamePerm relation demonstrates that given public input $(A, M) \in \mathbb{G}^2$ there exists $\sigma(1, \ldots, \ell)$ such that M is a commitment to $\sigma()$ and A is a commitment to $\boldsymbol{a}$. These commitments are blinded. In other words

$$R_{\mathsf{sameperm}} = \left\{ \begin{array}{ll} (A, M, \boldsymbol{a}); & (\sigma(), \boldsymbol{r}_A, \boldsymbol{r}_M) \end{array} \middle| \begin{array}{l} A = \sigma(\boldsymbol{a}) \times \boldsymbol{g} + \boldsymbol{r}_A \times \boldsymbol{h} \\ M = \sigma(1, \ldots, \ell) \times \boldsymbol{g} + \boldsymbol{r}_M \times \boldsymbol{h} \\ \sigma() \in \text{permutations over } [1, \ldots, \ell] \end{array} \right\}$$

SameScalar Relation

SameScalar relation demonstrates that given public input $(R, S, \mathsf{cm}_T, \mathsf{cm}_U)$ there exists k such that cm_T is a commitment to $T = kR$ and cm_U is a commitment to kS. In other words

$$R_{\mathsf{samescalar}} = \left\{ \begin{array}{ll} (R, S, \mathsf{cm}_T, \mathsf{cm}_U); & (k, r_U, r_T) \end{array} \middle| \begin{array}{l} \mathsf{cm}_T = \mathsf{GroupCommit}((G_T, H);\ kR;\ r_T) \\ \mathsf{cm}_U = \mathsf{GroupCommit}((G_U, H);\ kS;\ r_U) \end{array} \right\}$$

where G_U, G_T and H are fixed group elements.

SameMultiScalar Relation

SameMultiScalar relation demonstrates that two group elements $(T, U) \in \mathbb{G}^2$ are bound together by the same vector $\boldsymbol{x} \in \mathbb{F}^n$ under the bases $\boldsymbol{T} \in \mathbb{G}^n$ and $\boldsymbol{U} \in \mathbb{G}^n$. The vector $\boldsymbol{x}$ is contained in a commitment $A \in \mathbb{G}$ that is computed under a binding commitment key $\boldsymbol{G} \in \mathbb{G}^n$ which is not chosen by the prover. In other words

$$R_{\mathsf{samemsm}} = \left\{ \begin{array}{ll} (A, Z_T, Z_U, \boldsymbol{T}, \boldsymbol{U}); & \boldsymbol{x} \end{array} \middle| \begin{array}{l} A = \boldsymbol{x} \times \boldsymbol{G} \\ Z_T = \boldsymbol{x} \times \boldsymbol{T} \\ Z_U = \boldsymbol{x} \times \boldsymbol{U} \end{array} \right\}.$$

2.2.3 Full Zero Knowledge Construction

A formal description of Curdleproofs is provided in Figures 2.2 and 2.3. Here we describe the additional steps that we have added compared to the informal overview in Section 2.2.1 to achieve zero-knowledge. We defer the security proofs of zero-knowledge, and soundness to Section 2.2.4, Theorems 2.2.2 and 2.2.1.

Step 1: In the first step the prover and verifier both hash the instance to get a random vector of field elements $\boldsymbol{a} \in \mathbb{F}^{\ell}$. There are no secrets involved in this step. The verifier parses all inputs to check that they are group or field elements. When parsing $\boldsymbol{T}$ the verifier checks that $T_1 \neq 0$ to prevent the prover from choosing $k = 0$.

Step 2: In the second step the prover computes a commitment A to the permuted $\sigma(\boldsymbol{a})$. The vector $\sigma(\boldsymbol{a})$ is private because it reveals information about the secret permutation $\sigma()$. The prover therefore chooses a random blinding vector $\boldsymbol{r}_A \in \mathbb{F}^{n_{\mathsf{bl}}-2}$. Looking ahead, the same-permutation argument is only zero-knowledge provided $|\boldsymbol{r}_A| \geqslant 2$, thus we choose $n_{\mathsf{bl}} \geqslant 4$.

The prover outputs A together with a proof π_{sameperm} demonstrating that A is a blinded commitment to $\sigma(\boldsymbol{a})$ for $\sigma()$ committed to in the blinded commitment M. The verifier simply checks that this proof verifies.

Step 3: In the third step, the prover computes $R = \boldsymbol{a} \times \boldsymbol{R}$ and $S = \boldsymbol{a} \times \boldsymbol{S}$ and the verifier checks that R and S have been computed correctly. Note that due to the optimisations in **??** it is faster for the verifier to check correctness of R and S than it is to compute them itself. The prover then computes commitments $\mathsf{cm}_T = (r_T G_T, T + r_T H)$, $\mathsf{cm}_U = (r_U G_U, U + r_U H)$ to $T = kR$ and $U = kS$ respectively. The commitments are blinded with the masking values r_T and r_U. The prover then outputs $\mathsf{cm}_T, \mathsf{cm}_U$ together with a proof $\pi_{\mathsf{samescalar}}$ demonstrating that cm_T and cm_U open to (T, U) such that $T = kR$ and $U = kS$ for the same scalar k.

Step 4: In the fourth and final step, The prover and verifier first extends $A' = A + r_T G_T + r_U G_U$ such that A' includes the blinders r_T and r_U. They also extend the basis $\boldsymbol{G}$ such that A' is a commitment to $\boldsymbol{x} = (\sigma(\boldsymbol{a} \mid\mid \boldsymbol{r}_A \mid\mid r_T \mid\mid r_U))$ under the basis $\boldsymbol{G}$. Now if $\boldsymbol{T}' = (\boldsymbol{T} \mid\mid \boldsymbol{0} \mid\mid H \mid\mid 0)$ for $\boldsymbol{0}$ a vector of length $n_{\mathsf{bl}} - 2$ with every element equal to the identity element then

$$\mathsf{cm}_{T,2} = kR + r_T H = \boldsymbol{x} \times \boldsymbol{T}' = \sigma(\boldsymbol{a}) \times \boldsymbol{T} + r_T H$$

then we have that $kR = \sigma(\boldsymbol{a}) \times \boldsymbol{T}$ as required. A similar argument shows that $kS = \sigma(\boldsymbol{a}) \times \boldsymbol{U}$. Thus the prover outputs a proof π_{samemsm} demonstrating that cm_T and cm_U contain $\sigma(\boldsymbol{a}) \times \boldsymbol{T}$ and $\sigma(\boldsymbol{a}) \times \boldsymbol{U}$ respectively.

Outcome: The prover returns the proof

$$\pi_{\mathsf{shuffle}} = (A, \mathsf{cm}_T, \mathsf{cm}_U, R, S, \pi_{\mathsf{sameperm}}, \pi_{\mathsf{samescalar}}, \pi_{\mathsf{samemsm}}).$$

The verifier returns 1 if and only if all checks pass.

2.2.4 Security

We first prove the zero-knowledge of our arguments, i.e. a verifier learns no information from an honest proof except for the truth of the statement. The SamePerm and SameScalar are uncondi-

tionally zero-knowledge. The SameMultiScalar argument is zero-knowledge assuming that we set n_{bl} such that $n_{\mathsf{bl}} \geqslant 4$ and $n_{\mathsf{bl}} + \ell \geqslant 6$.

Theorem 2.2.1 (Shuffle argument is zero-knowledge)**.** *If SamePerm argument, SameScalar argument, and SameMultiScalar argument are zero-knowledge then the shuffle argument described in Figures 2.2 and 2.3 is zero-knowledge.*

Proof. We design a simulator Simulate that takes as input an instance

$$(\boldsymbol{R}, \boldsymbol{S}, \boldsymbol{T}, \boldsymbol{U}, M)$$

and outputs a proof π_{shuffle} that is indistinguishable from a proof generated by an honest prover that knows the witness. The simulator can program the random oracle on any points yet to be determined by the adversary.

During the setup the simulator chooses the $\mathsf{crs}_{\mathsf{shuffle}} = (\boldsymbol{g}, \boldsymbol{h}, G_T, G_U, H)$ uniformly at random. During proving the simulator Simulate proceeds as follows

1. In Step 1 they hash to find $a_1, \ldots, a_\ell$ identically to the honest prover.
2. In Step 2 they choose $A \xleftarrow{\$} \mathbb{G}$ uniformly at random and run
$$\pi_{\mathsf{sameperm}} = \mathsf{Simulate}_{\mathsf{sameperm}}((\boldsymbol{g}, \boldsymbol{h}, H);\ (A, M, \boldsymbol{a}))$$
3. In Step 3 they generate $R = \boldsymbol{a} \times \boldsymbol{R}$ and $S = \boldsymbol{a} \times \boldsymbol{S}$ identically to the honest prover. They sample $\mathsf{cm}_T, \mathsf{cm}_U \xleftarrow{\$} \mathbb{G}^4$ uniformly at random and run
$$\pi_{\mathsf{samescalar}} = \mathsf{Simulate}_{\mathsf{samescalar}}((G_T, G_U, H);\ (R, S, \mathsf{cm}_T, \mathsf{cm}_U))$$
4. In Step 3 they set $A' \leftarrow A + \mathsf{cm}_{T,1} + \mathsf{cm}_{U,1}$, $\boldsymbol{G} \leftarrow (\boldsymbol{g} \mid\mid \boldsymbol{h} \mid\mid G_T \mid\mid G_U)$, $\boldsymbol{T}' \leftarrow (\boldsymbol{T} \mid\mid \boldsymbol{0} \mid\mid H \mid\mid 0)$, $\boldsymbol{U}' \leftarrow (\boldsymbol{U} \mid\mid \boldsymbol{0} \mid\mid 0 \mid\mid H)$, identically to the honest prover and run
$$\pi_{\mathsf{samemsm}} = \mathsf{Simulate}_{\mathsf{samemsm}}(\boldsymbol{G};\ (A', \mathsf{cm}_{T,2}, \mathsf{cm}_{U,2}, \boldsymbol{T}', \boldsymbol{U}'))$$
Finally they return $\pi_{\mathsf{shuffle}} = (\pi_{\mathsf{sameperm}}, \pi_{\mathsf{samescalar}}, \pi_{\mathsf{samemsm}})$.

Now we must argue that the simulated proof is indistinguishable from the real proof.

We design an adversaries $\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3, \mathcal{B}_4, \mathcal{B}_5$ such that

$$\mathsf{Adv}^{\mathsf{shuffle}}_{\mathcal{A}}(\lambda) \leqslant 2(\mathsf{Adv}^{\mathsf{sameperm}}_{\mathcal{B}_1}(\lambda) + \mathsf{Adv}^{\mathsf{samescalar}}_{\mathcal{B}_2}(\lambda) + \mathsf{Adv}^{\mathsf{samemsm}}_{\mathcal{B}_3}(\lambda) + \mathsf{Adv}^{\mathsf{ddh}}_{\mathcal{B}_4}(\lambda) + \mathsf{Adv}^{\mathsf{ddh}}_{\mathcal{B}_5}(\lambda))$$

We proceed via a series of games $\mathsf{Game}_1, \mathsf{Game}_2, \mathsf{Game}_3, \mathsf{Game}_4, \mathsf{Game}_5$ such that

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{shuffle}}_{\mathcal{A}}(\lambda) &\leqslant 2\mathsf{Adv}^{\mathsf{sameperm}}_{\mathcal{B}_1}(\lambda) + \mathsf{Adv}^{\mathsf{Game}_1}_{\mathcal{A}}(\lambda) \\
\mathsf{Adv}^{\mathsf{Game}_1}_{\mathcal{A}}(\lambda) &\leqslant 2\mathsf{Adv}^{\mathsf{samescalar}}_{\mathcal{B}_2}(\lambda) + \mathsf{Adv}^{\mathsf{Game}_2}_{\mathcal{A}}(\lambda) \\
\mathsf{Adv}^{\mathsf{Game}_2}_{\mathcal{A}}(\lambda) &\leqslant 2\mathsf{Adv}^{\mathsf{samemsm}}_{\mathcal{B}_3}(\lambda) + \mathsf{Adv}^{\mathsf{Game}_3}_{\mathcal{A}}(\lambda) \\
\mathsf{Adv}^{\mathsf{Game}_3}_{\mathcal{A}}(\lambda) &\leqslant 2\mathsf{Adv}^{\mathsf{ddh}}_{\mathcal{B}_4}(\lambda) + \mathsf{Adv}^{\mathsf{Game}_4}_{\mathcal{A}}(\lambda) \\
\mathsf{Adv}^{\mathsf{Game}_4}_{\mathcal{A}}(\lambda) &\leqslant 2\mathsf{Adv}^{\mathsf{ddh}}_{\mathcal{B}_5}(\lambda) + \mathsf{Adv}^{\mathsf{Game}_5}_{\mathcal{A}}(\lambda) \\
\mathsf{Adv}^{\mathsf{Game}_5}_{\mathcal{A}}(\lambda) &= 0
\end{aligned}$$

which combined give us our final result.

$\underline{\mathsf{Game}_0 \mapsto \mathsf{Game}_1}$: Let Game_0 be the initial zero-knowledge game. Define Game_1 to run identicallly to Game_0 except that, the crs and the π_{sameperm} proof are generated by the $\mathsf{sameperm}$ simulator for both $b = 0$ and $b = 1$.

Let $\mathcal{B}_1$ be an adversary against the $\mathsf{sameperm}$ zero-knowledge game. Then $\mathcal{B}_1$ simulates the zero-knowledge game for $\mathcal{A}$. It takes as input $(\boldsymbol{g}, \boldsymbol{h}, H)$, generates the remaining terms (G_T, G_U) randomly, and runs $\mathcal{A}(\mathsf{crs})$. It flips a coin to get $b \in \{0, 1\}$. When $\mathcal{A}$ makes a valid prover query $(\boldsymbol{R}, \boldsymbol{S}, \boldsymbol{T}, \boldsymbol{U}, M), (\sigma(), k, \boldsymbol{r}_M)$, if $b = 0$ then $\mathcal{B}_1$ generates $A, \boldsymbol{a}, \boldsymbol{r}_A$ honestly and queries its oracle on input $(A, M, \boldsymbol{a})$ and $(\sigma(), \boldsymbol{r}_A, \boldsymbol{r}_M)$ for $A = \sigma(\boldsymbol{a}) \times \boldsymbol{g} + \boldsymbol{r}_A \times \boldsymbol{h}$. It receives back a proof π_{sameperm}. It computes the remaining proofs $\pi_{\mathsf{samescalar}}, \pi_{\mathsf{samemsm}}$ the same as the honest prover. Then $\mathcal{B}_1$ returns $(\pi_{\mathsf{sameperm}}, \pi_{\mathsf{samescalar}}, \pi_{\mathsf{samemsm}})$. If $b = 1$ then $\mathcal{B}_1$ runs the simulator to compute the response. When $\mathcal{A}$ returns b' then if $b = b'$ then $\mathcal{B}_1$ returns 0, else $\mathcal{B}_1$ returns 1.

Then

$$\Pr[\mathcal{B}_1(\mathsf{crs}) = 0 \mid \bar{b} = 0] = \frac{1}{2}\left(\Pr[\mathcal{A}(\mathsf{crs}) = 0 \mid \mathsf{Game}_0, b = 0] + \Pr[\mathcal{A}(\mathsf{crs}) = 1 \mid \mathsf{Game}_0, b = 1]\right)$$
$$\Pr[\mathcal{B}_1(\mathsf{crs}) = 1 \mid \bar{b} = 1] = \frac{1}{2}\left(2 - \Pr[\mathcal{A}(\mathsf{crs}) = 0 \mid \mathsf{Game}_1, b = 0] - \Pr[\mathcal{A}(\mathsf{crs}) = 1 \mid \mathsf{Game}_1, b = 1]\right)$$

and

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{sameperm}}_{\mathcal{B}_1}(\lambda) &= |1 - 2\Pr[\mathsf{Game}^{\mathsf{sameperm}}_{\mathcal{B}_1}(\lambda)]| \\
&= |1 - \Pr[\mathcal{B}_1(\mathsf{crs}) = 0 \mid \bar{b} = 0] - \Pr[\mathcal{B}_1(\mathsf{crs}) = 1 \mid \bar{b} = 1]| \\
&= \Big|1 - \frac{1}{2}\left(\Pr[\mathcal{A}(\mathsf{crs}) = 0 \mid \mathsf{Game}_0, b = 0] + \Pr[\mathcal{A}(\mathsf{crs}) = 1 \mid \mathsf{Game}_0, b = 1]\right) \\
&\qquad - \left(1 - \Pr[\mathcal{B}_1(\mathsf{crs}) = 0 \mid \bar{b} = 1]\right)\Big| \\
&= \frac{1}{2}\Big|1 - \Pr[\mathcal{A}(\mathsf{crs}) = 0 \mid \mathsf{Game}_0, b = 0] - \Pr[\mathcal{A}(\mathsf{crs}) = 1 \mid \mathsf{Game}_0, b = 1] \\
&\qquad\qquad - 1 + \Pr[\mathcal{A}(\mathsf{crs}) = 0 \mid \mathsf{Game}_1, b = 0] + \Pr[\mathcal{A}(\mathsf{crs}) = 1 \mid \mathsf{Game}_0, b = 1]\Big| \\
&= \frac{1}{2}|\mathsf{Adv}^{\mathsf{Game}_0}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{\mathsf{Game}_1}_{\mathcal{A}}(\lambda)|
\end{aligned}$$

This means that

$$\mathsf{Adv}^{\mathsf{Game}_0}_{\mathcal{A}}(\lambda) \leqslant 2\mathsf{Adv}^{\mathsf{sameperm}}_{\mathcal{B}_1}(\lambda) + \mathsf{Adv}^{\mathsf{Game}_1}_{\mathcal{A}}(\lambda)$$

$\underline{\mathsf{Game}_1 \mapsto \mathsf{Game}_2}$: Define Game_2 to run identicallly to Game_1 except that, the crs and the $\pi_{\mathsf{samescalar}}$ proof are generated by the $\mathsf{samescalar}$ simulator for both $b = 0$ and $b = 1$.

Let $\mathcal{B}_2$ be an adversary against the $\mathsf{samescalar}$ zero-knowledge game. Then $\mathcal{B}_2$ simulates the zero-knowledge game for $\mathcal{A}$. It takes as input (G_T, G_U), generates the remaining terms $\boldsymbol{g}, \boldsymbol{h}, H$ randomly, and runs $\mathcal{A}(\mathsf{crs})$. It flips a coin to get $b \in \{0, 1\}$. When $\mathcal{A}$ makes a valid prover query $(\boldsymbol{R}, \boldsymbol{S}, \boldsymbol{T}, \boldsymbol{U}, M), (\sigma(), k, \boldsymbol{r}_M)$, if $b = 0$ then $\mathcal{B}_2$ generates $\boldsymbol{a}, A$ honestly and simulates π_{sameperm}. Then $\mathcal{B}_2$ generates $r_T, r_U, R, S, \mathsf{cm}_T, \mathsf{cm}_U$ honestly and queries its oracle on input $(R, S, \mathsf{cm}_T, \mathsf{cm}_U)$ and (k, r_T, r_U). It receives back a proof $\pi_{\mathsf{samescalar}}$. It computes the remaining proof π_{samemsm} the

same as the honest prover. Then $\mathcal{B}_2$ returns $(\pi_{\mathsf{sameperm}}, \pi_{\mathsf{samescalar}}, \pi_{\mathsf{samemsm}})$. If $b = 1$ then $\mathcal{B}_2$ runs the simulator to compute the response. When $\mathcal{A}$ returns b' then if $b = b'$ then $\mathcal{B}_2$ returns 0, else $\mathcal{B}_2$ returns 1.

By the same argument as in $\mathsf{Game}_0 \mapsto \mathsf{Game}_1$ we have that

$$\mathsf{Adv}^{\mathsf{Game}_1}_{\mathcal{A}}(\lambda) \leqslant 2\mathsf{Adv}^{\mathsf{samescalar}}_{\mathcal{B}_2}(\lambda) + \mathsf{Adv}^{\mathsf{Game}_2}_{\mathcal{A}}(\lambda)$$

$\underline{\mathsf{Game}_2 \mapsto \mathsf{Game}_3}$: Define Game_3 to run identicallly to Game_2 except that, the crs and the π_{samemsm} proof are generated by the $\mathsf{samemsm}$ simulator for both $b = 0$ and $b = 1$.

Let $\mathcal{B}_3$ be an adversary against the $\mathsf{samemsm}$ zero-knowledge game. Then $\mathcal{B}_3$ simulates the zero-knowledge game for $\mathcal{A}$. It takes as input $(\boldsymbol{g} \mid\mid \boldsymbol{h} \mid\mid G_T \mid\mid G_U)$, generates the remaining terms H randomly, and runs $\mathcal{A}(\mathsf{crs})$. It flips a coin to get $b \in \{0, 1\}$. When $\mathcal{A}$ makes a valid prover query $(\boldsymbol{R}, \boldsymbol{S}, \boldsymbol{T}, \boldsymbol{U}, M), (\sigma(), k, \boldsymbol{r}_M)$, if $b = 0$ then $\mathcal{B}_3$ generates $\boldsymbol{a}, A, \mathsf{cm}_T, \mathsf{cm}_U$ honestly and simulates $\pi_{\mathsf{sameperm}}, \pi_{\mathsf{samescalar}}$. Then $\mathcal{B}_3$ generates $A', \boldsymbol{T}', \boldsymbol{U}', \boldsymbol{x}$ honestly and queries its oracle on input $(A', \mathsf{cm}_{T,2}, \mathsf{cm}_{U,2}, \boldsymbol{T}', \boldsymbol{U}')$ and $\boldsymbol{x}$. It receives back a proof π_{samemsm}. Then $\mathcal{B}_3$ returns $(\pi_{\mathsf{sameperm}}, \pi_{\mathsf{samescalar}}, \pi_{\mathsf{samemsm}})$. If $b = 1$ then $\mathcal{B}_3$ runs the three simulators to compute the response. When $\mathcal{A}$ returns b' then if $b = b'$ then $\mathcal{B}_3$ returns 0, else $\mathcal{B}_3$ returns 1.

By the same argument as in $\mathsf{Game}_0 \mapsto \mathsf{Game}_1$ we have that

$$\mathsf{Adv}^{\mathsf{Game}_2}_{\mathcal{A}}(\lambda) \leqslant 2\mathsf{Adv}^{\mathsf{samemsm}}_{\mathcal{B}_3}(\lambda) + \mathsf{Adv}^{\mathsf{Game}_3}_{\mathcal{A}}(\lambda)$$

$\underline{\mathsf{Game}_3 \mapsto \mathsf{Game}_4}$: Define Game_4 to run identically to Game_3 except that, for both $b = 0$ and $b = 1$, the cm_T is selected uniformly at random.

Let $\mathcal{B}_4$ be an adversary against the ddh. Then $\mathcal{B}_4$ gets as input (G_1, G_2, G_3, G_4) and aims to distinguish whether these equal (G_1, xG_1, yG_1, xyG_1) for some x, y or not. It simulates the zero-knowledge game for $\mathcal{A}$ setting $G_T = G_1, H = G_2$, and generating the remaining terms $(\boldsymbol{g}, \boldsymbol{h}, G_U)$ randomly, and runs $\mathcal{A}(\mathsf{crs})$. It flips a coin to get $b \in \{0, 1\}$. When $\mathcal{A}$ makes a valid prover query, if $b = 0$ then $\mathcal{B}_4$ computes all elements the same as in Game_3 apart from it sets $\mathsf{cm}_T = (G_3, kR + G_4)$. If $b = 1$ then $\mathcal{B}_4$ runs the simulator to compute the response. When $\mathcal{A}$ returns b' then if $b = b'$ then $\mathcal{B}_2$ returns 0, else $\mathcal{B}_2$ returns 1.

Then

$$\Pr[\mathcal{B}_1(G_1, G_2, G_3, G_4) = 0 \mid \bar{b} = 0] = \frac{1}{2}\left(\Pr[\mathcal{A}(\mathsf{crs}) = 0 \mid \mathsf{Game}_3, b = 0] + \Pr[\mathcal{A}(\mathsf{crs}) = 1 \mid \mathsf{Game}_3, b = 1]\right)$$

$$\Pr[\mathcal{B}_1(G_1, G_2, G_3, G_4) = 1 \mid \bar{b} = 1] = \frac{1}{2}\left(2 - \Pr[\mathcal{A}(\mathsf{crs}) = 0 \mid \mathsf{Game}_4, b = 0] - \Pr[\mathcal{A}(\mathsf{crs}) = 1 \mid \mathsf{Game}_4, b = 1]\right)$$

Thus by the same argument as in $\mathsf{Game}_0 \mapsto \mathsf{Game}_1$ we have that

$$\mathsf{Adv}^{\mathsf{Game}_3}_{\mathcal{A}}(\lambda) \leqslant 2\mathsf{Adv}^{\mathsf{ddh}}_{\mathcal{B}_4}(\lambda) + \mathsf{Adv}^{\mathsf{Game}_4}_{\mathcal{A}}(\lambda)$$

$\underline{\mathsf{Game}_4 \mapsto \mathsf{Game}_5}$: Define Game_5 to run identically to Game_4 except that, for both $b = 0$ and $b = 1$, the cm_U is selected uniformly at random. Then by the same argument as in $\mathsf{Game}_3 \mapsto \mathsf{Game}_4$ we

have that there exists $\mathcal{B}_5$ such that

$$\mathsf{Adv}^{\mathsf{Game}_4}_{\mathcal{A}}(\lambda) \leqslant 2\mathsf{Adv}^{\mathsf{ddh}}_{\mathcal{B}_5}(\lambda) + \mathsf{Adv}^{\mathsf{Game}_5}_{\mathcal{A}}(\lambda)$$

$\underline{\mathsf{Game}_5 \mapsto 0}$: In Game_5 the simulated proofs are generated identically and are thus indistinguishable. □

Theorem 2.2.2 (Shuffle Argument is knowledge-sound)**.** *If SamePerm argument, SameScalar argument, and SameMultiScalar argument are knowledge-sound, and the* q-dlog *assumption holds, then the Curdleproofs shuffle argument described in Figures 2.2 and 2.3 is knowledge-sound.*

Proof. We design an extractor $\mathcal{X}_{\mathsf{shuffle}}$ such that for any adversary $\mathcal{A}$ that convinces the verifier, with overwhelming probability returns either a discrete logarithm relation between $\boldsymbol{g}, \boldsymbol{h}, G_T, G_U, H$ or a permutation $\sigma()$ and field element k such that

$$((\boldsymbol{R}, \boldsymbol{S}, \boldsymbol{T}, \boldsymbol{U}, M);\ (\sigma(), k)) \in R_{\mathsf{shuffle}}.$$

By the knowledge-soundness of SamePerm argument, SameScalar argument, and SameMultiScalar argument there exist extractors $\mathcal{X}_{\mathsf{sameperm}}, \mathcal{X}_{\mathsf{samescalar}}, \mathcal{X}_{\mathsf{samemsm}}$ such that if $\mathcal{A}$ returns verifying $(\pi_{\mathsf{sameperm}}, \pi_{\mathsf{samescalar}}, \pi_{\mathsf{samemsm}})$ then they return valid witnesses for their respective languages with overwhelming probability.

The extractor $\mathcal{X}_{\mathsf{shuffle}}$ works as follows

1. Run $(\boldsymbol{R}, \boldsymbol{S}, \boldsymbol{T}, \boldsymbol{U}, M);\ (A, \mathsf{cm}_T, \mathsf{cm}_U, \pi_{\mathsf{sameperm}}, \pi_{\mathsf{samescalar}}, \pi_{\mathsf{samemsm}}) \leftarrow \mathcal{A}(\mathsf{crs})$.
2. Let $\mathcal{B}_1$ be the adversary that returns $((A, M, \boldsymbol{a}), \pi_{\mathsf{sameperm}})$ and $\mathcal{X}_{\mathsf{sameperm}}$ its corresponding SamePerm extractor. Extract $\sigma(), \boldsymbol{r}_A, \boldsymbol{r}_M$ such that
$$M = \sigma(1, \ldots, \ell) \times \boldsymbol{g} + \boldsymbol{r}_M \times \boldsymbol{h} \ \wedge\ A = \sigma(\boldsymbol{a}) \times \boldsymbol{g} + \boldsymbol{r}_A \times \boldsymbol{h}$$
3. Let $\mathcal{B}_2$ be the adversary that returns $((\mathsf{cm}_T, \mathsf{cm}_U), \pi_{\mathsf{samescalar}})$ and $\mathcal{X}_{\mathsf{samescalar}}$ its corresponding SameScalar extractor. Extract k, r_T, r_U such that
$$\mathsf{cm}_{T,1} = r_T G_T,\ \mathsf{cm}_{T,2} = kR + r_T H,\ \wedge\ \mathsf{cm}_{U,1} = r_U G_U,\ \mathsf{cm}_{U,2} = kS + r_U H.$$
Return $(\sigma(), k)$.

We must show that whenever $\mathcal{A}$ convinces the verifier, then either $\mathcal{X}_{\mathsf{shuffle}}$ succeeds or we can extract a discrete logarithm relation between $(\boldsymbol{g}, \boldsymbol{h}, G_T, G_U, H)$. First see that $\mathcal{X}_{\mathsf{shuffle}}$ terminates in polynomial time because $\mathcal{X}_{\mathsf{sameperm}}$ and $\mathcal{X}_{\mathsf{samescalar}}$ terminate in polynomial time.

We design adversaries $\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3, \mathcal{B}_4$ such that for all extractors $\mathcal{X}_{\mathsf{sameperm}}, \mathcal{X}_{\mathsf{samescalar}}, \mathcal{X}_{\mathsf{samemsm}}$ we have that

$$\mathsf{Adv}_{\mathcal{A}, \mathcal{X}_{\mathsf{shuffle}}}(\lambda) \leqslant \mathsf{Adv}^{\mathsf{sameperm}}_{\mathcal{B}_1, \mathcal{X}_{\mathsf{sameperm}}}(\lambda) + \mathsf{Adv}^{\mathsf{samescalar}}_{\mathcal{B}_2, \mathcal{X}_{\mathsf{sameperm}}}(\lambda) + \mathsf{Adv}^{\mathsf{samemsm}}_{\mathcal{B}_3, \mathcal{X}_{\mathsf{samemsm}}}(\lambda) + \mathsf{Adv}^{\mathsf{q\text{-}dlog}}_{\mathcal{B}_4}(\lambda) + \frac{q_H}{|\mathbb{F}|}$$

We proceed via a series of games $\mathsf{Game}_1, \mathsf{Game}_2, \mathsf{Game}_3, \mathsf{Game}_4$ such that

$$\begin{aligned}
\mathsf{Adv}_{\mathcal{A},\mathcal{X}_{\mathsf{shuffle}}}(\lambda) &\leqslant \mathsf{Adv}^{\mathsf{sameperm}}_{\mathcal{B}_1,\mathcal{X}_{\mathsf{sameperm}}}(\lambda) + \mathsf{Adv}^{\mathsf{Game}_1}_{\mathcal{A},\mathcal{X}_{\mathsf{shuffle}}}(\lambda) \\
\mathsf{Adv}^{\mathsf{Game}_1}_{\mathcal{A},\mathcal{X}_{\mathsf{shuffle}}}(\lambda) &\leqslant \mathsf{Adv}^{\mathsf{samescalar}}_{\mathcal{B}_2,\mathcal{X}_{\mathsf{samescalar}}}(\lambda) + \mathsf{Adv}^{\mathsf{Game}_2}_{\mathcal{A},\mathcal{X}_{\mathsf{shuffle}}}(\lambda) \\
\mathsf{Adv}^{\mathsf{Game}_2}_{\mathcal{A},\mathcal{X}_{\mathsf{shuffle}}}(\lambda) &\leqslant \mathsf{Adv}^{\mathsf{samemsm}}_{\mathcal{B}_3,\mathcal{X}_{\mathsf{samemsm}}}(\lambda) + \mathsf{Adv}^{\mathsf{Game}_3}_{\mathcal{A},\mathcal{X}_{\mathsf{shuffle}}}(\lambda) \\
\mathsf{Adv}^{\mathsf{Game}_3}_{\mathcal{A},\mathcal{X}_{\mathsf{shuffle}}}(\lambda) &\leqslant \mathsf{Adv}^{\mathsf{q\text{-}dlog}}_{\mathcal{B}_4}(\lambda) + \mathsf{Adv}^{\mathsf{Game}_4}_{\mathcal{A},\mathcal{X}_{\mathsf{shuffle}}}(\lambda) \\
\mathsf{Adv}^{\mathsf{Game}_4}_{\mathcal{A},\mathcal{X}_{\mathsf{shuffle}}}(\lambda) &\leqslant \frac{q_H}{\mathbb{F}}
\end{aligned}$$

which combined give us our final result.

$\underline{\mathsf{Game}_0 \mapsto \mathsf{Game}_1}$: Let Game_0 be the initial knowledge-soundness game. Then Game_1 is identical except in the following case. If $\mathcal{A}$ outputs a verifying proof then define $\mathcal{B}_1$ be the adversary that returns $((A, M, \boldsymbol{a}), \pi_{\mathsf{sameperm}})$ and $\mathcal{X}_{\mathsf{sameperm}}$ its corresponding extractor. Return 0 if $\mathcal{X}_{\mathsf{sameperm}}$ fails. If Game_0 returns 1 but Game_1 returns 0 then this means that π_{sameperm} verifies and $\mathcal{X}_{\mathsf{sameperm}}$ fails, and hence that $\mathcal{B}_1$ succeeds.

$\underline{\mathsf{Game}_1 \mapsto \mathsf{Game}_2}$: Define Game_2 to be identical to Game_1 except in the following case. If $\mathcal{A}$ outputs a verifying proof then define $\mathcal{B}_2$ be the adversary that returns $((\mathsf{cm}_T, \mathsf{cm}_U), \pi_{\mathsf{samescalar}})$ and $\mathcal{X}_{\mathsf{samescalar}}$ its corresponding extractor. Return 0 if $\mathcal{X}_{\mathsf{samescalar}}$ fails. such that $\mathcal{X}_{\mathsf{samescalar}}$ fails then Game_2 returns 0. This is in addition to returning 0 if $\mathcal{X}_{\mathsf{sameperm}}$ fails. If Game_1 returns 1 but Game_2 returns 0 then this means that $\pi_{\mathsf{samescalar}}$ verifies and $\mathcal{X}_{\mathsf{samescalar}}$ fails ,and hence that $\mathcal{B}_2$ succeeds.

$\underline{\mathsf{Game}_2 \mapsto \mathsf{Game}_3}$: Define Game_3 to be identical to Game_2 except in the following case. If $\mathcal{A}$ outputs a verifying proof then define $\mathcal{B}_3$ to be the adversary that returns $((B, \mathsf{cm}_{T,2}, \mathsf{cm}_{U,2}, \boldsymbol{V}, \boldsymbol{W}), \pi_{\mathsf{samemsm}})$ for

$$B = A + \mathsf{cm}_{T,1} + \mathsf{cm}_{U,1},\ \boldsymbol{V} = (\boldsymbol{T} \mid\mid \boldsymbol{0} \mid\mid H \mid\mid 0),\ \boldsymbol{W} = (\boldsymbol{U} \mid\mid \boldsymbol{0} \mid\mid 0 \mid\mid H)$$

and $\mathcal{X}_{\mathsf{samemsm}}$ its corresponding extractor. Return 0 if $\mathcal{X}_{\mathsf{samemsm}}$ fails. This is in addition to returning 0 if $\mathcal{X}_{\mathsf{sameperm}}$ fails or if $\mathcal{X}_{\mathsf{samescalar}}$ fails. If Game_2 returns 1 but Game_3 returns 0 then this means that π_{samemsm} verifies and $\mathcal{X}_{\mathsf{samemsm}}$ fails, and hence that $\mathcal{B}_3$ succeeds.

