A Shuffle Argument Protocol Specification
Work in Progress

The Ethereum Foundation Research Team

September 10, 2020

Abstract

In this document we describe a shuffle argument inspired by the work of Bayer and Groth [BG12]. This argument is motivated by its potential applications to secret leader elections which is a vital component of Eth 2.0. The argument runs over a public coin setup in any group where the DDH assumption holds.

Asymptotically the prover and the verifier both run in linear time in the number of ciphertexts. The proof size is logarithmic although we note that the instance (i.e. the shuffled ciphertexts) is linear size.

This is currently a work in progress.

Contents

1 Notation

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

$$R_{\mathsf{rel}} = \{ \ (\phi, w) \ \big| \ \text{properties that } \phi \text{ and } w \text{ satisfy } \}.$$

Proving algorithms Prove take as input $(\mathsf{crs}_{\mathsf{rel}}, \phi, w)$ where $\mathsf{crs}_{\mathsf{rel}}$ is a common reference string that includes a description of the relation R_{rel} and where $(\phi, w) \in R_{\mathsf{rel}}$. They return a proof π_{rel}.

Verification algorithms Verify 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.

2 Problem Statement

The aim of the construction is to build a shuffle argument of ciphertexts. More precisely, given a public set of El-Gamal ciphertexts

$$(R_1, S_1), \ldots, (R_\ell, S_\ell)$$

a shuffler computes a second set of El-Gamal ciphertexts

$$(T_1, U_1), \ldots, (T_\ell, U_\ell)$$

and proves in zero knowledge that there exists a permutation

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

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

$$T_i = R_{\sigma(i)}^r \ \wedge \ U_i = S_{\sigma(i)}^r.$$

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

$$R_{\mathsf{shuffle}} = \left\{ \begin{array}{l} (((R_1, S_1), \ldots, (R_\ell, S_\ell)) \in \mathbb{G}^{2\times\ell}, \\ ((T_1, U_1), \ldots, (T_\ell, U_\ell)) \in \mathbb{G}^{2\times\ell}), \\ (\sigma \in \text{permutations over } [1, \ldots, \ell], r \in \mathbb{F}) \end{array} \middle| \ T_i = R_{\sigma(i)}^r, U_i = S_{\sigma(i)}^r \text{ for } 1 \leq i \leq \ell \right\}$$

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.

3 Public Coin Setup

4 Construction

We begin by giving a full overview of the construction in Figures 1 and 2 which is proven secure in Theorems 7.1 and 7.2. As part of our full construction we require zero-knowledge algorithms for proving and verifying three additional relations, a grand-product relation, a same exponent relation, and a multiexponentiation relation. We specify these relations below and specify the proving and verifying algorithms in Sections ???.

4.1 Grand Product Relation

The grand-product relation demonstrates that given public input $(A \in \mathbb{G}, \mathsf{gprod} \in \mathbb{F})$ there exists $(a_1, \ldots, a_{\ell+4})$ such that $A = \mathsf{compute\_multiexp}(\mathsf{crs}, (a_1, \ldots, a_{\ell+4}))$ and $\mathsf{gprod} = \prod_{i=1}^{\ell} a_i$. In other words

$$R_{\mathsf{gprod}} = \left\{ \ (A_1, \mathsf{gprod}), (a_1, \ldots, a_{\ell+4}) \ \middle| \ A_1 = g_1^{a_1} \cdots g_{\ell+4}^{a_{\ell+4}} \wedge \mathsf{gprod} = a_1 \cdots a_\ell \ \right\}$$

4.2 Same Exponent Relation

The same exponent relation demonstrates that given public input $(R, S, T, U) \in \mathbb{G}^4$ there exists $r, s_1, s_2 \in \mathbb{F}$ such that $T = R^r g_t^{s_1}$ and $U = S^r g_u^{s_2}$. In other words

$$R_{\mathsf{sameexp}} = \left\{ \ (R, S, T, U) \in \mathbb{G}^4, (r, s_1, s_2) \in \mathbb{F}^3 \ \middle| \ T = R^r g_t^{s_1} \ \wedge \ U = S^r g_u^{s_2} \ \right\}$$

4.3 Multiexponentiation Relation

The multiexponentiation relation demonstrates that given public input $(T_1, U_1, \ldots, T_{\ell+4}, U_{\ell+4}) \in \mathbb{G}^{2\times(\ell+4)}$, $A \in \mathbb{G}$, and $(T, U) \in \mathbb{G}^2$ there exists $(a_1, \ldots, a_{\ell+4})$ such that

$$\begin{aligned} A &= g_1^{a_1} \cdots g_{\ell+4}^{a_{\ell+4}} \\ T &= T_1^{a_1} \cdots T_{\ell+4}^{a_{\ell+4}} \\ U &= U_1^{a_1} \cdots U_{\ell+4}^{a_{\ell+4}}. \end{aligned}$$

In other words

$$R_{\mathsf{multiexp}} = \left\{ \begin{array}{l} ((T_1, U_1, \ldots, T_{\ell+4}, U_{\ell+4}) \in \mathbb{G}^{2(\ell+4)}, A \in \mathbb{G}, T \in \mathbb{G}, U \in \mathbb{G}), \\ (a_1, \ldots, a_{\ell+4}) \in \mathbb{F}^{\ell+4} \end{array} \middle| \begin{array}{l} A = \prod_{i=1}^{\ell+4} g_i^{a_i} \ \wedge \\ T = \prod_{i=1}^{\ell+4} T_i^{a_i} \ \wedge \\ U = \prod_{i=1}^{\ell+4} U_i^{a_i} \end{array} \right\}.$$

4.4 Overview

A formal description of the shuffle argument is provided in Figures 1 and 2. Here we give an overview of the protocol. We prove zero-knowledge, and soundness in Section 7, Theorems 7.2 and 7.1.

