# A Shuffle Argument Protocol Specification
# Work in Progress

The Ethereum Foundation Research Team

September 28, 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 $R_{\mathsf{rel}}$ where a public instance $\phi$ and a private witness $w$ is in $R_{\mathsf{rel}}$ if and only if certain properties hold, we write

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

Proving algorithms $\mathsf{Prove}$ 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 $R_{\mathsf{rel}}$ and where $(\phi, w) \in R_{\mathsf{rel}}$. They return a proof $\pi_{\mathsf{rel}}$.

Verification algorithms $\mathsf{Verify}$ take as input $(\mathsf{crs}_{\mathsf{rel}}, \phi, \pi_{\mathsf{rel}})$ where $\mathsf{crs}_{\mathsf{rel}}$ is a common reference string, $\phi$ is an instance the prover is claiming to be in the language, and $\pi_{\mathsf{rel}}$ 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

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

a shuffler computes a second set of El-Gamal ciphertexts

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

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

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

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 9.1 and 9.6. 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

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

## 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

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

## 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 9, Theorems 9.6 and 9.1.

### 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 $\pi_{\mathsf{gprod}}$ 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 $\pi_{\mathsf{gprod}}$ for the instance $(A_1, \mathsf{gprod})$. 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 $1 \leq i \leq \ell$. They compute a same-exponentiation argument $\pi_{\mathsf{sameexp}}$ 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 $\pi_{\mathsf{sameexp}}$ 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 $(c_1, \dots, c_n)$ such that $A, T, U \in \mathbb{G}^3$ are equal to

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

In other words they run

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

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

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

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

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

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 9.2 and 9.7. As part of our construction we require zero-knowledge algorithms for proving and verifying an additional relation $R_{\mathsf{DLInner}}$. 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 $(b_1, \dots, b_{\ell+4})$ and $(c_1, \dots, c_{\ell+4})$ 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

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

```
$\underline{\mathsf{ShuffleProve}(\mathsf{crs}_{\mathsf{shuffle}}, R_1, S_1, T_1, U_1, \ldots, R_\ell, S_\ell, T_\ell, U_\ell, \sigma, r)}$
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:
$\mathsf{gprod} \leftarrow (a_1 + 1 \cdot \alpha + \beta)(a_2 + 2 \cdot \alpha + \beta) \cdots (a_\ell + \ell \cdot \alpha + \beta)$
$A_1 \leftarrow AM^\alpha (g_1 \ldots g_{\ell+4})^\beta$
$\pi_{\mathsf{gprod}} \leftarrow \mathsf{Prove}\begin{pmatrix} \mathsf{crs}_{\mathsf{gprod}}, (A_1, \mathsf{gprod}), \\ (a_{\sigma(1)} + \sigma(1)\alpha + \beta, \ldots, a_{\sigma(\ell)} + \sigma(\ell)\alpha + \beta, a_{\ell+1} + s_1\alpha + \beta, \ldots, a_{\ell+4} + s_4\alpha + \beta) \end{pmatrix}$

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:
$\pi_{\mathsf{multiexp}} \leftarrow \mathsf{Prove}\begin{pmatrix} \mathsf{crs}_{\mathsf{multiexp}}, ((T_1, U_1, \ldots, T_\ell, U_\ell, g_t^{\gamma_1}, g_u^{\delta_1}, \ldots, g_t^{\gamma_4}, g_u^{\delta_4}), A, (T, U)), \\ (a_{\sigma(1)}, \ldots, a_{\sigma(\ell)}, a_{\ell+1}, a_{\ell+2}) \end{pmatrix}$
return $(M, A, T, U, \pi_{\mathsf{gprod}}, \pi_{\mathsf{sameexp}}, \pi_{\mathsf{multiexp}})$
```

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

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

## 5.2 Grand Product Construction Overview

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

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

and select 4 random values $b_{\ell+1}, \dots, b_{\ell+4} \xleftarrow{\$} \mathbb{F}$ 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 $(B, \mathsf{bl}, \pi_{\mathsf{DLInner}}) = \pi_{\mathsf{gprod}}$. 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

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

and the corresponding opening to $C$ as

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

**Verifier:** The verifier computes the commitment

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

They then compute the generators

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

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

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

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 $\pi_{\mathsf{gpInner}}$ 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.

```
$\underline{\mathsf{GProdProve}(\mathsf{crs}_{\mathsf{gprod}}, A_1, \mathsf{gprod}, a_1, \ldots, a_n)}$
**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**:
$C \leftarrow A_1(g_1 \cdots g_\ell)^{-x^{-1}}$
$(c_1, \ldots, c_n) \leftarrow (a_2x - 1, a_3x^2 - x, \ldots, a_\ell x^{\ell-1} - x^{\ell-2}, a_1x^\ell - x^{\ell-1}, a_{\ell+1}x^{\ell+1}, \ldots, a_{\ell+4}x^{\ell+1})$
$\mathsf{crs}_h \leftarrow (g_2^{x^{-1}}, g_3^{x^{-2}}, \ldots, g_\ell^{x^{-(\ell-1)}}, g_1^{x^{-\ell}}, g_{\ell+1}^{x^{-(\ell+1)}}, \ldots, g_{\ell+4}^{x^{-(\ell+1)}})$

**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 $(a_1, \ldots, a_n)$.

## 6 Discrete Logarithm Inner Product Argument

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

$$R_{\mathsf{DLInner}} = \left\{ (B, C, z), ((b_1, \ldots, b_n), (c_1, \ldots, c_n)) \;\middle|\; \begin{array}{l} B = g_1^{b_1} \cdots g_n^{b_n} \wedge C = g_1^{c_1} \cdots g_n^{c_n} \\ \wedge\ z = b_1c_1 + \ldots + b_{\ell+4}c_n \end{array} \right\}$$

which is given in Figures 5 and 6. This protocol was originally written 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

```
$\underline{\mathsf{GProdVerify}(\mathsf{crs}_{\mathsf{gprod}}, A_1, \mathsf{gprod}, \pi_{\mathsf{gprod}})}$
**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 $(a_1, \ldots, a_n)$.