$\underline{\mathsf{Game}_3 \mapsto \mathsf{Game}_4}$: Define Game_4 to be identical to Game_3 except in the following case. If $\mathcal{X}_{\mathsf{sameperm}}$ and $\mathcal{X}_{\mathsf{samescalar}}$ output $(\sigma(), \boldsymbol{r}_A, \boldsymbol{r}_M)$ and (k, r_T, r_U) and if $\mathcal{X}_{\mathsf{samemsm}}$ outputs $\boldsymbol{x}$ such that

$$\boldsymbol{x} \neq (\sigma(\boldsymbol{a}) \mid\mid \boldsymbol{r}_A \mid\mid r_T \mid\mid r_U)$$

then return 0.

We define the adversary $\mathcal{B}_4$ against the $(\ell+n_{\mathsf{bl}}+3)$-dlog assumption that takes as input $(\ell+n_{\mathsf{bl}}+3)$ random group elements $\boldsymbol{g}'$ and aims to output two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ such that

$$\boldsymbol{x} \times \boldsymbol{g}' = \boldsymbol{y} \times \boldsymbol{g}' \ \wedge\ \boldsymbol{x} \neq \boldsymbol{y}$$

First $\mathcal{B}_4$ splits $(\boldsymbol{g} \mid\mid \boldsymbol{h} \mid\mid G_T \mid\mid G_U \mid\mid H) \leftarrow \boldsymbol{g}'$ and runs $\mathcal{A}$ on $\mathsf{crs} = (\boldsymbol{g} \mid\mid \boldsymbol{h} \mid\mid G_T \mid\mid G_U \mid\mid H)$. When $\mathcal{A}$ returns a verifying proof

$$((\boldsymbol{R}, \boldsymbol{S}, \boldsymbol{T}, \boldsymbol{U}, M);\ (A, \mathsf{cm}_T, \mathsf{cm}_U, \pi_{\mathsf{sameperm}}, \pi_{\mathsf{samescalar}}, \pi_{\mathsf{samemsm}})) \leftarrow \mathcal{A}(\mathsf{crs})$$

then $\mathcal{B}_4$ runs $\mathcal{X}_{\mathsf{sameperm}}$ and $\mathcal{X}_{\mathsf{samescalar}}$ to obtain $(\sigma(), \boldsymbol{r}_A, \boldsymbol{r}_M)$, (k, r_T, r_U) and $\mathcal{X}_{\mathsf{samemsm}}$ to obtain $\boldsymbol{x}$ such that

$$A + r_T G_T + r_U G_U = \boldsymbol{x} \times \boldsymbol{g}' = \sigma(\boldsymbol{a}) \times \boldsymbol{g} + \boldsymbol{r}_A \times \boldsymbol{h} + r_T G_T + r_U G_U$$

See that if Game_3 returns 1 then these extractors will succeed. Then $\mathcal{B}_4$ returns $(\boldsymbol{x},\ (\sigma(\boldsymbol{a}) \,||\, \boldsymbol{r}_A \,||\, r_T \,||\, r_U))$. If Game_4 returns 0 then this output is a valid $\mathsf{q\text{-}dlog}$ solution.

$\underline{\mathsf{Game}_4 \mapsto \mathsf{negl}(\lambda)}$: If Game_4 returns 1 then we have that $\mathcal{X}_{\mathsf{shuffle}}$ outputs $(\sigma(), \boldsymbol{r}_M, k)$ such that $M = \sigma(\boldsymbol{a}) \times \boldsymbol{g} + \boldsymbol{r}_M \boldsymbol{h}$ and there exists $\boldsymbol{r}_A$ with

$$\begin{aligned} A + r_T G_T + r_U G_U &= (\sigma(\boldsymbol{a}) \,||\, \boldsymbol{r}_A \,||\, r_T \,||\, r_U) \ \times\ (\boldsymbol{g} \,||\, \boldsymbol{h} \,||\, G_T \,||\, G_U) \\ kR + r_T H &= (\sigma(\boldsymbol{a}) \,||\, \boldsymbol{r}_A \,||\, r_T \,||\, r_U) \ \times\ (\boldsymbol{T} \,||\, \boldsymbol{0} \,||\, H \,||\, 0) \\ kS + r_U H &= (\sigma(\boldsymbol{a}) \,||\, \boldsymbol{r}_A \,||\, r_T \,||\, r_U) \ \times\ (\boldsymbol{U} \,||\, \boldsymbol{0} \,||\, 0 \,||\, H) \end{aligned}$$

Thus

$$\begin{aligned} kR + r_T H &= \sigma(\boldsymbol{a}) \times \boldsymbol{T} + r_T H \\ kS + r_U H &= \sigma(\boldsymbol{a}) \times \boldsymbol{U} + r_U H \end{aligned}$$

and

$$\boldsymbol{a} \times k\boldsymbol{R} = \sigma(\boldsymbol{a}) \times \boldsymbol{T} \ \wedge\ \boldsymbol{a} \times k\boldsymbol{S} = \sigma(\boldsymbol{a}) \times \boldsymbol{U}$$

Where $\boldsymbol{a}$ is selected after $(\boldsymbol{R}, \boldsymbol{S}, \boldsymbol{T}, \boldsymbol{U}, M)$ are determined, and $\ell > 1$, this happens with maximum probability $q_H/|\mathbb{F}|$. □

$\underline{\mathsf{Prove}_{\mathsf{shuffle}}(\mathsf{crs}_{\mathsf{shuffle}};\ (\boldsymbol{R}, \boldsymbol{S}, \boldsymbol{T}, \boldsymbol{U}, M);\ (\sigma(), k, \boldsymbol{r}_M))}$

Step 1:

$(\boldsymbol{g}, \boldsymbol{h}, G_T, G_U, H) \leftarrow \mathsf{parse}(\mathsf{crs}_{\mathsf{shuffle}})$

$\boldsymbol{a} = (a_1, \ldots, a_\ell) \leftarrow \mathsf{Hash}(\boldsymbol{R}, \boldsymbol{S}, \boldsymbol{T}, \boldsymbol{U}, M)$

Step 2:

$\boldsymbol{r}_A \xleftarrow{\$} \mathbb{F}^{n_{\mathsf{bl}}-2}$

$\boldsymbol{r}'_A \leftarrow (\boldsymbol{r}_A \mid\mid (0,0))$

$A \leftarrow \sigma(\boldsymbol{a}) \times \boldsymbol{g} + \boldsymbol{r}'_A \times \boldsymbol{h}$

$\pi_{\mathsf{sameperm}} \leftarrow \mathsf{Prove}_{\mathsf{sameperm}}\left((\boldsymbol{g}, \boldsymbol{h}, H);\ (A, M, \boldsymbol{a});\ (\sigma(), \boldsymbol{r}'_A, \boldsymbol{r}_M)\right)$

Step 3:

$r_T, r_U \xleftarrow{\$} \mathbb{F}$

$R \leftarrow \boldsymbol{a} \times \boldsymbol{R}$

$S \leftarrow \boldsymbol{a} \times \boldsymbol{S}$

$\mathsf{cm}_T \leftarrow \mathsf{GroupCommit}((G_T, H);\ kR;\ r_T)$

$\mathsf{cm}_U \leftarrow \mathsf{GroupCommit}((G_U, H);\ kS;\ r_U)$

$\pi_{\mathsf{samescalar}} \leftarrow \mathsf{Prove}_{\mathsf{samescalar}}((G_T, G_U, H);\ (R, S, \mathsf{cm}_T, \mathsf{cm}_U);\ (k, r_T, r_U))$

Step 4:

$A' \leftarrow A + \mathsf{cm}_{T,1} + \mathsf{cm}_{U,1}$

$\boldsymbol{G} \leftarrow (\boldsymbol{g} \mid\mid \boldsymbol{h}_{[:n_{\mathsf{bl}}-2]} \mid\mid G_T \mid\mid G_U)$

$\boldsymbol{T}' \leftarrow (\boldsymbol{T} \mid\mid \boldsymbol{0} \mid\mid H \mid\mid 0)$

$\boldsymbol{U}' \leftarrow (\boldsymbol{U} \mid\mid \boldsymbol{0} \mid\mid 0 \mid\mid H)$

$\boldsymbol{x} \leftarrow (\sigma(\boldsymbol{a}) \mid\mid \boldsymbol{r}_A \mid\mid r_T \mid\mid r_U)$

$\pi_{\mathsf{samemsm}} \leftarrow \mathsf{Prove}_{\mathsf{samemsm}}\left(\ \boldsymbol{G};\ (A', \mathsf{cm}_{T,2}, \mathsf{cm}_{U,2}, \boldsymbol{T}', \boldsymbol{U}');\ \boldsymbol{x}\ \right)$

return $(A, \mathsf{cm}_T, \mathsf{cm}_U, R, S, \pi_{\mathsf{sameperm}}, \pi_{\mathsf{samescalar}}, \pi_{\mathsf{samemsm}})$

Figure 2.2: The Curdleproofs proving algorithm to demonstrate that $\boldsymbol{T}, \boldsymbol{U} = \sigma(k\boldsymbol{R}), \sigma(k\boldsymbol{S})$ for some field element k and permutation σ committed in M.

$\underline{\mathsf{Verify}_{\mathsf{shuffle}}(\mathsf{crs}_{\mathsf{shuffle}}, ;\ \phi_{\mathsf{shuffle}};\ \pi_{\mathsf{shuffle}})}$
Step 1:
$(\boldsymbol{g}, \boldsymbol{h}, G_T, G_U, H) \leftarrow \mathsf{parse}(\mathsf{crs}_{\mathsf{shuffle}})$
$(\boldsymbol{R}, \boldsymbol{S}, \boldsymbol{T}, \boldsymbol{U}, M) \leftarrow \mathsf{parse}(\phi_{\mathsf{shuffle}})$
$(A, \mathsf{cm}_T, \mathsf{cm}_U, R, S, \pi_{\mathsf{sameperm}}, \pi_{\mathsf{samescalar}}, \pi_{\mathsf{samemsm}}) \leftarrow \mathsf{parse}(\pi_{\mathsf{shuffle}})$
$\boldsymbol{a} = (a_1, \ldots, a_\ell) \leftarrow \mathsf{Hash}(\boldsymbol{R}, \boldsymbol{S}, \boldsymbol{T}, \boldsymbol{U}, M)$

Step 2:
$\mathsf{check}_1 \leftarrow \mathsf{Verify}_{\mathsf{sameperm}}((\boldsymbol{g}, \boldsymbol{h}, H);\ (A, M);\ \pi_{\mathsf{sameperm}})$

Step 3:
$R \leftarrow \boldsymbol{a} \times \boldsymbol{R}$
$S \leftarrow \boldsymbol{a} \times \boldsymbol{S}$
$\mathsf{check}_2 \leftarrow \mathsf{Verify}_{\mathsf{samescalar}}((G_T, G_U, H);\ (R, S, \mathsf{cm}_T, \mathsf{cm}_U);\ \pi_{\mathsf{samescalar}})$

Step 4:
$A' \leftarrow A + \mathsf{cm}_{T,1} + \mathsf{cm}_{U,1}$
$\boldsymbol{G} \leftarrow (\boldsymbol{g} \mid\mid \boldsymbol{h}_{[:n_{\mathsf{bl}}-2]} \mid\mid G_T \mid\mid G_U)$
$\boldsymbol{T}' \leftarrow (\boldsymbol{T} \mid\mid \boldsymbol{0} \mid\mid H \mid\mid 0)$
$\boldsymbol{U}' \leftarrow (\boldsymbol{U} \mid\mid \boldsymbol{0} \mid\mid 0 \mid\mid H)$
$\mathsf{check}_3 \leftarrow \mathsf{Verify}_{\mathsf{samemsm}}(\boldsymbol{G};\ (A', \mathsf{cm}_{T,2}, \mathsf{cm}_{U,2}, \boldsymbol{T}', \boldsymbol{U}'); \pi_{\mathsf{samemsm}})$

return 1 if $(\mathsf{check}_1, \mathsf{check}_2, \mathsf{check}_3) = (1, 1, 1)$
else return 0

Figure 2.3: The Curdleproofs verification algorithm to check that $\boldsymbol{T}, \boldsymbol{U} = \sigma(k\boldsymbol{R}), \sigma(k\boldsymbol{S})$ for some unknown field element k and unknown permutation σ committed in M.

Chapter 3

SameScalar Argument

In this chapter we discuss a zero knowledge argument for the relation

$$R_{\mathsf{samescalar}} = \left\{ \begin{array}{l|l} (R, S, \mathsf{cm}_T, \mathsf{cm}_U); \quad (k, r_U, r_T) & \begin{array}{l} \mathsf{cm}_T = \mathsf{GroupCommit}((G_T, H);\ kR;\ r_T) \\ \mathsf{cm}_U = \mathsf{GroupCommit}((G_U, H);\ kS;\ r_U) \end{array} \end{array} \right\}$$

It demonstrates that given public input $(R, S, \mathsf{cm}_T, \mathsf{cm}_U)$ there exists k such that cm_T is a commitment to $T = kR$ and cm_U is a commitment to kS.

The same scalar argument does not depend on any subroutines. This chapter consists of a single section discussing the argument. We first describe the full zero-knowledge SameScalar construction. We then prove its security in Theorems 3.0.1 and 3.0.2.

3.0.1 Full Zero-Knowledge Construction

A formal description of SameScalar argument is provided in Figure 3.1. The protocol is a simple sigma-protocol and makes use of the additive homomorphism of the commitment scheme.

In order to convince the verifier the prover chooses a random statement that satisfies the same-scalar relation. In other words it chooses a random scalar r_k and computes two group elements $A = r_k R$ and $B = r_k S$ with the same scalar. The prover then outputs the commitments: (1) cm_A a commitment to A under randomness r_A; and (2) cm_B a commitment to B under randomness r_B. These commitments are hashed, together with the instance, to get a challenge α.

The commitment scheme is homomorphic and thus $\mathsf{cm}_A + \alpha\mathsf{cm}_T$ is a commitment to $A + \alpha T$ where T is the contents of cm_T. Similarly $\mathsf{cm}_B + \alpha\mathsf{cm}_U$ is a commitment to $B + \alpha U$ where U is the contents of cm_U. If $T = kR$, $U = kS$, $A = r_k R$, and $B = r_k S$ then we have that $A + \alpha T$ and $B + \alpha U$ have the same scalar (namely $r_k + \alpha k$). This is negligibly unlikely to occur if either T and U or A and B do not have the same scalar because α is chosen randomly. Thus the prover returns $z_k = r_k + \alpha k$ together with the commitment randomness $z_T = r_T + \alpha r_A$ and $z_U = r_k + \alpha r_B$. The verifier checks that (1) $\mathsf{cm}_A + \alpha\mathsf{cm}_T$ is a commitment to $z_k R$ under randomness z_T; and (2) $\mathsf{cm}_B + \alpha\mathsf{cm}_U$ is a commitment to $z_k S$ under randomness z_U.

3.0.2 Security

Theorem 3.0.1 (SameScalar argument is zero-knowledge)**.** *SameScalar argument in Figure 3.1 is zero-knowledge in the random oracle model.*

$\underline{\mathsf{Prove}_{\mathsf{samescalar}}(\mathsf{crs}_{\mathsf{samescalar}};\ (R, S, \mathsf{cm}_T, \mathsf{cm}_U);\ (k, r_T, r_U))}$

Step 1:

$(G_T, G_U, H) \leftarrow \mathsf{parse}(\mathsf{crs})$

$r_A, r_B, r_k \xleftarrow{\$} \mathbb{F}$

$\mathsf{cm}_A \leftarrow \mathsf{GroupCommit}((G_T, H);\ r_k R;\ r_A)$

$\mathsf{cm}_B \leftarrow \mathsf{GroupCommit}((G_U, H);\ r_k S;\ r_B)$

$\alpha \leftarrow \mathsf{Hash}(R, S, \mathsf{cm}_T, \mathsf{cm}_U, \mathsf{cm}_A, \mathsf{cm}_B)$

Step 2:

$z_k \leftarrow r_k + \alpha k$

$z_T \leftarrow r_A + \alpha r_T$

$z_U \leftarrow r_B + \alpha r_U$

return $(\mathsf{cm}_A, \mathsf{cm}_B, z_k, z_T, z_U)$

$\underline{\mathsf{Verify}_{\mathsf{samescalar}}(\mathsf{crs}_{\mathsf{samescalar}};\ \phi_{\mathsf{samescalar}};\ \pi_{\mathsf{samescalar}})}$

Step 1:

$(G_T, G_U, H) \leftarrow \mathsf{parse}(\mathsf{crs}_{\mathsf{samescalar}})$

$(R, S, \mathsf{cm}_T, \mathsf{cm}_U) \leftarrow \mathsf{parse}(\phi_{\mathsf{samescalar}})$

$(\mathsf{cm}_A, \mathsf{cm}_B, z_k, z_T, z_U) \leftarrow \mathsf{parse}(\pi_{\mathsf{samescalar}})$

$\alpha \leftarrow \mathsf{Hash}(R, S, \mathsf{cm}_T, \mathsf{cm}_U, \mathsf{cm}_A, \mathsf{cm}_B)$

Step 2:

check $\mathsf{cm}_A + \alpha \mathsf{cm}_T = \mathsf{GroupCommit}((G_T, H);\ z_k R;\ z_T)$

check $\mathsf{cm}_B + \alpha \mathsf{cm}_U = \mathsf{GroupCommit}((G_U, H);\ z_k S;\ z_U)$

return 1 if both checks pass, else return 0

Figure 3.1: Proving and verifying algorithms to demonstrate that cm_T and cm_U open to some T and U such that $T = kR$ amd $U = kS$ for the same scalar k.

Proof. We design a simulator $\mathsf{Simulate}$ that takes as input an instance

$$(R, S, \mathsf{cm}_T, \mathsf{cm}_U)$$

and outputs a proof $\pi_{\mathsf{samescalar}}$ that is indistinguishable from a proof generated by an honest prover that knows the witness. The simulator can program the random oracle on any points yet to be queried by the adversary.

The simulator chooses $z_k, z_T, z_U, \alpha \xleftarrow{\$} \mathbb{F}$. They set

$$\mathsf{cm}_A = \mathsf{cm}_T - \alpha\mathsf{GroupCommit}((G_T, H);\ z_k R;\ z_T)$$
$$\mathsf{cm}_B = \mathsf{cm}_U - \alpha\mathsf{GroupCommit}((G_U, H);\ z_k S;\ z_U)$$

They program the random oracle to return α on input $(R, S, \mathsf{cm}_T, \mathsf{cm}_U, \mathsf{cm}_A, \mathsf{cm}_B)$ and return $(\mathsf{cm}_A, \mathsf{cm}_B, z_k, z_T, z_U)$.

First observe that $\mathsf{cm}_A, \mathsf{cm}_B$ are randomised and thus with overwhelming probability the oracle will not have already been programmed at this point. Second we see that if the commitments cm_T and cm_U are in $R_{\mathsf{samescalar}}$ then there exists k, r_T, r_U such that

$$\mathsf{cm}_T = \mathsf{GroupCommit}((G_T, H);\ kR;\ r_T)$$
$$\mathsf{cm}_U = \mathsf{GroupCommit}((G_U, H);\ kS;\ r_U)$$

Thus for $r'_k = (r_k - \alpha z_k)$, $r_A = (r_T - \alpha z_T)$ and $r_B = (r_U - \alpha z_U)$ we have that

$$\mathsf{cm}_A = \mathsf{GroupCommit}((G_T, H);\ r'_k R;\ r_A)$$
$$\mathsf{cm}_B = \mathsf{GroupCommit}((G_U, H);\ r'_k S;\ r_B)$$

Where r'_k, r_A, r_B are randomised by z_k, z_T, z_U respectively we see that these outputs are indistinguishable from the honest provers output.

The remaining outputs z_k, z_T, z_U are the unique openings of $\mathsf{cm}_A + \alpha\mathsf{cm}_T$ and $\mathsf{cm}_B + \alpha\mathsf{cm}_U$ and thus the only values that satisfy the verifiers equations for both prover and verifier. This is because our commitment scheme is perfectly binding. Thus the prover and simulator values are sampled from the same distribution. □

Theorem 3.0.2 ($\mathsf{SameScalar}$ argument is knowledge-sound)**.** *The same scalar argument described in Figure 3.1 is statistically knowledge-sound in the random oracle model.*

Proof. We design an extractor $\mathcal{X}_{\mathsf{sameperm}}$ such that: if there exists an adversary $\mathcal{A}$ that convinces the verifier with non-negligible probability then with overwhelming probability $\mathcal{X}_{\mathsf{sameperm}}$ returns field elements k, r_T, r_U such that

$$((R, S, \mathsf{cm}_T, \mathsf{cm}_U);\ (k, r_T, r_U)) \in R_{\mathsf{samescalar}}.$$

The extractor $\mathcal{X}_{\mathsf{sameperm}}$ works as follows

1. Generate $\mathsf{crs} = (G_T, G_U, H)$ and set $\mathsf{trans} = 0$. While $\mathsf{trans} = 0$ run $\mathsf{trans} \leftarrow \mathsf{Game}^{\mathsf{fork}}_{\mathcal{A}}(\mathsf{crs})$ for $\mathsf{Game}^{\mathsf{fork}}_{\mathcal{A}}(\mathsf{crs})$ defined in Section 7.0.2. Parse

$$(\alpha, \alpha'), ((R, S, \mathsf{cm}_T, \mathsf{cm}_U);\ (\mathsf{cm}_A, \mathsf{cm}_B)), (z_k, z_T, z_U), (z'_k, z'_T, z'_U)) = \mathsf{trans}$$

2. Compute

$$r_k = (z_k - z'_k)/(\alpha - \alpha')$$
$$r_T = (z_T - z'_T)/(\alpha - \alpha')$$
$$r_U = (z_U - z'_U)/(\alpha - \alpha')$$

and return (r_k, r_T, r_U).

We must show that whenever $\mathcal{A}$ convinces the verifier then $\mathcal{X}_{\mathsf{sameperm}}$ succeeds.

First see that $\mathcal{X}_{\mathsf{sameperm}}$ terminates in polynomial time with overwhelming probability. Let τ be the run time of $\mathcal{A}$, ϵ be the probability that $\mathcal{A}$ outputs a valid response and q_H be the total number of hash queries that $\mathcal{A}$ can make. Assuming $\frac{8q_H}{|\mathbb{F}|} < \epsilon$ then the game $\mathsf{Game}^{\mathsf{fork}}_{\mathcal{A}}(\mathsf{par})$ runs in time at most $\frac{8\tau q_H}{\epsilon} \cdot \ln(8/\epsilon)$ and is successful with probability at least $\epsilon/8$. Thus the expected run time of $\mathcal{X}_{\mathsf{samemsm}}$ is less than $\tau q_H \cdot \ln(8/\epsilon)$, which is polynomial time assuming that τ, ϵ and q_H are polynomial in the security parameter.

Second see that where both proofs verify we have that

$$\mathsf{cm}_A + \alpha\mathsf{cm}_T = \mathsf{GroupCommit}((G_T, H);\ z_k R;\ z_T)$$
$$\mathsf{cm}_A + \alpha'\mathsf{cm}_T = \mathsf{GroupCommit}((G_T, H);\ z'_k R;\ z'_T)$$

and hence

$$(\alpha - \alpha')\mathsf{cm}_T = \mathsf{GroupCommit}((G_T, H);\ (z_k - z'_k)R;\ z_T - z'_T)$$

Multiplying both sides by $(\alpha - \alpha')^{-1}$ yields

$$\mathsf{cm}_T = \mathsf{GroupCommit}\left((G_T, H);\ \frac{z_k - z'_k}{\alpha - \alpha'}R;\ \frac{z_T - z'_T}{\alpha - \alpha'}\right) = \mathsf{GroupCommit}((G_T, H);\ r_k R;\ r_T)$$

Similarly see that where both proofs verify we have that

$$\mathsf{cm}_B + \alpha\mathsf{cm}_U = \mathsf{GroupCommit}((G_U, H);\ z_k S;\ z_U)$$
$$\mathsf{cm}_B + \alpha'\mathsf{cm}_U = \mathsf{GroupCommit}((G_U, H);\ z'_k S;\ z'_U)$$

and hence

$$(\alpha - \alpha')\mathsf{cm}_U = \mathsf{GroupCommit}((G_U, H);\ (z_k - z'_k)S;\ z_U - z'_U)$$

Multiplying both sides by $(\alpha - \alpha')^{-1}$ yields

$$\mathsf{cm}_U = \mathsf{GroupCommit}\left((G_U, H);\ \frac{z_k - z'_k}{\alpha - \alpha'}S;\ \frac{z_U - z'_U}{\alpha - \alpha'}\right) = \mathsf{GroupCommit}((G_T, H);\ r_k S;\ r_U)$$

Thus (r_k, r_T, r_U) is a valid witness. □

Chapter 4

SameMultiscalar Argument

In this chapter we discuss a zero knowledge argument for the relation

$$R_{\mathsf{samemsm}} = \left\{ (A, Z_T, Z_U, \boldsymbol{T}, \boldsymbol{U}); \quad \boldsymbol{x} \;\middle|\; \begin{array}{l} A = \boldsymbol{x} \times \boldsymbol{G} \\ Z_T = \boldsymbol{x} \times \boldsymbol{T} \\ Z_U = \boldsymbol{x} \times \boldsymbol{U} \end{array} \right\}.$$

SameMultiScalar argument does not depend on any subroutines. This chapter consists of a single section discussing the argument. We first provide an informal overview and then describe the full zero-knowledge SameMultiScalar construction. We finish by proving its security in Theorems 4.0.1 and 4.0.3.

4.0.1 Informal Overview

Our SameMultiScalar relation can be seen as a form of inner product relation where one is interested in verifying whether $A = \boldsymbol{x} \times \boldsymbol{G}$, $Z_T = \boldsymbol{x} \times \boldsymbol{T}$ and $Z_U = \boldsymbol{x} \times \boldsymbol{U}$ for some $\boldsymbol{x}$. Inner product relations have proven popular in recent years and have been the focus both of a long line of both academic work [BCC+16, BBB+18, WTS+18, LMR19, HKR19, JT20, BMM+21, ACF21, GT21, BCS21, RMM21] and implementation work. By expressing our multiscalar relation as an inner product we can thus capitalise on this preexisting work.

In our case we consider that A is a commitment to $\boldsymbol{x}$ and $\boldsymbol{T}$ is the identity commitment to $\boldsymbol{T}$. We then wish to show that $Z_T = \boldsymbol{x} \times \boldsymbol{T}$. Here $\boldsymbol{x}$ is private while $\boldsymbol{T}$ is known to the verifier. For simplicity we ignore the proof that $Z_U = \boldsymbol{x} \times \boldsymbol{U}$ because this behaves identically. The inner product argument is recursive. At each stage of the recursion, the aim is to find new commitments $A', \boldsymbol{T}'$ to values $\boldsymbol{x}', \boldsymbol{T}'$ of half the length. Further we need a new Z'_T such that $Z'_T = \boldsymbol{x}' \times \boldsymbol{T}'$ if and only if $Z_T = \boldsymbol{x} \times \boldsymbol{T}$. After sufficient rounds of recursion we have that $\boldsymbol{x}'$ is a vector of length 1, and thus can be sent in the clear. The verifier checks that the inner product relation holds for the final revealed openings, and this suffices to show that the relation holds for the original longer openings.

Each round of the recursion proceeds as follows. The prover first computes auxiliary cross product commitments (that will later be used to define A' and T') as

$$L_A = \boldsymbol{x}_{[:n]} \times \boldsymbol{G}_{[n:]}, \; R_A = \boldsymbol{x}_{[n:]} \times \boldsymbol{G}_{[:n]}, \; L_T = \boldsymbol{x}_{[:n]} \times \boldsymbol{T}_{[n:]}, \; R_T = \boldsymbol{x}_{[n:]} \times \boldsymbol{T}_{[:n]}$$

Here n is a power of two. These are then hashed to find a random challenge γ .

The verifier updates the claimed inner product result to $Z'_T = \gamma L_T + Z_T + \gamma^{-1} R_T$ and the prover updates the commitment contents to

$$\boldsymbol{x}' = \boldsymbol{x}_{[:n]} + \gamma^{-1} \boldsymbol{c}_{[n:]},\ \boldsymbol{T}' = \boldsymbol{T}_{[:n]} + \gamma \boldsymbol{T}_{[n:]}$$

such that $Z'_T = \boldsymbol{x}' \times \boldsymbol{T}'$. See that $\boldsymbol{x}'$ and $\boldsymbol{T}'$ are half the length of $\boldsymbol{x}$ and $\boldsymbol{T}$. We then update the commitment A to $\boldsymbol{x}$ and the commitment key $\boldsymbol{G}$ as

$$A' = \gamma L_A + A + \gamma^{-1} R_A,\ \boldsymbol{G}' = \boldsymbol{G}_{[:n]} + \gamma^{-1} \boldsymbol{G}_{[n:]}$$

such that $A' = \boldsymbol{x}' \times \boldsymbol{G}'$ is a commitment to $\boldsymbol{x}'$.

Putting this together means we have $(A', T') = (\boldsymbol{x}' \times \boldsymbol{G}', \boldsymbol{x}' \times \boldsymbol{T}')$ for some $\boldsymbol{x}'$ that is half the length of $\boldsymbol{x}$. Due to the randomised nature of γ this statement is true if and only if the original $(A, Z_T) = (\boldsymbol{x} \times \boldsymbol{G}, \boldsymbol{x} \times \boldsymbol{T})$ for some $\boldsymbol{x}$. The protocol then recurses until the final round, where $\boldsymbol{x}$ and $\boldsymbol{T}$ have length 1. Then the prover sends $\boldsymbol{x} = x_1$ in the clear and verifier accepts if and only if $Z_T = x_1 T_1$. Note that the full protocol has some additional masking values that are included to ensure zero-knowledge. For simplicity we have ignored these terms in this overview.

4.0.2 Full Zero Knowledge Construction

A formal description of SameMultiScalar construction is provided in Figures 4.1 and 4.2. Inner product arguments are not, by default, zero-knowledge. In order to get a zero-knowledge argument we introduce a step at the beginning to randomise the provers witness. In particular the prover first blinds the argument by sampling $\boldsymbol{r}$ randomly. They compute $B_A, B_T, B_U = (\boldsymbol{r} \times \boldsymbol{G}, \boldsymbol{r} \times \boldsymbol{T}, \boldsymbol{r} \times \boldsymbol{U})$ to blind the witness relating to A, Z_T and Z_U respectively. They hash to obtain the field element α. The prover resets the private inputs to equal $\boldsymbol{r} + \alpha \boldsymbol{x}$ and the verifier resets the public inputs to equal

$$A = B_A + \alpha A \text{ and } Z_T = B_T + \alpha Z_T \text{ and } Z_U = B_U + \alpha Z_U$$

At this point the provers private input $\boldsymbol{x}$ is fully randomised and the prover could, theoretically, reveal it in the clear. Doing so however would increase the proof size significantly. Instead we run the inner product argument as specified in Section 4.0.1.

4.0.3 Security

Theorem 4.0.1 (SameMultiScalar argument is zero-knowledge)**.** *SameMultiScalar argument in Figures 4.1 and 4.2 is zero-knowledge in the random oracle model.*

Proof. We design a simulator Simulate that takes as input an instance

$$(A, Z_T, Z_U, \boldsymbol{T}, \boldsymbol{U})$$

and outputs a proof π_{samemsm} that is indistinguishable from a proof generated by an honest prover that knows the witness. The simulator can program the random oracle on any points yet to be determined by the adversary.

The simulator Simulate samples $\mathbf{x}', \alpha \xleftarrow{\$} \mathbb{F}$ and computes

$$\begin{aligned} A' &= \boldsymbol{x}' \times \boldsymbol{G} \\ Z'_T &= \boldsymbol{x}' \times \boldsymbol{T} \\ Z'_U &= \boldsymbol{x}' \times \boldsymbol{U} \end{aligned}$$

They set

$$B_A = A' - \alpha A$$
$$B_T = Z'_T - \alpha Z_T$$
$$B_U = Z'_U - \alpha Z_U$$

program $\mathsf{Hash}(A, Z_T, Z_U, \boldsymbol{T}, \boldsymbol{U}, B_A, B_T, B_U)$ to equal α. In the remaining steps they behave exactly as the honest prover with respect to the inputs $\boldsymbol{x}'$.

Now we must argue that the simulated proof is indistinguishable from the real proof and that the simulator doesn't abort. First observe that B_A, B_T and B_U are randomly sampled so the probability that the adversary has already queried these points (causing the simulator to fail) is negligible. Second observe that the provers commitment openings $\boldsymbol{x} + \alpha\mathbf{r}$ and the simulaters commitment openings $\boldsymbol{x}'$ are distributed uniformly at random. These random values completely determine the form of the honest provers output and the simulators output.

Indeed, for convenience denote the provers $\boldsymbol{x}_P = \boldsymbol{r} + \alpha\mathbf{x}$ Then the provers elements

$$\{B_A, B_T, B_U\} = \{\boldsymbol{x}_P \times \boldsymbol{G} - \alpha A, \boldsymbol{x}_P \times \boldsymbol{T} - \alpha Z_T, \boldsymbol{x}_P \times \boldsymbol{U} - \alpha Z_U\}$$

are distributed identically to the simulated elements. Likewise $\boldsymbol{\pi}, x$ are determined according to the same recursive argument. □

Lemma 4.0.2. *The algorithm in Steps 2 and 3 of the prover and verifier in Figures 4.1 and 4.2 is a knowledge sound argument for the relation R_{samemsm} assuming the* q-dlog *problem holds.*

Proof. The algorithms implement a generalised inner product argument with respect to the commitment scheme

$$(\boldsymbol{G}, \boldsymbol{k}, k) \xleftarrow{\$} \mathsf{Setup}(\mathbb{G})$$
$$\begin{pmatrix} \boldsymbol{x} \times \boldsymbol{G} \\ (\boldsymbol{k} \circ \boldsymbol{T}, \boldsymbol{k} \circ \boldsymbol{U}) \\ (kZ_T, kZ_U) \end{pmatrix} \leftarrow \mathsf{Commit} \left(\begin{array}{c|c} \boldsymbol{G} & \boldsymbol{x} \\ \boldsymbol{k} & (\boldsymbol{T}, \boldsymbol{U}) \\ k & (Z_T, Z_U) \end{array} \right)$$

and the inner product

$$\cdot : \mathbb{F} \times \mathbb{G}^2 \mapsto \mathbb{G}^2, \; x \cdot (T, U) = (xT, \; xU)$$

Note that Figures 4.1 and 4.2 have been optimised such that the commitments $(\boldsymbol{k}\circ\boldsymbol{T}, \boldsymbol{k}\circ\boldsymbol{U}, kZ_T, kZ_U)$ are neither computed nor sent. This is because the verifier can compute the openings of these commitments in the final round of recursion for itself.

By Theorem 7.0.5 it suffices to show that $(\mathsf{Setup}, \mathsf{Commit}, \cdot)$ is an inner product commitment. We first show that $(\mathsf{Setup}, \mathsf{Commit})$ is binding. We second show that $(\mathsf{Setup}, \mathsf{Commit})$ is doubly homomorphic. We third show the existence of a correct $\mathsf{Collapse}$ function (Definition 7.0.8). Together these suffice to prove the lemma.