Step 1

Prover: The prover samples $s_1, s_2, s_3, s_4 \xleftarrow{\$} \mathbb{F}$ randomly from $\mathbb{F}$ as masking values. They compute $M \in \mathbb{G}$ as

$$M = g_{\ell+1}^{s_1} g_{\ell+2}^{s_2} g_{\ell+3}^{s_3} g_{\ell+4}^{s_4} \prod_{i=1}^{\ell} g_i^{\sigma(i)}$$

where $\mathsf{crs}_g = (g_1, \ldots, g_{\ell+4}) \in \mathbb{G}^{\ell+4}$. They hash the shuffled ciphertexts (T_1, U_1), …, $(T_\ell, U_\ell) \in \mathbb{G}^{2\times\ell}$ and M to obtain the field elements $a_1, \ldots, a_\ell \in \mathbb{F}^\ell$.

Verifier: The verifier hashes the shuffled ciphertexts (T_1, U_1), …, $(T_\ell, U_\ell) \in \mathbb{G}^{2\times\ell}$ and $M \in \mathbb{G}$ to obtain the field elements $a_1, \ldots, a_\ell \in \mathbb{F}^\ell$.

Step 2

Prover: The prover samples $a_{\ell+1}, a_{\ell+2}, a_{\ell+3}, a_{\ell+4} \xleftarrow{\$} \mathbb{F}$ randomly from $\mathbb{F}$ as masking values. They compute $A \in \mathbb{G}$ as

$$A = g_{\ell+1}^{a_{\ell+1}} g_{\ell+2}^{a_{\ell+2}} g_{\ell+3}^{a_{\ell+3}} g_{\ell+4}^{a_{\ell+4}} \prod_{i=1}^{\ell} g_i^{a_{\sigma(i)}}$$

They hash A to obtain the field elements $\alpha, \beta \in \mathbb{F}$.

Verifier: The verifier hashes A to obtain the field element $\alpha, \beta \in \mathbb{F}$.

Step 3

Prover: The prover computes $\mathsf{gprod} \in \mathbb{F}$ as $\prod_{i=1}^{\ell}(a_i+i\alpha+\beta)$. They set $A_1 \in \mathbb{G}$ as $AM^{\alpha}(g_1 \dots g_{\ell+4})^{\beta}$. They compute a prove π_{gprod} that they know $(c_1, \dots, c_{\ell+4})$ such that $A_1 = \prod_{i=1}^{\ell+4} g_i^{c_i}$ and $\prod_{i=1}^{\ell+4} g_i = \mathsf{gprod}$. In other words they run

$$\pi_{\mathsf{gprod}} = \mathsf{Prove}(\mathsf{crs}_{\mathsf{gprod}}, (A_1, \mathsf{gprod}), (a_{\sigma(1)}+\sigma(1)\alpha+\beta, \dots, a_{\sigma(\ell)}+\sigma(\ell)\alpha+\beta, a_{\ell+1}+\alpha s_1+\beta, \dots, a_{\ell+4}+\alpha s_4+\beta)).$$

Verifier: The verifier computes $\mathsf{gprod} \in \mathbb{F}$ as $\prod_{i=1}^{\ell}(a_i+i\alpha+\beta)$. They set $A_1 \in \mathbb{G}$ as $AM^{\alpha}(g_1 \dots g_{\ell+4})^{\beta}$ where $\mathsf{crs}_g = (g_1, \dots, g_{\ell+4}) \in \mathbb{G}^{\ell+4}$. They verify the grand-product proof π_{gprod} for the instance (A_1, gprod). In other words they run

$$b_1 = \mathsf{Verify}(\mathsf{crs}_{\mathsf{gprod}}, (A_1, \mathsf{gprod}), \pi_{\mathsf{gprod}}).$$

and reject the shuffle proof if $b_1 = 0$.

Step 4

Prover: The prover computes $R, S \in \mathbb{G}^2$ as the multiexponentiations $R = \prod_{i=1}^{\ell} R_i^{a_i}$ and $S = \prod_{i=1}^{\ell} S_i^{a_i}$. They hash A to obtain the field elements $\gamma_1, \delta_1, \dots, \gamma_4, \delta_4 \in \mathbb{F}^4$. They compute $T, U \in \mathbb{G}^2$ as $T = R^r g_t^{\gamma_1 a_{\ell+1} + \dots \gamma_4 a_{\ell+4}}$ and $U = S^r g_u^{\delta_1 a_{\ell+1} + \dots \delta_4 a_{\ell+4}}$ where $r \in \mathbb{F}$ is the provers initial secret such that $T_i = R_{\sigma(i)}^r$ and $U_i = S_{\sigma(i)}^r$ for all $1 \le i \le \ell$. They compute a same-exponentiation argument π_{sameexp} for the instance (R, S, T, U). In other words they run

$$\pi_{\mathsf{sameexp}} = \mathsf{Prove}(R_{\mathsf{sameexp}}, (R, S, T, U), (r, (\gamma_1 a_{\ell+1} + \dots + \gamma_4 a_{\ell+4}), (\delta_1 a_{\ell+1} + \dots + \delta_4 a_{\ell+4}))).$$

Verifier: The verifier computes $R, S \in \mathbb{G}^2$ as the multiexponentiations $R = \prod_{i=1}^{\ell} R_i^{a_i}$ and $S = \prod_{i=1}^{\ell} S_i^{a_i}$. They hash A to obtain the field elements $\gamma_1, \delta_1, \dots, \gamma_4, \delta_4 \in \mathbb{F}^4$. They verify the same-exponentiation proof π_{sameexp} for the instance $(R, S, T, U) \in \mathbb{G}^4$. In other words they run

$$b_2 = \mathsf{Verify}(R_{\mathsf{sameexp}}, (R, S, T, U), \pi_{\mathsf{sameexp}}).$$

and reject the shuffle proof if $b_2 = 0$.

Step 5
Prover: The prover runs a multi-exponentiation argument to demonstrate knowledge of $(c_1, \ldots, c_n)$ such that $A, T, U \in \mathbb{G}^3$ are equal to

$$A = \prod_{i=1}^{\ell+4} g_i^{c_i} \quad T = g_t^{\gamma_1 c_{\ell+1} + \ldots + \gamma_4 c_{\ell+4}} \prod_{i=1}^{\ell} T_i^{c_i} \quad U = g_t^{\delta_1 c_{\ell+1} + \ldots + \delta_4 c_{\ell+4}} \prod_{i=1}^{\ell} U_i^{c_i}.$$

In other words they run

$$\pi_{\mathsf{multiexp}} = \mathsf{Prove}(\mathsf{crs}_{\mathsf{multiexp}}, (T_1, U_1, \ldots, T_\ell, U_\ell, g_t^{\gamma_1}, g_u^{\delta_1}, \ldots, g_t^{\gamma_4}, g_u^{\delta_4}, A, T, U), (a_{\sigma(1)}, \ldots, a_{\sigma(\ell)}, a_{\ell+1}, \ldots, a_{\ell+4})).$$

Verifier: The verifier verifies the multiexponentiation proof π_{multiexp} with respect to the instance $(T_1, U_1, \ldots, T_\ell, U_\ell, g_t^{\gamma_1}, g_u^{\delta_1}, \ldots, g_t^{\gamma_4}, g_u^{\delta_4}, A, T, U) \in \mathbb{G}^{2\ell+11}$. In other words they run

$$b_3 = \mathsf{Verify}(\mathsf{crs}_{\mathsf{multiexp}}, (T_1, U_1, \ldots, T_\ell, U_\ell, g_t^{\gamma_1}, g_u^{\delta_1}, \ldots, g_t^{\gamma_4}, g_u^{\delta_4}, A, T, U), \pi_{\mathsf{multiexp}}).$$

and reject the shuffle proof if $b_3 = 0$.

Outcome
The prover returns the proof $\pi_{\mathsf{shuffle}} = (M, A, T, U, \pi_{\mathsf{gprod}}, \pi_{\mathsf{sameexp}}, \pi_{\mathsf{multiexp}})$.

The verifier returns 1 if $(b_1, b_2, b_3) = (1, 1, 1)$ and otherwise returns 0.

5 Grand Product Argument

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

$$R_{\mathsf{gprod}} = \left\{ \; (A_1, \mathsf{gprod}), (a_1, \ldots, a_n) \;\middle|\; A_1 = g_1^{a_1} \ldots g_n^{a_n} \wedge \mathsf{gprod} = a_1 \ldots a_\ell \; \right\}$$

which is given in Figures 3 and 4. An overview of our argument is given in Section 5.2. We prove our construction sound and zero-knowledge in Theorems ???. As part of our construction we require zero-knowledge algorithms for proving and verifying an additional relation R_{DLInner}. We specify this inner product discrete logarithm relation below and its proving and verifying algorithms in Section 6.

5.1 Discrete logarithm inner product relation

The discrete-logarithm inner product relation demonstrates that given public input $B, C \in \mathbb{G}, z \in \mathbb{F}$ there exists $(b_1, \ldots, b_{\ell+4})$ and $(c_1, \ldots, c_{\ell+4})$ such that $B = \prod_{i=1}^{\ell+4} g_i^{b_i}$ and $C = \prod_{i=1}^{\ell+4} g_i^{c_i}$. In other words

$$R_{\mathsf{DLInner}} = \left\{ \begin{array}{l} (B, C, z), ((b_1, \ldots, b_{\ell+2}), (c_1, \ldots, c_{\ell+2})) \end{array} \middle| \begin{array}{l} B = g_1^{b_1} \cdots g_{\ell+4}^{b_{\ell+4}} \wedge C = g_1^{c_1} \cdots g_{\ell+4}^{c_{\ell+4}} \\ \wedge \; z = b_1 c_1 + \ldots + b_{\ell+4} c_{\ell+4} \end{array} \right\}$$

$\underline{\mathsf{ShuffleProve}(\mathsf{crs}_{\mathsf{shuffle}}, R_1, S_1, T_1, U_1, \ldots, R_\ell, S_\ell, T_\ell, U_\ell, \sigma, r)}$

Step 1:

$(R_{\mathsf{shuffle}}, (\mathsf{crs}_g, u)) \leftarrow \mathsf{parse}(\mathsf{crs}_{\mathsf{shuffle}})$

$s_1, s_2, s_3, s_4 \xleftarrow{\$} \mathbb{F}$

$M \leftarrow g_1^{\sigma(1)} \cdots g_\ell^{\sigma(\ell)} g_{\ell+1}^{s_1} \cdots g_{\ell+4}^{s_4}$

$(a_1, \ldots, a_\ell) \leftarrow \mathsf{Hash}(T_1, \ldots, T_\ell, U_1, \ldots, U_\ell, M)$

Step 2:

$a_{\ell+1}, \ldots, a_{\ell+4} \xleftarrow{\$} \mathbb{F}$

$A \leftarrow g_1^{a_{\sigma(1)}} \cdots g_\ell^{a_{\sigma(\ell)}} g_{\ell+1}^{a_{\ell+1}} \cdots g_{\ell+4}^{a_{\ell+4}}$

$\alpha, \beta \leftarrow \mathsf{Hash}(A)$

Step 3:

$\mathsf{gprod} \leftarrow (a_1 + 1 \cdot \alpha + \beta)(a_2 + 2 \cdot \alpha + \beta) \cdots (a_\ell + \ell \cdot \alpha + \beta)$

$A_1 \leftarrow AM^\alpha (g_1 \ldots g_{\ell+4})^\beta$

$$\pi_{\mathsf{gprod}} \leftarrow \mathsf{Prove}\left(\begin{array}{l}\mathsf{crs}_{\mathsf{gprod}}, (A_1, \mathsf{gprod}), \\ (a_{\sigma(1)} + \sigma(1)\alpha + \beta, \ldots, a_{\sigma(\ell)} + \sigma(\ell)\alpha + \beta, a_{\ell+1} + s_1\alpha + \beta, \ldots, a_{\ell+4} + s_4\alpha + \beta)\end{array}\right)$$

Step 4:

$\gamma_1, \delta_1, \ldots, \gamma_4, \delta_4 \leftarrow \mathsf{Hash}(A)$

$R \leftarrow R_1^{a_1} \cdots R_\ell^{a_\ell}$

$S \leftarrow S_1^{s_1} \cdots S_\ell^{a_\ell}$

$T \leftarrow R^r g_t^{\gamma_1 a_{\ell+1} + \ldots + \gamma_4 a_{\ell+4}}$

$U \leftarrow S^r g_u^{\delta_1 a_{\ell+1} + \ldots + \delta_4 a_{\ell+4}}$

$\pi_{\mathsf{sameexp}} \leftarrow \mathsf{Prove}(R_{\mathsf{sameexp}}, (R, S, T, U), r)$