Binding commitment: Let $\mathcal{A}$ be an adversary that breaks binding. We describe an adversary $\mathcal{B}$ against q-dlog. The adversary $\mathcal{B}$ takes as input $\boldsymbol{G}$ and sets $\boldsymbol{a}, \boldsymbol{k}, k$ to have random entries in $\mathbb{F}^\times$. Then $\mathcal{B}$ runs

$$\begin{pmatrix} A \\ (\boldsymbol{V}, \boldsymbol{W}) \\ (Z_V, Z_W) \end{pmatrix}, \begin{pmatrix} \boldsymbol{x} \\ (\boldsymbol{T}, \boldsymbol{U}) \\ (Z_T, Z_U) \end{pmatrix}, \begin{pmatrix} \boldsymbol{x}' \\ (\boldsymbol{T}', \boldsymbol{U}') \\ (Z'_T, Z'_U) \end{pmatrix} \leftarrow \mathcal{A}(\boldsymbol{G}, \boldsymbol{k}, k)$$

and then $\mathcal{B}$ returns $(\boldsymbol{x}, \boldsymbol{x}')$.

If $\mathcal{A}$ wins then $A = \boldsymbol{x} \times \boldsymbol{G} = \boldsymbol{x}' \times \boldsymbol{G}$ for $\boldsymbol{x} \neq \boldsymbol{x}'$ Thus $\mathcal{B}$ returns a correct $\mathsf{q\text{-}dlog}$ response and

$$\mathsf{Adv}^{\mathsf{binding}}_{\mathcal{A}}(\lambda) \leqslant \mathsf{Adv}^{\mathsf{q\text{-}dlog}}_{\mathcal{B}}(\lambda)$$

Doubly homomorphic commitment: First see that the key space is homomorphic

$$\begin{aligned}
\mathsf{Commit}\left(\begin{array}{c|c} \boldsymbol{G}+\boldsymbol{G}' & \boldsymbol{x} \\ \boldsymbol{k}+\boldsymbol{k}' & (\boldsymbol{T},\boldsymbol{U}) \\ k+k' & (Z_T, Z_U) \end{array}\right) &= \left(\begin{array}{c} \boldsymbol{x} \times (\boldsymbol{G}+\boldsymbol{G}') \\ ((\boldsymbol{k}+\boldsymbol{k}') \circ \boldsymbol{T}, (\boldsymbol{k}+\boldsymbol{k}') \circ \boldsymbol{U}) \\ ((k+k')Z_T, (k+k')Z_U) \end{array}\right) \\
&= \left(\begin{array}{c} \boldsymbol{x} \times \boldsymbol{G} \\ (\boldsymbol{k} \circ \boldsymbol{T}, \boldsymbol{k} \circ \boldsymbol{U}) \\ (kZ_T, kZ_U) \end{array}\right) + \left(\begin{array}{c} \boldsymbol{x} \times \boldsymbol{G}' \\ (\boldsymbol{k}' \circ \boldsymbol{T}, \boldsymbol{k}' \circ \boldsymbol{U}) \\ (k'Z_T, k'Z_U) \end{array}\right) \\
&= \mathsf{Commit}\left(\begin{array}{c|c} \boldsymbol{G} & \boldsymbol{x} \\ \boldsymbol{k} & (\boldsymbol{T},\boldsymbol{U}) \\ k & (Z_T, Z_U) \end{array}\right) + \mathsf{Commit}\left(\begin{array}{c|c} \boldsymbol{G}' & \boldsymbol{x} \\ \boldsymbol{k}' & (\boldsymbol{T},\boldsymbol{U}) \\ k' & (Z_T, Z_U) \end{array}\right)
\end{aligned}$$

Second see that the message space is homomorphic

$$\begin{aligned}
\mathsf{Commit}\left(\begin{array}{c|c} \boldsymbol{G} & \boldsymbol{x}+\boldsymbol{x}' \\ \boldsymbol{k} & (\boldsymbol{T},\boldsymbol{U}) + (\boldsymbol{T}',\boldsymbol{U}') \\ k & (Z_T, Z_U) + (Z'_T, Z'_U) \end{array}\right) &= \left(\begin{array}{c} (\boldsymbol{x}+\boldsymbol{x}') \times \boldsymbol{G} \\ (\boldsymbol{k} \circ (\boldsymbol{T}+\boldsymbol{T}'), \boldsymbol{k} \circ (\boldsymbol{U}+\boldsymbol{U}')) \\ k(Z_T + Z'_T, Z_U + Z'_U) \end{array}\right) \\
&= \left(\begin{array}{c} \boldsymbol{x} \times \boldsymbol{G} \\ (\boldsymbol{k} \circ \boldsymbol{T}, \boldsymbol{k} \circ \boldsymbol{U}) \\ (kZ_T, kZ_U) \end{array}\right) + \left(\begin{array}{c} \boldsymbol{x}' \times \boldsymbol{G} \\ (\boldsymbol{k} \circ \boldsymbol{T}', \boldsymbol{k} \circ \boldsymbol{U}') \\ k(Z'_T, Z'_U) \end{array}\right) \\
&= \mathsf{Commit}\left(\begin{array}{c|c} \boldsymbol{G} & \boldsymbol{x} \\ \boldsymbol{k} & (\boldsymbol{T},\boldsymbol{U}) \\ k & (Z_T, Z_U) \end{array}\right) + \mathsf{Commit}\left(\begin{array}{c|c} \boldsymbol{G} & \boldsymbol{x}' \\ \boldsymbol{k} & (\boldsymbol{T}',\boldsymbol{U}') \\ k & (Z'_T, Z'_U) \end{array}\right)
\end{aligned}$$

Collapsible commitment: Let $\mathsf{Collapse}$ be the function

$$\mathsf{Collapse}\left(\begin{array}{c} A \\ ((\boldsymbol{V}_1 \mid\mid \boldsymbol{V}_2), (\boldsymbol{W}_1 \mid\mid \boldsymbol{W}_2)) \\ (Z_V, Z_W) \end{array}\right) \mapsto \left(\begin{array}{c} A \\ (\boldsymbol{V}_1 + \boldsymbol{V}_2, \boldsymbol{W}_1 + \boldsymbol{W}_2) \\ (Z_V, Z_W) \end{array}\right)$$

Then

$$\begin{aligned}
\mathsf{Collapse}\left(\mathsf{Commit}\left(\begin{array}{c|c} \boldsymbol{G} \mid\mid \boldsymbol{G}' & \boldsymbol{x} \mid\mid \boldsymbol{x} \\ \boldsymbol{k} \mid\mid \boldsymbol{k}' & ((\boldsymbol{T} \mid\mid \boldsymbol{T}), (\boldsymbol{U} \mid\mid \boldsymbol{U})) \\ k & (Z_T, Z_U) \end{array}\right)\right) &= \mathsf{Collapse}\left(\begin{array}{c} \boldsymbol{x} \times \boldsymbol{G} + \boldsymbol{x} \times \boldsymbol{G}' \\ ((\boldsymbol{k} \circ \boldsymbol{T} \mid\mid \boldsymbol{k}' \circ \boldsymbol{T}), (\boldsymbol{k} \circ \boldsymbol{U} \mid\mid \boldsymbol{k}' \circ \boldsymbol{U})) \\ (kZ_T, kZ_U) \end{array}\right) \\
&= \left(\begin{array}{c} \boldsymbol{x} \times (\boldsymbol{G}+\boldsymbol{G}') \\ ((\boldsymbol{k} \circ \boldsymbol{T} + \boldsymbol{k}' \circ \boldsymbol{T}), (\boldsymbol{k} \circ \boldsymbol{U} + \boldsymbol{k}' \circ \boldsymbol{U})) \\ (kZ_T, kZ_U) \end{array}\right) \\
&= \left(\begin{array}{c} \boldsymbol{x} \times (\boldsymbol{G}+\boldsymbol{G}') \\ ((\boldsymbol{k}+\boldsymbol{k}') \circ \boldsymbol{T}, (\boldsymbol{k}+\boldsymbol{k}') \circ \boldsymbol{U}) \\ (kZ_T, kZ_U) \end{array}\right) \\
&= \mathsf{Commit}\left(\begin{array}{c|c} \boldsymbol{G}+\boldsymbol{G}' & \boldsymbol{x} \\ \boldsymbol{k}+\boldsymbol{k}' & (\boldsymbol{T},\boldsymbol{U}) \\ k & (Z_T, Z_U) \end{array}\right)
\end{aligned}$$

as required.

□

Theorem 4.0.3 (SameMultiScalar argument is knowledge-sound)**.** *SameMultiScalar argument described in Figures 4.1 and 4.2 is knowledge-sound in the random oracle model assuming the* q-dlog *is hard.*

Proof. We design an extractor $\mathcal{X}_{\mathsf{samemsm}}$ such that: if there exists an adversary $\mathcal{A}$ that convinces the verifier with non-negligible probability then with overwhelming probability $\mathcal{X}_{\mathsf{samescalar}}$ returns field elements $\boldsymbol{x}$ such that

$$((A, Z_T, Z_U, \boldsymbol{T}, \boldsymbol{U});\ \boldsymbol{x}) \in R_{\mathsf{samemsm}}.$$

By Lemma 4.0.2, whenever an adversary $\mathcal{B}$ outputs a valid proof, there exists an extractor $\mathcal{X}_{\mathcal{B}}$ that takes as input $\mathcal{B}$'s transcript and outputs $\boldsymbol{x}$ such that

$$(B_A, B_T, B_U) + \alpha(A, Z_T, Z_U) = (\boldsymbol{x} \times \boldsymbol{G},\ \boldsymbol{x} \times \boldsymbol{T},\ \boldsymbol{x} \times \boldsymbol{U})$$

such that $\mathsf{Adv}^{\mathsf{samemsm2}}_{\mathcal{B},\mathcal{X}_B}(\lambda)$ is negligible assuming the dlog problem is hard. Here samemsm2 refers to the knowledge-soundness game for the (non-zk) protocol in steps 2 and 3 of Figures 4.1 and 4.2.

The extractor $\mathcal{X}_{\mathsf{samemsm}}$ works as follows

1. Randomly sample coins ω.
2. Define an adversary $\mathcal{B}_1$ that behaves as follows:
   - Generate $\mathsf{crs} = \boldsymbol{G}$ and set $\mathsf{trans} = 0$. While $\mathsf{trans} = 0$ run $\mathsf{trans} \leftarrow \mathsf{Game}^{\mathsf{fork}}_{\mathcal{A}}(\mathsf{crs}; \omega)$ for $\mathsf{Game}^{\mathsf{fork}}_{\mathcal{A}}(\mathsf{crs})$ defined in Section 7.0.2. Parse
   $$(\alpha, \alpha'), ((A, Z_T, Z_U, \boldsymbol{T}, \boldsymbol{U});\ (B_A, B_T, B_U), (\boldsymbol{\pi}, x), (\boldsymbol{\pi'}, x')) = \mathsf{trans}$$
   - Return $(B_A + \alpha A, B_T + \alpha Z_T, B_U + \alpha Z_U), (\boldsymbol{\pi}, x)$
3. Define an adversary $\mathcal{B}_2$ that behaves as follows:
   - Compute
   $$(\alpha, \alpha'), ((A, Z_T, Z_U, \boldsymbol{T}, \boldsymbol{U});\ (B_A, B_T, B_U), (\boldsymbol{\pi}, x), (\boldsymbol{\pi'}, x')) = \mathsf{trans}$$
   the same as $\mathcal{B}_0$
   - Return $(B_A + \alpha' A, B_T + \alpha' Z_T, B_U + \alpha' Z_U), (\boldsymbol{\pi'}, x')$
4. Let $\mathcal{X}_{\mathcal{B}_1}$ be $\mathcal{B}_1$'s samemsm2 extractor. Extract $\boldsymbol{y}$ such that $B_A + \alpha A = \boldsymbol{y} \times \boldsymbol{G}$, $B_T + \alpha Z_T = \boldsymbol{y} \times \boldsymbol{T}$, $B_U + \alpha Z_U = \boldsymbol{y} \times \boldsymbol{U}$.
5. Let $\mathcal{X}_{\mathcal{B}_2}$ be $\mathcal{B}_2$'s samemsm2 extractor. Extract $\boldsymbol{y'}$ such that $B_A + \alpha' A = \boldsymbol{y'} \times \boldsymbol{G}$, $B_T + \alpha' Z_T = \boldsymbol{y'} \times \boldsymbol{T}$, $B_U + \alpha' Z_U = \boldsymbol{y'} \times \boldsymbol{U}$.
6. Compute $\boldsymbol{x} = (\alpha - \alpha')^{-1}(\boldsymbol{y} - \boldsymbol{y'})$ and return $\boldsymbol{x}$.

We must show that whenever $\mathcal{A}$ convinces the verifier then $\mathcal{X}_{\mathsf{samemsm}}$ succeeds.

First see that $\mathcal{X}_{\mathsf{samemsm}}$ terminates in polynomial time with overwhelming probability. Let τ be the run time of $\mathcal{A}$, ϵ be the probability that $\mathcal{A}$ outputs a valid response and q_H be the total number of hash queries that $\mathcal{A}$ can make. Assuming $\frac{8q_H}{|\mathbb{F}|} < \epsilon$ then the game $\mathsf{Game}^{\mathsf{fork}}_{\mathcal{A}}(\mathsf{par})$ runs in time at most $\frac{8\tau q_H}{\epsilon} \cdot \ln(8/\epsilon)$ and is successful with probability at least $\epsilon/8$. The expected run time of $\mathcal{B}_1, \mathcal{B}_2$ is less than $\tau q_H \cdot \ln(8/\epsilon)$, which is polynomial time assuming that τ, ϵ and q_H are polynomial in the security parameter. By Lemma 4.0.2 this means that the expected run times of $\mathcal{X}_{\mathcal{B}_1}$ and $\mathcal{X}_{B_2}$ are also polynomial.

We show that for all extractors $\mathcal{X}_{\mathcal{B}_1}, \mathcal{X}_{\mathcal{B}_2}$ we have that

$$\mathsf{Adv}_{\mathcal{A},\mathcal{X}_{\mathsf{samemsm}}}(\lambda) \leqslant \mathsf{Adv}^{\mathsf{samemsm2}}_{\mathcal{B}_1,\mathcal{X}_1}(\lambda) + \mathsf{Adv}^{\mathsf{samemsm2}}_{\mathcal{B}_2,\mathcal{X}_2}(\lambda)$$

We proceed via a series of games $\mathsf{Game}_1, \mathsf{Game}_2$ such that

$$\begin{aligned}
\mathsf{Adv}_{\mathcal{A},\mathcal{X}_{\mathsf{samemsm}}}(\lambda) &\leqslant \mathsf{Adv}^{\mathsf{samemsm2}}_{\mathcal{B}_1,\mathcal{X}_{\mathcal{B}_1}}(\lambda) + \mathsf{Adv}^{\mathsf{Game}_1}_{\mathcal{A},\mathcal{X}_{\mathsf{samemsm}}}(\lambda) \\
\mathsf{Adv}^{\mathsf{Game}_1}_{\mathcal{A},\mathcal{X}_{\mathsf{samemsm}}}(\lambda) &\leqslant \mathsf{Adv}^{\mathsf{samemsm2}}_{\mathcal{B}_2,\mathcal{X}_{\mathcal{B}_2}}(\lambda) + \mathsf{Adv}^{\mathsf{Game}_2}_{\mathcal{A},\mathcal{X}_{\mathsf{samemsm}}}(\lambda) \\
\mathsf{Adv}^{\mathsf{Game}_2}_{\mathcal{A},\mathcal{X}_{\mathsf{samemsm}}}(\lambda) &= 0
\end{aligned}$$

which combined give us our final result.

$\underline{\mathsf{Game}_0 \mapsto \mathsf{Game}_1}$: Let Game_0 be the initial knowledge-soundness game. Then Game_1 is identical except in the following case. If $\mathcal{A}$ outputs a verifying proof then define $\mathcal{B}_1$ be the adversary as in $\mathcal{X}_{\mathsf{samemsm}}$ that returns $(B_A + \alpha A, B_T + \alpha Z_T, B_U + \alpha Z_U), (\boldsymbol{\pi}, x)$ and $\mathcal{X}_{\mathcal{B}_1}$ its corresponding extractor. Return 0 if $\mathcal{X}_{\mathcal{B}_1}$ fails. If Game_0 returns 1 but Game_1 returns 0 then this means that $(\boldsymbol{\pi}, x)$ verifies and $\mathcal{X}_{\mathcal{B}_1}$ fails, and hence that $\mathcal{B}_1$ succeeds. By Lemma 4.0.2 the probability of this is negligible if the q-dlog assumption holds.

$\underline{\mathsf{Game}_1 \mapsto \mathsf{Game}_2}$: Define Game_2 to be identical to Game_1 except in the following case. If $\mathcal{A}$ outputs a verifying proof then define $\mathcal{B}_2$ be the adversary as in $\mathcal{X}_{\mathsf{samemsm}}$ that returns $(B_A + \alpha' A, B_T + \alpha' Z_T, B_U + \alpha' Z_U), (\boldsymbol{\pi}', x')$ and $\mathcal{X}_{\mathcal{B}_2}$ its corresponding extractor. Return 0 if $\mathcal{X}_{\mathcal{B}_2}$ fails. If Game_1 returns 1 but Game_2 returns 0 then this means that $(\boldsymbol{\pi}', x')$ verifies and $\mathcal{X}_{\mathcal{B}_2}$ fails, and hence that $\mathcal{B}_2$ succeeds. By Lemma 4.0.2 the probability of this is negligible if the q-dlog assumption holds.

$\underline{\mathsf{Game}_2 \mapsto \mathsf{negl}(\lambda)}$: See that

$$\begin{pmatrix} B_A + \alpha A \\ B_T + \alpha Z_T \\ B_U + \alpha Z_U \end{pmatrix} = \begin{pmatrix} \boldsymbol{y} \times \boldsymbol{G} \\ \boldsymbol{y} \times \boldsymbol{T} \\ \boldsymbol{y} \times \boldsymbol{U} \end{pmatrix} \text{ and } \begin{pmatrix} B_A + \alpha' A \\ B_T + \alpha' Z_T \\ B_U + \alpha' Z_U \end{pmatrix} = \begin{pmatrix} \boldsymbol{y}' \times \boldsymbol{G} \\ \boldsymbol{y}' \times \boldsymbol{T} \\ \boldsymbol{y}' \times \boldsymbol{U} \end{pmatrix}$$

$$\Rightarrow \begin{pmatrix} (\alpha - \alpha')A \\ (\alpha - \alpha')Z_T \\ (\alpha - \alpha')Z_U \end{pmatrix} = \begin{pmatrix} (\boldsymbol{y} - \boldsymbol{y}') \times \boldsymbol{G} \\ (\boldsymbol{y} - \boldsymbol{y}') \times \boldsymbol{T} \\ (\boldsymbol{y} - \boldsymbol{y}') \times \boldsymbol{U} \end{pmatrix}$$

Thus the value $\boldsymbol{x} = (\alpha - \alpha')^{-1}(\boldsymbol{y} - \boldsymbol{y}')$ output by $\mathcal{X}_{\mathsf{samemsm}}$ in Game_2 is a correct witness for $(A, Z_T, Z_U, \boldsymbol{T}, \boldsymbol{U})$. □

$\underline{\mathsf{Prove}_{\mathsf{samemsm}}(\mathsf{crs}_{\mathsf{samemsm}};\ (A, Z_T, Z_U, \boldsymbol{T}, \boldsymbol{U});\ \boldsymbol{x})}$

Step 1:

$\boldsymbol{G} \leftarrow \mathsf{parse}(\mathsf{crs}_{\mathsf{samemsm}})$

$\boldsymbol{r} \xleftarrow{\$} \mathbb{F}^n$

$B_A \leftarrow \boldsymbol{r} \times \boldsymbol{G}$

$B_T \leftarrow \boldsymbol{r} \times \boldsymbol{T}$

$B_U \leftarrow \boldsymbol{r} \times \boldsymbol{U}$

$\alpha \leftarrow \mathsf{Hash}(A, Z_T, Z_U, \boldsymbol{T}, \boldsymbol{U}, B_A, B_T, B_U)$

$\boldsymbol{x} \leftarrow \boldsymbol{r} + \alpha \boldsymbol{x}$

Step 2:

$m \leftarrow n$

while $1 \leqslant j \leqslant m$:

$n \leftarrow \frac{n}{2}$

$L_{A,j} \leftarrow \boldsymbol{x}_{[:n]} \times \boldsymbol{G}_{[n:]}$

$L_{T,j} \leftarrow \boldsymbol{x}_{[:n]} \times \boldsymbol{T}_{[n:]}$

$L_{U,j} \leftarrow \boldsymbol{x}_{[:n]} \times \boldsymbol{U}_{[n:]}$

$R_{A,j} \leftarrow \boldsymbol{x}_{[n:]} \times \boldsymbol{G}_{[:n]}$

$R_{T,j} \leftarrow \boldsymbol{x}_{[n:]} \times \boldsymbol{T}_{[:n]}$

$R_{U,j} \leftarrow \boldsymbol{x}_{[n:]} \times \boldsymbol{U}_{[:n]}$

$\pi_j \leftarrow (L_{A,j}, L_{T,j}, L_{U,j}, R_{A,j}, R_{T,j}, R_{U,j})$

$\gamma_j \leftarrow \mathsf{Hash}(\pi_j)$

$\boldsymbol{x} \leftarrow \boldsymbol{x}_{[:n]} + \gamma_j^{-1} \boldsymbol{x}_{[n:]}$

$\boldsymbol{T} \leftarrow \boldsymbol{T}_{[:n]} + \gamma_j \boldsymbol{T}_{[n:]}$

$\boldsymbol{U} \leftarrow \boldsymbol{U}_{[:n]} + \gamma_j \boldsymbol{U}_{[n:]}$

$\boldsymbol{G} \leftarrow \boldsymbol{G}_{[:n]} + \gamma_j \boldsymbol{G}_{[n:]}$

Step 3:

$x \leftarrow x_1$

return $(B_A, B_T, B_U, \boldsymbol{\pi}, x)$

Figure 4.1: Proving algorithm to demonstrate that $(A, Z_T, Z_U) = (\boldsymbol{x} \times \boldsymbol{G}, \boldsymbol{x} \times \boldsymbol{T}, \boldsymbol{x} \times \boldsymbol{U})$ for some vector of field elements $\boldsymbol{x}$.

$\underline{\mathsf{Verify}_{\mathsf{samemsm}}(\mathsf{crs}_{\mathsf{samemsm}};\ \phi_{\mathsf{samemsm}};\ \pi_{\mathsf{samemsm}})}$

Step 1:

$\boldsymbol{G} \leftarrow \mathsf{parse}(\mathsf{crs}_{\mathsf{samemsm}})$

$(A, Z_T, Z_U, \boldsymbol{T}, \boldsymbol{U}) \leftarrow \mathsf{parse}(\phi_{\mathsf{samemsm}})$

$(B_A, B_T, B_U, \boldsymbol{\pi}, x) \leftarrow \mathsf{parse}(\pi_{\mathsf{samemsm}})$

$\alpha \leftarrow \mathsf{Hash}(A, Z_T, Z_U, \boldsymbol{T}, \boldsymbol{U}, B_A, B_T, B_U)$

$A \leftarrow B_A + \alpha A$

$Z_T \leftarrow B_T + \alpha Z_T$

$Z_U \leftarrow B_U + \alpha Z_U$

Step 2:

$m \leftarrow n$

while $1 \leqslant j \leqslant m$:

$n \leftarrow \frac{n}{2}$

$(L_{A,j}, L_{T,j}, L_{U,j}, R_{A,j}, R_{T,j}, R_{U,j}) \leftarrow \mathsf{parse}(\pi_j)$

$\gamma_j \leftarrow \mathsf{Hash}(\pi_j)$

$A \leftarrow \gamma_j L_{A,j} + A + \gamma_j^{-1} R_{A,j}$

$Z_T \leftarrow \gamma_j L_{T,j} + Z_T + \gamma_j^{-1} R_{T,j}$

$Z_U \leftarrow \gamma_j L_{U,j} + Z_U + \gamma_j^{-1} R_{U,j}$

$\boldsymbol{G} \leftarrow \boldsymbol{G}_{[:n]} + \gamma_j \boldsymbol{G}_{[n:]}$

$\boldsymbol{T} \leftarrow \boldsymbol{T}_{[:n]} + \gamma_j \boldsymbol{T}_{[n:]}$

$\boldsymbol{U} \leftarrow \boldsymbol{U}_{[:n]} + \gamma_j \boldsymbol{U}_{[n:]}$

Step 3:

$\mathsf{check}_1 \leftarrow A \stackrel{?}{=} xG_1$

$\mathsf{check}_2 \leftarrow Z_T \stackrel{?}{=} xT_1$

$\mathsf{check}_3 \leftarrow Z_U \stackrel{?}{=} xU_1$

return $(\mathsf{check}_1, \mathsf{check}_2, \mathsf{check}_3) = (1, 1, 1)$

else return 0.

Figure 4.2: Verify algorithm to check that that $(A, Z_T, Z_U) = (\boldsymbol{x} \times \boldsymbol{G},\ \boldsymbol{x} \times \boldsymbol{T},\ \boldsymbol{x} \times \boldsymbol{U})$ for some vector of field elements $\boldsymbol{x}$.

Chapter 5

Same Permutation Argument

In this chapter we discuss a zero knowledge argument for the relation

$$R_{\mathsf{sameperm}} = \left\{ (A, M, \boldsymbol{a}); \quad (\sigma(), \boldsymbol{r}_A, \boldsymbol{r}_M) \;\middle|\; \begin{array}{l} A = \sigma(\boldsymbol{a}) \times \boldsymbol{g} + \boldsymbol{r}_A \times \boldsymbol{h} \\ M = \sigma(1, \ldots, \ell) \times \boldsymbol{g} + \boldsymbol{r}_M \times \boldsymbol{h} \\ \sigma() \in \text{permutations over } [1, \ldots, \ell] \end{array} \right\}$$

This chapter consists of three sections each discussing a unique argument: (1) SamePerm argument; (2) GrandProd argument; and (3) an inner product for pedersen commitments. SamePerm uses GrandProd as a subroutine. The grand-product then uses the inner-product as a subroutine.

5.1 Same Permutation Argument

We begin by giving a full overview of the same-permutation construction. For an informal overview see Section 5.1.2 and for the formal construction see Figure 5.1. The security arguments are given in Theorems 5.1.1 and 5.1.2. The construction makes use of GrandProd argument as a subprotocol. We specify GrandProd relations below and describe GrandProd construction in Section 5.2.

5.1.1 Neff's Trick

The argument takes advantage of an observation (first applied in the proof context by Neff [Nef01]) that two polynomials are equal if and only if their roots are the same up to permutation. In other words

$$\sigma(\boldsymbol{a}) = \boldsymbol{c} \quad \Leftrightarrow \quad (a_1 + Y)(a_2 + Y) \cdots (a_\ell + Y) = (c_1 + Y)(c_2 + Y) \cdots (c_\ell + Y)$$

as polynomials of Y. We can additionally bind $\boldsymbol{a}$ and $\boldsymbol{c}$ to a specific permutation $\sigma()$ through including an additional indeterminate X. Indeed whenever the polynomial equation

$$(a_1 + X + Y)(a_2 + 2X + Y) \cdots (a_\ell + \ell X + Y) = (c_1 + m_1 X + Y)(c_2 + m_2 X + Y) \cdots (c_\ell + m_\ell X + Y)$$

holds we have that there exists $\sigma()$ such that

$$\sigma(a_1 + X, a_2 + 2X, \ldots, a_\ell + \ell X) = (c_1 + m_1 X, c_2 + m_2 X, \ldots, c_\ell + m_\ell X)$$

This implies that $\sigma()$ is a permutation, $\sigma(\boldsymbol{a}) = \boldsymbol{c}$ and $\sigma(1, \ldots, \ell) = \boldsymbol{m}$.

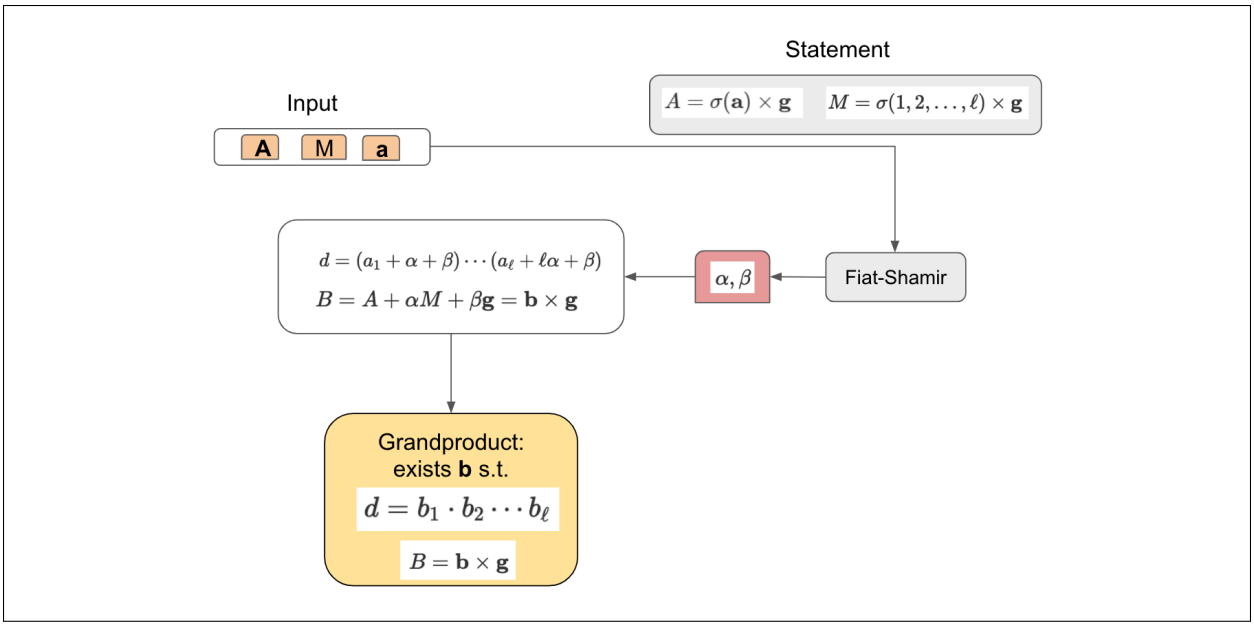

Figure 5.1: Overview of SamePerm argument. The protocol uses GrandProd argument as a subroutine.

5.1.2 Informal Overview

The prover will take as input the $A, M, \boldsymbol{a}$ and aims to prove knowledge of $\sigma()$ such that:

- $A = \sigma(\boldsymbol{a}) \times \boldsymbol{g}$ is a commitment to $\sigma(\boldsymbol{a})$
- $M = \sigma(1, 2, \ldots, \ell) \times \boldsymbol{g}$ is a commitment to $\sigma()$

The verifier wishes to check that A and M are commitments to $\boldsymbol{c}$ and $\boldsymbol{m}$ respectively such that

$$(a_1 + X + Y)(a_2 + 2X + Y) \cdots (a_\ell + \ell X + Y) = (c_1 + m_1 X + Y)(c_2 + m_2 X + Y) \cdots (c_\ell + m_\ell X + Y)$$

Initially all the public inputs $(A, M, \boldsymbol{a})$ are hashed to get challenges α, β and we must show that

$$(a_1 + \alpha + \beta)(a_2 + 2\alpha + \beta) \cdots (a_\ell + \ell\alpha + \beta) = (c_1 + m_1\alpha + \beta)(c_2 + m_2\alpha + \beta) \cdots (c_\ell + m_\ell\alpha + \beta)$$

By the Schwartz-Zippel Lemma this implies that the polynomial expression holds except with negligible probability.

Next the prover and verifier both compute values $p = \prod_{i=1}^{\ell}(a_i + i\alpha + \beta)$ and $B = A + \alpha M + \boldsymbol{\beta} \times \boldsymbol{g}$ where $\boldsymbol{\beta} = (\beta, \beta, \ldots, \beta)$. By the homomorphic properties of the Pedersen commitment we see that B is thus a commitment to

$$\boldsymbol{b} = \boldsymbol{c} + \alpha\boldsymbol{m} + \beta\mathbf{1} = (c_1 + m_1\alpha + \beta, c_2 + m_2\alpha + \beta, \ldots, c_\ell + m_\ell\alpha + \beta)$$

Here $\mathbf{1} = (1, \ldots, 1)$ is the length ℓ vector where every entry equals 1. Then the prover uses a grand-product argument to describe knowledge of $\boldsymbol{b}$ such that B is a commitment to $\boldsymbol{b}$ and p is a grandproduct of $\boldsymbol{b}$. This implies that

$$\prod_{i=1}^{\ell}(a_i + i\alpha + \beta) = \prod_{i=1}^{\ell}(c_i + m_i\alpha + \beta)$$

and hence that $\boldsymbol{m} = \sigma(1, \ldots, \ell)$, $\boldsymbol{c} = \sigma(\boldsymbol{a})$ for some $\sigma()$.

Note that the full same-permutation protocol has some additional masking values that are included to ensure zero-knowledge. For simplicity we have ignored these terms in this overview.

5.1.3 GrandProd Relation

GrandProd relation demonstrates that given public input $(B, p) \in \mathbb{G} \times \mathbb{F}$ there exists $\boldsymbol{b}$ such that B is a commitment to $\boldsymbol{b}$ and p is the grand-product of $\boldsymbol{b}$. This commitment is blinded. In other words

$$R_{\mathsf{gprod}} = \left\{ \begin{array}{l|l} (B, p); \quad (\boldsymbol{b}, \boldsymbol{r}_B) & B = \boldsymbol{b} \times \boldsymbol{g} + \boldsymbol{r}_B \times \boldsymbol{h} \\ & p = \prod_{i=1}^{\ell} b_i \end{array} \right\}$$

$\mathsf{Prove}_{\mathsf{sameperm}}(\mathsf{crs}_{\mathsf{sameperm}};\ (A, M, \boldsymbol{a});\ (\sigma(), \boldsymbol{r}_A, \boldsymbol{r}_M))$

Step 1:
$(\boldsymbol{g}, \boldsymbol{h}, H) \leftarrow \mathsf{parse}(\mathsf{crs}_{\mathsf{sameperm}})$
$(\alpha, \beta) \leftarrow \mathsf{Hash}(A, M, \boldsymbol{a})$

Step 2:
$\boldsymbol{b} \leftarrow \sigma(\boldsymbol{a_i}) + \sigma((1 \ldots \ell))\alpha + \beta$
$p \leftarrow \prod_{i=1}^{\ell} b_i$
$B \leftarrow A + \alpha M + \boldsymbol{\beta} \times \boldsymbol{g}$
$\boldsymbol{r}_B \leftarrow \boldsymbol{r}_A + \alpha \boldsymbol{r}_M$
$\pi_{\mathsf{gprod}} \leftarrow \mathsf{Prove}_{\mathsf{gprod}}((\boldsymbol{g}, \boldsymbol{h}, H);\ (B, p);\ (\boldsymbol{b}, \boldsymbol{r}_B))$

return $(B, \pi_{\mathsf{gprod}})$

$\mathsf{Verify}_{\mathsf{sameperm}}(\mathsf{crs}_{\mathsf{sameperm}}, ;\ \phi_{\mathsf{sameperm}};\ \pi_{\mathsf{sameperm}})$

Step 1:
$(\boldsymbol{g}, \boldsymbol{h}, H) \leftarrow \mathsf{parse}(\mathsf{crs}_{\mathsf{sameperm}})$
$(A, M, \boldsymbol{a}) \leftarrow \mathsf{parse}(\phi_{\mathsf{sameperm}})$
$(B, \pi_{\mathsf{gprod}}) \leftarrow \mathsf{parse}(\pi_{\mathsf{sameperm}})$
$(\alpha, \beta) \leftarrow \mathsf{Hash}(A, M, \boldsymbol{a})$

Step 2:
$p \leftarrow \prod_{i=1}^{\ell}(a_i + i\alpha + \beta)$
$\mathsf{check}_1 \leftarrow B \stackrel{?}{=} A + \alpha M + \boldsymbol{\beta} \times \boldsymbol{g}$
$\mathsf{check}_2 \leftarrow \mathsf{Verify}_{\mathsf{gprod}}((\boldsymbol{g}, \boldsymbol{h}, H);\ (B, p);\ \pi_{\mathsf{gprod}})$

return 1 if $(\mathsf{check}_1, \mathsf{check}_2) = (1, 1)$
else return 0

Figure 5.2: Proving and verifying algorithms to demonstrate that A, M are commitments to $\sigma(\boldsymbol{a}), \sigma(1, \ldots, \ell)$ for some permutation σ. Here we denote $\boldsymbol{\beta} = (\beta, \beta, \ldots, \beta)$

5.1.4 Full Zero Knowledge Same-Permutation Construction

The full zero-knowledge construction for SamePerm argument is given in Figure 5.2. In the first step the prover and verifier both hash the instance $(A, M, \boldsymbol{a})$ to get a challenge α, β.

In the second step the prover and verifer both compute the grandproduct $p \leftarrow \prod_{i=1}^{\ell}(a_i + i\alpha + \beta)$ and a value B which is a commitment to $\boldsymbol{b} = \boldsymbol{c} + \alpha\boldsymbol{m} + \boldsymbol{\beta}$ where

$$A = \boldsymbol{c} \times \boldsymbol{g} + \boldsymbol{r}_A \times \boldsymbol{h} \ \wedge \ M = \boldsymbol{m} \times \boldsymbol{g} + \boldsymbol{r}_M \times \boldsymbol{h}$$

and where $\boldsymbol{\beta}$ is a vector of length ℓ in which every entry is β. For later optimisations to the verifier, the verifier checks that B is correct rather than computing the value for itself. The prover additionally computes the randomness $\boldsymbol{r}_B$ of the commitment B such that

$$B = \boldsymbol{b} \times \boldsymbol{g} + \boldsymbol{r}_B \times \boldsymbol{h}$$

Finally the prover runs a grandproduct argument to demonstrate that the prover knows $(\boldsymbol{b}, \boldsymbol{r}_B)$ such that B is as above and $p = \prod_{i=1}^{\ell} b_i$.

The final proof is simply the grand-product proof π_{gprod} and thus π_{sameperm} is zero-knowledge provided that π_{gprod} is zero-knowledge.

5.1.5 Security

Theorem 5.1.1 (Same-permutation argument is zero-knowledge)**.** *If the grandproduct argument is zero-knowledge, then the same-permutation argument described in Figures 5.2 and 5.2 is zero-knowledge.*

Proof. We design a simulator Simulate that takes as input an instance

$$(A, M, \boldsymbol{a})$$

and outputs a proof π_{sameperm} that is indistinguishable from a proof generated by an honest prover that knows the witness. The simulator can program the random oracle on any points yet to be determined by the adversary.

During the setup the simulator chooses the $\mathsf{crs}_{\mathsf{gprod}} = (\boldsymbol{g}, \boldsymbol{h}, H)$ uniformly at random. During proving the simulator Simulate proceeds as follows

1. They compute p, B identically to the honest prover.
2. They run
$$\pi_{\mathsf{gprod}} = \mathsf{Simulate}_{\mathsf{dl\text{-}inner}}((\boldsymbol{g}, \boldsymbol{h}, H), (B, p))$$
and return π_{gprod}.

Now we must argue that the simulated proof is indistinguishable from the real proof.

We design an adversary $\mathcal{B}$ such that

$$\mathsf{Adv}_{\mathcal{A}}(\lambda) \leqslant 2\mathsf{Adv}^{\mathsf{gprod}}_{\mathcal{B}}(\lambda)$$

$\underline{\mathsf{Game}_0 \mapsto \mathsf{Game}_1}$: Let Game_0 be the initial zero-knowledge game. Define Game_1 to run identicallly to Game_0 except that, the crs and the π_{gprod} proof are generated by the gprod simulator for both $b = 0$ and $b = 1$.

Let $\mathcal{B}$ be an adversary against the gprod zero-knowledge game. Then $\mathcal{B}$ simulates the zero-knowledge game for $\mathcal{A}$. It takes as input crs and runs $\mathcal{A}(\mathsf{crs})$ on the same input. It flips a coin to get $b \in \{0,1\}$. When $\mathcal{A}$ makes a prover query, if $b = 0$ then $\mathcal{B}$ generates (B, p) honestly and queries its oracle on input (B, p) and $(\boldsymbol{b}, \boldsymbol{r}_B)$ for $\boldsymbol{b} = \sigma(\boldsymbol{a}) + \alpha\sigma([1, \ell]) + \boldsymbol{\beta}$ and $\boldsymbol{r}_B = \boldsymbol{r}_A + \alpha \boldsymbol{r}_M$. It receives back a proof π_{gprod}. Then $\mathcal{B}$ returns π_{gprod}. If $b = 1$ then $\mathcal{B}$ runs the simulator to compute the response. When $\mathcal{A}$ returns b' then if $b = b'$ then $\mathcal{B}$ returns 0, else $\mathcal{B}$ returns 1.

Then

$$\Pr[\mathcal{B}(\mathsf{crs}) = 0 \mid \bar{b} = 0] = \frac{1}{2}\left(\Pr[\mathcal{A}(\mathsf{crs}) = 0 \mid \mathsf{Game}_0, b = 0] + \Pr[\mathcal{A}(\mathsf{crs}) = 1 \mid \mathsf{Game}_0, b = 1]\right)$$
$$\Pr[\mathcal{B}(\mathsf{crs}) = 1 \mid \bar{b} = 1] = \frac{1}{2}\left(2 - \Pr[\mathcal{A}(\mathsf{crs}) = 0 \mid \mathsf{Game}_1, b = 0] - \Pr[\mathcal{A}(\mathsf{crs}) = 1 \mid \mathsf{Game}_1, b = 1]\right)$$

and

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{zk}}_{\mathcal{B}}(\lambda) &= |1 - 2\Pr[\mathsf{Game}^{\mathsf{gprod}}_{\mathcal{B}}(\lambda)]| \\
&= |1 - \Pr[\mathcal{B}(\mathsf{crs}) = 0 \mid \bar{b} = 0] - \Pr[\mathcal{B}(\mathsf{crs}) = 1 \mid \bar{b} = 1]| \\
&= \Big|1 - \frac{1}{2}\left(\Pr[\mathcal{A}(\mathsf{crs}) = 0 \mid \mathsf{Game}_0, b = 0] + \Pr[\mathcal{A}(\mathsf{crs}) = 1 \mid \mathsf{Game}_0, b = 1]\right) \\
&\qquad - \left(1 - \Pr[\mathcal{B}(\mathsf{crs}) = 0 \mid \bar{b} = 1]\right)\Big| \\
&= \frac{1}{2}\Big|1 - \Pr[\mathcal{A}(\mathsf{crs}) = 0 \mid \mathsf{Game}_0, b = 0] - \Pr[\mathcal{A}(\mathsf{crs}) = 1 \mid \mathsf{Game}_0, b = 1] \\
&\qquad\qquad - 1 + \Pr[\mathcal{A}(\mathsf{crs}) = 0 \mid \mathsf{Game}_1, b = 0] + \Pr[\mathcal{A}(\mathsf{crs}) = 1 \mid \mathsf{Game}_0, b = 1]\Big| \\
&= \frac{1}{2}|\mathsf{Adv}^{\mathsf{Game}_0}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{\mathsf{Game}_1}_{\mathcal{A}}(\lambda)|
\end{aligned}$$

This means that

$$\mathsf{Adv}^{\mathsf{Game}_0}_{\mathcal{A}}(\lambda) \leqslant 2\mathsf{Adv}^{\mathsf{gprod}}_{\mathcal{B}}(\lambda) + \mathsf{Adv}^{\mathsf{Game}_1}_{\mathcal{A}}(\lambda)$$

$\underline{\mathsf{Game}_1 \mapsto 0}$: In Game_1 the simulated proofs are generated identically and are thus indistinguishable. □

Theorem 5.1.2 (SamePerm argument is knowledge-sound)**.** *If GrandProd argument is knowledge-sound, and the* q-dlog *assumption holds, then the SamePerm argument described in Figure 5.2 is knowledge-sound.*

Proof. We design an extractor $\mathcal{X}_{\mathsf{sameperm}}$ such that for any adversary $\mathcal{A}$ that convinces the verifier, with overwhelming probability returns either a discrete logarithm relation between $\boldsymbol{g}, \boldsymbol{h}$ or a permutation $\sigma()$ and randomness $\boldsymbol{r}_A$, $\boldsymbol{r}_M$ such that

$$((A, M, \boldsymbol{a});\ (\sigma(), \boldsymbol{r}_A, \boldsymbol{r}_M)) \in R_{\mathsf{sameperm}}.$$

By the knowledge-soundness of the grand-product argument there exists an extractor $\mathcal{X}_{\mathsf{gprod}}$ such that if $\mathcal{A}$ returns verifying $(B, p, \pi_{\mathsf{gprod}})$ then they return valid witnesses for their respective languages with overwhelming probability.

The extractor $\mathcal{X}_{\mathsf{sameperm}}$ works as follows

1. Randomly sample coins ω.
2. Define an adversary $\mathcal{B}_1$ that behaves as follows:
   - Generate $\mathsf{crs} = (\boldsymbol{g}, \boldsymbol{h}, H)$ and set $\mathsf{trans} = 0$. While $\mathsf{trans} = 0$ run $\mathsf{trans} \leftarrow \mathsf{Game}^{\mathsf{fork}}_{\mathcal{A}}(\mathsf{crs}; \omega)$ for $\mathsf{Game}^{\mathsf{fork}}_{\mathcal{A}}(\mathsf{crs})$ defined in Section 7.0.2. Parse
   $$((\alpha, \beta), (\alpha', \beta')), (A, M, \boldsymbol{a}); \ \cdot\ , (\pi_{\mathsf{gprod}}, \pi'_{\mathsf{gprod}}) = \mathsf{trans}$$
   - Return $(A + \alpha M + \boldsymbol{\beta} \times \boldsymbol{g}, \pi_{\mathsf{gprod}})$
3. Define an adversary $\mathcal{B}_2$ that behaves as follows:
   - Generate $\mathsf{crs} = (\boldsymbol{g}, \boldsymbol{h}, H)$ and set $\mathsf{trans} = 0$. While $\mathsf{trans} = 0$ run $\mathsf{trans} \leftarrow \mathsf{Game}^{\mathsf{fork}}_{\mathcal{A}}(\mathsf{crs}; \omega)$ for $\mathsf{Game}^{\mathsf{fork}}_{\mathcal{A}}(\mathsf{crs})$ defined in Section 7.0.2. Parse
   $$((\alpha, \beta), (\alpha', \beta')), (A, M, \boldsymbol{a}); \ \cdot, (\pi_{\mathsf{gprod}}, \pi'_{\mathsf{gprod}}) = \mathsf{trans}$$
   - Return $(A + \alpha' M + \boldsymbol{\beta}' \times \boldsymbol{g}, \pi'_{\mathsf{gprod}})$
4. Let $\mathcal{X}_{\mathcal{B}_1}$ be $\mathcal{B}_1$'s gprod extractor. Extract $\boldsymbol{b}$, $\boldsymbol{r}_B$ such that $B = \boldsymbol{b} \times \boldsymbol{g} + \boldsymbol{r}_B \times \boldsymbol{h}$ and $p = \prod_i b_i$.
5. Let $\mathcal{X}_{\mathcal{B}_2}$ be $\mathcal{B}_2$'s gprod extractor. Extract $\boldsymbol{b}'$, $\boldsymbol{r}'_B$ such that $B' = \boldsymbol{b}' \times \boldsymbol{g} + \boldsymbol{r}'_B \times \boldsymbol{h}$ and $p' = \prod_i b_i$.
6. Set $\boldsymbol{m} \leftarrow (\alpha - \alpha')^{-1}((\boldsymbol{b} - \boldsymbol{\beta}) - (\boldsymbol{b}' - \boldsymbol{\beta}'))$
7. Set $\boldsymbol{r}_M \leftarrow (\alpha - \alpha')^{-1}(\boldsymbol{r}_B - \boldsymbol{r}'_B)$
8. Set $\boldsymbol{r}_A \leftarrow \boldsymbol{r}_B - \alpha \boldsymbol{r}_M$ and return $\boldsymbol{m}, \boldsymbol{r}_A, \boldsymbol{r}_M$.

We must show that whenever $\mathcal{A}$ convinces the verifier, then either $\mathcal{X}_{\mathsf{sameperm}}$ succeeds or we can extract a discrete logarithm relation between $(\boldsymbol{g} \ || \ \boldsymbol{h})$. First see that $\mathcal{X}_{\mathsf{sameperm}}$ terminates in polynomial time. Let τ be the run time of $\mathcal{A}$, ϵ be the probability that $\mathcal{A}$ outputs a valid response and q_H be the total number of hash queries that $\mathcal{A}$ can make. Assuming $\frac{8q_H}{|\mathbb{F}|} < \epsilon$ then the game $\mathsf{Game}^{\mathsf{fork}}_{\mathcal{A}}(\mathsf{par})$ runs in time at most $\frac{8\tau q_H}{\epsilon} \cdot \ln(8/\epsilon)$ and is successful with probability at least $\epsilon/8$. The expected run time of $\mathcal{B}_1, \mathcal{B}_2$ is less than $\tau q_H \cdot \ln(8/\epsilon)$, which is polynomial time assuming that τ, ϵ and q_H are polynomial in the security parameter. By the knowledge soundness of the gprod argument this means that the expected run times of $\mathcal{X}_{\mathcal{B}_1}$ and $\mathcal{X}_{B_2}$ are also polynomial.

We design an $\mathcal{B}_3, \mathcal{B}_4$ such that for all extractors $\mathcal{X}_{\mathcal{B}_1}, \mathcal{X}_{\mathcal{B}_2}, \mathcal{X}_{\mathcal{B}_3}$ we have that

$$\mathsf{Adv}_{\mathcal{A}, \mathcal{X}_{\mathsf{sameperm}}}(\lambda) \leqslant \mathsf{Adv}^{\mathsf{gprod}}_{\mathcal{B}_1, \mathcal{X}_1}(\lambda) + \mathsf{Adv}^{\mathsf{gprod}}_{\mathcal{B}_2, \mathcal{X}_2}(\lambda) + \mathsf{Adv}^{\mathsf{gprod}}_{\mathcal{B}_3, \mathcal{X}_3}(\lambda) + \mathsf{Adv}^{\mathsf{q\text{-}dlog}}_{\mathcal{B}_4}(\lambda) + \frac{q_H}{|\mathbb{F}|}$$

We proceed via a series of games $\mathsf{Game}_1, \mathsf{Game}_2, \mathsf{Game}_3$ such that

$$\begin{aligned}
\mathsf{Adv}_{\mathcal{A}, \mathcal{X}_{\mathsf{sameperm}}}(\lambda) &\leqslant \mathsf{Adv}^{\mathsf{gprod}}_{\mathcal{B}_1, \mathcal{X}_{\mathcal{B}_1}}(\lambda) + \mathsf{Adv}^{\mathsf{Game}_1}_{\mathcal{A}, \mathcal{X}_{\mathsf{sameperm}}}(\lambda) \\
\mathsf{Adv}^{\mathsf{Game}_1}_{\mathcal{A}, \mathcal{X}_{\mathsf{sameperm}}}(\lambda) &\leqslant \mathsf{Adv}^{\mathsf{gprod}}_{\mathcal{B}_2, \mathcal{X}_{\mathcal{B}_2}}(\lambda) + \mathsf{Adv}^{\mathsf{Game}_2}_{\mathcal{A}, \mathcal{X}_{\mathsf{sameperm}}}(\lambda) \\
\mathsf{Adv}^{\mathsf{Game}_2}_{\mathcal{A}, \mathcal{X}_{\mathsf{sameperm}}}(\lambda) &\leqslant \mathsf{Adv}^{\mathsf{gprod}}_{\mathcal{B}_3, \mathcal{X}_{\mathcal{B}_3}}(\lambda) + + \mathsf{Adv}^{\mathsf{q\text{-}dlog}}_{\mathcal{B}_4}(\lambda) + \mathsf{Adv}^{\mathsf{Game}_3}_{\mathcal{A}, \mathcal{X}_{\mathsf{sameperm}}}(\lambda) \\
\mathsf{Adv}^{\mathsf{Game}_3}_{\mathcal{A}, \mathcal{X}_{\mathsf{sameperm}}}(\lambda) &\leqslant \frac{q_H}{\mathbb{F}}
\end{aligned}$$

which combined give us our final result.

$\underline{\mathsf{Game}_0 \mapsto \mathsf{Game}_1}$: Let Game_0 be the initial knowledge-soundness game. Then Game_1 is identical except in the following case. If $\mathcal{A}$ outputs a verifying proof then define $\mathcal{B}_1$ be the adversary as in $\mathcal{X}_{\mathsf{sameperm}}$ that returns $(A + \alpha M + \boldsymbol{\beta} \times \boldsymbol{g}, \pi_{\mathsf{gprod}})$ and $\mathcal{X}_{\mathcal{B}_1}$ its corresponding extractor. Return 0 if $\mathcal{X}_{\mathcal{B}_1}$ fails. If Game_0 returns 1 but Game_1 returns 0 then this means that π_{gprod} verifies and $\mathcal{X}_{\mathcal{B}_1}$ fails, and hence that $\mathcal{B}_1$ succeeds.

$\underline{\mathsf{Game}_1 \mapsto \mathsf{Game}_2}$: Define Game_2 to be identical to Game_1 except in the following case. If $\mathcal{A}$ outputs a verifying proof then define $\mathcal{B}_2$ be the adversary as in $\mathcal{X}_{\mathsf{sameperm}}$ that returns $(A+\alpha M+\boldsymbol{\beta}' \times \boldsymbol{g}, \pi'_{\mathsf{gprod}})$ and $\mathcal{X}_{\mathcal{B}_2}$ its corresponding extractor. Return 0 if $\mathcal{X}_{\mathcal{B}_2}$ fails. If Game_1 returns 1 but Game_2 returns 0 then this means that π'_{gprod} verifies and $\mathcal{X}_{\mathcal{B}_2}$ fails, and hence that $\mathcal{B}_2$ succeeds.

$\underline{\mathsf{Game}_2 \mapsto \mathsf{Game}_3}$: Define Game_3 to be identical to Game_2 except in the following case. We define an adversary $\mathcal{B}_3$ that behaves as follows:

- Generate $\mathsf{crs} = (\boldsymbol{g}, \boldsymbol{h}, H)$ and set $\mathsf{trans} = 0$. Choose random coins ω' such that during the $\mathsf{Game}^{\mathsf{fork}}_{\mathcal{A}}(\mathsf{crs})$ the original f values are sampled identically but the f' values are sampled differently. While $\mathsf{trans} = 0$ run $\mathsf{trans} \leftarrow \mathsf{Game}^{\mathsf{fork}}_{\mathcal{A}}(\mathsf{crs}; \omega')$ for $\mathsf{Game}^{\mathsf{fork}}_{\mathcal{A}}(\mathsf{crs})$ defined in Section 7.0.2. Parse
$$((\alpha, \beta), (\alpha'', \beta'')), (A, M, \boldsymbol{a});\ \cdot, (\pi_{\mathsf{gprod}}, \pi''_{\mathsf{gprod}}) = \mathsf{trans}$$
- Return $(A + \alpha M + \boldsymbol{\beta} \times \boldsymbol{g}, \pi''_{\mathsf{gprod}})$

Let $\mathcal{X}_{\mathcal{B}_3}$ be $\mathcal{B}_3$'s gprod extractor. Then Game_3 runs $\mathcal{B}_3$ and then $\mathcal{X}_{\mathcal{B}_3}$ to extract $\boldsymbol{b}'', \boldsymbol{r}''_B$ such that $B = \boldsymbol{b}'' \times \boldsymbol{g} + \boldsymbol{r}''_B \times \boldsymbol{h}$ and $p = \prod_i b''_i$. It computes $\boldsymbol{x} \leftarrow \boldsymbol{b} - \alpha \boldsymbol{m} - \boldsymbol{\beta}$. If

$$(\boldsymbol{b}'' \mid\mid \boldsymbol{r}''_B) \neq (\boldsymbol{x} + \alpha'' \boldsymbol{m} + \boldsymbol{\beta}'' \mid\mid \boldsymbol{r}_A + \alpha'' \boldsymbol{r}_M)$$

then return 0.

Let $\mathcal{B}_4$ be the adversary that takes as input $\boldsymbol{g}, \boldsymbol{h}$, samples H randomly, and runs $\mathcal{B}_3(\mathsf{crs};\ \omega)$ and $\mathcal{X}_{\mathcal{B}_3}$ on $\mathcal{B}_3$'s transcript, and returns

$$\big((\boldsymbol{b}'' \mid\mid \boldsymbol{r}''_B),\ (\boldsymbol{x} + \alpha'' \boldsymbol{m} + \boldsymbol{\beta}'' \mid\mid \boldsymbol{r}_A + \alpha'' \boldsymbol{r}_M)\big)$$

If Game_2 returns 1 but Game_3 returns 0 then this means that either (1) π''_{gprod} verifies and $\mathcal{X}_{\mathcal{B}_3}$ fails, and hence that $\mathcal{B}_3$ succeeds; or (2) $(\boldsymbol{b}'' \mid\mid \boldsymbol{r}''_B) \neq (\boldsymbol{x} + \alpha'' \boldsymbol{m} + \boldsymbol{\beta}'' \mid\mid \boldsymbol{r}_A + \alpha'' \boldsymbol{r}_M)$.

In the latter case we have that

$$A + \alpha \boldsymbol{m} \times \boldsymbol{g} + \alpha \boldsymbol{r}_M \times \boldsymbol{h} + \boldsymbol{\beta} = \boldsymbol{b} \times \boldsymbol{g} + \boldsymbol{r}_B \times \boldsymbol{h}$$

implies that $A = \boldsymbol{x} \times \boldsymbol{g} + \boldsymbol{r}_A \times \boldsymbol{h}$. Thus

$$\begin{aligned} A + \alpha M + \boldsymbol{\beta} \times \boldsymbol{g} &= \boldsymbol{b}'' \times \boldsymbol{g} + \boldsymbol{r}''_B \times \boldsymbol{h} \\ &= (\boldsymbol{x} + \alpha'' \boldsymbol{m} + \boldsymbol{\beta}'') \times \boldsymbol{g} + (\boldsymbol{r}_A + \alpha'' \boldsymbol{r}_M) \times \boldsymbol{h} \end{aligned}$$

and so $\mathcal{B}_4$ breaks the $\mathsf{q\text{-}dlog}$ assumption.

$\underline{\mathsf{Game}_3 \mapsto \mathsf{negl}(\lambda)}$: If Game_4 returns 1 then we have that $\mathcal{X}_{\mathsf{sameperm}}$ outputs $(\boldsymbol{m}, \boldsymbol{r}_A, \boldsymbol{r}_M)$ such that for $\boldsymbol{x}$ defined in Game_3 and for random α'', β'' we have that

$$\prod_{i=1}^{\ell}(a_i + i\alpha'' + \beta'') = \prod_{i=1}^{\ell}(x_i + \alpha'' m_i + \beta'')$$

Where α'', β'' is selected after $(\boldsymbol{x}, \boldsymbol{m}, \boldsymbol{r}_A, \boldsymbol{r}_M)$ are determined, this happens with maximum probability $q_H/|\mathbb{F}|$ unless $\boldsymbol{x} = \sigma(\boldsymbol{a})$ and $\boldsymbol{m} = \sigma(1, \ldots, \ell)$. □

5.2 Grand-Product Argument

In this section we discuss a zero knowledge argument for the relation

$$R_{\mathsf{gprod}} = \left\{ \begin{array}{ll} (B, p); & (\boldsymbol{b}, \boldsymbol{r}_B) \end{array} \middle| \begin{array}{l} B = \boldsymbol{b} \times \boldsymbol{g} + \boldsymbol{r}_B \times \boldsymbol{h} \\ p = \prod_{i=1}^{\ell} b_i \end{array} \right\}$$

For an informal overview see Section 5.2.1 and for the formal construction see Figures 5.4 and 5.5. The security arguments are deferred to Theorems 5.2.1 and 5.2.3. The construction makes use of a discrete-logarithm inner product argument as a subprotocol. We specify the inner-product relations below and describe the inner-product construction in Section 5.3.

5.2.1 Informal Overview

The prover will take as input the B, p and aims to prove knowledge of $\boldsymbol{b}$ such that:

- $B = \boldsymbol{b} \times \boldsymbol{g}$ is a commitment to $\boldsymbol{b}$
- $p = \prod_{i=1}^{\ell} b_i$ is the grandproduct of $\boldsymbol{b}$

On a high level we aim to express this relation as an inner product argument. Doing this consists of the following steps:

1. We *separate* the grandproduct into multiple single product equations;
2. We *compress* all our equations into a polynomial;
3. We *rearrange* the polynomial into an inner product equation;
4. We *compile* the proving system by obtaining commitments to the inputs to the inner product equation;

See Figure 5.3.

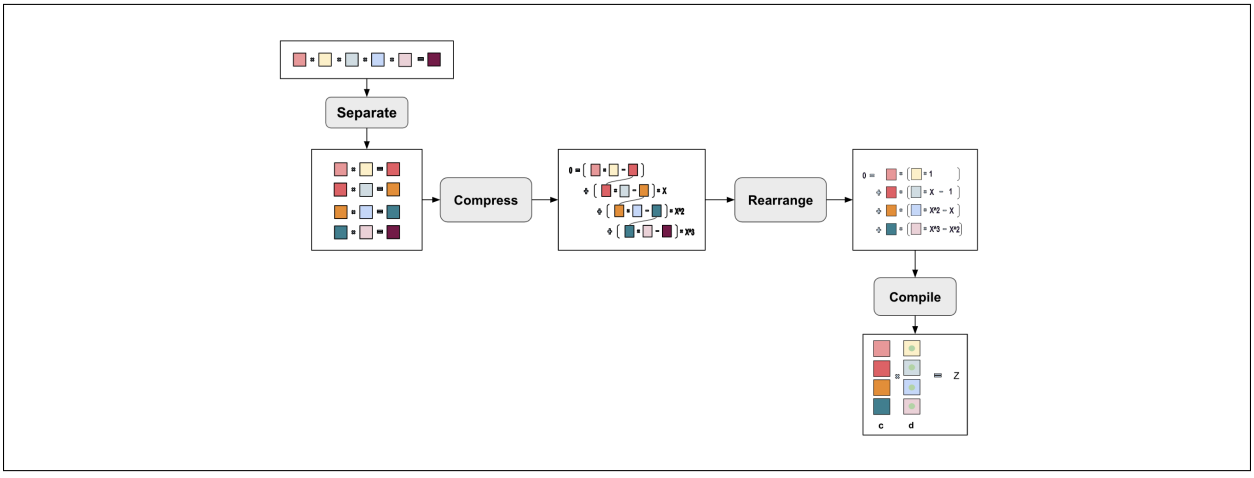

Figure 5.3: The grandproduct argument is compiled into an inner product argument.

Separate

The product $p = \prod_{i=1}^{\ell} b_i$ consists of $\ell - 1$ multiplications. Initially we *separate* these multiplications into $\ell + 1$ separate multiplication checks

$$c_1 = 1 \ \wedge \ c_{i+1} = b_i c_i, \ i \in [1, \ell) \ \wedge \ p = b_\ell c_\ell$$

that iteratively define a vector $\boldsymbol{c}$. The final check enforces that $p = \prod_{i=1}^{\ell} b_i$ is the grandproduct of $\boldsymbol{b}$.

Compress

To ensure that each of our multiplication checks hold we compress them into a single polynomial equation

$$0 = (1 - c_1) + (b_1 c_1 - c_2)X + (b_2 c_2 - c_3)X^2 + \ldots + (b_{\ell-1} c_{\ell-1} - c_\ell)X^{\ell-1} + (b_\ell c_\ell - p)X^\ell$$

or equivalently

$$0 = (1 - c_1) + \sum_{i=1}^{\ell-1} (b_i c_i - c_{i+1})X^i + (b_\ell c_\ell - p)X^\ell$$

in the indeterminate X where each coefficient is checking a single constraint.

Rearrange

Our eventual goal is to express (5.1) as an inner product equation such that we can run an inner product argument. We thus rearrange the $\boldsymbol{c}$ terms and see that

$$pX^\ell - 1 = c_1(Xb_1 - 1) + c_2(X^2 b_2 - X) + \ldots + c_{\ell-1}(X^{\ell-1} b_{\ell-1} - X^{\ell-2}) + c_\ell(X^\ell b_\ell - X^{\ell-1})$$

or equivalently

$$pX^\ell - 1 = \sum_{i=1}^{\ell} c_i(X^i b_i - X^{i-1})$$

Compile

By the Schwartz-Zippel Lemma our inner product equation holds with overwhelming probability if at a random point β

$$p\beta^\ell - 1 = \sum_{i=1}^{\ell} c_i(\beta^i b_i - \beta^{i-1}) \tag{5.1}$$

Equivalently

$$z = \boldsymbol{c} \times \boldsymbol{d}$$

where

$$z = p\beta^\ell - 1 \ \wedge \ d_i = (\beta^i b_i - \beta^{i-1}), \ i \in [1, \ell]$$

We thus require a commitment to $\boldsymbol{c}$ and $\boldsymbol{d}$.

Initially the prover provides a commitment $C = \boldsymbol{c} \times \boldsymbol{g}$ to

$$\boldsymbol{c} = (1, b_1, b_1 b_2, b_1 b_2 b_3, \ldots, b_1 \ldots b_{\ell-1})$$

The commitment C is hashed to get β. We now require a commitment D to the vector $\boldsymbol{d}$. We have a commitment $B = \boldsymbol{b} \times \boldsymbol{g}$ to $\boldsymbol{b}$. Recall that

$$\boldsymbol{v} \times \boldsymbol{w} = (a_1 v_1, \ldots, a_\ell v_\ell) \times (a_1^{-1} w_1, \ldots, a_\ell^{-1} w_\ell)$$

for all invertible $\boldsymbol{a}$. Thus we can view B as being a commitment to a rescaled vector $\boldsymbol{b}'$ under an appropriately rescaled commitment key $\boldsymbol{g}'$

$$\begin{aligned}
\boldsymbol{b}' &= (\beta^1 b_1, \ldots, \beta^\ell b_\ell) \\
\boldsymbol{g}' &= (\beta^{-1} g_1, \ldots, \beta^{-(\ell)} g_\ell) \\
B &= \boldsymbol{b}' \times \boldsymbol{g}'
\end{aligned}$$

Now

$$\boldsymbol{d} = \boldsymbol{b}' - (1, \beta, \ldots, \beta^{\ell-1})$$

Hence the prover and verifier compute

$$D = B - \sum_{i=1}^{\ell} \beta^{i-1} g'_i$$

such that $D = \boldsymbol{d} \times \boldsymbol{g}'$ is a commitment to $\boldsymbol{d}$ under $\boldsymbol{g}'$.

To finish, the prover provides a discrete log inner product argument, the relation for which is formally defined below, attesting to the existence of $\boldsymbol{c}$ and $\boldsymbol{d}$ such that

$$C = \boldsymbol{c} \times \boldsymbol{g},\ D = \boldsymbol{d} \times \boldsymbol{g}',\ p\beta^\ell - 1 = \boldsymbol{c} \times \boldsymbol{d}$$

By design there exists a non-trivial relation between $\boldsymbol{g}$ and $\boldsymbol{g}'$. The full construction has some additional masking values that are included to ensure zero-knowledge. For simplicity we have ignored these terms in this overview.

5.2.2 Discrete Logarithm Inner Product Relation

The discrete logarithm inner product relation demonstrates that given public input $C, D \in \mathbb{G}, \boldsymbol{v} \in \mathbb{F}^n, z \in \mathbb{F}$ there exists $\boldsymbol{c}$ and $\boldsymbol{d}$ such that $C = \boldsymbol{c} \times \boldsymbol{G}$, $D = \boldsymbol{d} \times \boldsymbol{G}'$ and $z = \boldsymbol{c} \times \boldsymbol{d}$. In other words

$$R_{\mathsf{dl\text{-}inner}} = \left\{ (C, D, z);\quad (\boldsymbol{c}, \boldsymbol{d}) \;\middle|\; \begin{array}{l} C = \boldsymbol{c} \times \boldsymbol{G} \\ D = \boldsymbol{d} \times \boldsymbol{G}' \\ z = \boldsymbol{c} \times \boldsymbol{d} \end{array} \right\}$$

When we use this relation we will have that the adversary knows a non-trivial relation between $\boldsymbol{G}$ and $\boldsymbol{G}'$ but it will not know: (1) any non-trivial relations between the elements in $\boldsymbol{G}$; (1) any non-trivial relations between the elements in $\boldsymbol{G}'$.

5.2.3 Full Zero Knowledge Grand Product Construction

A formal description of the grand-product argument is provided in Figures 5.4 and 5.5. Here we describe the additional steps that we have added compared to the informal overview in Section 5.2.1

to achieve zero-knowledge. We defer the security proofs of zero-knowledge, and soundness to Section 5.2.4, Theorems 5.2.3 and 5.2.1.

Step 1: In the first step the prover and verifier both hash the instance to get a random value α. This allows the prover to mask $\boldsymbol{r}_B$ in the next step even when $\boldsymbol{r}_B = \mathbf{0}$. There are no secrets in this step. The verifier parses all inputs to check that they are group or field elements.

Step 2: In the second step the prover computes a commitment C to $\boldsymbol{c}$. The vector $\boldsymbol{c}$ depends on $\boldsymbol{b}$ and thus must be kept private. Thus the prover chooses a random blinding vector $\boldsymbol{r}_C \in \mathbb{F}^{n_{\mathsf{bl}}}$. This vector $\boldsymbol{r}_C$ is included in the inner product argument in the final step, and thus the prover provides a field element $r_p = (\boldsymbol{r}_B + \alpha\mathbf{1}) \times \boldsymbol{r}_C$ that cancels out the blinders contributions to the inner product. See here that the $\alpha(\mathbf{1} \times \boldsymbol{r}_C)$ component ensures that r_p is satistically blinded provided that $|\boldsymbol{r}_C| \geqslant 2$.

Step 3: In the third step the prover and verifier compute $\boldsymbol{h}' = \beta^{-(\ell+1)}\boldsymbol{h}$ as the rescaled part of the commitment key that is used for blinding commitments. The prover additionally computes randomness $\boldsymbol{r}_D = \beta^{\ell+1}(\boldsymbol{r}_B + \alpha\mathbf{1})$ such that $D = \boldsymbol{d} \times \boldsymbol{g}' + \boldsymbol{r}_D \times \boldsymbol{h}'$ is a commitment to $\boldsymbol{d}$. Here $\beta^{\ell+1}$ does not overlap with the $(\beta, \beta^2, \ldots, \beta^\ell)$ values that are used to rescale $\boldsymbol{b}'$.

Step 4: In the fourth and final step the prover and verifier compute the commitment key $\boldsymbol{G} = (\boldsymbol{g} \mid\mid \boldsymbol{h})$ so that they can view $\boldsymbol{C}$ as a commitment to the extended vector $(\boldsymbol{c} \mid\mid \boldsymbol{r}_C)$. They do the same for $\boldsymbol{G}'$ such that $\boldsymbol{D}$ is a commitment to the extended vector $(\boldsymbol{d} \mid\mid \boldsymbol{r}_D)$. They compute $z = p\beta^\ell + r_p\beta^{\ell+1} - 1$ as the inner product of the extended vectors $z = (\boldsymbol{c} \mid\mid \boldsymbol{r}_C) \times (\boldsymbol{d} \mid\mid \boldsymbol{r}_D)$. See that $r_p\beta^{\ell+1} = \boldsymbol{r}_C \times \boldsymbol{r}_D$. There are no secrets involved in this step.