Step 5:

$$\pi_{\mathsf{multiexp}} \leftarrow \mathsf{Prove}\left(\begin{array}{l}\mathsf{crs}_{\mathsf{multiexp}}, ((T_1, U_1, \ldots, T_\ell, U_\ell, g_t^{\gamma_1}, g_u^{\delta_1}, \ldots, g_t^{\gamma_4}, g_u^{\delta_4}), A, (T, U)), \\ (a_{\sigma(1)}, \ldots, a_{\sigma(\ell)}, a_{\ell+1}, a_{\ell+2})\end{array}\right)$$

return $(M, A, T, U, \pi_{\mathsf{gprod}}, \pi_{\mathsf{sameexp}}, \pi_{\mathsf{multiexp}})$

Figure 1: Proving algorithm to demonstrate that $(T_1, U_1), \ldots, (T_\ell, U_\ell) = (R_{\sigma(1)}^r, S_{\sigma(1)}^r), \ldots, (R_{\sigma(\ell)}^r, S_{\sigma(\ell)}^r)$ for some field element r and permutation σ.

$\underline{\mathsf{ShuffleVerify}(\mathsf{crs}_{\mathsf{shuffle}}, R_1, S_1, T_1, U_1, \ldots, R_\ell, S_\ell, T_\ell, U_\ell, \pi_{\mathsf{shuffle}})}$

Step 1:
$(R_{\mathsf{shuffle}}, (\mathsf{crs}_g, u)) \leftarrow \mathsf{parse}(\mathsf{crs}_{\mathsf{shuffle}})$
$(M, A, T, U, \pi_{\mathsf{gprod}}, \pi_{\mathsf{sameexp}}, \pi_{\mathsf{multiexp}}) \leftarrow \mathsf{parse}(\pi_{\mathsf{shuffle}})$
$(a_1, \ldots, a_\ell) \leftarrow \mathsf{Hash}(T_1, \ldots, T_\ell, U_1, \ldots, U_\ell, M)$

Step 2:
$\alpha \leftarrow \mathsf{Hash}(A, M)$

Step 3:
$\mathsf{gprod} \leftarrow (a_1 + 1 \cdot \alpha + \beta)(a_2 + 2 \cdot \alpha + \beta) \cdots (a_\ell + \ell \cdot \alpha + \beta)$
$A_1 \leftarrow AM^{\alpha}(g_1 \ldots g_n)^{\beta}$
$b_1 \leftarrow \mathsf{Verify}(\mathsf{crs}_{\mathsf{gprod}}, (A_1, \mathsf{gprod}), \pi_{\mathsf{gprod}})$

Step 4:
$\gamma_1, \delta_1, \ldots, \gamma_4, \delta_4 \leftarrow \mathsf{Hash}(A)$
$R \leftarrow R_1^{a_1} \cdots R_\ell^{a_\ell}$
$S \leftarrow S_1^{a_1} \cdots S_\ell^{a_\ell}$
$b_2 \leftarrow \mathsf{Verify}(R_{\mathsf{sameexp}}, (R, S, T, U), \pi_{\mathsf{sameexp}})$

Step 5:
$b_3 \leftarrow \mathsf{Verify}(\mathsf{crs}_{\mathsf{multiexp}}, ((T_1, U_1, \ldots, T_\ell, U_\ell, g_t^{\gamma_1}, g_u^{\delta_1}, \ldots, , g_t^{\gamma_4}, g_u^{\delta_4}), A, (T, U)), \pi_{\mathsf{multiexp}})$
return 1 if $(b_1, b_2, b_3) = (1, 1, 1)$
else return 0

Figure 2: Verify algorithm to check that $(T_1, U_1), \ldots, (T_\ell, U_\ell) = (R^r_{\sigma(1)}, S^r_{\sigma(1)}), \ldots, (R^r_{\sigma(\ell)}, S^r_{\sigma(\ell)})$ for some unknown field element r and unknown permutation σ.

5.2 Grand Product Construction Overview

Step 1:
Prover: The prover takes as input $A_1 \in \mathbb{G}$, $\mathsf{gprod} \in \mathbb{F}$, $(a_1, \ldots, a_{\ell+4}) \in \mathbb{F}^{\ell+4}$ such that $A_1 = \prod_{i=1}^{\ell+4} g_i^{a_i}$ and $\mathsf{gprod} = \prod_{i=1}^{\ell} a_i$. They set

$$(b_1, \ldots, b_\ell) = (1, a_2, a_2a_3, a_2a_3a_4, \ldots, a_2 \cdots a_\ell)$$

and select 4 random values $b_{\ell+1}, \ldots, b_{\ell+4} \xleftarrow{\$} \mathbb{F}$ from the field. They compute

$$B = \prod_{i=\ell+1}^{\ell+4} g_i^{b_i} \ \wedge \mathsf{bl} = a_{\ell+1}b_{\ell+1} + a_{\ell+2}b_{\ell+2} + a_{\ell+3}b_{\ell+3} + a_{\ell+4}b_{\ell+4}$$

and hash $B, \mathsf{bl} \in \mathbb{G} \times \mathbb{F}$ to obtain the field element x.

Verifier: The verifier takes as input the instance $A_1 \in \mathbb{G}$ and $\mathsf{gprod} \in \mathbb{F}$ and a proof π_{DLInner}. They hash $B, \mathsf{bl} \in \mathbb{G} \times \mathbb{F}$ to obtain the randomness $x \in \mathbb{F}$.