5.2.4 Grand-Product Security

Theorem 5.2.1 (Grand product argument is zero-knowledge)**.** *The grand-product argument described in Figures 5.4 and 5.5 is zero-knowledge when $|\boldsymbol{h}| \geqslant 2$ and the discrete logarithm inner product argument is zero-knowledge.*

Proof. We design a simulator $\mathsf{Simulate}$ that takes as input an instance (B, p) and outputs a proof π_{gprod} that is indistinguishable from a proof generated by an honest prover that knows the witness.

During the setup the simulator chooses the $\mathsf{crs}_{\mathsf{gprod}} = (\boldsymbol{g}, \boldsymbol{h}, H)$ uniformly at random. During proving the simulator $\mathsf{Simulate}$ proceeds as follows

1. They sample $C \xleftarrow{\$} \mathbb{G}$ and $r_p \xleftarrow{\$} \mathbb{F}$.
2. They compute $\boldsymbol{G}, \boldsymbol{G}', D, z$ identically to the honest prover.
3. They run
$$\pi_{\mathsf{dl\text{-}inner}} = \mathsf{Simulate}_{\mathsf{dl\text{-}inner}}((\boldsymbol{G}, \boldsymbol{G}', H), (C, D, z))$$
and return $(C, r_p, \pi_{\mathsf{dl\text{-}inner}})$.

Now we must argue that the simulated proof is indistinguishable from the real proof.

We design an adversary $\mathcal{B}$ such that

$$\mathsf{Adv}_{\mathcal{A}}(\lambda) \leqslant 2\mathsf{Adv}^{\mathsf{dl\text{-}inner}}_{\mathcal{B}}(\lambda) + \frac{1}{|\mathbb{F}|}$$

$\underline{\mathsf{Prove}_{\mathsf{gprod}}(\mathsf{crs}_{\mathsf{gprod}};\ (B,p);\ (\boldsymbol{b},\boldsymbol{r}_B))}$

Step 1:
$(\boldsymbol{g},\boldsymbol{h},H) \leftarrow \mathsf{parse}(\mathsf{crs}_{\mathsf{gprod}})$
$\alpha \leftarrow \mathsf{Hash}(B,p)$

Step 2:
$\boldsymbol{c} \leftarrow (1, b_1, b_1b_2, b_1b_2b_3, \ldots, b_1\cdots b_{\ell-1})$
$\boldsymbol{r}_C \xleftarrow{\$} \mathbb{F}^{n_{\mathsf{bl}}}$
$C \leftarrow \boldsymbol{c} \times \boldsymbol{g} + \boldsymbol{r}_C \times \boldsymbol{h}$
$r_p \leftarrow (\boldsymbol{r}_B + \alpha\mathbf{1}) \times \boldsymbol{r}_C$
$\beta \leftarrow \mathsf{Hash}(C, r_p)$

Step 3:
$\boldsymbol{g}' \leftarrow (\beta^{-1}g_1, \beta^{-2}g_2, \ldots, \beta^{-\ell}g_\ell)$
$\boldsymbol{h}' \leftarrow \beta^{-(\ell+1)}\boldsymbol{h}$
$\boldsymbol{b}' \leftarrow (b_1\beta,\ b_2\beta^2,\ \ldots,\ b_\ell\beta^\ell)$
$\boldsymbol{d} \leftarrow \boldsymbol{b}' - (1, \beta, \ldots, \beta^{\ell-1})$
$\boldsymbol{r}_D \leftarrow \beta^{\ell+1}(\boldsymbol{r}_B + \alpha\mathbf{1})$
$D \leftarrow B - (1, \beta, \ldots, \beta^{\ell-1}) \times \boldsymbol{g}' + \alpha\beta^{\ell+1}\mathbf{1} \times \boldsymbol{h}'$

Step 4:
$\boldsymbol{G} \leftarrow (\boldsymbol{g} \mid\mid \boldsymbol{h})$
$\boldsymbol{G}' \leftarrow (\boldsymbol{g}' \mid\mid \boldsymbol{h}')$
$z \leftarrow p\beta^\ell + r_p\beta^{\ell+1} - 1$
$\pi_{\mathsf{dl\text{-}inner}} \leftarrow \mathsf{Prove}_{\mathsf{dl\text{-}inner}}\big(\ (\boldsymbol{G},\boldsymbol{G}',H);\ (C,D,z);((\boldsymbol{c} \mid\mid \boldsymbol{r}_C),(\boldsymbol{d} \mid\mid \boldsymbol{r}_D))\ \big)$

return $(C, r_p, \pi_{\mathsf{dl\text{-}inner}})$

Figure 5.4: Proving algorithm to demonstrate that (B,p) is such that $B = \boldsymbol{b} \times \boldsymbol{g} + \boldsymbol{r}_B \times \boldsymbol{h}$ such that $p = \prod_{i=1}^{\ell} b_i$. Here $\mathbf{1} = (1,\ldots,1)$ denotes the length ℓ vector with all entries equal to 1.

$\underline{\mathsf{Verify}_{\mathsf{gprod}}(\mathsf{crs}_{\mathsf{gprod}};\ \phi_{\mathsf{gprod}};\ \pi_{\mathsf{gprod}})}$

Step 1:
$(\boldsymbol{g}, \boldsymbol{h}, H) \leftarrow \mathsf{parse}(\mathsf{crs}_{\mathsf{gprod}})$
$(B, p) \leftarrow \mathsf{parse}(\phi_{\mathsf{gprod}})$
$(C, r_p, \pi_{\mathsf{dl\text{-}inner}}) \leftarrow \mathsf{parse}(\pi_{\mathsf{gprod}})$
$\alpha \leftarrow \mathsf{Hash}(B, p)$

Step 2:
$\beta \leftarrow \mathsf{Hash}(C, r_p)$

Step 3:
$\boldsymbol{g}' \leftarrow (\beta^{-1} g_1, \beta^{-2} g_2, \ldots, \beta^{-\ell} g_\ell)$
$\boldsymbol{h}' \leftarrow \beta^{-(\ell+1)} \mathbf{1} \times \boldsymbol{h}$
$D \leftarrow B - (1, \beta, \ldots, \beta^{\ell-1}) \times \boldsymbol{g}' + \alpha \beta^{\ell+1} \mathbf{1} \times \boldsymbol{h}'$

Step 4:
$\boldsymbol{G} \leftarrow (\boldsymbol{g} \mid\mid \boldsymbol{h})$
$\boldsymbol{G}' \leftarrow (\boldsymbol{g}' \mid\mid \boldsymbol{h}')$
$z \leftarrow p\beta^{\ell} + r_p \beta^{\ell+1} - 1$
$\mathsf{check} \leftarrow \mathsf{Verify}_{\mathsf{dl\text{-}inner}}((\boldsymbol{G}, \boldsymbol{G}', H), (C, D, z), \pi_{\mathsf{dl\text{-}inner}})$

return 1 if $\mathsf{check} = 1$
else return 0

Figure 5.5: Verify algorithm to check that the prover knows $\boldsymbol{b}, \boldsymbol{r}_B$ such that $B = \boldsymbol{b} \times \boldsymbol{g} + \boldsymbol{r}_B \times \boldsymbol{h}$ such that $p = \prod_{i=1}^{\ell} b_i$.

$\underline{\mathsf{Game}_0 \mapsto \mathsf{Game}_1}$: Let Game_0 be the initial zero-knowledge game. Define Game_1 to run identicallly to Game_0 except that, the crs and the $\pi_{\mathsf{dl\text{-}inner}}$ proof are generated by the $\mathsf{dl\text{-}inner}$ simulator for both $b = 0$ and $b = 1$.

Let $\mathcal{B}$ be an adversary against the $\mathsf{dl\text{-}inner}$ zero-knowledge game. Then $\mathcal{B}$ simulates the zero-knowledge game for $\mathcal{A}$. It takes as input crs and runs $\mathcal{A}(\mathsf{crs})$ on the same input. It flips a coin to get $b \in \{0, 1\}$. When $\mathcal{A}$ makes a prover query, if $b = 0$ then $\mathcal{B}$ generates (C, r_p) honestly and queries its oracle on input (C, D, z) and $(\boldsymbol{c} \mid\mid \boldsymbol{r}_C)$, $(\boldsymbol{d} \mid\mid \boldsymbol{r}_D)$ to get a proof $\pi_{\mathsf{dl\text{-}inner}}$. Then $\mathcal{B}$ returns $(C, r_p, \pi_{\mathsf{dl\text{-}inner}})$. If $b = 1$ then $\mathcal{B}$ runs the simulator to compute the response. When $\mathcal{A}$ returns b' then if $b = b'$ then $\mathcal{B}$ returns 0, else $\mathcal{B}$ returns 1.

Then

$$\Pr[\mathcal{B}(\mathsf{crs}) = 0 \mid \bar{b} = 0] = \frac{1}{2}\left(\Pr[\mathcal{A}(\mathsf{crs}) = 0 \mid \mathsf{Game}_0, b = 0] + \Pr[\mathcal{A}(\mathsf{crs}) = 1 \mid \mathsf{Game}_0, b = 1]\right)$$
$$\Pr[\mathcal{B}(\mathsf{crs}) = 1 \mid \bar{b} = 1] = \frac{1}{2}\left(2 - \Pr[\mathcal{A}(\mathsf{crs}) = 0 \mid \mathsf{Game}_1, b = 0] - \Pr[\mathcal{A}(\mathsf{crs}) = 1 \mid \mathsf{Game}_1, b = 1]\right)$$

and

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{zk}}_{\mathcal{B}}(\lambda) &= |1 - 2\Pr[\mathsf{Game}^{\mathsf{dl\text{-}inner}}_{\mathcal{B}}(\lambda)]| \\
&= |1 - \Pr[\mathcal{B}(\mathsf{crs}) = 0 \mid \bar{b} = 0] - \Pr[\mathcal{B}(\mathsf{crs}) = 1 \mid \bar{b} = 1]| \\
&= \Big|1 - \frac{1}{2}\left(\Pr[\mathcal{A}(\mathsf{crs}) = 0 \mid \mathsf{Game}_0, b = 0] + \Pr[\mathcal{A}(\mathsf{crs}) = 1 \mid \mathsf{Game}_0, b = 1]\right) \\
&\qquad - \left(1 - \Pr[\mathcal{B}(\mathsf{crs}) = 0 \mid \bar{b} = 1]\right)\Big| \\
&= \frac{1}{2}\Big|1 - \Pr[\mathcal{A}(\mathsf{crs}) = 0 \mid \mathsf{Game}_0, b = 0] - \Pr[\mathcal{A}(\mathsf{crs}) = 1 \mid \mathsf{Game}_0, b = 1] \\
&\qquad\qquad - 1 + \Pr[\mathcal{A}(\mathsf{crs}) = 0 \mid \mathsf{Game}_1, b = 0] + \Pr[\mathcal{A}(\mathsf{crs}) = 1 \mid \mathsf{Game}_0, b = 1]\Big| \\
&= \frac{1}{2}|\mathsf{Adv}^{\mathsf{Game}_0}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{\mathsf{Game}_1}_{\mathcal{A}}(\lambda)|
\end{aligned}$$

This means that

$$\mathsf{Adv}^{\mathsf{Game}_0}_{\mathcal{A}}(\lambda) \leqslant 2\mathsf{Adv}^{\mathsf{dl\text{-}inner}}_{\mathcal{B}}(\lambda) + \mathsf{Adv}^{\mathsf{Game}_1}_{\mathcal{A}}(\lambda)$$

$\underline{\mathsf{Game}_1 \mapsto \mathsf{negl}(\lambda)}$: In Game_1 the simulated proofs are generated identically. The remaining honest prover elements C, r_p are such that C is randomised by $\boldsymbol{r}_C$. The value $r_p = \boldsymbol{r}_C \times (\boldsymbol{r}_B + \alpha\mathbf{1})$ is randomised by $\boldsymbol{r}_C$ provided that $|\boldsymbol{r}_C| > 1$ and $\boldsymbol{r}_B \neq \alpha\mathbf{1}$. The latter happens with maximum probability $\frac{1}{|\mathbb{F}|}$. The simulated elements C, r_p are also generated randomly and are thus indistinguishable. □

Lemma 5.2.2. *There does not exist an adversary $\mathcal{A}$ such that*

$$\begin{aligned}(\mathsf{aux}, (\boldsymbol{a}, \boldsymbol{b}, c), (\boldsymbol{x}, \boldsymbol{y}, z)) &\xleftarrow{\$} \mathcal{A}(\boldsymbol{g}, \boldsymbol{h}, H)\\ \beta &\leftarrow \mathsf{Hash}(\mathsf{aux})\\ \boldsymbol{G} &\leftarrow (\boldsymbol{g} \mid\mid \boldsymbol{h})\\ \boldsymbol{G}' &\leftarrow ((\beta^{-1}g_1, \beta^{-2}g_2, \ldots, \beta^{-\ell}g_\ell) \mid\mid \beta^{-(\ell+1)}\boldsymbol{h})\\ \mathsf{Commit}((\boldsymbol{G}, \boldsymbol{G}', H), (\boldsymbol{a}, \boldsymbol{b}, c)) &= \mathsf{Commit}((\boldsymbol{G}, \boldsymbol{G}', H), (\boldsymbol{x}, \boldsymbol{y}, z))\\ (\boldsymbol{a}, \boldsymbol{b}, c) &\neq (\boldsymbol{x}, \boldsymbol{y}, z)\end{aligned}$$

where

$$(\boldsymbol{x} \times \boldsymbol{G} + zH,\ \boldsymbol{y} \times \boldsymbol{G}') \leftarrow \mathsf{Commit}((\boldsymbol{G}, \boldsymbol{G}', H);\ (\boldsymbol{x}, \boldsymbol{y}, z))$$

assuming the q-dlog *assumption holds.*

Proof. We show the existence of an adversary $\mathcal{B}$ such that

$$\mathsf{Adv}_{\mathcal{A}}(\lambda) \leqslant \mathsf{Adv}^{\mathsf{q\text{-}dlog}}_{\mathcal{B}}(\lambda)$$

Initially $\mathcal{B}$ takes as input a q-dlog instance $(\boldsymbol{g} \mid\mid \boldsymbol{h} \mid\mid H)$ and runs

$$(\mathsf{aux}, (\boldsymbol{a}, \boldsymbol{b}, c), (\boldsymbol{x}, \boldsymbol{y}, z)) \xleftarrow{\$} \mathcal{A}(\boldsymbol{g}, \boldsymbol{h}, H)$$

If $\mathcal{A}$ succeeds then

$$(\boldsymbol{a} \times \boldsymbol{G} + cH,\ \boldsymbol{b} \times \boldsymbol{G}') = (\boldsymbol{x} \times \boldsymbol{G} + zH,\ \boldsymbol{y} \times \boldsymbol{G}')$$

If $(\boldsymbol{a}, c) \neq (\boldsymbol{x}, z)$ then $\mathcal{B}$ returns $(\boldsymbol{a} \mid\mid c), (\boldsymbol{x} \mid\mid z)$.

Else $\mathcal{B}$ sets

$$\begin{aligned}\boldsymbol{b}' &\leftarrow ((\beta^{-1}b_1, \ldots, \beta^{-\ell}b_\ell),\ \mid\mid \beta^{-(\ell+1)}\boldsymbol{b}_{[\ell+1:]} \mid\mid 0)\\ \boldsymbol{y}' &\leftarrow (\beta^{-1}y_1, \ldots, \beta^{-\ell}y_\ell),\ \mid\mid \beta^{-(\ell+1)}\boldsymbol{y}_{[\ell+1:]} \mid\mid 0)\end{aligned}$$

and returns $(\boldsymbol{b}', \boldsymbol{y}')$.

If $\mathcal{A}$ succeeds then

$$(\boldsymbol{a} \mid\mid c) \times (\boldsymbol{G}||h) = (\boldsymbol{x} \mid\mid z) \times (\boldsymbol{G}||h)$$

and so if $(\boldsymbol{a}, c) \neq (\boldsymbol{x}, z)$ then $\mathcal{B}$ succeeds. If $(\boldsymbol{a}, c) = (\boldsymbol{x}, z)$ then $\boldsymbol{b} \neq \boldsymbol{y}$. In this case

$$\boldsymbol{b}' \times (\boldsymbol{G}||h) = \boldsymbol{y}' \times (\boldsymbol{G}||h)$$

and $\boldsymbol{b}' \neq \boldsymbol{y}'$, hence $\mathcal{B}$ also succeeds.

□

Theorem 5.2.3 (Grand Product Argument is knowledge-sound)**.** *Suppose the* dl-inner *argument is knowledge sound whenever* $\mathsf{crs}_{\mathsf{dl\text{-}inner}} = (\boldsymbol{G}, \boldsymbol{G}', H)$ *is sampled such that*

$$(\boldsymbol{x} \times \boldsymbol{G} + zH,\ \boldsymbol{y} \times \boldsymbol{G}') \leftarrow \mathsf{Commit}((\boldsymbol{G}, \boldsymbol{G}', H);\ (\boldsymbol{x}, \boldsymbol{y}, z))$$

is a binding commitment. Then whenever the q-dlog *assumption holds, the shuffle argument described in Figures 5.4 and 5.5 is knowledge-sound.*

Proof. We design an extractor $\mathcal{X}_{\mathsf{gprod}}$ such that: if there exists an adversary $\mathcal{A}$ that convinces the verifier with non-negligible probability then with overwhelming probability $\mathcal{X}_{\mathsf{gprod}}$ returns field elements $(\boldsymbol{b}, \boldsymbol{r}_B)$ such that

$$((B, p);\ (\boldsymbol{b}, \boldsymbol{r}_B)) \in R_{\mathsf{gprod}}.$$

By Lemma 5.2.2 $\mathsf{Commit}()$ is a binding commitment with respect to $\mathsf{crs}_{\mathsf{dl\text{-}inner}}$ and hence we can assume that the dl-inner argument is knowledge sound. By the knowledge-soundness of the discrete-log inner product argument there exists an extractor $\mathcal{X}_{\mathcal{B}}$ such that if $\mathcal{B}$ returns verifying $\pi_{\mathsf{dl\text{-}inner}}$ then they return valid witnesses for $R_{\mathsf{dl\text{-}inner}}$ with overwhelming probability.

The extractor $\mathcal{X}_{\mathsf{gprod}}$ works as follows

1. Randomly sample coins ω.
2. Run $((B, p);\ (C, r_p, \pi_{\mathsf{dl\text{-}inner}})) \leftarrow \mathcal{A}(\mathsf{crs};\ \omega)$.

   Define an adversary $\mathcal{B}_1$ that behaves as follows:

   - Generate $\mathsf{crs} = (\boldsymbol{g}, \boldsymbol{h}, H)$ and set $\mathsf{trans} = 0$. While $\mathsf{trans} = 0$ run $\mathsf{trans} \leftarrow \mathsf{Game}^{\mathsf{fork}}_{\mathcal{A}}(\mathsf{crs}; \omega)$ for $\mathsf{Game}^{\mathsf{fork}}_{\mathcal{A}}(\mathsf{crs})$ defined in Section 7.0.2. Parse

   $$((\alpha, \beta), (\alpha, \bar{\beta})), ((B, p);\ (C, r_p), (\pi_{\mathsf{dl\text{-}inner}}, \bar{\pi}_{\mathsf{dl\text{-}inner}})) = \mathsf{trans}$$

   - Computes D, z the same as the honest verifier wrt β.
   - Return $(C, D, z), \pi'_{\mathsf{dl\text{-}inner}}$
3. Let $\mathcal{X}_{\mathcal{B}_1}$ be $\mathcal{B}_1$'s corresponding discrete-logarithm inner-product extractor. Extract $(\boldsymbol{c} \mid\mid \boldsymbol{r}_C)$ and $(\boldsymbol{d} \mid\mid \boldsymbol{r}_D)$ such that

   $$C = (\boldsymbol{c} \mid\mid \boldsymbol{r}_C) \times \boldsymbol{G} \ \wedge\ D = (\boldsymbol{d} \mid\mid \boldsymbol{r}_D) \times \boldsymbol{G}'$$

4. Return

   $$\boldsymbol{b} = (d_1\beta^{-1}, \ldots, d_\ell\beta^{-\ell})) \ \wedge\ \boldsymbol{r}_B = \beta^{-(\ell+1)}\boldsymbol{r}_D - \alpha\boldsymbol{1}$$

We must show that whenever $\mathcal{A}$ convinces the verifier, then either $\mathcal{X}_{\mathsf{gprod}}$ succeeds or we can extract a discrete logarithm relation between $(\boldsymbol{g}, \boldsymbol{h}, H)$.

First see that $\mathcal{X}_{\mathsf{gprod}}$ terminates in polynomial time with overwhelming probability. Let τ be the run time of $\mathcal{A}$, ϵ be the probability that $\mathcal{A}$ outputs a valid response and q_H be the total number of hash queries that $\mathcal{A}$ can make. Assuming $\frac{8q_H}{|\mathbb{F}|} < \epsilon$ then the game $\mathsf{Game}^{\mathsf{fork}}_{\mathcal{A}}(\mathsf{par})$ runs in time at most $\frac{8\tau q_H}{\epsilon} \cdot \ln(8/\epsilon)$ and is successful with probability at least $\epsilon/8$. The expected run time of $\mathcal{B}_1$ is less than $\tau q_H \cdot \ln(8/\epsilon)$, which is polynomial time assuming that τ, ϵ and q_H are polynomial in the security parameter. By the assumption that dl-inner is knowledge sound this means that the expected run time of $\mathcal{X}_{\mathcal{B}_1}$ is also polynomial.

We design adversaries $\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3$ such that for all extractors $\mathcal{X}_{\mathcal{B}_1}, \mathcal{X}_{\mathcal{B}_2}$ we have that

$$\mathsf{Adv}_{\mathcal{A}, \mathcal{X}_{\mathsf{gprod}}}(\lambda) \leqslant \mathsf{Adv}^{\mathsf{dl\text{-}inner}}_{\mathcal{B}_1, \mathcal{X}_{\mathcal{B}_1}}(\lambda) + \mathsf{Adv}^{\mathsf{dl\text{-}inner}}_{\mathcal{B}_2, \mathcal{X}_{\mathcal{B}_2}}(\lambda) + \mathsf{Adv}^{\mathsf{q\text{-}dlog}}_{\mathcal{B}_3}(\lambda) + \frac{q_H}{|\mathbb{F}|}$$

for q_H the maximum number of hash queries the adversary can make. We proceed via a series of games $\mathsf{Game}_1, \mathsf{Game}_2, \mathsf{Game}_3$ such that

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{Game}_0}_{\mathcal{A},\mathcal{X}_{\mathsf{gprod}}}(\lambda) &\leqslant \mathsf{Adv}^{\mathsf{dl\text{-}inner}}_{\mathcal{B}_1,\mathcal{X}_{\mathcal{B}_1}} + \mathsf{Adv}^{\mathsf{Game}_1}_{\mathcal{A},\mathcal{X}_{\mathsf{gprod}}}(\lambda) \\
\mathsf{Adv}^{\mathsf{Game}_1}_{\mathcal{A},\mathcal{X}_{\mathsf{gprod}}}(\lambda) &\leqslant +\mathsf{Adv}^{\mathsf{dl\text{-}inner}}_{\mathcal{B}_2,\mathcal{X}_{\mathcal{B}_2}}(\lambda) + \mathsf{Adv}^{\mathsf{Game}_2}_{\mathcal{A},\mathcal{X}_{\mathsf{gprod}}}(\lambda) \\
\mathsf{Adv}^{\mathsf{Game}_2}_{\mathcal{A},\mathcal{X}_{\mathsf{gprod}}}(\lambda) &\leqslant \mathsf{Adv}^{\mathsf{q\text{-}dlog}}_{\mathcal{B}_3}(\lambda) + \mathsf{Adv}^{\mathsf{Game}_3}_{\mathcal{A},\mathcal{X}_{\mathsf{gprod}}}(\lambda) \\
\mathsf{Adv}^{\mathsf{Game}_3}_{\mathcal{A},\mathcal{X}_{\mathsf{gprod}}}(\lambda) &\leqslant \frac{q_H}{|\mathbb{F}|}
\end{aligned}$$

which combined give us our final result.

$\underline{\mathsf{Game}_0 \mapsto \mathsf{Game}_1}$: Let Game_0 be the initial knowledge-soundness game and define Game_1 to initially run identicallly to Game_0. However, in Game_1, when $\mathcal{A}$ outputs a verifying proof then define $\mathcal{B}_1$ to be the adversary in $\mathcal{X}_{\mathsf{gprod}}$ that returns $((C, D, z), \pi_{\mathsf{dl\text{-}inner}})$ and $\mathcal{X}_{\mathcal{B}_1}$ its corresponding extractor. Return 0 if $\mathcal{X}_{\mathsf{dl\text{-}inner}}$ fails.

If Game_0 returns 1 but Game_1 returns 0 then this means that $\pi_{\mathsf{dl\text{-}inner}}$ verifies and $\mathcal{X}_{\mathcal{B}_1}$ fails, and hence that $\mathcal{B}_1$ succeeds.

$\underline{\mathsf{Game}_1 \mapsto \mathsf{Game}_2}$: Let Game_2 be identical to Game_1 except in the following case.

Define an adversary $\mathcal{B}_2$ that behaves as follows:

- Generate $\mathsf{crs} = (\boldsymbol{g}, \boldsymbol{h}, H)$ and set $\mathsf{trans} = 0$. While $\mathsf{trans} = 0$ run $\mathsf{trans} \leftarrow \mathsf{Game}^{\mathsf{fork}}_{\mathcal{A}}(\mathsf{crs}; \omega)$ for $\mathsf{Game}^{\mathsf{fork}}_{\mathcal{A}}(\mathsf{crs})$ defined in Section 7.0.2. Parse

$$((\alpha, \beta), (\alpha, \bar{\beta})), ((B, p);\ (C, r_p), (\pi_{\mathsf{dl\text{-}inner}}, \bar{\pi}_{\mathsf{dl\text{-}inner}})) = \mathsf{trans}$$

- Computes $\bar{D}, \bar{z}$ the same as the honest verifier wrt $\bar{\beta}$.
- Return $(C, \bar{D}, \bar{z}), \bar{\pi}_{\mathsf{dl\text{-}inner}}$

Let $\mathcal{X}_{\mathcal{B}_2}$ be $\mathcal{B}_2$'s corresponding discrete-logarithm inner-product extractor. Then Game_2 runs $\mathcal{B}_2$ and $\mathcal{X}_{\mathcal{B}_2}$ to extract $(\bar{\boldsymbol{c}} \mid\mid \bar{\boldsymbol{r}}_C)$ and $(\bar{\boldsymbol{d}} \mid\mid \bar{\boldsymbol{r}}_D)$. Then Game_2 checks whether

$$C = (\bar{\boldsymbol{c}} \mid\mid \bar{\boldsymbol{r}}_C) \times \boldsymbol{G} \ \wedge\ D = (\bar{\boldsymbol{d}} \mid\mid \bar{\boldsymbol{r}}_D) \times \bar{\boldsymbol{G}}' \ \wedge\ (\bar{\boldsymbol{c}} \mid\mid \bar{\boldsymbol{r}}_C) \times (\bar{\boldsymbol{d}} \mid\mid \bar{\boldsymbol{r}}_D) = \bar{z}$$

and returns 0 if not.

First see that Game_2 terminates in polynomial time with overwhelming probability. By the same argument as $\mathcal{B}_1, \mathcal{X}_{\mathcal{B}_1}$ we have that the expected run time of $\mathcal{B}_2, \mathcal{X}_{\mathcal{B}_2}$ is polynomial. If Game_1 returns 1 but Game_2 returns 0 then this means that $\bar{\pi}_{\mathsf{dl\text{-}inner}}$ verifies and $\mathcal{X}_{\mathcal{B}_2}$ fails, and hence that $\mathcal{B}_2$ succeeds.

$\underline{\mathsf{Game}_2 \mapsto \mathsf{Game}_3}$: Define Game_3 to be identical to Game_2 except in the following cases. When $\mathcal{X}_{\mathcal{B}_1}$ outputs $((\boldsymbol{c} \mid\mid \boldsymbol{r}_C), (\boldsymbol{d} \mid\mid \boldsymbol{r}_D))$ compute

$$\boldsymbol{b} = (d_1\beta^{-1}, \ldots, d_\ell\beta^{-\ell})) \ \wedge\ \boldsymbol{r}_B = \beta^{-(\ell+1)}\boldsymbol{r}_D - \alpha\mathbf{1}$$

After $\mathcal{X}_{\mathcal{B}_2}$ has output $((\bar{\boldsymbol{c}} \mid\mid \bar{\boldsymbol{r}}_C), (\bar{\boldsymbol{d}} \mid\mid \bar{\boldsymbol{r}}_D))$ then return 0 if $(\boldsymbol{c} \mid\mid \boldsymbol{r}_C) \neq (\bar{\boldsymbol{c}} \mid\mid \bar{\boldsymbol{r}}_C)$ or if

$$\bar{\boldsymbol{d}} \neq (b_1\beta',\ b_2\beta'^2,\ \ldots,\ b_\ell\beta'^\ell) - (1, \beta', \ldots, \beta'^\ell)$$
$$\bar{\boldsymbol{r}}_D \neq \beta'^{\ell+1}(\boldsymbol{r}_B + \alpha\mathbf{1})$$

Let $\mathcal{B}_3$ be the adversary that takes as input a q-dlog challenge $(\boldsymbol{g} \mid\mid \boldsymbol{h} \mid\mid H)$ and runs $\mathcal{B}_1$, $\mathcal{X}_{\mathcal{B}_1}$, $\mathcal{B}_2$ and $\mathcal{X}_{\mathcal{B}_2}$ as subroutines under the crs with respect to random coins ω. When $\mathcal{X}_{\mathcal{B}_1}$ and $\mathcal{X}_{\mathcal{B}_2}$ have returned $((\boldsymbol{c} \mid\mid \boldsymbol{r}_C), (\boldsymbol{d} \mid\mid \boldsymbol{r}_D))$ and $((\bar{\boldsymbol{c}} \mid\mid \bar{\boldsymbol{r}}_C), (\bar{\boldsymbol{d}} \mid\mid \bar{\boldsymbol{r}}_D))$. If $((\boldsymbol{c} \mid\mid \boldsymbol{r}_C) \neq ((\bar{\boldsymbol{c}} \mid\mid \bar{\boldsymbol{r}}_C)$ then $\mathcal{B}_3$ returns these as a correct q-dlog output.

Else compute

$$\boldsymbol{b} = (d_1\beta^{-1}, \ldots, d_\ell\beta^{-\ell}) \ \wedge\ \boldsymbol{r}_B = \beta^{-(\ell+1)}\boldsymbol{r}_D - \alpha\mathbf{1}$$

and

$$\bar{\boldsymbol{b}} = (\bar{d}_1(\beta')^{-1}, \ldots, \bar{d}_\ell(\beta')^{-\ell}) \ \wedge\ \bar{\boldsymbol{r}}_B = (\beta')^{-(\ell+1)}\bar{\boldsymbol{r}}_D - \alpha\mathbf{1}$$

If $(\boldsymbol{b} \mid\mid \boldsymbol{r}_D) \neq (\bar{\boldsymbol{b}} \mid\mid \bar{\boldsymbol{r}}_B)$ then $\mathcal{B}_3$ returns these as a correct q-dlog output.

If Game_2 returns 1 but Game_3 returns 0 then this means that $\mathcal{B}_3$ succeeds.

$\underline{\mathsf{Game}_3 \mapsto \mathsf{negl}(\lambda)}$: If Game_3 returns 0 then

$$B = \boldsymbol{b} \times \boldsymbol{g} + \boldsymbol{r}_B \times \boldsymbol{h}$$
$$C = \boldsymbol{c} \times \boldsymbol{g} + \boldsymbol{r}_C \times \boldsymbol{h}$$

is such that, for random β', we have that

$$p\beta'^\ell + r_p\beta'^{\ell+1} - 1 = (\boldsymbol{c} \mid\mid \boldsymbol{r}_C) \times (\boldsymbol{d}' \mid\mid \boldsymbol{r}'_D)$$

for random β'. Substituting for $\boldsymbol{d}'$ and $\boldsymbol{r}'_D$

$$p\beta'^\ell + r_p\beta'^{\ell+1} - 1 = \boldsymbol{c} \times \Big((b_1\beta',\ b_2\beta'^2,\ \ldots,\ b_\ell\beta'^\ell) - (1, \beta', \ldots, \beta'^{\ell-1})\Big) + \boldsymbol{r}_C \times \beta'^{\ell+1}(\boldsymbol{r}_B + \alpha\mathbf{1})$$

Expanding we see that

$$p\beta'^\ell + r_p\beta'^{\ell+1} - 1 = c_1(b_1\beta' - 1) + \ldots + c_\ell(b_\ell\beta'^\ell - \beta'^{\ell-1}) + \beta'^{\ell+1}\boldsymbol{r}_C \times (\boldsymbol{r}_B + \alpha\mathbf{1})$$

By the Schwatz-Zippel Lemma this holds with maximum probability $\frac{q_H}{|\mathbb{F}|}$ unless

$$pX^\ell + r_pX^{\ell+1} - 1 = c_1(b_1X - 1) + \ldots + c_\ell(b_\ell X^\ell - X^{\ell-1}) + X^{\ell+1}\boldsymbol{r}_C \times (\boldsymbol{r}_B + \alpha\mathbf{1})$$

If this polynomial expression holds then $r_p = \boldsymbol{r}_C \times (\boldsymbol{r}_B + \alpha\mathbf{1})$ and

$$\begin{aligned} pX^\ell - 1 &= c_1(b_1X - 1) + \ldots + c_\ell(b_\ell X^\ell - X^{\ell-1}) \\ &= -c_1 + (c_1b_1 - c_2)X + \ldots + (c_{\ell-1}b_{\ell-1} - c_\ell)X^{\ell-1} + c_\ell b_\ell X^\ell \end{aligned}$$

Thus

$$c_1 = 1,\ c_2 = b_1,\ \ldots,\ c_\ell = b_1 \ldots b_{\ell-1},\ b_1 \ldots b_\ell = p$$

This means that p is the grandproduct of $\boldsymbol{b}$ extracted by $\mathcal{X}_{\mathsf{gprod}}$ as required. □

5.3 Discrete Logarithm Inner Product Argument

In this section we discuss a zero knowledge argument for the relation

$$R_{\mathsf{dl\text{-}inner}} = \left\{ (C, D, z); \quad (\boldsymbol{c}, \boldsymbol{d}) \;\middle|\; \begin{array}{l} C = \boldsymbol{c} \times \boldsymbol{G} \\ D = \boldsymbol{d} \times \boldsymbol{G}' \\ z = \boldsymbol{c} \times \boldsymbol{d} \end{array} \right\}$$

which is given in Figures 5.6 and 5.7. This protocol was originally by Bootle et al. [BCC+16]. We make minor adjustments in order to achieve zero-knowledge. We did not use all the optimisations by Bünz et al. [BBB+18] because we decided that the improvements to the proof size is not justified by the additional cost to the verifier for our application. However we did use their method for inserting the inner product into the commitment. We prove our construction sound and zero-knowledge in Theorems 5.3.1 and 5.3.3.

5.3.1 Informal Overview

Our relation is a form of inner product relation where one is interested in verifying whether we know $(\boldsymbol{c}, \boldsymbol{d})$ such that $C = \boldsymbol{c} \times \boldsymbol{G} + zH$, $D = \boldsymbol{d} \times \boldsymbol{G}'$ where $z = \boldsymbol{c} \times \boldsymbol{d}$. The inner product argument is recursive. At each stage of the recursion, the aim is to find new commitments C', D' to values $\boldsymbol{c}'$, $\boldsymbol{d}'$ of half the length. Further we need a new z' such that $z' = \boldsymbol{c}' \times \boldsymbol{d}'$ if and only if $z = \boldsymbol{c} \times \boldsymbol{d}$. After sufficient rounds of recursion we have that $\boldsymbol{c}, \boldsymbol{d}$ are vectors of length 1, and thus can be sent in the clear. The verifier checks that the inner product relation holds for the final revealed openings, and this suffices to show that the relation holds for the original longer openings.

One initial subtlety is that C is a commitment to $(\boldsymbol{c} \,||\, 0)$ whereas the inner product argument as described in Section 5.3.1 assumes that C is a commitment to $(\boldsymbol{c} \,||\, z)$. We thus have an initial step where: (1) we obtain a random challenge by hashing the public inputs $\beta = \mathsf{Hash}(C, D, z)$; (2) the verifier updates the public input

$$C = C + z\beta H$$

to include z; (3) we update $H = \beta H$. Here the random β term prevents a cheating prover from initially providing C that is not a commitment to $(\,\cdot \,||\, 0)$.

Each round of the recursion proceeds as follows. The prover first computes cross product commitments (that will later be used to define C' and C') as

$$L_C = \boldsymbol{c}_{[:n]} \times \boldsymbol{G}_{[n:]} + (\boldsymbol{c}_{[:n]} \times \boldsymbol{d}_{[n:]})H \qquad R_C = \boldsymbol{c}_{[n:]} \times \boldsymbol{G}_{[:n]} + (\boldsymbol{c}_{[n:]} \times \boldsymbol{d}_{[:n]})H$$
$$L_D = \boldsymbol{d}_{[n:]} \times \boldsymbol{G}'_{[:n]} \qquad R_D = \boldsymbol{d}_{[:n]} \times \boldsymbol{G}'_{[n:]}$$

These are then hashed to find a random challenge γ .

The prover updates the commitment contents to

$$\boldsymbol{c}' = \boldsymbol{c}_{[:n]} + \gamma^{-1}\boldsymbol{c}_{[n:]}, \;\; \boldsymbol{d}' = \boldsymbol{d}_{[:n]} + \gamma\boldsymbol{d}_{[n:]}, z = \gamma(\boldsymbol{c}_{[:n]} \times \boldsymbol{d}_{[n:]}) + z + \gamma^{-1}(\boldsymbol{c}_{[n:]} \times \boldsymbol{d}_{[:n]})$$

such that $z' = \boldsymbol{c}' \times \boldsymbol{d}'$. See that $\boldsymbol{c}'$ and $\boldsymbol{d}'$ are half the length of $\boldsymbol{c}$ and $\boldsymbol{d}$. We then update the commitments C, D to $\boldsymbol{c}$, $\boldsymbol{d}$ and the commitment keys $\boldsymbol{G}$, $\boldsymbol{G}'$ as

$$C' = \gamma L_C + C + \gamma^{-1}R_C, \;\; D' = \gamma L_D + D + \gamma^{-1}R_D, \;\; \bar{\boldsymbol{G}} = \boldsymbol{G}_{[:n]} + \gamma\boldsymbol{G}_{[n:]}, \;\; \bar{\boldsymbol{G}}' = \boldsymbol{G}_{[:n]} + \gamma^{-1}\boldsymbol{G}_{[n:]}$$

such that $C' = \boldsymbol{c}' \times \bar{\boldsymbol{G}} + z'H$ is a commitment to $(\boldsymbol{c}', z')$ and $D' = \boldsymbol{d}' \times \bar{\boldsymbol{D}}$ is a commitment to $\boldsymbol{d}'$.

Putting this together means we have $(C', D') = (\boldsymbol{c}' \times \bar{\boldsymbol{G}} + zH, \boldsymbol{d}' \times \bar{\boldsymbol{G}}')$ for some $\boldsymbol{c}'$, $\boldsymbol{d}'$ that are half the length of $\boldsymbol{c}$, $\boldsymbol{d}$. Due to the randomised nature of γ this statement is true if and only if the original $(C, D) = (\boldsymbol{c} \times \boldsymbol{G} + (\boldsymbol{c} \times \boldsymbol{d})H, \boldsymbol{d} \times \boldsymbol{G}')$ for some $\boldsymbol{c}$, $\boldsymbol{d}$. The protocol then recurses until the final round, where $\boldsymbol{c}$ and $\boldsymbol{d}$ have length 1. Then the prover sends $\boldsymbol{c} = c_1$ and $\boldsymbol{d} = d_1$ in the clear and verifier accepts if and only if $C = cG_1 + zH$, $D = dG_1'$. Note that the full protocol has some additional masking values that are included to ensure zero-knowledge. For simplicity we have ignored these terms in this overview.

5.3.2 Full Zero-Knowledge DL Inner Product Construction

The full zero-knowledge construction for the same-permutation argument is given in Figures 5.6 and 5.7. Inner product arguments are not, by default, zero-knowledge. In order to get a zero-knowledge argument we introduce an intermediary step to randomise the provers witness.

Step 1: The prover blinds the argument by sampling $\boldsymbol{r}_C, \boldsymbol{r}_D$ randomly such that

$$\boldsymbol{r}_C \times \boldsymbol{d} + \boldsymbol{r}_D \times \boldsymbol{c} = 0 \text{ and } \boldsymbol{r}_C \times \boldsymbol{r}_D = 0$$

Then the prover computes

$$(B_C, B_D) = (\boldsymbol{r}_C \times \boldsymbol{G}, \boldsymbol{r}_D \times \boldsymbol{G}')$$

to blind the witness relating to $\boldsymbol{c}$, $\boldsymbol{d}$ respectively.

Next they hash to obtain the field elements α, β. The prover resets the private inputs to equal $\boldsymbol{r}_C + \alpha\boldsymbol{c}$ and $\boldsymbol{r}_D + \alpha\boldsymbol{d}$ and the verifier resets the public inputs to equal

$$C = B_C + \alpha C + \alpha^2 zH \text{ and } D = B_D + \alpha D$$

Observe that the updated C is a commitment to $(\boldsymbol{r}_C + \alpha\boldsymbol{c}, \alpha^2 z)$ and D is a commitment to $(\boldsymbol{r}_D + \alpha\boldsymbol{d})$. Thus

$$(\boldsymbol{r}_C + \alpha\boldsymbol{c}) \times (\boldsymbol{r}_D + \alpha\boldsymbol{d}) = \boldsymbol{r}_C \times \boldsymbol{r}_D + \alpha(\boldsymbol{r}_C \times \boldsymbol{d} + \boldsymbol{r}_D \times \boldsymbol{c}) + \alpha^2 z$$

and $\alpha^2 z$ is the correct inner product of the updated commitments.

Step 3: We now run the inner product argument as specified in Section 5.3.1. If $|\boldsymbol{G}| \geqslant 8$ then the randomisers given in $\boldsymbol{r}_C$ and $\boldsymbol{r}_D$ suffice to fully blind the inner product argument, which we argue formally in Theorem 5.3.1.

5.3.3 Security

Theorem 5.3.1 (DL inner product argument is zero-knowledge)**.** *The discrete logarithm inner product argument described in Figures 5.6 and 5.7 is zero-knowledge in the random oracle model provided that* $|\boldsymbol{G}| \geqslant 8$.

Proof. We design a simulator $\mathsf{Simulate}$ that takes as input an instance

$$(B, C, z)$$

and outputs a proof $\pi_{\mathsf{dl\text{-}inner}}$ that is indistinguishable from a proof generated by an honest prover that knows the witness. The simulator can program the random oracle on any points yet to be determined by the adversary.

The simulator $\mathsf{Simulate}$ takes as input $(\boldsymbol{G}, \boldsymbol{G}', H;\ C, D, z)$, and samples $\bar{\boldsymbol{c}}, \bar{\boldsymbol{d}}, \alpha, \beta$ randomly from the field and compute

$$\bar{C} = \bar{\boldsymbol{c}} \times \boldsymbol{G} + (\bar{\boldsymbol{c}} \times \bar{\boldsymbol{d}})H$$
$$\bar{D} = \bar{\boldsymbol{d}} \times \boldsymbol{G}'$$

Then they set

$$B_C = \bar{C} - \alpha C - \alpha^2 \beta z H$$
$$B_D = \bar{D} - \alpha D$$

and program $\mathsf{Hash}(B_C, B_D)$ to equal α, β. They rescale queries $\beta \leftarrow \mathsf{Hash}(C, D, z)$. In the remaining steps they behave exactly as the honest prover with respect to the inputs $\bar{\boldsymbol{c}}, \bar{\boldsymbol{d}}$.

Now we must argue that the simulated proof is indistinguishable from the real proof and that the simulator doesn't abort. First observe that B_C, B_D are randomly sampled so the probability that the adversary has already queried these points (causing the simulator to fail) is negligible.

Second observe that the provers commitment openings

$$\boldsymbol{r}_D + \beta \boldsymbol{d}$$

and the simulaters commitment openings $\bar{\boldsymbol{d}}$ are distributed uniformly at random and could be revealed in the clear. Hence elements relating to these values $B_D, L_{D,j}, R_{D,j}$ are identically distributed.

The proof element B_C are blinded by $r_{C,0}$ for the honest prover and $\bar{c}_0$ for the simulator. The proof elements $L_{C,1}$ are blinded by $r_{C,1}$ for the honest prover and $\bar{c}_1$ for the simulator. The proof elements $R_{C,1}$ are blinded by $r_{C,\frac{n}{2}}$ for the honest prover and $\bar{c}_{\frac{n}{2}}$ for the simulator. The updated commitment $\gamma L_{C,1} + C + \gamma^{-1} R_{C,1}$ contains

$$\bar{\boldsymbol{c}}' = \bar{\boldsymbol{c}}_{[:n]} + \gamma^{-1} \bar{\boldsymbol{c}}_{[n:]}$$

If $|\bar{\boldsymbol{c}}| \geqslant 8$ then $\bar{c}'_0$ is blinded by $\bar{c}_{\frac{n}{2}+1}$. The remaining $\bar{\boldsymbol{c}}'_{[1:]}$ is blinded by $\bar{\boldsymbol{c}}_{[1:]}$. Thus the openings $\bar{\boldsymbol{c}}', \bar{\boldsymbol{d}}', \bar{z}'$ could theoretically be revealed in the clear for $\bar{z}' = \bar{\boldsymbol{c}}' \times \bar{\boldsymbol{d}}'$. Where the remaining proof elements are deterministically computed from these openings, the provers and simulators responses are distributed identically. □

Lemma 5.3.2. *The algorithm in Steps 2 and 3 of the prover and verifier in Figures 5.6 and 5.7 is a knowledge sound argument for the relation $R_{\mathsf{dl\text{-}inner}}$ assuming the* q-dlog *problem holds.*

Proof. The algorithms implement a generalised inner product argument with respect to the commitment scheme

$$(\boldsymbol{G}, \boldsymbol{G}', H) \xleftarrow{\$} \mathsf{Setup}(\mathbb{G})$$
$$\begin{pmatrix} \boldsymbol{c} \times \boldsymbol{G} + zH \\ \boldsymbol{d} \times \boldsymbol{G}' \end{pmatrix} \leftarrow \mathsf{Commit}\left(\begin{array}{c|c} \boldsymbol{G} & \boldsymbol{c} \\ \boldsymbol{G}' & \boldsymbol{d} \\ H & z \end{array}\right)$$

and the inner product

$$\cdot : \mathbb{G} \times \mathbb{G},\ \boldsymbol{c} \cdot \boldsymbol{d} = \boldsymbol{c} \times \boldsymbol{d}$$

By Theorem 7.0.5 it suffices to show that $(\mathsf{Setup}, \mathsf{Commit}, \cdot)$ is an inner product commitment. We first show that $(\mathsf{Setup}, \mathsf{Commit})$ is binding. We second show that $(\mathsf{Setup}, \mathsf{Commit})$ is doubly homomorphic. We third show the existence of a correct $\mathsf{Collapse}$ function (Definition 7.0.8). Together these suffice to prove the lemma.

Binding commitment: Binding follows from the q-dlog assumption by Lemma 5.2.2.

Doubly homomorphic commitment: First see that the key space is homomorphic

$$\begin{aligned}
\mathsf{Commit}\left(\begin{array}{c|c} \boldsymbol{G}+\bar{\boldsymbol{G}} & \boldsymbol{c} \\ \boldsymbol{G}'+\bar{\boldsymbol{G}}' & \boldsymbol{d} \\ H+\bar{H} & z \end{array}\right) &= \begin{pmatrix} \boldsymbol{c}\times(\boldsymbol{G}+\bar{\boldsymbol{G}})+z(H+\bar{H}) \\ \boldsymbol{d}\times(\boldsymbol{G}'+\bar{\boldsymbol{G}}') \end{pmatrix} \\
&= \begin{pmatrix} \boldsymbol{c}\times\boldsymbol{G}+zH \\ \boldsymbol{d}\times\boldsymbol{G}' \end{pmatrix} + \begin{pmatrix} \boldsymbol{c}\times\bar{\boldsymbol{G}}+z\bar{H} \\ \boldsymbol{d}\times\bar{\boldsymbol{G}}' \end{pmatrix} \\
&= \mathsf{Commit}\left(\begin{array}{c|c} \boldsymbol{G} & \boldsymbol{c} \\ \boldsymbol{G}' & \boldsymbol{d} \\ H & z \end{array}\right) + \mathsf{Commit}\left(\begin{array}{c|c} \bar{\boldsymbol{G}} & \boldsymbol{c} \\ \bar{\boldsymbol{G}}' & \boldsymbol{d} \\ \bar{H} & z \end{array}\right)
\end{aligned}$$

Second see that the message space is homomorphic

$$\begin{aligned}
\mathsf{Commit}\left(\begin{array}{c|c} \boldsymbol{G} & \boldsymbol{c}+\bar{\boldsymbol{c}} \\ \boldsymbol{G}' & \boldsymbol{d}+\bar{\boldsymbol{d}} \\ H & z+\bar{z} \end{array}\right) &= \begin{pmatrix} (\boldsymbol{c}+\bar{\boldsymbol{c}})\times\boldsymbol{G}+(z+\bar{z})H \\ (\boldsymbol{d}+\bar{\boldsymbol{d}})\times\boldsymbol{G}' \end{pmatrix} \\
&= \begin{pmatrix} \boldsymbol{c}\times\boldsymbol{G}+zH \\ \boldsymbol{d}\times\boldsymbol{G}' \end{pmatrix} + \begin{pmatrix} \bar{\boldsymbol{c}}\times\boldsymbol{G}+\bar{z}H \\ \bar{\boldsymbol{d}}\times\boldsymbol{G}' \end{pmatrix} \\
&= \mathsf{Commit}\left(\begin{array}{c|c} \boldsymbol{G} & \boldsymbol{c} \\ \boldsymbol{G}' & \boldsymbol{d} \\ H & z \end{array}\right) + \mathsf{Commit}\left(\begin{array}{c|c} \boldsymbol{G} & \bar{\boldsymbol{c}} \\ \boldsymbol{G}' & \bar{\boldsymbol{d}} \\ H & \bar{z} \end{array}\right)
\end{aligned}$$

Collapsible commitment: Let $\mathsf{Collapse}$ be the identity function

$$\mathsf{Collapse}\begin{pmatrix} C \\ D \end{pmatrix} \mapsto \begin{pmatrix} C \\ D \end{pmatrix}$$

Then

$$\begin{aligned}
\mathsf{Collapse}\left(\mathsf{Commit}\left(\begin{array}{c|c} \boldsymbol{G}\,||\,\bar{\boldsymbol{G}} & (\boldsymbol{c}\,||\,\boldsymbol{c}) \\ \boldsymbol{G}'\,||\,\bar{\boldsymbol{G}}' & (\boldsymbol{d}\,||\,\boldsymbol{d}) \\ H & z \end{array}\right)\right) &= \mathsf{Collapse}\begin{pmatrix} \boldsymbol{c}\times\boldsymbol{G}+\boldsymbol{c}\times\bar{\boldsymbol{G}}+zH \\ \boldsymbol{d}\times\boldsymbol{G}'+\boldsymbol{d}\times\bar{\boldsymbol{G}}' \end{pmatrix} \\
&= \begin{pmatrix} \boldsymbol{c}\times(\boldsymbol{G}+\bar{\boldsymbol{G}})+zH \\ \boldsymbol{d}\times(\boldsymbol{G}'+\bar{\boldsymbol{G}}') \end{pmatrix} \\
&= \mathsf{Commit}\left(\begin{array}{c|c} \boldsymbol{G}+\bar{\boldsymbol{G}} & \boldsymbol{c} \\ \boldsymbol{G}'+\bar{\boldsymbol{G}}' & \boldsymbol{d} \\ H & z \end{array}\right)
\end{aligned}$$

□

Theorem 5.3.3 (DL inner product argument is knowledge-sound)**.** *The inner product argument described in Figures 5.6 and 5.7 is knowledge-sound in the random oracle model assuming the* q-dlog *is hard.*

Proof. We design an extractor $\mathcal{X}_{\mathsf{dl\text{-}inner}}$ such that: if there exists an adversary $\mathcal{A}$ that convinces the verifier with non-negligible probability then with overwhelming probability $\mathcal{X}_{\mathsf{dl\text{-}inner}}$ returns field elements $(\boldsymbol{c}, \boldsymbol{d})$ such that

$$((C, D, z);\ (\boldsymbol{c}, \boldsymbol{d})) \in R_{\mathsf{dl\text{-}inner}}.$$

By Lemma 5.3.2, whenever an adversary $\mathcal{B}$ outputs a valid proof, there exists an extractor $\mathcal{X}_{\mathcal{B}}$ that takes as input $\mathcal{B}$'s transcript and outputs $(\boldsymbol{x}, \boldsymbol{y})$ such that

$$(B_C, B_D) + \alpha(C, D) = (\boldsymbol{x} \times \boldsymbol{G} + (\boldsymbol{x} \times \boldsymbol{y})H,\ \boldsymbol{y} \times \boldsymbol{G}')$$

such that $\mathsf{Adv}^{\mathsf{dl\text{-}inner2}}_{\mathcal{B},\mathcal{X}_B}(\lambda)$ is negligible assuming the q-dlog problem is hard. Here dl-inner2 refers to the knowledge-soundness game for the (non-zk) protocol in steps 3 and 4 of Figures 5.6 and 5.7.

The extractor $\mathcal{X}_{\mathsf{dl\text{-}inner}}$ works as follows

1. Randomly sample coins ω.
2. Define an adversary $\mathcal{B}_1$ that behaves as follows:
   - Generate $\mathsf{crs} = \boldsymbol{G}, \boldsymbol{G}', H$ and set $\mathsf{trans} = 0$. While $\mathsf{trans} = 0$ run $\mathsf{trans} \leftarrow \mathsf{Game}^{\mathsf{fork}}_{\mathcal{A}}(\mathsf{crs}; \omega)$ for $\mathsf{Game}^{\mathsf{fork}}_{\mathcal{A}}(\mathsf{crs})$ defined in Section 7.0.2. Parse
     $$((\alpha, \beta), (\alpha', \beta')), ((C, D, z);\ (B_C, B_D), (\boldsymbol{\pi}, x, y), (\boldsymbol{\pi}', x', y')) = \mathsf{trans}$$
   - Return $(B_C + \alpha C + \alpha^2 \beta z H, B_D + \beta D), (\boldsymbol{\pi}, c, d)$
3. Define an adversary $\mathcal{B}_2$ that behaves as follows:
   - Compute
     $$((\alpha, \beta), (\alpha', \beta')), ((C, D, z);\ (B_C, B_D), (\boldsymbol{\pi}, x, y), (\boldsymbol{\pi}', x', y')) = \mathsf{trans}$$
     the same as $\mathcal{B}_1$
   - Return $(B_C + \beta' C + \alpha'(\beta')^2 z H, B_D + \beta' D), (\boldsymbol{\pi}', x', y')$
4. Let $\mathcal{X}_{\mathcal{B}_1}$ be $\mathcal{B}_1$'s dl-inner2 extractor. Extract $(\boldsymbol{x}, \boldsymbol{y})$ such that $B_C + \alpha C + \alpha^2 \beta z H = \boldsymbol{x} \times \boldsymbol{G} + (\boldsymbol{x} \times \boldsymbol{y})\beta H$, $B_D + \alpha D = \boldsymbol{y} \times \boldsymbol{G}'$.
5. Let $\mathcal{X}_{\mathcal{B}_2}$ be $\mathcal{B}_2$'s dl-inner2 extractor. Extract $(\boldsymbol{x}', \boldsymbol{y}')$ such that $B_C + \alpha' C + (\alpha')^2 \beta' z H = \boldsymbol{x}' \times \boldsymbol{G} + (\boldsymbol{x}' \times \boldsymbol{y}')\beta' H$, $B_D + \alpha' D = \boldsymbol{y}' \times \boldsymbol{G}'$.
6. Compute $\boldsymbol{c} = (\beta - \beta')^{-1}(\boldsymbol{x} - \boldsymbol{x}')$, $\boldsymbol{d} = (\beta - \beta')^{-1}(\boldsymbol{y} - \boldsymbol{y}')$ and return $(\boldsymbol{c}, \boldsymbol{d})$.

We must show that whenever $\mathcal{A}$ convinces the verifier then $\mathcal{X}_{\mathsf{dl\text{-}inner}}$ succeeds.

First see that $\mathcal{X}_{\mathsf{dl\text{-}inner}}$ terminates in polynomial time with overwhelming probability. Let τ be the run time of $\mathcal{A}$, ϵ be the probability that $\mathcal{A}$ outputs a valid response and q_H be the total number of hash queries that $\mathcal{A}$ can make. Assuming $\frac{8q_H}{|\mathbb{F}|} < \epsilon$ then the game $\mathsf{Game}^{\mathsf{fork}}_{\mathcal{A}}(\mathsf{par})$ runs in time at most $\frac{8\tau q_H}{\epsilon} \cdot \ln(8/\epsilon)$ and is successful with probability at least $\epsilon/8$. The expected run time of $\mathcal{B}_1, \mathcal{B}_2$

is less than $\tau q_H \cdot \ln(8/\epsilon)$, which is polynomial time assuming that τ, ϵ and q_H are polynomial in the security parameter. By Lemma 4.0.2 this means that the expected run times of $\mathcal{X}_{\mathcal{B}_1}$ and $\mathcal{X}_{B_2}$ are also polynomial.

We show that there exists $\mathcal{B}_3, \mathcal{B}_4$ such that for all extractors $\mathcal{X}_{\mathcal{B}_1}, \mathcal{X}_{\mathcal{B}_2}, \mathcal{X}_{\mathcal{B}_3}$ we have that

$$\mathsf{Adv}_{\mathcal{A},\mathcal{X}_{\mathsf{dl\text{-}inner}}}(\lambda) \leqslant \mathsf{Adv}^{\mathsf{dl\text{-}inner2}}_{\mathcal{B}_1,\mathcal{X}_1}(\lambda) + \mathsf{Adv}^{\mathsf{dl\text{-}inner2}}_{\mathcal{B}_2,\mathcal{X}_2}(\lambda) + \mathsf{Adv}^{\mathsf{dl\text{-}inner2}}_{\mathcal{B}_3,\mathcal{X}_3}(\lambda) + \mathsf{Adv}^{\mathsf{q\text{-}dlog}}_{\mathcal{B}_4}(\lambda) + \frac{q_H}{|\mathbb{F}|}$$

We proceed via a series of games $\mathsf{Game}_1, \mathsf{Game}_2, \mathsf{Game}_3$ such that

$$\begin{aligned}
\mathsf{Adv}_{\mathcal{A},\mathcal{X}_{\mathsf{dl\text{-}inner}}}(\lambda) &\leqslant \mathsf{Adv}^{\mathsf{dl\text{-}inner2}}_{\mathcal{B}_1,\mathcal{X}_{\mathcal{B}_1}}(\lambda) + \mathsf{Adv}^{\mathsf{Game}_1}_{\mathcal{A},\mathcal{X}_{\mathsf{dl\text{-}inner}}}(\lambda) \\
\mathsf{Adv}^{\mathsf{Game}_1}_{\mathcal{A},\mathcal{X}_{\mathsf{dl\text{-}inner}}}(\lambda) &\leqslant \mathsf{Adv}^{\mathsf{dl\text{-}inner2}}_{\mathcal{B}_2,\mathcal{X}_{\mathcal{B}_2}}(\lambda) + \mathsf{Adv}^{\mathsf{Game}_2}_{\mathcal{A},\mathcal{X}_{\mathsf{dl\text{-}inner}}}(\lambda) \\
\mathsf{Adv}^{\mathsf{Game}_2}_{\mathcal{A},\mathcal{X}_{\mathsf{dl\text{-}inner}}}(\lambda) &\leqslant \mathsf{Adv}^{\mathsf{dl\text{-}inner2}}_{\mathcal{B}_3,\mathcal{X}_{\mathcal{B}_3}}(\lambda) + \mathsf{Adv}^{\mathsf{dlog}}_{\mathcal{B}_4}(\lambda) + \mathsf{Adv}^{\mathsf{Game}_3}_{\mathcal{A},\mathcal{X}_{\mathsf{dl\text{-}inner}}}(\lambda) \\
\mathsf{Adv}^{\mathsf{Game}_3}_{\mathcal{A},\mathcal{X}_{\mathsf{dl\text{-}inner}}}(\lambda) &\leqslant \frac{q_H}{|\mathbb{F}|}
\end{aligned}$$

which combined give us our final result.

$\underline{\mathsf{Game}_0 \mapsto \mathsf{Game}_1}$: Let Game_0 be the initial knowledge-soundness game. Then Game_1 is identical except in the following case. If $\mathcal{A}$ outputs a verifying proof then define $\mathcal{B}_1$ be the adversary as in $\mathcal{X}_{\mathsf{dl\text{-}inner}}$ that returns $(B_C + \alpha C + \alpha^2\beta H, B_D + \alpha D), (\boldsymbol{\pi}, x, y)$ and $\mathcal{X}_{\mathcal{B}_1}$ its corresponding extractor. Return 0 if $\mathcal{X}_{\mathcal{B}_1}$ fails. If Game_0 returns 1 but Game_1 returns 0 then this means that $(\boldsymbol{\pi}, x, y)$ verifies and $\mathcal{X}_{\mathcal{B}_1}$ fails, and hence that $\mathcal{B}_1$ succeeds. By Lemma 4.0.2 the probability of this is negligible if the $\mathsf{q\text{-}dlog}$ assumption holds.

$\underline{\mathsf{Game}_1 \mapsto \mathsf{Game}_2}$: Define Game_2 to be identical to Game_1 except in the following case. If $\mathcal{A}$ outputs a verifying proof then define $\mathcal{B}_2$ be the adversary as in $\mathcal{X}_{\mathsf{dl\text{-}inner}}$ that returns $(B_C + \alpha' C + (\alpha')^2\beta' H, B_D + \alpha' D), (\boldsymbol{\pi}', x', y')$ and $\mathcal{X}_{\mathcal{B}_2}$ its corresponding extractor. Return 0 if $\mathcal{X}_{\mathcal{B}_2}$ fails. If Game_1 returns 1 but Game_2 returns 0 then this means that $(\boldsymbol{\pi}', x', y')$ verifies and $\mathcal{X}_{\mathcal{B}_2}$ fails, and hence that $\mathcal{B}_2$ succeeds. By Lemma 4.0.2 the probability of this is negligible if the $\mathsf{q\text{-}dlog}$ assumption holds.

$\underline{\mathsf{Game}_2 \mapsto \mathsf{Game}_3}$: Define Game_3 to be identical to Game_2 except in the following case.

We define an adversary $\mathcal{B}_3$ that behaves as follows:

- Generate $\mathsf{crs} = (\boldsymbol{G}, \boldsymbol{G}', H)$ and set $\mathsf{trans} = 0$. Choose random coins ω' such that during the $\mathsf{Game}^{\mathsf{fork}}_{\mathcal{A}}(\mathsf{crs})$ the original f values are sampled identically but the f' values are sampled differently. While $\mathsf{trans} = 0$ run $\mathsf{trans} \leftarrow \mathsf{Game}^{\mathsf{fork}}_{\mathcal{A}}(\mathsf{crs}; \omega')$ for $\mathsf{Game}^{\mathsf{fork}}_{\mathcal{A}}(\mathsf{crs})$ defined in Section 7.0.2. Parse
$$((\alpha, \beta), (\alpha'', \beta'')), (C, D, z); (B_C, B_D), (\boldsymbol{\pi}, x, y), (\boldsymbol{\pi}'', x'', y'')) = \mathsf{trans}$$
- Return $(B''_C + \alpha'' C + (\alpha'')^2\beta'' z H, (\boldsymbol{\pi}'', x'', y''))$

Let $\mathcal{X}_{\mathcal{B}_3}$ be $\mathcal{B}_3$'s $\mathsf{dl\text{-}inner2}$ extractor.

Then Game_3 runs $\mathcal{B}_3$ and then $\mathcal{X}_{\mathcal{B}_3}$ to extract $\boldsymbol{x}'', \boldsymbol{y}''$. Using $\mathcal{B}_1, \mathcal{X}_{\mathcal{B}_1}, \mathcal{B}_2, \mathcal{X}_{\mathcal{B}_2}$ it extracts $\boldsymbol{c}, \boldsymbol{d}, z_c$ such that

$$C = \boldsymbol{c} \times \boldsymbol{G} + z_c H, \;\; D = \boldsymbol{d} \times \boldsymbol{G}'$$

where $\boldsymbol{c}, \boldsymbol{d}$ are computed the same as in $\mathcal{X}_{\mathsf{dl\text{-}inner}}$, and

$$z_C = \beta(\alpha - \alpha')^{-1}\left(\boldsymbol{x} \times \boldsymbol{y} - \boldsymbol{x}' \times \boldsymbol{y}' - (\alpha^2 - (\alpha')^2)z\right)$$

Additionally Game_3 sets

$$\boldsymbol{r}_C = \boldsymbol{x} - \alpha\boldsymbol{c}, \boldsymbol{r}_D = \boldsymbol{y} - \alpha\boldsymbol{d},\ r_z = \beta\left(\boldsymbol{x} \times \boldsymbol{y} - \alpha z_C - \alpha^2 z\right)$$

If

$$(\boldsymbol{r}_C + \beta''\boldsymbol{c}) \times (\boldsymbol{r}_D + \alpha''\boldsymbol{d}) = (r_z + \alpha'' z_C + (\alpha'')^2 z)$$

then Game_3 returns 0.

Let $\mathcal{B}_4$ be the adversary that takes as input $\boldsymbol{G}, \boldsymbol{G}', H$ and runs $\mathcal{B}_3(\mathsf{crs};\ \omega)$ and $\mathcal{X}_{\mathcal{B}_3}$ on $\mathcal{B}_3$'s transcript, and returns either

$$\left((\boldsymbol{x}'' \mid\mid \boldsymbol{x}'' \times \boldsymbol{y}''),\ (\boldsymbol{r}_C + \alpha''\boldsymbol{c} \mid\mid r_z + \alpha'' z_C + (\alpha'')^2\beta'' z)\right)$$

or

$$\left(\boldsymbol{y}'',\ (\boldsymbol{r}_D + \alpha''\boldsymbol{d})\right)$$

If Game_2 returns 1 but Game_3 returns 0 then this means that either (1) $\boldsymbol{\pi}'', x'', y''$ verifies and $\mathcal{X}_{\mathcal{B}_3}$ fails, and hence that $\mathcal{B}_3$ succeeds; or (2) $(\boldsymbol{x}'' \mid\mid \boldsymbol{x}'' \times \boldsymbol{y}'') \neq (\boldsymbol{r}_C + \alpha''\boldsymbol{c} \mid\mid r_z + \alpha'' z_c + (\alpha'')^2\beta'' z)$ or (3) $\boldsymbol{y}'' \neq (\boldsymbol{r}_D + \alpha''\boldsymbol{d})$

In the second case we have that

$$(\boldsymbol{r}_C + \alpha''\boldsymbol{c}) \times \boldsymbol{G} + (r_z + \alpha'' z_C + (\alpha'')^2\beta'' z) = \boldsymbol{x}'' \times \boldsymbol{G} + (\boldsymbol{x}'' \times \boldsymbol{y}'')\beta'' H$$

implies that $\mathcal{B}_4$ returns a valid discrete logarithm solution. In the third case we have that

$$(\boldsymbol{r}_D + \alpha''\boldsymbol{d}) \times \boldsymbol{G}' = \boldsymbol{y}'' \times \boldsymbol{G}'$$

implies that $\mathcal{B}_4$ returns a valid discrete logarithm solution. Thus $\mathcal{B}_4$ breaks the $\mathsf{q\text{-}dlog}$ assumption.

$\underline{\mathsf{Game}_3 \mapsto \mathsf{negl}(\lambda)}$: If Game_3 returns 1 then we have that

$$(\boldsymbol{r}_C + \alpha''\boldsymbol{c}) \times (\boldsymbol{r}_D + \alpha''\boldsymbol{d})\beta'' = (r_z + \alpha'' z_C + (\alpha'')^2\beta'' z)$$

Where $\boldsymbol{r}_C, \boldsymbol{c}, \boldsymbol{r}_D, \boldsymbol{d}, r_z, z_C$ are determined before α'', β'' the probability of this occurring is bounded by $\frac{q_H}{|\mathbb{F}|}$ unless $r_z = z_C = 0$ and $z = \boldsymbol{c} \times \boldsymbol{d}$. Thus the values $\boldsymbol{c} = (\alpha - \alpha')^{-1}(\boldsymbol{x} - \boldsymbol{x}')$ and $\boldsymbol{d} = (\alpha - \alpha')^{-1}(\boldsymbol{y} - \boldsymbol{y}')$ output by $\mathcal{X}_{\mathsf{dl\text{-}inner}}$ in Game_3 is a correct witness for (C, D, z). □

$\underline{\mathsf{Prove}_{\mathsf{dl\text{-}inner}}(\mathsf{crs}_{\mathsf{dl\text{-}inner}}, (C, D, z), (\boldsymbol{c}, \boldsymbol{d}))}$

Step 1:

$\boldsymbol{r}_C, \boldsymbol{r}_D \xleftarrow{\$} \mathbb{F}^n$ such that $(\boldsymbol{r}_C \times \boldsymbol{d} + \boldsymbol{r}_D \times \boldsymbol{c}) = 0$ and $\boldsymbol{r}_C \times \boldsymbol{r}_D = 0$

$B_C \leftarrow \boldsymbol{r}_C \times \boldsymbol{G}$

$B_D \leftarrow \boldsymbol{r}_D \times \boldsymbol{G}'$

$\alpha, \beta \leftarrow \mathsf{Hash}(C, D, z, B_C, B_D)$

$\boldsymbol{c} \leftarrow \boldsymbol{r}_C + \alpha \boldsymbol{c}$

$\boldsymbol{d} \leftarrow \boldsymbol{r}_D + \alpha \boldsymbol{d}$

$H \leftarrow \beta H$

Step 2:

$m \leftarrow n$

while $1 \leqslant j \leqslant \log(m)$:

- $n \leftarrow \frac{n}{2}$
- $L_{C,j} \leftarrow \boldsymbol{c}_{[:n]} \times \boldsymbol{G}_{[n:]} + (\boldsymbol{c}_{[:n]} \times \boldsymbol{d}_{[n:]})H$
- $L_{D,j} \leftarrow \boldsymbol{d}_{[n:]} \times \boldsymbol{G}'_{[:n]}$
- $R_{C,j} \leftarrow \boldsymbol{c}_{[n:]} \times \boldsymbol{G}_{[:n]} + (\boldsymbol{c}_{[n:]} \times \boldsymbol{d}_{[:n]})H$
- $R_{D,j} \leftarrow \boldsymbol{d}_{[:n]} \times \boldsymbol{G}'_{[n:]}$
- $\pi_j \leftarrow (L_{C,j}, L_{D,j}, R_{C,j}, R_{D,j})$
- $\gamma_j \leftarrow \mathsf{Hash}(\pi_j)$
- $\boldsymbol{c} \leftarrow \boldsymbol{c}_{[:n]} + \gamma_j^{-1} \boldsymbol{c}_{[n:]}$
- $\boldsymbol{d} \leftarrow \boldsymbol{d}_{[:n]} + \gamma_j \boldsymbol{d}_{[n:]}$
- $\boldsymbol{G} \leftarrow \boldsymbol{G}_{[:n]} + \gamma_j \boldsymbol{G}_{[n:]}$
- $\boldsymbol{G}' \leftarrow \boldsymbol{G}'_{[:n]} + \gamma_j^{-1} \boldsymbol{G}'_{[n:]}$

Step 3:

$c \leftarrow c_1$

$d \leftarrow d_1$

return $(B_C, B_D, \boldsymbol{\pi}, c, d)$

Figure 5.6: Proving algorithm to demonstrate that $(C, D, z) = (\boldsymbol{c} \times \boldsymbol{G}, \boldsymbol{d} \times \boldsymbol{G}', \boldsymbol{c} \times \boldsymbol{d})$ for some $\boldsymbol{c}$, $\boldsymbol{d}$.