Step 2:
Prover: The prover computes the commitment

$$C = A_1(g_1 \cdots g_\ell)^{-x^{-1}}.$$

They then compute the generators

$$(h_1, \ldots, h_n) = (g_2^{x^{-1}}, g_3^{x^{-2}}, \ldots, g_\ell^{x^{\ell-1}}, g_1^{x^\ell}, g_{\ell+1}^{x^{-\ell-1}}, \ldots g_n^{x^{-\ell-1}}).$$

and the corresponding opening to C as

$$(c_1, \ldots, c_n) = (a_2x - 1, a_3x^2 - x, \ldots, a_\ell x^{\ell-1} - x^{\ell-2}, a_1x^\ell - x^{\ell-1}, a_{\ell+1}x^{\ell+1}, \ldots, a_nx^{\ell+1}).$$

Verifier: The verifier computes the commitment

$$C = A_1(g_1 \cdots g_\ell)^{-x^{-1}}.$$

They then compute the generators

$$(h_1, \ldots, h_n) = (g_2^{x^{-1}}, g_3^{x^{-2}}, \ldots, g_\ell^{x^{\ell-1}}, g_1^{x^\ell}, g_{\ell+1}^{x^{-\ell-1}}, \ldots g_n^{x^{-\ell-1}}).$$

Remark: Observe that

$$\begin{aligned} < (b_1, \ldots, b_n), (c_1, \ldots, c_n) >= \ & (b_1a_2x - b_1) + (b_2a_3x^2 - b_2x) + \ldots + (b_{\ell-1}a_\ell x^{\ell-1} - b_{\ell-1}x^{\ell-2}) \\ & + (b_\ell a_1 x^\ell - b_\ell x^{\ell-1}) + (b_{\ell+1}a_{\ell+1} + \ldots + b_n a_n)x^{\ell+1}. \end{aligned}$$

If this value is equal to $\mathsf{gprod}x^\ell + \mathsf{bl}x^{\ell+1} - 1$ at a randomly sampled point x then we have that

$$\begin{aligned}
-b_1 &= -1 && \Rightarrow b_1 = 1 \\
b_1 a_2 - b_2 &= 0 && \Rightarrow a_2 = b_2 \\
b_2 a_3 - b_3 &= 0 && \Rightarrow b_3 = a_2 a_3 \\
&\vdots \\
b_{\ell-1} a_\ell - b_\ell &= 0 && \Rightarrow b_\ell = a_2 \cdots a_\ell \\
b_\ell a_1 &= \mathsf{gprod} && \Rightarrow \mathsf{gprod} = a_1 \cdots a_\ell \\
b_{\ell+1} a_{\ell+1} + \ldots + b_n a_n &= \mathsf{bl} && \Rightarrow \text{N/A}
\end{aligned}$$

Step 3:
Prover: The prover computes a prove π_{DLInner} that they know $(b_1, \ldots, b_{\ell+4})$ and $(c_1, \ldots, c_{\ell+4})$ such that $B = \prod_{i=1}^{\ell+4} g_i^{b_i}$ and $C = \prod_{i=1}^{\ell+4} h_i^{c_i}$ and $< (b_1, \ldots, b_{\ell+4}), ((c_1, \ldots, c_{\ell+4})) >= \mathsf{gprod}x^\ell + \mathsf{bl}x^{\ell+1} - 1$. In other words they run

$$\pi_{\mathsf{gpInner}} = \mathsf{Prove}(\mathsf{crs}_{\mathsf{DL}}, (B, C, \mathsf{gprod}x^\ell + \mathsf{bl}x^{\ell+1} - 1), ((b_1, \ldots, b_{\ell+2}), (c_1, \ldots, c_{\ell+2}))).$$

Crucially this proof is sound even if an adversary knows a discrete logarithm relation between e.g. g_1 and h_1.

Verifier: The verifier checks that π_{gpInner} verifies with respect to B, C, and $\mathsf{gprod}x^\ell + \mathsf{bl}x^{\ell+1} - 1$. In other words they run

$$b = \mathsf{Verify}(\mathsf{crs}_{\mathsf{DL}}, (B, C, \mathsf{gprod}x^\ell + \mathsf{bl}x^{\ell+1} - 1), \pi_{\mathsf{gpInner}}).$$

Outcome:
The prover returns the proof $\pi_{\mathsf{gprod}} = (B, \mathsf{bl}, \pi_{\mathsf{gpInner}})$.

The verifier returns 1 if $b = 1$ and otherwise returns 0.

6 Discrete Logarithm Inner Product

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

$$R_{\mathsf{DLInner}} = \left\{ (B, C, z), ((b_1, \ldots, b_{\ell+2}), (c_1, \ldots, c_{\ell+2})) \;\middle|\; \begin{array}{l} B = g_1^{b_1} \cdots g_{\ell+4}^{b_{\ell+4}} \wedge C = g_1^{c_1} \cdots g_{\ell+4}^{c_{\ell+4}} \\ \wedge\ z = b_1 c_1 + \ldots + b_{\ell+4} c_{\ell+4} \end{array} \right\}$$

that was written by Bootle et al. [BCC+16]. We make minor adjustments in order to achieve zero-knowledge. We do not use 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. An overview of our argument is given in Section 6.1. We prove our construction sound and zero-knowledge in Theorems ???.

$\underline{\mathsf{GProdProve}(\mathsf{crs}_{\mathsf{gprod}}, A_1, \mathsf{gprod}, a_1, \ldots, a_n)}$

Step 1:

$(R_{\mathsf{gprod}}, (\mathsf{crs}_g, u)) \leftarrow \mathsf{parse}(\mathsf{crs}_{\mathsf{shuffle}})$

$b_1, b_2, b_3, b_4 \xleftarrow{\$} \mathbb{F}$

$B \leftarrow g_{\ell+1}^{b_1} \cdots g_{\ell+4}^{b_4}$

$\mathsf{bl} \leftarrow a_{\ell+1}b_1 + a_{\ell+2}b_2 + a_{\ell+3}b_3 + a_{\ell+4}b_4$

$x \leftarrow \mathsf{Hash}(B, \mathsf{bl})$

Step 2:

$C \leftarrow A_1(g_1 \cdots g_\ell)^{-x^{-1}}$

$(c_1, \ldots, c_n) \leftarrow (a_2 x - 1, a_3 x^2 - x, \ldots, a_\ell x^{\ell-1} - x^{\ell-2}, a_1 x^\ell - x^{\ell-1}, a_{\ell+1} x^{\ell+1}, \ldots, a_{\ell+4} x^{\ell+1})$