$\underline{\mathsf{Verify}_{\mathsf{dl\text{-}inner}}(\mathsf{crs}_{\mathsf{dl\text{-}inner}};\ \phi_{\mathsf{dl\text{-}inner}};\ \pi_{\mathsf{dl\text{-}inner}})}$

Step 1:

$(\boldsymbol{G}, \boldsymbol{G}', H) \leftarrow \mathsf{parse}(\mathsf{crs}_{\mathsf{dl\text{-}inner}})$

$(C, D, z) \leftarrow \mathsf{parse}(\phi_{\mathsf{dl\text{-}inner}})$

$(B_C, B_D, \boldsymbol{\pi}, c, d) \leftarrow \mathsf{parse}(\pi_{\mathsf{dl\text{-}inner}})$

$\alpha, \beta \leftarrow \mathsf{Hash}(C, D, z, B_C, B_D)$

$H \leftarrow \beta H$

$C \leftarrow B_C + \alpha C + (\alpha^2 z)H$

$D \leftarrow B_D + \alpha D$

Step 2:

$m \leftarrow \log(n)$

for $1 \leqslant j \leqslant m$:

$n \leftarrow \frac{n}{2}$

$(L_{C,j}, L_{D,j}, R_{C,j}, R_{D,j}) \leftarrow \mathsf{parse}(\pi_j)$

$\gamma_j \leftarrow \mathsf{Hash}(\pi_j)$

$C \leftarrow \gamma_j L_{C,j} + C + \gamma_j^{-1} R_{C,j}$

$D \leftarrow \gamma_j L_{D,j} + D + \gamma_j^{-1} R_{D,j}$

$\boldsymbol{G} \leftarrow \boldsymbol{G}_{[:n]} + \gamma_j \boldsymbol{G}_{[n:]}$

$\boldsymbol{G}' \leftarrow \boldsymbol{G}'_{[:n]} + \gamma_j^{-1} \boldsymbol{G}'_{[n:]}$

Step 3:

check $C = c \times G_1 + cdH$

check $D = d \times G'_1$

return 1 if both checks pass, else return 0.

Figure 5.7: Verify algorithm to check that $(C, D, z) = (\boldsymbol{c} \times \boldsymbol{G}, \boldsymbol{d} \times \boldsymbol{G}', \boldsymbol{c} \times \boldsymbol{d})$.

Chapter 6

Efficiency

In this section we provide a breakdown of the costs in our shuffle argument. We first provide a full overview of costs and then explain how we got to each of these numbers. We also explain any optimisations that have been used.

6.1 Full Curdleproofs Construction Efficiency

The proof consists of $18 + 10\log(\ell + 4)$ group elements and 7 field elements. To see this observe that

$$\pi_{\mathsf{shuffle}} = (A, \mathsf{cm}_T, \mathsf{cm}_U, R, S, \pi_{\mathsf{sameperm}}, \pi_{\mathsf{samescalar}}, \pi_{\mathsf{samemsm}})$$

for cm_T and cm_U group commitments and A, R, S group elements. Each group commitment consists of 2 group elements. The proofs $\pi_{\mathsf{sameperm}}, \pi_{\mathsf{samescalar}}, \pi_{\mathsf{samemsm}}$ contribute a total of $11 + 10\log(\ell + 4)$ group elements and 7 field elements.

The prover computes $30\ell + 2\log(\ell+4) + 102$ scalar multiplications. It computes $A = \sigma(\boldsymbol{a}) + \boldsymbol{r}'_A \times \boldsymbol{h}$ costing $\ell + 4$, $R = \boldsymbol{a} \times \boldsymbol{R}$ and $\boldsymbol{S} = \boldsymbol{a} \times \boldsymbol{S}$ costing 2ℓ. The group commitments to $\mathsf{cm}_T, \mathsf{cm}_U$ cost 2 each (4). It computes A' using only additions. Computing the proofs contributes a total of $23\ell + 2\log(\ell + 4) + 98$.

The verifier computes $5\ell + 10\log(\ell + 4) + 32$ scalar multiplications. Computing A' requires only additions. Computing check_4 and check_5 requires accumulating 2 MSMs, costing 2 scalar

Protocol	Proof Size	Prover Computation	Verifier Computation
Shuffle	$18 + 10\log(\ell + 4)$ $\mathbb{G}$, 7 $\mathbb{F}$	$30\ell + 2\log(\ell + 4) + 102$	$5\ell + 10\log(\ell + 4) + 32$
Same Scalar	4 $\mathbb{G}$, 3 $\mathbb{F}$	6	10
Same Permutation	$4 + 4\log(\ell + 4)$ $\mathbb{G}$, 3 $\mathbb{F}$	$11\ell + 2\log(\ell + 4) + 44$	$4\log(\ell + 4) + 9$
Same Multi Scalar	$3 + 6\log(\ell + 4)$ $\mathbb{G}$, 1 $\mathbb{F}$	$12\ell + 48$	$6\log(\ell + 4) + 6$
Accumulated MSMs	0	0	$5\ell + 5$

Table 6.1: Proof size is counted by number of group elements $\mathbb{G}$ and number of field elements $\mathbb{F}$. Prover computation counts the number of scalar multiplications. Verifier computation also counts the number of scalar multiplications.

multiplications. Verifying all of the proofs costs $10\log(\ell+4)+25$ and checking the accumulated MSMs at the end costs $5\ell+5$.

6.1.1 Verifier Optimisation: Accumulate MSM Operations

Throughout the protocol there are checks of the form

$$C \stackrel{?}{=} \boldsymbol{x} \times (\boldsymbol{g} \,||\, \boldsymbol{h} \,||\, G_T \,||\, G_U \,||\, H \,||\, \boldsymbol{R} \,||\, \boldsymbol{S} \,||\, \boldsymbol{T} \,||\, \boldsymbol{U})$$

These checks form the bottleneck of the verifiers computation and we can save significant amounts of work by accumulating these checks into a single multiscalar multiplication that is checked at the end of the protocol. This accumulated check costs $|(\boldsymbol{g} \,||\, \boldsymbol{h} \,||\, G_T \,||\, G_U \,||\, H \,||\, \boldsymbol{R} \,||\, \boldsymbol{S} \,||\, \boldsymbol{T} \,||\, \boldsymbol{U})|$ group multiplications for the verifier to check, i.e. $5\ell+5$.

We use the fact that

$$C_1 = \boldsymbol{x}_1 \times \boldsymbol{V} \text{ and } C_2 = \boldsymbol{x}_2 \times \boldsymbol{V} \Rightarrow \qquad \alpha_1 C_1 + \alpha_2 C_2 = (\alpha_1 \boldsymbol{x}_1 + \alpha_2 \boldsymbol{x}_2) \times \boldsymbol{V}$$

for all α_1, α_2. If $\alpha_1, \alpha_2 \xleftarrow{\$} \mathbb{F}$ then the probability that

$$\alpha_1 C_1 + \alpha_2 C_2 \stackrel{?}{=} (\alpha_1 \boldsymbol{x}_1 + \alpha_2 \boldsymbol{x}_2) \times \boldsymbol{V}$$

passes is negligible unless $C_1 = \boldsymbol{x}_1 \times \boldsymbol{V}$ and $C_2 = \boldsymbol{x}_2 \times \boldsymbol{V}$.

We define an operation, AccumulateCheck() that accumulates checks of the form $C \stackrel{?}{=} \boldsymbol{x} \times \boldsymbol{V}$ into a single check $A_C \stackrel{?}{=} \boldsymbol{w} \times \boldsymbol{W}$. Running AccumulateCheck() costs a single group multiplication. Then AccumulateVerify() is defined to return 1 if and only if the accumulated check passes. See Figure 6.1.

The inputs $C, \boldsymbol{x}, \boldsymbol{V}$ to AccumulateCheck must be such that $\boldsymbol{V} \subset \boldsymbol{W}$. The AccumulateCheck algorithm first generates $\boldsymbol{x}'$ which is a padded version of the $\boldsymbol{x}$ vector that has 0 entries whenever $V_i \notin \boldsymbol{W}$. It then updates $A_C = A_C + \alpha C$ and $\boldsymbol{w} = \boldsymbol{w} + \alpha \boldsymbol{x}'$ such that they include this new check. Here $\alpha \xleftarrow{\$} \mathbb{F}$ is sampled uniformly at random. To use this accumulating msms optimisation we must edit our Curdleproofs construction slightly to initialise $\boldsymbol{W}, A_C, \boldsymbol{w}$ and to run a final accumulated msm check. See Figure 6.2

Initialise $\boldsymbol{W} \leftarrow (\boldsymbol{g} \mid\mid \boldsymbol{h} \mid\mid G_T \mid\mid G_U \mid\mid H \mid\mid \boldsymbol{R} \mid\mid \boldsymbol{S} \mid\mid \boldsymbol{T} \mid\mid \boldsymbol{U})$
Initialise $A_C \leftarrow 0$
Initialise $\boldsymbol{w} \leftarrow \boldsymbol{0}$

$\underline{\mathsf{AccumulateCheck}(C \stackrel{?}{=} \boldsymbol{x} \times \boldsymbol{V})}$
for $1 \leqslant j \leqslant |\boldsymbol{W}|$:
  if $W_j = V_i$ for some i:
    $x'_j \leftarrow x_i$
  else $x'_j \leftarrow 0$

$\alpha \stackrel{\$}{\leftarrow} \mathbb{F}$
$A_C \leftarrow A_C + \alpha C$
$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha \boldsymbol{x}'$

$\underline{\mathsf{AccumulateVerify}()}$
return $A_C \stackrel{?}{=} \boldsymbol{w} \times \boldsymbol{W}$

Figure 6.1: Accumulator that gathers msm checks and verifier that evaluates the acculated msm check. The AccumulateCheck and AccumulateVerify algorithms are stateful.

6.2 Breakdown of Efficiency

6.2.1 Same Scalar Efficiency

The proof consists of 4 group elements and 3 field elements. To see this observe that

$$\pi_{\mathsf{samescalar}} = (\mathsf{cm}_A, \mathsf{cm}_B, z_k, z_T, z_U)$$

for cm_A and cm_B group commitments and z_k, z_T, z_U field elements. Each group commitment consists of 2 group elements.

The prover computes 6 scalar multiplications. It computes $A = r_k R$ and $B = r_k S$ in addition to 2 group commitments. Each group commitment costs 2 scalar multiplications.

The verifier computes 10 scalar multiplications. There are 4 from computing $\alpha\mathsf{cm}_T$ and $\alpha\mathsf{cm}_U$ where multiplying group commitments costs 2 scalar multiplications. There are 2 from computing $z_k R$ and $z_k S$. There are 4 from computing 2 group commitments.

6.2.2 Same Multiscalar Efficiency

The proof consists of $3 + 6\log(\ell + 4)$ group elements and 1 field elements. To see this observe that

$$\pi_{\mathsf{samemsm}} = (B_A, B_T, B_U, \boldsymbol{\pi}, x)$$

for B_A, B_T, B_U group elements and x a field element. The proof $\boldsymbol{\pi}$ consists of

$$\{L_{A,j}, L_{T,j}, L_{U,j}, R_{A,j}, R_{T,j}, R_{U,j}\}_{j=1}^{\log(n)}$$

for $n = |\boldsymbol{G}|$. We have that $\boldsymbol{G} = (\boldsymbol{g} \mid\mid \boldsymbol{h}_{[:n_{\mathsf{bl}}-2]} \mid\mid G_T \mid\mid G_U)$ where $|\boldsymbol{g}| = \ell$ and $|\boldsymbol{h}_{[:n_{\mathsf{bl}}-2]}| = n_{\mathsf{bl}} - 2 \geqslant 2$. Assuming $\ell \geqslant 4$ we can set $n_{\mathsf{bl}} - 2 \geqslant 2$ such that $\ell + n_{\mathsf{bl}}$ is a power of 2. We optimistically assume $n_{\mathsf{bl}} = 4$ in our efficiency evaluation.

The prover computes $12\ell + 48$ scalar multiplications. It computes $B_A = \boldsymbol{r} \times \boldsymbol{G}$, $B_T = \boldsymbol{r} \times \boldsymbol{T}$, $B_U = \boldsymbol{r} \times \boldsymbol{U}$ costing $n = |\boldsymbol{G}|$ operations each $(3n)$. During the recursion, computing $L_{A,1}, \ldots, L_{A,m}$ costs n operations (n). This is because the size of $|\boldsymbol{G}|$ is halving in each step of the recursion, thus computing $L_{A,1}$ costs $n/2$, $L_{A,2}$ costs $n/4$, ..., $L_{A,m}$ costs n/n. Together this gives us $n = (\frac{n}{2} + \frac{n}{4} + \ldots \frac{n}{n})$. Similarly, during the recursion, computing $\boldsymbol{L}_T, \boldsymbol{L}_U, \boldsymbol{R}_A, \boldsymbol{R}_T, \boldsymbol{R}_U$ costs n operations each $(5n)$ and updating $\boldsymbol{T}, \boldsymbol{U}, \boldsymbol{G}$ costs n operations each $(3n)$. Assuming $n = \ell + 4$ this gives us $12(\ell + 4)$ scalar multiplications.

The verifier computes $6\log(\ell + 4) + 6$ scalar multiplications. These operations all occur during the 3 checks added to the MSM accumulator

$$\mathsf{AccumulateCheck}(\boldsymbol{\gamma} \times \boldsymbol{L}_A + (B_A + \alpha A) + \boldsymbol{\gamma}^{-1}\boldsymbol{R}_A \stackrel{?}{=} x\boldsymbol{s} \times \boldsymbol{G})$$
$$\mathsf{AccumulateCheck}(\boldsymbol{\gamma} \times \boldsymbol{L}_T + (B_T + \alpha Z_T) + \boldsymbol{\gamma}^{-1}\boldsymbol{R}_T \stackrel{?}{=} x\boldsymbol{s} \times \boldsymbol{T})$$
$$\mathsf{AccumulateCheck}(\boldsymbol{\gamma} \times \boldsymbol{L}_U + (B_U + \alpha Z_U) + \boldsymbol{\gamma}^{-1}\boldsymbol{R}_U \stackrel{?}{=} x\boldsymbol{s} \times \boldsymbol{U})$$

Computing $\boldsymbol{\gamma} \times \boldsymbol{L}_A + (B_A + \alpha A) + \boldsymbol{\gamma}^{-1}\boldsymbol{R}_A$ costs $2\log(n) + 1$ for $n = |\boldsymbol{G}| = \ell + 4$. Similarly computing the inputs to the other accumulated checks also costs $2\log(n) + 1$. Each accumulation costs 1 scalar multiplication.

IPA Verifier Optimisation

In Figure 6.3 we describe an optimised version of our SameMultiScalar verifier. In particular we use an optimisation by Bünz et al [BBB+18] to reduce the verifier overhead in inner product arguments that is also used in the Dalek implementation of Bulletproofs.

The verifier computes only three checks in the entire SameMultiScalar argument: namely in Step 3 it checks that $A = xG_1$, $Z_T = xT_1$, and $Z_U = xU_1$. This means that although the prover needs to compute the intermediate vectors $\boldsymbol{G}, \boldsymbol{T}, \boldsymbol{U}$ at each step in order to compute the π_j values, the verifier does not and it can compute the final $A, Z_T, Z_U, G_1, T_1, U_1$ directly from the initial $A, Z_T, Z_U, \boldsymbol{G}, \boldsymbol{T}, \boldsymbol{U}$ and the $B_A, B_T, B_U, \boldsymbol{\gamma}$ values.

Using a simple example where the starting $|\boldsymbol{G}| = 8$, we see that $\boldsymbol{G}$ gets changed as follows:

$$\begin{array}{lllllllllllllllll}
\text{Start} & \boldsymbol{G} = & (G'_1 & || & G'_2 & || & G'_3 & || & G'_4 & || & G'_5 & || & G'_6 & || & G'_7 & || & G'_8) \\
\text{Fold 1} & \boldsymbol{G} = & (G'_1 & + & \gamma_1 G'_5 & || & G'_2 & + & \gamma_1 G'_6 & || & G'_3 & + & \gamma_1 G'_7 & || & G'_4 & + & \gamma_1 G'_8) \\
\text{Fold 2} & \boldsymbol{G} = & (G'_1 & + & \gamma_2 G'_3 & + & \gamma_1 G'_5 & + & \gamma_1\gamma_2 G'_7 & || & G'_2 & + & \gamma_2 G'_4 & + & \gamma_1 G'_6 & + & \gamma_1\gamma_2 G'_8) \\
\text{Fold 3} & \boldsymbol{G} = & (G'_1 & + & \gamma_3 G'_2 & + & \gamma_2 G'_3 & + & \gamma_2\gamma_3 G'_4 & + & \gamma_1 G'_5 & + & \gamma_1\gamma_3 G'_6 & + & \gamma_1\gamma_2 G'_7 & + & \gamma_1\gamma_2\gamma_3 G'_8)
\end{array}$$

such that the final G_1 value is equal to

$$(1,\ \gamma_3,\ \gamma_2,\ \gamma_2\gamma_3,\ \gamma_1,\ \gamma_1\gamma_3,\ \gamma_1\gamma_2,\ \gamma_1\gamma_2\gamma_3) \times \boldsymbol{G}$$

If we set $\boldsymbol{\delta} = (\gamma_m, \ldots, \gamma_1)$ to be the reverse of $\boldsymbol{\gamma}$ then we see a useful structure

$$G_1 = (1,\ \delta_1,\ \delta_2,\ \delta_1\delta_2,\ \delta_3,\ \delta_1\delta_3,\ \delta_2\delta_3,\ \delta_1\delta_2\delta_3) \times \boldsymbol{G}$$

namely that $G_1 = \boldsymbol{s} \times \boldsymbol{G}$ where

$$s_i = \sum_{j=1}^{m} \delta_j^{b_{i,j}} \text{ for } b_{i,j} \text{ such that } i = \sum_{j=1}^{m} b_{i,j} 2^j \text{ is the binary decomposition of } i$$

Protocol	Proof Size	Prover Computation	Verifier Computation
Same Permutation	$4 + 4\log(\ell + 4)$ $\mathbb{G}$, 3 $\mathbb{F}$	$11\ell + 2\log(\ell + 4) + 44$	$4\log(\ell + 4) + 9$
Grand Product	$3 + 4\log(\ell + 4)$ $\mathbb{G}$, 3 $\mathbb{F}$	$10\ell + 2\log(\ell + 4) + 43$	$4\log(\ell + 4) + 7$
DL IPA	$2 + 4\log(\ell + 4)$ $\mathbb{G}$, 2 $\mathbb{F}$	$8\ell + 2\log(\ell + 4) + 33$	$4\log(\ell + 4) + 5$

Table 6.2: Proof size is counted by number of group elements $\mathbb{G}$ and number of field elements $\mathbb{F}$. Prover computation counts the number of scalar multiplications. Verifier computation also counts the number of scalar multiplications.

6.2.3 Same Permutation Efficiency

The SamePerm argument uses a grandproduct argument which uses a DL IPA argument. We give a breakdown of efficiencies for each of these components in Section 6.2.3 and describe how we calculated these costs.

In Figure 6.5 we describe an optimised version of the SamePerm verifier.

The proof consists of $4 + 4\log(\ell + 4)$ group elements and 3 field elements. To see this observe that

$$\pi_{\mathsf{sameperm}} = (B, \pi_{\mathsf{gprod}})$$

for B a group element and π_{gprod} a gprod proof. The gprod argument has $3 + 4\log(\ell + 4)$ group elements and 2 field elements.

The prover computes $11\ell + 2\log(\ell + 4) + 44$ scalar multiplications. They compute $B = A + \alpha M + \boldsymbol{\beta} \times \boldsymbol{g}$ costing $\ell + 1$ operations. The prover also computes π_{gprod} costing $10\ell + 2\log(\ell + 4) + 43$ scalar multiplications.

The verifier computes $4\log(\ell + 4) + 9$ scalar multiplications. Computing p requires only field elements. They add one check to the MSM accumulator

$$\mathsf{AccumulateCheck}(B - A - \alpha M \stackrel{?}{=} \boldsymbol{\beta} \times \boldsymbol{g})$$

Computing $B - A - \alpha M$ costs 1 and the accumulation costs 1 scalar multiplication. Verifying a gprod argument uses $4\log(\ell + 4) + 7$.

Grand Product Efficiency

In Figure 6.5 we describe an optimised version of the gprod verifier.

The proof consists of $3 + 4\log(\ell + 4)$ group elements and 3 field elements. To see this observe that

$$\pi_{\mathsf{sameperm}} = \pi_{\mathsf{gprod}}$$

which has the claimed number of elements.

The prover computes $10\ell + 2\log(\ell + 4) + 43$ scalar multiplications. They compute $C = \boldsymbol{c} \times \boldsymbol{g} + \boldsymbol{r}_C \times \boldsymbol{h}$ costing $\ell + n_{\mathsf{bl}}$ operations where $n_{\mathsf{bl}} = 4$. The prover also computes $\boldsymbol{g}'$ using ℓ operations and $\boldsymbol{h}'$ using $n_{\mathsf{bl}} = 4$ operations. Setting $g_{\mathsf{sum}} = \sum_{i=1}^{\ell} g_i$ and $h_{\mathsf{sum}} = \sum_{i=1}^{\ell} h_i$ the prover computes

$$D \leftarrow B - \beta^{-1} g_{\mathsf{sum}} + \alpha h_{\mathsf{sum}}$$

in just 2 scalar multiplications. Finally the prover computes $\pi_{\mathsf{dl\text{-}inner}}$ costing $8\ell + 2\log(\ell+4) + 33$ scalar multiplications.

The verifier computes $4\log(\ell+4) + 7$ scalar multiplications. Using the first optimisation in Section 6.2.3 we see that the verifier can compute a vector $\boldsymbol{u}$ rather than $\boldsymbol{G}'$ using only field operations. Using the second optimisation in Section 6.2.3 the verifier computes D using 2 scalar multiplications. Finally the verifier checks $\pi_{\mathsf{dl\text{-}inner}}$. Verifying a dl-inner argument uses $4\log(\ell+4)+5$.

Grandproduct Verifier Optimisations

The non-optimised grandproduct verifier is required to compute a vector $\boldsymbol{G}' = \boldsymbol{u} \circ \boldsymbol{G}$ for some public vector $\boldsymbol{u}$. Then $\boldsymbol{G}'$ is used as input to the dl-inner common reference string. Computing $\boldsymbol{G}' = \boldsymbol{u} \circ \boldsymbol{G}$ would cost n scalar multiplications that cannot be accumulated efficiently. However, when we look into how the vector $\boldsymbol{G}'$ is used in dl-inner, it is used only once during

$$\mathsf{AccumulateCheck}(\boldsymbol{\gamma} \times \boldsymbol{L}_D + (B_D + \alpha D) + \boldsymbol{\gamma}^{-1} \times \boldsymbol{R}_D \stackrel{?}{=} d\boldsymbol{s}' \times \boldsymbol{G}')$$

This check is equivalent to accumulating the check

$$\mathsf{AccumulateCheck}(\boldsymbol{\gamma} \times \boldsymbol{L}_D + (B_D + \alpha D) + \boldsymbol{\gamma}^{-1} \times \boldsymbol{R}_D \stackrel{?}{=} (d\boldsymbol{s} \circ \boldsymbol{u}) \times \boldsymbol{G})$$

We thus edit the dl-inner verifier to only take the original generators as input in $\mathsf{crs}_{\mathsf{dl\text{-}inner}} = (\boldsymbol{G}, H)$, however to take $\boldsymbol{u}$ one of the public inputs $\phi_{\mathsf{dl\text{-}inner}} = (C, D, z, \boldsymbol{u})$. The accumulated check can then be run efficiently.

A second optimisation we run is that the non-optimised grandproduct verifier is required to compute a group element

$$D \leftarrow B - (1, \beta, \ldots, \beta^{\ell-1}) \times \boldsymbol{g}' + \alpha\beta^{\ell+1}\mathbf{1} \times \boldsymbol{h}'$$

for

$$\boldsymbol{g}' \leftarrow (\beta^{-1}g_1, \beta^{-2}g_2, \ldots, \beta^{-\ell}g_\ell) \ \wedge \ \boldsymbol{h}' \leftarrow \beta^{-(\ell+1)}\boldsymbol{h}$$

Expanding we see that

$$\begin{aligned}(1, \beta, \ldots, \beta^{\ell-1}) \times \boldsymbol{g}' &= (1, \beta, \ldots, \beta^{\ell-1}) \times (\beta^{-1}g_1, \beta^{-2}g_2, \ldots, \beta^{-\ell}g_\ell) \\ &= (\beta^{-1}g_1, \beta^{-1}g_2, \ldots, \beta^{-1}g_\ell) \\ &= \beta^{-1}\sum_{i=1}^{\ell} g_i\end{aligned}$$

Similarly

$$\begin{aligned}\alpha\beta^{\ell+1}\mathbf{1} \times \boldsymbol{h}' &= \alpha\beta^{\ell+1}\mathbf{1} \times \beta^{-(\ell+1)}\boldsymbol{h} \\ &= \alpha\sum_{i=1}^{n_{\mathsf{bl}}} h_i\end{aligned}$$

If we store $g_{\mathsf{sum}} = \sum_{i=1}^{\ell} g_i$ and $h_{\mathsf{sum}} = \sum_{i=1}^{\ell} h_i$ in the CRS then we can compute

$$D \leftarrow B - \beta^{-1}g_{\mathsf{sum}} + \alpha h_{\mathsf{sum}}$$

in just 2 scalar multiplications.

DL Inner Product Efficiency

The proof consists of $2 + 4\log(\ell + 4)$ group elements and 2 field elements. To see this observe that

$$\pi_{\mathsf{samemsm}} = (B_C, B_D, \boldsymbol{\pi}, c, d)$$

for B_C, B_D group elements and c, d field elements. The proof $\boldsymbol{\pi}$ consists of

$$\{L_{C,j}, L_{D,j}, R_{C,j}, R_{D,j}\}_{j=1}^{\log(n)}$$

for $n = |\boldsymbol{G}|$. We have that $\boldsymbol{G} = (\boldsymbol{g} \mid\mid \boldsymbol{h})$ where $|\boldsymbol{g}| = \ell$ and $|\boldsymbol{h}| = n_{\mathsf{bl}} = 4$. Hence $n = \ell + 4$

The prover computes $8\ell + 2\log(\ell + 4) + 33$ scalar multiplications. Set $n = |\boldsymbol{G}|$. The prover computes $B_C = \boldsymbol{r}_C \times \boldsymbol{G} + \boldsymbol{r}_C \times \boldsymbol{r}_D H$, $B_D = \boldsymbol{r}_D \times \boldsymbol{G}'$ costing $n+1$ and n operations $(2n+1)$. During the recursion, computing $L_{C,1}, \ldots, L_{C,m}$ costs $n + \log(n)$ operations $(n + \log(n)\ )$. This is because the size of $|\boldsymbol{G}|$ is halving in each step of the recursion, thus computing $L_{A,1}$ costs $n/2+1$, $L_{A,2}$ costs $n/4+1$, …, $L_{A,m}$ costs $n/n+1$. Together this gives us $n+\log(n) = (\frac{n}{2}+\frac{n}{4}+\ldots\frac{n}{n})+\log(n)$. Similarly, during the recursion, computing $\boldsymbol{R}_C$ costs $n + \log(n)$ operations and $\boldsymbol{L}_D, \boldsymbol{R}_D$ costs n operations each $(3n + \log(n))$ and updating $\boldsymbol{G}, \boldsymbol{G}'$ costs n operations each $(2n)$. Assuming $n = \ell + 4$ this gives us $8n + 2\log(n) + 1 = 8(\ell + 4) + 2\log(\ell + 4) + 1$ scalar multiplications.

The verifier computes $4\log(\ell + 4) + 5$ scalar multiplications. These operations all occur during the 2 checks added to the MSM accumulator

$$\mathsf{AccumulateCheck}(\boldsymbol{\gamma} \times \boldsymbol{L}_C + (B_C + \alpha C + (\alpha^2 z)H) + \boldsymbol{\gamma}^{-1} \times \boldsymbol{R}_C \stackrel{?}{=} (c\boldsymbol{s} \mid\mid cd\beta) \times (\boldsymbol{G} \mid\mid H))$$

$$\mathsf{AccumulateCheck}(\boldsymbol{\gamma} \times \boldsymbol{L}_D + (B_D + \alpha D) + \boldsymbol{\gamma}^{-1} \times \boldsymbol{R}_D \stackrel{?}{=} d\boldsymbol{s}' \times \boldsymbol{G}')$$

Computing $\boldsymbol{\gamma} \times \boldsymbol{L}_C + (B_C + \alpha C + (\alpha^2 z)H) + \boldsymbol{\gamma}^{-1} \times \boldsymbol{R}_C$ costs $2\log(n) + 2$ for $n = |\boldsymbol{G}| = \ell + 4$. Computing $\boldsymbol{\gamma} \times \boldsymbol{L}_D + (B_D + \alpha D) + \boldsymbol{\gamma}^{-1} \times \boldsymbol{R}_D$ costs $2\log(n) + 1$. Each accumulation costs 1 scalar multiplication.

IPA Verifier Optimisation: In Figure 6.6 we describe an optimised version of our dl-inner verifier that uses the same optimisation as in Section 6.2.2 to avoid rescaling the generators at each fold in the recursion.

6.3 Figures of Optimised Constructions

$\underline{\mathsf{Verify}_{\mathsf{shuffle}}(\mathsf{crs}_{\mathsf{shuffle}}, ;\ \phi_{\mathsf{shuffle}};\ \pi_{\mathsf{shuffle}})}$
Step 1:
$(\boldsymbol{g}, \boldsymbol{h}, G_T, G_U, H, g_{\mathsf{sum}}, h_{\mathsf{sum}}) \leftarrow \mathsf{parse}(\mathsf{crs}_{\mathsf{shuffle}})$
$(\boldsymbol{R}, \boldsymbol{S}, \boldsymbol{T}, \boldsymbol{U}, M) \leftarrow \mathsf{parse}(\phi_{\mathsf{shuffle}})$
$(A, \mathsf{cm}_T, \mathsf{cm}_U, R, S, \pi_{\mathsf{sameperm}}, \pi_{\mathsf{samescalar}}, \pi_{\mathsf{samemsm}}) \leftarrow \mathsf{parse}(\pi_{\mathsf{shuffle}})$
$\boldsymbol{a} = (a_1, \ldots, a_\ell) \leftarrow \mathsf{Hash}(\boldsymbol{R}, \boldsymbol{S}, \boldsymbol{T}, \boldsymbol{U}, M)$

$\boldsymbol{W} \leftarrow (\boldsymbol{g} \mid\mid \boldsymbol{h} \mid\mid G_T \mid\mid G_U \mid\mid H \mid\mid \boldsymbol{R} \mid\mid \boldsymbol{S} \mid\mid \boldsymbol{T} \mid\mid \boldsymbol{U})$
$A_C \leftarrow 0$
$\boldsymbol{w} \leftarrow \boldsymbol{0}$

Step 2:
$\mathsf{check}_1 \leftarrow \mathsf{Verify}_{\mathsf{sameperm}}((\boldsymbol{g}, \boldsymbol{h}, H, g_{\mathsf{sum}}, h_{\mathsf{sum}});\ (A, M);\ \pi_{\mathsf{sameperm}})$

Step 3:
$\mathsf{check}_2 \leftarrow \mathsf{Verify}_{\mathsf{samescalar}}((G_T, G_U, H);\ (R, S, \mathsf{cm}_T, \mathsf{cm}_U);\ \pi_{\mathsf{samescalar}})$

Step 4:
$A' \leftarrow A + \mathsf{cm}_{T,1} + \mathsf{cm}_{U,1}$
$\boldsymbol{G} \leftarrow (\boldsymbol{g} \mid\mid \boldsymbol{h}_{[:n_{\mathsf{bl}}-2]} \mid\mid G_T \mid\mid G_U)$
$\boldsymbol{T}' \leftarrow (\boldsymbol{T} \mid\mid \boldsymbol{0} \mid\mid H \mid\mid 0)$
$\boldsymbol{U}' \leftarrow (\boldsymbol{U} \mid\mid \boldsymbol{0} \mid\mid 0 \mid\mid H)$
$\mathsf{check}_3 \leftarrow \mathsf{Verify}_{\mathsf{samemsm}}(\boldsymbol{G};\ (A', \mathsf{cm}_{T,2}, \mathsf{cm}_{U,2}, \boldsymbol{T}', \boldsymbol{U}'); \pi_{\mathsf{samemsm}})$

$\mathsf{check}_4 \leftarrow \mathsf{AccumulateCheck}(R \stackrel{?}{=} \boldsymbol{a} \times \boldsymbol{R})$
$\mathsf{check}_5 \leftarrow \mathsf{AccumulateCheck}(S \stackrel{?}{=} \boldsymbol{a} \times \boldsymbol{S})$
$\mathsf{check}_6 \leftarrow \mathsf{AccumulateVerify}()$
return 1 if $(\mathsf{check}_1, \mathsf{check}_2, \mathsf{check}_3, \mathsf{check}_4, \mathsf{check}_5, \mathsf{check}_6) = (1, 1, 1, 1, 1, 1)$
else return 0

Figure 6.2: The optimised Curdleproofs verification algorithm to check that $\boldsymbol{T}, \boldsymbol{U} = \sigma(k\boldsymbol{R}), \sigma(k\boldsymbol{S})$ for some unknown field element k and unknown permutation σ committed in M.