$\mathsf{crs}_h \leftarrow (g_2^{x^{-1}}, g_3^{x^{-2}}, \ldots, g_\ell^{x^{\ell-1}}, g_1^{x^\ell}, g_{\ell+1}^{x^{-\ell-1}}, \ldots, g_{\ell+4}^{x^{-\ell-1}})$

Step 3:

$\pi_{\mathsf{DLInner}} \leftarrow \mathsf{Prove}\big(\ (R_{\mathsf{DLInner}}, \mathsf{crs}_g, \mathsf{crs}_h, u), (B, C, \mathsf{gprod} x^\ell + \mathsf{bl} x^{\ell+1} - 1), (b_1, \ldots, b_n), (c_1, \ldots, c_n) \ \big)$

return $(B, \mathsf{bl}, \pi_{\mathsf{DLInner}})$

Figure 3: Proving algorithm to demonstrate that $(A_1, \mathsf{gprod}) = (g_1^{a_1} \cdots g_n^{a_n}, a_1 \cdots a_\ell)$ for some field elements $(a_1, \ldots, a_n)$.

$\underline{\mathsf{GProdVerify}(\mathsf{crs}_{\mathsf{gprod}}, A_1, \mathsf{gprod}, \pi_{\mathsf{gprod}})}$

Step 1:

$(R_{\mathsf{gprod}}, (\mathsf{crs}_g, u)) \leftarrow \mathsf{parse}(\mathsf{crs}_{\mathsf{shuffle}})$

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

$x \leftarrow \mathsf{Hash}(B, \mathsf{bl})$

Step 2:

$C \leftarrow A_1(g_1 \cdots g_\ell)^{-x^{-1}}$

$\mathsf{crs}_h \leftarrow (g_2^{x^{-1}}, g_3^{x^{-2}}, \ldots, g_\ell^{x^{\ell-1}}, g_1^{x^\ell}, g_{\ell+1}^{x^{-\ell-1}}, \ldots, g_{\ell+4}^{x^{-\ell-1}})$

Step 3:

$b \leftarrow \mathsf{Verify}((R_{\mathsf{DLInner}}, \mathsf{crs}_g, \mathsf{crs}_h, u), (B, C, \mathsf{gprod} x^\ell + \mathsf{bl} x^{\ell+1} - 1), \pi_{\mathsf{DLInner}})$

return b

Figure 4: Verify algorithm to check that $(A_1, \mathsf{gprod}) = (g_1^{a_1} \cdots g_n^{a_n}, a_1 \cdots a_\ell)$ for some unknown field elements $(a_1, \ldots, a_n)$.

6.1 Overview

Step 1

Prover: The prover first blinds the argument by sampling $r_1, \ldots, r_n \xleftarrow{\$} \mathbb{F}$ randomly from $\mathbb{F}$ as masking values. They compute $R \in \mathbb{G}$ as

$$R = h_1^{r_1} \cdots h_n^{r_n}$$

and set

$$\mathsf{bl} = b_1 r_1 + \ldots + b_n r_n$$

They hash the shuffled ciphertexts R, bl obtain the field elements x. The prover resets the inner product to equal

$$z = z + \mathsf{bl} x$$

and the inputs

$$(c_1, \ldots, c_n) = (c_1 + x r_1, \ldots, c_n + x r_n)$$

Verifier: The verifier receives as input (R, bl) The verifier hashes $R, \mathsf{bl} \in \mathbb{G} \times \mathbb{F}$ to obtain the field element $x \in \mathbb{F}$. The verifier resets the inner product to equal

$$z = z + \mathsf{bl} x$$

Step 2

Prover: They now insert the claimed inner product into the exponent of B. They compute $x = \mathsf{Hash}(z)$ and reset

$$u = u^x$$

and reset

$$B = B u^{xz}$$

Verifier: They now insert the claimed inner product into the exponent of B. They compute $x = \mathsf{Hash}(z)$ and reset

$$u = u^x$$

and reset

$$B = B u^{xz}$$

Step 3

Prover: The prover now runs a recursive argument.

Outcome

The prover returns the proof $\pi_{\mathsf{shuffle}} = (M, A, T, U, \pi_{\mathsf{gprod}}, \pi_{\mathsf{sameexp}}, \pi_{\mathsf{multiexp}})$.

The verifier returns 1 if $(b_1, b_2, b_3) = (1, 1, 1)$ and otherwise returns 0. s

7 Security Proofs

7.1 Zero-Knowledge

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.

Theorem 7.1 (Shuffle Argument is zero-knowledge)**.** *If the grand-product argument, the same-exponentiation argument, and the multiexponentiation argument are zero-knowledge, then the shuffle argument described in Figures 1 and 2 is zero-knowledge.*

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

$$(R_1, S_1), \ldots, (R_\ell, S_\ell)), (T_1, U_1), \ldots, (T_\ell, U_\ell)$$

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.

The simulator $\mathsf{Simulate}$ proceeds as follows

1. In the first step they sample $M \xleftarrow{\$} \mathbb{G}$. They compute $(a_1, \ldots, a_\ell) = \mathsf{Hash}(T_1, \ldots, T_\ell, U_1, \ldots, U_\ell, M)$.
2. In the second step they sample $A \xleftarrow{\$} \mathbb{G}$ and compute $(\alpha, \beta) = \mathsf{Hash}(A)$
3. In the third step they compute $\mathsf{gprod} = a_1 \cdots a_\ell$ and set $A_1 = AM^\alpha (g_1 \cdots g_{\ell+4})^\beta$. They compute
$$\pi_{\mathsf{gprod}} = \mathsf{Simulate}(\mathsf{crs}_{\mathsf{gprod}}, (A_1, \mathsf{gprod})).$$
4. In the fourth step they compute $R = \prod_{i=1}^{\ell} R_i^{a_i}$, $S = \prod_{i=1}^{\ell}$, and $\gamma_1, \delta_1, \ldots, \gamma_4, \delta_4 = \mathsf{Hash}(A)$. They sample $T, U \xleftarrow{\$} \mathbb{G}$ and compute
$$\pi_{\mathsf{sameexp}} = \mathsf{Simulate}(R_{\mathsf{sameexp}}, R, S, T, U)$$
5. In the fifth step they simulate
$$\pi_{\mathsf{multiexp}} = \mathsf{Simulate}(\mathsf{crs}_{\mathsf{multiexp}}, (T_1, U_1, \ldots, T_\ell, U_\ell, g_t^{\gamma_1}, g_u^{\delta_1}, \ldots, g_t^{\gamma_4}, g_u^{\delta_4}), A, (T, U))$$
They return $(M, A, T, U, \pi_{\mathsf{gprod}}, \pi_{\mathsf{sameexp}}, \pi_{\mathsf{multiexp}})$.