$\underline{\mathsf{Verify}_{\mathsf{samemsm}}(\mathsf{crs}_{\mathsf{samemsm}};\ \phi_{\mathsf{samemsm}};\ \pi_{\mathsf{samemsm}})}$

Step 1:

$\boldsymbol{G} \leftarrow \mathsf{parse}(\mathsf{crs}_{\mathsf{samemsm}})$

$(A, Z_T, Z_U, \boldsymbol{T}, \boldsymbol{U}) \leftarrow \mathsf{parse}(\phi_{\mathsf{samemsm}})$

$(B_A, B_T, B_U, \boldsymbol{\pi}, x) \leftarrow \mathsf{parse}(\pi_{\mathsf{samemsm}})$

$\alpha \leftarrow \mathsf{Hash}(A, Z_T, Z_U, \boldsymbol{T}, \boldsymbol{U}, B_A, B_T, B_U)$

Step 2:

$m \leftarrow \log(n)$

for $1 \leqslant j \leqslant m$:

$(L_{A,j}, L_{T,j}, L_{U,j}, R_{A,j}, R_{T,j}, R_{U,j}) \leftarrow \mathsf{parse}(\pi_j)$

$\gamma_j \leftarrow \mathsf{Hash}(\pi_j)$

Step 3:

$\boldsymbol{\delta} \leftarrow (\gamma_m, \ldots, \gamma_1)$

for $1 \leqslant j \leqslant n$:

$s_i = \sum_{j=1}^{m} \delta_j^{b_{i,j}}$ for $b_{i,j} \in \{0, 1\}$ such that $i = \sum_{j=1}^{m} b_{i,j} 2^j$

$\mathsf{AccumulateCheck}(\boldsymbol{\gamma} \times \boldsymbol{L}_A + (B_A + \alpha A) + \boldsymbol{\gamma}^{-1} \boldsymbol{R}_A \stackrel{?}{=} x\boldsymbol{s} \times \boldsymbol{G})$

$\mathsf{AccumulateCheck}(\boldsymbol{\gamma} \times \boldsymbol{L}_T + (B_T + \alpha Z_T) + \boldsymbol{\gamma}^{-1} \boldsymbol{R}_T \stackrel{?}{=} x\boldsymbol{s} \times \boldsymbol{T})$

$\mathsf{AccumulateCheck}(\boldsymbol{\gamma} \times \boldsymbol{L}_U + (B_U + \alpha Z_U) + \boldsymbol{\gamma}^{-1} \boldsymbol{R}_U \stackrel{?}{=} x\boldsymbol{s} \times \boldsymbol{U})$

return 1

Figure 6.3: Optimised SameMultiScalar verification algorithm to check that that $(A, Z_T, Z_U) = (\boldsymbol{x} \times \boldsymbol{G}, \boldsymbol{x} \times \boldsymbol{T}, \boldsymbol{x} \times \boldsymbol{U})$ for some vector of field elements $\boldsymbol{x}$. Here we use $\boldsymbol{\gamma}^{-1}$ to denote $(\gamma_1^{-1}, \ldots, \gamma_m^{-1})$. This verifier is satisfied if and only if the verifier in Figure 4.2 is satisfied.

$\underline{\mathsf{Verify}_{\mathsf{sameperm}}(\mathsf{crs}_{\mathsf{sameperm}}, ;\ \phi_{\mathsf{sameperm}};\ \pi_{\mathsf{sameperm}})}$

Step 1:

$(\boldsymbol{g}, \boldsymbol{h}, H, g_{\mathsf{sum}}, h_{\mathsf{sum}}) \leftarrow \mathsf{parse}(\mathsf{crs}_{\mathsf{sameperm}})$

$(A, M, \boldsymbol{a}) \leftarrow \mathsf{parse}(\phi_{\mathsf{sameperm}})$

$(B, \pi_{\mathsf{gprod}}) \leftarrow \mathsf{parse}(\pi_{\mathsf{sameperm}})$

$(\alpha, \beta) \leftarrow \mathsf{Hash}(\boldsymbol{R}, \boldsymbol{S}, \boldsymbol{T}, \boldsymbol{U}, M)$

Step 2:

$p \leftarrow \prod_{i=1}^{\ell}(a_i + i\alpha + \beta)$

$\mathsf{AccumulateCheck}(B - A - \alpha M \stackrel{?}{=} \boldsymbol{\beta} \times \boldsymbol{g})$

$\mathsf{check} \leftarrow \mathsf{Verify}_{\mathsf{gprod}}((\boldsymbol{g}, \boldsymbol{h}, H);\ (B, p);\ \pi_{\mathsf{gprod}})$

return 1 if $\mathsf{check} = 1$

else return 0

Figure 6.4: Optimised sameperm verification algorithm to check that A, M are commitments to $\sigma(\boldsymbol{a}), \sigma(1, \ldots, \ell)$ for some permutation σ. Here we denote $\boldsymbol{\beta} = (\beta, \beta^2, \ldots, \beta^\ell)$

$\underline{\mathsf{Verify}_{\mathsf{gprod}}(\mathsf{crs}_{\mathsf{gprod}};\ \phi_{\mathsf{gprod}};\ \pi_{\mathsf{gprod}})}$

Step 1:
$(\boldsymbol{g}, \boldsymbol{h}, H, g_{\mathsf{sum}}, h_{\mathsf{sum}}) \leftarrow \mathsf{parse}(\mathsf{crs}_{\mathsf{gprod}})$
$(B, p) \leftarrow \mathsf{parse}(\phi_{\mathsf{gprod}})$
$(C, r_p, \pi_{\mathsf{dl\text{-}inner}}) \leftarrow \mathsf{parse}(\pi_{\mathsf{gprod}})$
$\alpha \leftarrow \mathsf{Hash}(B, p)$

Step 2:
$\beta \leftarrow \mathsf{Hash}(C, r_p)$

Step 3:
$\boldsymbol{u} \leftarrow (\beta^{-1}, \beta^{-2}, \ldots,\ \beta^{-\ell}\ ||\ \beta^{-(\ell+1)}\mathbf{1})$
$D \leftarrow B - \beta^{-1} g_{\mathsf{sum}} + \alpha h_{\mathsf{sum}}$

Step 4:
$\boldsymbol{G} \leftarrow (\boldsymbol{g}\ ||\ \boldsymbol{h})$
$z \leftarrow p\beta^{\ell} + r_p \beta^{\ell+1} - 1$
$\mathsf{check} \leftarrow \mathsf{Verify}_{\mathsf{dl\text{-}inner}}((\boldsymbol{G}, H), (C, D, z, \boldsymbol{u}), \pi_{\mathsf{dl\text{-}inner}})$

return 1 if $\mathsf{check} = 1$
else return 0

Figure 6.5: Verify algorithm to check that the prover knows $\boldsymbol{b}, \boldsymbol{r}_B$ such that $B = \boldsymbol{b} \times \boldsymbol{g} + \boldsymbol{r}_B \times \boldsymbol{h}$ such that $p = \prod_{i=1}^{\ell} b_i$. Here $g_{\mathsf{sum}} = \sum_{i=1}^{\ell} g_i$ and $h_{\sum} = \sum_{i=1}^{n_{\mathsf{bl}}} h_i$

$\underline{\mathsf{Verify}_{\mathsf{dl\text{-}inner}}(\mathsf{crs}_{\mathsf{dl\text{-}inner}};\ \phi_{\mathsf{dl\text{-}inner}};\ \pi_{\mathsf{dl\text{-}inner}})}$

Step 1:

$(\boldsymbol{G}, H) \leftarrow \mathsf{parse}(\mathsf{crs}_{\mathsf{dl\text{-}inner}})$

$(C, D, z, \boldsymbol{u}) \leftarrow \mathsf{parse}(\phi_{\mathsf{dl\text{-}inner}})$

$(B_C, B_D, \boldsymbol{\pi}, c, d) \leftarrow \mathsf{parse}(\pi_{\mathsf{dl\text{-}inner}})$

$\alpha, \beta \leftarrow \mathsf{Hash}(C, D, z, B_C, B_D)$

Step 2:

$m \leftarrow \log(n)$

for $1 \leqslant j \leqslant m$:

$n \leftarrow \frac{n}{2}$

$(L_{C,j}, L_{D,j}, R_{C,j}, R_{D,j}) \leftarrow \mathsf{parse}(\pi_j)$

$\gamma_j \leftarrow \mathsf{Hash}(\pi_j)$

Step 3:

$\boldsymbol{\delta} \leftarrow (\gamma_m, \ldots, \gamma_1)$

for $1 \leqslant j \leqslant n$:

$s_i = \sum_{j=1}^{m} \delta_j^{b_{i,j}}$ for $b_{i,j} \in \{0, 1\}$ such that $i = \sum_{j=1}^{m} b_{i,j} 2^j$

$s'_i = \sum_{j=1}^{m} \delta_j^{-b_{i,j}}$

$\mathsf{AccumulateCheck}(\boldsymbol{\gamma} \times \boldsymbol{L}_C + (B_C + \alpha C + (\alpha^2 z)H) + \boldsymbol{\gamma}^{-1} \times \boldsymbol{R}_C \stackrel{?}{=} (c\boldsymbol{s} \mid\mid cd\beta) \times (\boldsymbol{G} \mid\mid H))$

$\mathsf{AccumulateCheck}(\boldsymbol{\gamma} \times \boldsymbol{L}_D + (B_D + \alpha D) + \boldsymbol{\gamma}^{-1} \times \boldsymbol{R}_D \stackrel{?}{=} d(\boldsymbol{s}' \circ \boldsymbol{u}) \times \boldsymbol{G})$

return 1

Figure 6.6: Optimised dl-inner verification algorithm to check that $(C, D, z) = (\boldsymbol{c} \times \boldsymbol{G}, \boldsymbol{d} \times \boldsymbol{G}', \boldsymbol{c} \times \boldsymbol{d})$.

Chapter 7

Deferred Security Preliminaries

Assumption 7.0.1 (q-ddh assumption)**.** *For an adversary $\mathcal{A}$, define* $\mathsf{Adv}^{q\text{-ddh}}_{\mathcal{A}}(\lambda) = |1-2\Pr[\mathsf{Game}^{q\text{-ddh}}_{\mathcal{A}}]|$ *where* $\mathsf{Game}^{q\text{-ddh}}_{\mathcal{A}}$ *is given by*

$$\frac{\text{MAIN } \mathsf{Game}^{q\text{-ddh}}_{\mathcal{A}}(\lambda)}{\mathbb{G} \leftarrow \mathsf{GroupGen}(\lambda)}$$
$$b \xleftarrow{\$} \{0,1\} \text{ if } b = 0\text{:}$$
$$\quad \boldsymbol{A} \xleftarrow{\$} \mathbb{G}^q; \quad y \xleftarrow{\$} \mathbb{F},\ B \leftarrow y \times g; \quad \boldsymbol{C} \leftarrow y\boldsymbol{A}$$
$$\text{if } b = 1\text{:}$$
$$\quad \boldsymbol{A} \xleftarrow{\$} \mathbb{G}^q; \quad y \xleftarrow{\$} \mathbb{F},\ B \leftarrow y \times g; \quad \boldsymbol{C} \xleftarrow{\$} \mathbb{G}^q$$
$$b' \xleftarrow{\$} \mathcal{A}(\mathbb{G}, \boldsymbol{A}, B, \boldsymbol{C})$$
$$\text{return } b' = b$$

The q-ddh assumption holds if for all PPT adversaries $\mathcal{A}$ we have that $\mathsf{Adv}^{q\text{-ddh}}_{\mathcal{A}}(\lambda) \leqslant \mathsf{negl}(\lambda)$ *is negligible in λ.*

Lemma 7.0.2. *The q-ddh assumption is implied by the* ddh *assumption.*

Proof. We shall show that $\mathsf{Adv}^{q\text{-ddh}}_{\mathcal{A}}(\lambda) \leqslant 2q\mathsf{Adv}^{\mathsf{ddh}}_{\mathcal{B}}(\lambda)$. We do this by hopping through a series of hybrids $\mathsf{Game}^1, \ldots, \mathsf{Game}^q$ such that Game^q is statistically hard.

$\underline{\mathsf{Game}_i \mapsto \mathsf{Game}_{i+1}}$: Let Game_0 be the initial q-ddh game. Define Game_1 to run identically to Game_0 except that, C_1 is chosen randomly for both $b = 0$ and $b = 1$. Similarly define Game_i to run identically to Game_{i-1} except that, C_i is chosen randomly for both $b = 0$ and $b = 1$.

Let $\mathcal{B}_i$ be an adversary against the ddh game. Then $\mathcal{B}_i$ simulates Game_i for $\mathcal{A}$. It takes as input $(\mathbb{G}, R, S, T)$ and selects $a_1, \ldots, a_q \xleftarrow{\$} \mathbb{F}$. Then $\mathcal{B}_i$ flips a coin to get $b \in \{0,1\}$. If $b = 0$ then $\mathcal{B}_i$ sets

$$A_j, C_j \leftarrow \begin{cases} g^{a_j}, S^{a_j} \text{ if } j < i \\ R, T \text{ if } j = i \\ \text{random if } j > i \end{cases}, \quad B \leftarrow S$$

and $b = 1$ then $\mathcal{B}_i$ generates $\boldsymbol{A}, B, \boldsymbol{C}$ randomly. Next $\mathcal{B}_i$ sets $\phi_i = (\mathbb{G}, \boldsymbol{A}, B, \boldsymbol{C})$. Finally $\mathcal{B}_i$ runs $b' \xleftarrow{\$} \mathcal{A}(\phi_i)$. When $\mathcal{A}$ returns b', then if $b = b'$ then $\mathcal{B}_i$ returns 0, else $\mathcal{B}_i$ returns 1.

Then

$$\begin{aligned}
&\Pr[\mathcal{B}_i(\mathbb{G}, R, S, T) = 0 \mid \bar{b} = 0] \\
&\quad = \frac{1}{2}\left(\Pr[\mathcal{A}(\phi_i) = 0 \mid \mathsf{Game}_{i-1}, b = 0] + \Pr[\mathcal{A}(\phi_i) = 1 \mid \mathsf{Game}_{i-1}, b = 1]\right) \\
&\Pr[\mathcal{B}_i(\mathbb{G}, R, S, T) = 1 \mid \bar{b} = 1] \\
&\quad = \frac{1}{2}\left(2 - \Pr[\mathcal{A}(\phi_i) = 0 \mid \mathsf{Game}_i, b = 0] - \Pr[\mathcal{A}(\phi_i) = 1 \mid \mathsf{Game}_i, b = 1]\right)
\end{aligned}$$

and

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{ddh}}_{\mathcal{B}_i}(\lambda) &= |1 - 2\Pr[\mathsf{Game}^{\mathsf{ddh}}_{\mathcal{B}_i}(\lambda)]| \\
&= |1 - \Pr[\mathcal{B}(\mathbb{G}, R, S, T) = 0 \mid \bar{b} = 0] - \Pr[\mathcal{B}(\mathbb{G}, R, S, T) = 1 \mid \bar{b} = 1]| \\
&= \Big|1 - \frac{1}{2}\left(\Pr[\mathcal{A}(\phi_i) = 0 \mid \mathsf{Game}_0, b = 0] + \Pr[\mathcal{A}(\phi_i) = 1 \mid \mathsf{Game}_0, b = 1]\right) \\
&\qquad - \left(1 - \Pr[\mathcal{B}(\mathbb{G}, R, S, T) = 0 \mid \bar{b} = 1]\right)\Big| \\
&= \frac{1}{2}\Big|1 - \Pr[\mathcal{A}(\phi_i) = 0 \mid \mathsf{Game}_0, b = 0] - \Pr[\mathcal{A}(\phi_i) = 1 \mid \mathsf{Game}_0, b = 1] \\
&\qquad\quad - 1 + \Pr[\mathcal{A}(\phi_i) = 0 \mid \mathsf{Game}_1, b = 0] + \Pr[\mathcal{A}(\phi_i) = 1 \mid \mathsf{Game}_0, b = 1]\Big| \\
&= \frac{1}{2}|\mathsf{Adv}^{\mathsf{Game}_{i-1}}_{\mathcal{A}}(\lambda) - \mathsf{Adv}^{\mathsf{Game}_i}_{\mathcal{A}}(\lambda)|
\end{aligned}$$

This means that

$$\mathsf{Adv}^{\mathsf{Game}_{i-1}}_{\mathcal{A}}(\lambda) \leqslant 2\mathsf{Adv}^{\mathsf{ddh}}_{\mathcal{B}}(\lambda) + \mathsf{Adv}^{\mathsf{Game}_i}_{\mathcal{A}}(\lambda)$$

Observe that $\mathsf{Adv}^{\mathsf{Game}_q}_{\mathcal{A}}(\lambda) = 0$ for all $\mathcal{A}$ because the adversary receives identically distributed inputs on both coin flips. Hence we have that $\mathsf{Adv}^{\mathsf{Game}_{q\text{-}\mathsf{ddh}}}_{\mathcal{A}}(\lambda) \leqslant 2q\mathsf{Adv}^{\mathsf{ddh}}_{\mathcal{B}}(\lambda)$ □

Assumption 7.0.3 (q-dlog assumption)**.** *For an adversary $\mathcal{A}$ define* $\mathsf{Adv}^{\mathsf{q\text{-}dlog}}_{\mathcal{A}}$ *to be*

$$\mathsf{Adv}^{\mathsf{q\text{-}dlog}}_{\mathcal{A}}(\lambda) = \Pr[\boldsymbol{x} \times \boldsymbol{g} = \boldsymbol{y} \times \boldsymbol{g} \ \wedge \ \boldsymbol{x} \neq \boldsymbol{y} \mid \mathbb{G} \leftarrow \mathsf{GroupGen}(\lambda),\ \boldsymbol{g} \xleftarrow{\$} \mathbb{G}^q, (\boldsymbol{x}, \boldsymbol{y}) \xleftarrow{\$} \mathcal{A}(\mathbb{G}, \boldsymbol{g})]$$

The q-dlog *assumption holds if for all PPT adversaries $\mathcal{A}$ we have that* $\mathsf{Adv}^{\mathsf{q\text{-}dlog}}_{\mathcal{A}}(\lambda) \leqslant \mathsf{negl}(\lambda)$ *is negligible in λ.*

Lemma 7.0.4. *The* q-dlog *assumption is implied by the* dlog *assumption.*

Proof. Let $\mathcal{A}$ be an adversary that breaks q-dlog. We describe an adversary $\mathcal{B}$ against the discrete logarithm assumption. The adversary $\mathcal{B}$ takes as input (g, h) and chooses $i \xleftarrow{\$} [1, q]$ (for q the length of $\boldsymbol{g}$). It chooses $\boldsymbol{a}$ to have random entries in $\mathbb{F}$ such that $a_i = 0$ but all other values are strictly non-zero. It sets $\boldsymbol{g}$ such that $g_i = h$ and $g_j = a_j g$ for all other $j \neq i$. Observe that $\boldsymbol{g}$ is perfectly distributed as a valid q-dlog instance. Then $\mathcal{B}$ runs

$$(\boldsymbol{x}, \boldsymbol{y}) \xleftarrow{\$} \mathcal{A}(\boldsymbol{g})$$

If $\mathcal{A}$ wins then $\boldsymbol{x} \times \boldsymbol{g} = \boldsymbol{y} \times \boldsymbol{g}$ for $\boldsymbol{x} \neq \boldsymbol{y}$.

Thus

$$(\boldsymbol{x} - \boldsymbol{y}) \times \boldsymbol{g} = 0 \Rightarrow (y_i - x_i)h = ((\boldsymbol{x} - \boldsymbol{y}) \times \boldsymbol{a})g$$

If $x_i \neq y_i$ then $\mathcal{B}$ returns the discrete logarithm $(y_i - x_i)^{-1}((\boldsymbol{x} - \boldsymbol{y}) \times \boldsymbol{a})$.

If $\boldsymbol{x} \neq \boldsymbol{y}$ then $y_i \neq x_i$ with probability $\frac{1}{q}$ and thus

$$\mathsf{Adv}^{\mathsf{q\text{-}dlog}}_{\mathcal{A}}(\lambda) \leqslant q\mathsf{Adv}^{\mathsf{dlog}}_{\mathcal{B}}(\lambda)$$

□

7.0.1 Generalised Inner Product Arguments

Bünz et al. [BMM+21] prove a general theorem for the knowledge soundness of inner product arugments that we refer to during our security proofs. We state this theorem below. This theorem is given in the interactive model. It can be compiled into a non-interactive argument by replacing the verifier challenges with hash queries. This is secure in the random oracle model due to the transforms by Attema et al. [AFK21] and by Wikström [Wik21]. Alternatively it has been proven that compiling inner product arguments into a non-interactive arguments is secure in the algebraic group/commitment model [GT21, BMM+21].

Theorem 7.0.5 (Theorem 1 from [BMM+21])**.** *If* $((\mathsf{Setup}, \mathsf{Commit}), \cdot)$ *is a binding inner product commitment (see Definition 7.0.8), then* $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *from [[BMM+21], Figure 1] has completeness and knowledge soundness for the relation*

$$R_{\mathsf{IPA}} = \left(\ (\mathsf{ck}, \mathsf{cm});\ (\boldsymbol{a}, \boldsymbol{b}) \ \middle|\ \mathsf{cm} = \mathsf{Commit}(\mathsf{ck};\ (\boldsymbol{a}, \boldsymbol{b}, \langle \boldsymbol{a}, \boldsymbol{b} \rangle)) \ \right)$$

To prove the knowledge soundness of an inner product argument that instantiates their generalised argument for a specific commitment scheme, it thus suffices to show that the commitment is an *inner product commitment*. We summarise their definition of an inner product commitment in Section 7.0.1.

Inner Product Commitment

A commitment scheme consists of two algorithms $(\mathsf{Setup}, \mathsf{Commit})$ where

- $\mathsf{ck} \xleftarrow{\$} \mathsf{Setup}(\mathbb{G})$ takes as input some public parameters defined by a security parameter (in our case the group description $\mathbb{G}$) and outputs a commitment key ck. We denote the key space $\mathcal{K}$ to be the set of possible commitment keys.
- $\mathsf{cm} \xleftarrow{\$} \mathsf{Commit}(\mathsf{ck};\ a;\ r)$ takes as input the commitment key, a message, and some randomness. Ir outputs a commitment cm. We denote the message space $\mathcal{M}$ to be the set of possible message inputs. If Commit is deterministic (i.e. $r = \perp$) then we simply write $\mathsf{Commit}(\mathsf{ck};\ a)$

We say that a commitment scheme is binding (see [BMM+21], Definition 9) if it is hard to find a, r, a', r' such that $\mathsf{Commit}(\mathsf{ck};\ a; r) = \mathsf{Commit}(\mathsf{ck};\ a';\ r')$ but $m \neq m'$.

Definition 7.0.6 (Doubly homomorphic commitment scheme [BMM+21])**.** *A binding commitment scheme* $(\mathsf{Setup}, \mathsf{Commit})$ *is doubly homomorphic if* $(\mathcal{K}, +), (\mathcal{M}, +)$ *and* $(\mathsf{Image}(\mathsf{Commit}), +)$ *define abelian groups such that for all* $\mathsf{ck}, \mathsf{ck}' \in \mathcal{K}$ *and all* $a, a' \in \mathcal{M}$ *it holds that*

1. $\mathsf{Commit}(\mathsf{ck};\ a) + \mathsf{Commit}(\mathsf{ck};\ a') = \mathsf{Commit}(\mathsf{ck};\ a + a')$

2. $\mathsf{Commit}(\mathsf{ck};\ a) + \mathsf{Commit}(\mathsf{ck}';\ a) = \mathsf{Commit}(\mathsf{ck} + \mathsf{ck}';\ a)$

Definition 7.0.7 (Inner product [BMM+21])**.** *A function $\cdot : \mathcal{M}_1, \mathcal{M}_2 \mapsto \mathcal{M}_3$ from two groups of prime order p to a third group of prime order p is and* inner product map *if for all $a, b \in \mathbb{G}_1$ and $c, d \in \mathbb{G}_2$ we have that*

$$(a + b) \cdot (c + d) = a \cdot c + a \cdot d + b \cdot c + b \cdot d$$

Given an inner product $\cdot$ between groups we define the inner product between vector spaces $\langle\rangle : \mathcal{M}_1^m \times \mathcal{M}_2^m \mapsto \mathcal{M}_3$ to be

$$\langle \boldsymbol{a}, \boldsymbol{b} \rangle = \sum_{i=1}^{m} a_i \cdot b_i$$

Definition 7.0.8 (Inner Product Commitment [BMM+21])**.** *Let* $(\mathsf{Setup}, \mathsf{Commit})$ *be a doubly homomorphic commitment with message space $\mathcal{M} = \mathcal{M}_1^m \times \mathcal{M}_2^m \times \mathcal{M}_3$ and key space $\mathcal{K} = \mathcal{K}_1^m \times \mathcal{K}_2^m \times \mathcal{K}_3$ defined for all $m \in \mathbb{N}$, where $|\mathcal{M}_i| = |\mathcal{K}_i| = p$ is prime for $1 \leqslant i \leqslant 3$. Let $\cdot : \mathcal{M}_1 \times \mathcal{M}_2 \mapsto \mathcal{M}_3$. We call* $(\mathsf{Setup}, \mathsf{Commit}, \cdot)$ *an inner product commitment if there exists an efficient deterministic functions* $\mathsf{Collapse}$ *such that for all $m \in \{2^j\}_{j \in \mathbb{N}}$, $a \in \mathcal{M}$, and $\mathsf{ck}, \mathsf{ck}' \in \mathcal{K}$ such that $\mathsf{ck}_3 = \mathsf{ck}'_3$ it holds that*

$$\mathsf{Collapse}\left(\mathsf{Commit}\left(\begin{array}{cc|c} \mathsf{ck}_1 & \mathsf{ck}'_1 & a_1 \\ \mathsf{ck}_2 & \mathsf{ck}'_2 & a_2 \\ \mathsf{ck}_3 & & a_3 \end{array}\right)\right) = \mathsf{Commit}\left(\begin{array}{c|c} \mathsf{ck}_1 + \mathsf{ck}'_1 & a_1 \\ \mathsf{ck}_2 + \mathsf{ck}'_2 & a_2 \\ \mathsf{ck}_3 & a_3 \end{array}\right)$$

We refer to the requirement above as the collapsing property.

7.0.2 The Generalized Forking Lemma

Several of the non-interactive protocols in this work rely on rewinding an adversary in order to extract a valid witness. The extractors will require at least two successful transcripts with respect to the same first messsage. We thus use the generalised forking lemma as proven by Bagherzandi et al.[BCJ08] (first stated in [BN06]) to quantify the probability of an extractor obtaining these transcripts. Consider an algorithm $\mathcal{A}$ that takes as input some parameters par and some randomness $f = (\omega, h_1, \ldots, h_Q)$, where ω is $\mathcal{A}$'s random coins and $h_1, \ldots, h_Q$ are responses received by querying a random oracle $\mathsf{Hash} : \{0,1\}^* \mapsto \mathbb{F}$, and Q is the maximal number of the hash queries. Suppose $\mathcal{A}$ outputs a pair $(L, \{\mathsf{out}_\ell\}_{\ell \in L})$, where L is a set of hash responses, and each out_ℓ is the corresponding output message for $\ell \in L$. Let ϵ be the probability that the output of $\mathcal{A}(\mathsf{par}, f)$ is successful. We define the game $\mathsf{Game}^{\mathsf{fork}}_{\mathcal{A}}(\mathsf{par})$ as in Section 7.0.2.

Lemma 7.0.9 (Generalized Forking Lemma [BCJ08])**.** *Let* $\mathsf{Hash} : \{0,1\}^* \mapsto \mathbb{F}$ *be a uniformly random function, and $\mathcal{A}$ be an adversary that runs in time τ and succeeds with probability ϵ. If $|\mathbb{F}| > 8mQ/\epsilon$, then the game* $\mathsf{Game}^{\mathsf{fork}}_{\mathcal{A}}(\mathsf{par})$ *runs in time at most $\tau \cdot 8m^2Q/\epsilon \cdot \ln(8m/\epsilon)$, and is successful with probability at least $\epsilon/8$.*

$\underline{\text{MAIN}(\mathsf{par})}$

$f = (\omega, h_1, \ldots, h_Q) \xleftarrow{\$} \mathbb{F}$

$(L, \{\mathsf{out}_\ell\}_{\ell \in L}) \xleftarrow{\$} \mathcal{A}(\mathsf{par}, f)$

If $L = \varnothing$ output 0

Else, let $L = (\ell_1, \ldots, \ell_m)$ be such that $\ell_1 \leqslant \cdots \leqslant \ell_m$

for $1 \leqslant i \leqslant m$:

  set $\mathsf{succ}_i \leftarrow 0$, $k_i \leftarrow 0$, $k_{max} \leftarrow 8mQ/\epsilon \cdot (\ln(8m/\epsilon))$

  while $\mathsf{succ}_i = 0$ and $k_i < k_{max}$:

    $f' \xleftarrow{\$} \mathbb{F}$ such that $f'_{|\ell_i} = f_{|\ell_i}$

    let $f' = (\omega, h_1, \ldots, h_{\ell_{i-1}}, h'_{\ell_i}, \ldots, h'_Q)$

    $(L', \{\mathsf{out}'_\ell\}_{\ell \in L'}) \leftarrow \mathcal{A}(\mathsf{par}, f')$

    if $h'_{\ell_i} \neq h_{\ell_i}$ and $L' \neq \varnothing$ and $\ell_i \in L'$ then keep out'_{ℓ_i} and set $\mathsf{succ} \leftarrow 1$

    else set $k_i \leftarrow k_i + 1$

If $\mathsf{succ}_i = 1$ for all $i \in [m]$, then output $(L, \{\mathsf{out}_\ell\}_{\ell \in L}, \{\mathsf{out}'_\ell\}_{\ell \in L})$

Else output 0

Figure 7.1: Game $\mathsf{Game}^{\mathsf{fork}}_{\mathsf{par},\mathcal{A}}$ where an algorithm is forked in the Generalized Forking Lemma.

Bibliography

[ABG+21] Diego F. Aranha, Carsten Baum, Kristian Gjøsteen, Tjerand Silde, and Thor Tunge. Lattice-based proof of shuffle and applications to electronic voting. In Kenneth G. Paterson, editor, *Topics in Cryptology - CT-RSA 2021 - Cryptographers' Track at the RSA Conference 2021, Virtual Event, May 17-20, 2021, Proceedings*, volume 12704 of *Lecture Notes in Computer Science*, pages 227–251. Springer, 2021.

[AC21] Sarah Azouvi and Daniele Cappelletti. Private attacks in longest chain proof-of-stake protocols with single secret leader elections. In Foteini Baldimtsi and Tim Roughgarden, editors, *AFT '21: 3rd ACM Conference on Advances in Financial Technologies, Arlington, Virginia, USA, September 26 - 28, 2021*, pages 170–182. ACM, 2021.

[ACF21] Thomas Attema, Ronald Cramer, and Serge Fehr. Compressing proofs of k-out-of-n partial knowledge. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part IV*, volume 12828 of *Lecture Notes in Computer Science*, pages 65–91. Springer, 2021.

[AFK21] Thomas Attema, Serge Fehr, and Michael Klooß. Fiat-shamir transformation of multi-round interactive proofs. *IACR Cryptol. ePrint Arch.*, page 1377, 2021.

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Gregory Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, pages 315–334, 2018.

[BCC+16] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. 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*, pages 327–357, 2016.

[BCJ08] Ali Bagherzandi, Jung Hee Cheon, and Stanislaw Jarecki. Multisignatures secure under the discrete logarithm assumption and a generalized forking lemma. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *Proceedings of the 2008 ACM Conference on Computer and Communications Security, CCS 2008, Alexandria, Virginia, USA, October 27-31, 2008*, pages 449–458. ACM, 2008.

[BCS21] Jonathan Bootle, Alessandro Chiesa, and Katerina Sotiraki. Sumcheck arguments and their applications. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology*

- *CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part I*, volume 12825 of *Lecture Notes in Computer Science*, pages 742–773. Springer, 2021.

[BEHG20] Dan Boneh, Saba Eskandarian, Lucjan Hanzlik, and Nicola Greco. Single secret leader election. In *AFT '20: 2nd ACM Conference on Advances in Financial Technologies, New York, NY, USA, October 21-23, 2020*, pages 12–24. ACM, 2020.

[BG12] Stephanie Bayer and Jens Groth. 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*, pages 263–280, 2012.

[BMM+21] Benedikt Bünz, Mary Maller, Pratyush Mishra, Nirvan Tyagi, and Psi Vesely. Proofs for inner pairing products and applications. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part III*, volume 13092 of *Lecture Notes in Computer Science*, pages 65–97. Springer, 2021.

[BN06] Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, October 30 - November 3, 2006*, pages 390–399. ACM, 2006.

[CFG21] Dario Catalano, Dario Fiore, and Emanuele Giunta. Efficient and universally composable single secret leader election from pairings. *IACR Cryptol. ePrint Arch.*, page 344, 2021.

[Cha03] David Chaum. Untraceable electronic mail, return addresses and digital pseudonyms. In Dimitris Gritzalis, editor, *Secure Electronic Voting*, volume 7 of *Advances in Information Security*, pages 211–219. Springer, 2003.

[CMM19] Núria Costa, Ramiro Martínez, and Paz Morillo. Lattice-based proof of a shuffle. In Andrea Bracciali, Jeremy Clark, Federico Pintore, Peter B. Rønne, and Massimiliano Sala, editors, *Financial Cryptography and Data Security - FC 2019 International Workshops, VOTING and WTSC, St. Kitts, St. Kitts and Nevis, February 18-22, 2019, Revised Selected Papers*, volume 11599 of *Lecture Notes in Computer Science*, pages 330–346. Springer, 2019.

[DK00] Yvo Desmedt and Kaoru Kurosawa. How to break a practical MIX and design a new one. In Bart Preneel, editor, *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, volume 1807 of *Lecture Notes in Computer Science*, pages 557–572. Springer, 2000.

[FLSZ17] Prastudy Fauzi, Helger Lipmaa, Janno Siim, and Michal Zajac. An efficient pairing-based shuffle argument. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in*

Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part II, volume 10625 of *Lecture Notes in Computer Science*, pages 97–127. Springer, 2017.

[GT21] Ashrujit Ghoshal and Stefano Tessaro. Tight state-restoration soundness in the algebraic group model. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part III*, volume 12827 of *Lecture Notes in Computer Science*, pages 64–93. Springer, 2021.

[HKR19] Max Hoffmann, Michael Klooß, and Andy Rupp. Efficient zero-knowledge arguments in the discrete log setting, revisited. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 2093–2110. ACM, 2019.

[HMS21] Javier Herranz, Ramiro Martínez, and Manuel Sánchez. Shorter lattice-based zero-knowledge proofs for the correctness of a shuffle. In Matthew Bernhard, Andrea Bracciali, Lewis Gudgeon, Thomas Haines, Ariah Klages-Mundt, Shin'ichiro Matsuo, Daniel Perez, Massimiliano Sala, and Sam Werner, editors, *Financial Cryptography and Data Security. FC 2021 International Workshops - CoDecFin, DeFi, VOTING, and WTSC, Virtual Event, March 5, 2021, Revised Selected Papers*, volume 12676 of *Lecture Notes in Computer Science*, pages 315–329. Springer, 2021.

[JT20] Joseph Jaeger and Stefano Tessaro. Expected-time cryptography: Generic techniques and applications to concrete soundness. In Rafael Pass and Krzysztof Pietrzak, editors, *Theory of Cryptography - 18th International Conference, TCC 2020, Durham, NC, USA, November 16-19, 2020, Proceedings, Part III*, volume 12552 of *Lecture Notes in Computer Science*, pages 414–443. Springer, 2020.

[LMR19] Russell W. F. Lai, Giulio Malavolta, and Viktoria Ronge. Succinct arguments for bilinear group arithmetic: Practical structure-preserving cryptography. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 2057–2074. ACM, 2019.

[Nef01] C. Andrew Neff. A verifiable secret shuffle and its application to e-voting. In Michael K. Reiter and Pierangela Samarati, editors, *CCS 2001, Proceedings of the 8th ACM Conference on Computer and Communications Security, Philadelphia, Pennsylvania, USA, November 6-8, 2001*, pages 116–125. ACM, 2001.

[RMM21] Michael Rosenberg, Mary Maller, and Ian Miers. Snarkblock: Federated anonymous blocklisting from hidden common input aggregate proofs. *IACR Cryptol. ePrint Arch.*, page 1577, 2021.

[Wik21] Douglas Wikström. Special soundness in the random oracle model. *IACR Cryptol. ePrint Arch.*, page 1265, 2021.

[WTS+18] Riad S. Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zksnarks without trusted setup. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, pages 926–943. IEEE Computer Society, 2018.