Now we must argue that the simulated proof is indistinguishable from the real proof. To do this first note that $\pi_{\mathsf{gprod}}, \pi_{\mathsf{sameexp}}, \pi_{\mathsf{multiexp}}$ are zero-knowledge proofs and thus the simulated proofs are indistinguishable from the real proofs. The provers value M is randomised by $s_1, \ldots, s_4$ and thus is indistinguishable from the simulators random M. The provers values A, T, U are randomised by $a_{\ell+1}, a_{\ell+2}, a_{\ell+3}, a_{\ell+4}$ and thus are indistinguishable from the simulators random A, T, U. This completes the proof. □

7.2 Soundness

We prove the knowledge-soundness of our arguments i.e. for any prover that convinces an honest verifier, there exists an extractor that outputs a valid witness.

Theorem 7.2 (Shuffle Argument is sound)**.** *If the grand-product argument, the same-exponentiation argument, and the multiexponentiation argument are knowledge-sound, and the discrete logarithm assumption holds, then the shuffle argument described in Figures 1 and 2 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 $g_1, \ldots, g_{\ell+4}$ or a permutation σ and field element r such that

$$((((R_1, S_1), \ldots, (R_\ell, S_\ell)), ((T_1, U_1), \ldots, (T_\ell, U_\ell))), (\sigma, r)) \in R_{\mathsf{shuffle}}.$$

By the knowledge-soundness of the grand product argument, the same exponentiation argument, and the multiexponentiation argument there exists extractors $\mathcal{X}_{\mathsf{gprod}}, \mathcal{X}_{\mathsf{sameexp}}, \mathcal{X}_{\mathsf{multiexp}}$ such that if $\mathcal{A}$ returns verifying $(\pi_{\mathsf{gprod}}, \pi_{\mathsf{sameexp}}, \pi_{\mathsf{multiexp}})$ then they return valid witnesses for their respective languages with overwhelming probability.

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

1. Run $\mathcal{A}$ using random coins r_{coin}. In Step 2 obtain $A \in \mathbb{G}$. Compute $\alpha_1, \beta_1 = \mathsf{Hash}(A)$.

2. Using the grand-product extractor $\mathcal{X}_{\mathsf{gprod}}$, extract $d_1, \ldots, d_{\ell+4}$ such that $AM^{\alpha_1}(g_1 \ldots g_{\ell+4})^{\beta_1} = g_1^{d_1} \cdots g_{\ell+4}^{d_{\ell+4}}$. Sample $\alpha_2, \beta_2 \xleftarrow{\$} \mathbb{F}$ and program $\mathsf{Hash}(A) = (\alpha_2, \beta_2)$. Run $\mathcal{A}$ again on the same random coins r_{coin}. Using $\mathcal{X}_{\mathsf{gprod}}$, extract $d'_1, \ldots, d'_{\ell+2}$ such that $AM^{\alpha_1}(g_1 \ldots g_{\ell+4})^{\beta_1} = g_1^{d'_1} \cdots g_{\ell+4}^{d'_{\ell+4}}$. Abort if $\mathcal{X}_{\mathsf{gprod}}$ fails.

3. Set $m_1, \ldots, m_{\ell+4} = \frac{(d_1-\beta_1)-(d'_1-\beta_2)}{\alpha_1-\alpha_2}, \ldots, \frac{(d_{\ell+4}-\beta_1)-(d'_{\ell+4}-\beta_2)}{\alpha_1-\alpha_2}$ and $c_1, \ldots, c_{\ell+4} = (d_1 - m_1\alpha_1, \ldots, d_{\ell+4} - m_{\ell+4}\alpha_{\ell+4})$. Abort if $A \neq \prod_{i=1}^{\ell+4} g_i^{c_i}$ or $M \neq \prod_{i=1}^{\ell+4} g_i^{m_i}$.

4. Sample $\alpha_3, \beta_3 \xleftarrow{\$} \mathbb{F}$ and program $\mathsf{Hash}(A) = (\alpha_3, \beta_3)$. Run $\mathcal{A}$ again on the same random coins r_{coin}.

5. Using the grand-product extractor $\mathcal{X}_{\mathsf{gprod}}$, extract $d_1, \ldots, d_{\ell+4}$ such that $AM^{\alpha_3}(g_1 \ldots g_{\ell+4})^{\beta_3} = g_1^{d_1} \cdots g_{\ell+4}^{d_{\ell+4}}$ and $\prod_{i=1}^{\ell} d_i = \prod_{i=1}^{\ell}(a_i + i \cdot \alpha + \beta)$. Abort if $\mathcal{X}_{\mathsf{gprod}}$ fails. If $(d_1 + \alpha m_1 + \beta, \ldots, d_{\ell+4} + \alpha m_{\ell+4} + \alpha) \neq (c_1, \ldots, c_{\ell+4})$ return the discrete log relation $(d_1 + \alpha m_1 + \beta, \ldots, d_{\ell+2} + \alpha m_{\ell+2} + \alpha), (c_1, \ldots, c_{\ell+2})$.

6. Set $\sigma = (m_1, \ldots, m_\ell)$. Abort if σ is not a permutation of $(1, \ldots, \ell)$. Abort if $(c_1, \ldots, c_\ell) \neq (a_{\sigma(1)}, \ldots, a_{\sigma(\ell)})$.

7. Using the same exponent extractor $\mathcal{X}_{\mathsf{sameexp}}$, extract r, r_t, r_u such that $T = (\prod_{i=1}^{\ell} R_i^{a_i})^r g_t^{r_t}$ and $U = (\prod_{i=1}^{\ell} S_i^{a_i})^r g_u^{r_u}$. Abort if $\mathcal{X}_{\mathsf{sameexp}}$ fails.

8. Using the multi exponentiation extractor $\mathcal{X}_{\mathsf{multiexp}}$, extract $(d_1, \ldots, d_{\ell+4})$ such that $A = g_1^{d_1} \cdots g_{\ell+4}^{d_{\ell+4}}$, $T = \prod_{i=1}^{\ell} T_i^{d_i} g_t^{\sum_{i=1}^{4} \gamma_i d_{\ell+i}}$ and $U = \prod_{i=1}^{\ell} U_i^{d_i} g_u^{\sum_{i=1}^{4} \delta_i d_{\ell+i}}$. Abort if $\mathcal{X}_{\mathsf{multiexp}}$ fails. If $(d_1, \ldots, d_{\ell+4}) \neq (c_1, \ldots, c_{\ell+4})$ return the discrete log relation $(d_1 + \alpha, \ldots, d_{\ell+4} + \alpha), (c_1, \ldots, c_{\ell+4})$.

9. If (σ, r) is such that $(T_i, U_i) = (R^r_{\sigma(i)}, S^r_{\sigma(i)})$ for all $1 \leq i \leq \ell$ then return (σ, r). Else abort.

Extractor runs in polynomial time: This follows directly from the fact that $\mathcal{X}_{\mathsf{gprod}}, \mathcal{X}_{\mathsf{sameexp}}, \mathcal{X}_{\mathsf{multiexp}}$ run in polynomial time.

Extractor does not abort We now must argue that $\mathcal{X}_{\mathsf{shuffle}}$ does not abort. In (2) the extractor aborts only if $\mathcal{X}_{\mathsf{gprod}}$ fails. By the knowledge-soundness of the grand-product argument this happens only with negligible probability.

In (3) the extractor aborts if $A \neq \prod_{i=1}^{\ell+2} h_i^{c_i}$ or $M \neq \prod_{i=1}^{\ell+2} h_i^{m_i}$. However from the success of the extractor we have that

$$AM^{\alpha_1} = h_1^{d_1-\beta_1} \cdots h_{\ell+2}^{d_{\ell+2}-\beta_1} \ \wedge \ AM^{\alpha_2} = h_1^{d'_1-\beta_2} \cdots h_{\ell+2}^{d'_{\ell+2}-\beta_2}.$$

Thus

$$M^{\alpha_1-\alpha_2} = h_1^{(d_1-\beta_1)-(d'_1-\beta_2)} \cdots h_{\ell+2}^{(d_{\ell+2}-\beta_1)-(d'_\ell-\beta_2)}$$

and $(m_1, \ldots, m_{\ell+2})$ is correct unless $\alpha_1 = \alpha_2$ (which happens only with negligible probability). This implies that A also has the correct discrete logarithm and the extractor does not abort. Similarly in (5) the extractor aborts only if $\mathcal{X}_{\mathsf{gprod}}$ fails. By the knowledge-soundness of the grand-product argument this happens only with negligible probability.

In (5) the extractor aborts only if $\mathcal{X}_{\mathsf{gprod}}$ fails. By the knowledge-soundness of the grand-product argument this happens only with negligible probability.

In (6) the extractor aborts if $(m_1, \ldots, m_\ell)$ is not a permutation of $(1, \ldots, \ell)$ or if $(c_1, \ldots, c_\ell) \neq (a_{m_1}, \ldots, a_{m_\ell})$. Given that (5) did not abort, we have that at a randomly sampled α,

$$(c_1 + m_1\alpha + \beta) \cdots (c_\ell + m_\ell\alpha + \beta) = (a_1 + 1 \cdot \alpha + \beta) \cdots (a_\ell + \ell\alpha + \beta).$$

Using that M is fixed in Step 1 before $a_1, \ldots, a_\ell$ are sampled, by the Schwartz-Zippel Lemma, this happens with negligible probability unless

$$(c_1 + m_1 X + Y) \cdots (c_\ell + m_\ell X + Y) = (a_1 + 1 \cdot X + Y) \cdots (a_\ell + \ell \cdot X + Y).$$

Thus $(c_1, m_1), \ldots, (c_\ell, m_\ell)$ is a permutation of $(a_1, 1), \ldots, (a_\ell, \ell)$ with overwhelming probability. and $\mathcal{X}_{\mathsf{shuffle}}$ does not abort.

In (7) the extractor aborts if $\mathcal{X}_{\mathsf{sameexp}}$ fails. By the knowledge-soundness of the same exponent argument this happens only with negligible probability.

In (8) the extractor aborts only if $\mathcal{X}_{\mathsf{multiexp}}$ fails. By the knowledge-soundness of the multiexponentiation argument this happens only with negligible probability.

In (9) the extractor aborts only if $(T_i, U_i) \neq (R_i^r, S_i^r)$ for some i. Since the same exponent and the multiexponentiation arguments to not abort and the values extracted from the multiexponentiation argument equal $(c_1, \ldots, c_{\ell+4}) = (a_{\sigma(1)}, \ldots, a_{\sigma(\ell)}, a_{\ell+1}, \ldots, a_{\ell+4})$ we have that

$$\prod_{i=1}^{\ell} T_i^{a_{\sigma(i)}} g_t^{\sum_{i=1}^4 \gamma_i a_{\ell+i}} = \prod_{i=1}^{\ell} R_i^{r a_i} g_t^{r_t} \ \wedge \ \prod_{i=1}^{\ell} U_i^{a_{\sigma(i)}} g_u^{\sum_{i=1}^4 \delta_i a_{\ell+i}} = \prod_{i=1}^{\ell} S_i^{r a_i} g_u^{r_u}.$$

Where $a_1, \ldots, a_\ell$ are random elements chosen only after $(\sigma, T_1, U_1, \ldots, T_\ell, U_\ell)$ have been determined in Step 1, this happens with negligible probability. □

References

[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.

[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.