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 exponent. We prove our construction sound and zero-knowledge in Theorems 9.3 and 9.8.

We now give an overview of our zero-knowledge argument for $R_{\mathsf{DLInner}}$.

## Step 1

**Prover:** The prover first blinds the argument by sampling $r_1, s_1 \ldots, r_n, s_n \overset{\$}{\leftarrow} \mathbb{F}$ randomly from $\mathbb{F}$ as masking values. They compute $R, S \in \mathbb{G}$ and $\mathsf{bl}_1, \mathsf{bl}_2 \in \mathbb{F}$ as

$$\begin{array}{ll} R = g_1^{r_1} \cdots g_n^{r_n} & \mathsf{bl}_1 = (b_1 s_1 + c_1 r_1) + \ldots + (b_n s_n + c_n r_n) \\ S = h_1^{s_1} \cdots h_n^{s_n} & \mathsf{bl}_2 = r_1 s_1 + \ldots + r_n s_n \end{array}$$

They hash $R, S, \mathsf{bl}_1, \mathsf{bl}_2$ obtain the field element $x$. The prover resets the inputs to equal

$$\begin{array}{c} (b_1, \ldots, b_n) = (b_1 + x r_1, \ldots, b_n + x r_n) \\ (c_1, \ldots, c_n) = (c_1 + x s_1, \ldots, c_n + x s_n) \end{array}$$

**Verifier:** The verifier receives as input $(R, S, \mathsf{bl}_1, \mathsf{bl}_2, \pi_{\mathsf{recurse},1}, \ldots, \pi_{\mathsf{recurse},\log_n}, b_f, c_f)$ They hash $R, \mathsf{bl} \in \mathbb{G} \times \mathbb{F}$ to obtain the field element $x \in \mathbb{F}$. The verifier resets the inputs to equal

$$B = BR^x \text{ and } C = CS^x \text{ and } z = z + x\mathsf{bl}_1 + x^2\mathsf{bl}_2$$

## Step 2

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

$$u = u^x$$

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

$$u = u^x \text{ and } B = Bu^{xz}$$

**Step 3**
**Prover:** The prover now runs a recursive argument. For $1 \leq j \leq \log_2(n)$ they do the following

1. Reset $n = \frac{n}{2}$
2. Set
$$\begin{array}{ll} c_{L,b} = \prod_{i=1}^{n} g_i^{b_{n+i}} u^{\sum_{i=1}^{n} b_{n+i}c_i} & c_{R,b} = \prod_{i=1}^{n} g_{n+i}^{b_i} u^{\sum_{i=1}^{n} b_i c_{n+i}} \\ c_{L,c} = \prod_{i=1}^{n} h_{n+i}^{c_i} & c_{R,c} = \prod_{i=1}^{n} h_i^{c_{n+i}} \end{array}$$
and
$$\pi_{\mathsf{recurse},j} = (c_{L,b}, c_{L,c}, c_{R,b}, c_{R,c}).$$
Compute $x = \mathsf{Hash}(c_{L,b}, c_{L,c}, c_{R,b}, c_{R,c})$.
3. Reset
$$\begin{array}{ll} \boldsymbol{b} = (b_1 + xb_{n+1}, \ldots, b_n + xb_{2n}) & \mathsf{crs}_g = (g_1 g_{n+1}^{x^{-1}}, \ldots, g_n g_{2n}^{x^{-1}}) \\ \boldsymbol{c} = (c_1 + x^{-1}c_{n+1}, \ldots, c_n + x^{-1}c_{2n}) & \mathsf{crs}_h = (h_1 h_{n+1}^{x}, \ldots, h_n h_{2n}^{x}) \end{array}$$

**Verifier:** The verifier now runs a recursive argument. For $1 \leq j \leq \log_2(n)$ they do the following

1. Reset $n = \frac{n}{2}$.
2. Parse $(c_{L,b}, c_{L,c}, c_{R,b}, c_{R,c}) = \pi_{\mathsf{recurse},j}$ and compute $x = \mathsf{Hash}(c_{L,b}, c_{L,c}, c_{R,b}, c_{R,c})$.
3. Reset
$$\begin{array}{ll} B = c_{L,b}^{x} \cdot B \cdot c_{R,b}^{x^{-1}} & \mathsf{crs}_g = (g_1 g_{n+1}^{x^{-1}}, \ldots, g_n g_{2n}^{x^{-1}}) \\ C = c_{L,c}^{x} \cdot C \cdot c_{R,c}^{x^{-1}} & \mathsf{crs}_h = (h_1 h_{n+1}^{x}, \ldots, h_n h_{2n}^{x}) \end{array}$$

**Step 4**
**Prover:** The prover sets

$$b_f = b_1 \text{ and } c_f = c_1$$

to be the final inputs from $\boldsymbol{b}$ and $\boldsymbol{c}$ in the recursion. Note in the final round these vectors will have length 1.

**Verifier:** The verifier lets $b$ equal 1 if

$$B = g_1^{b_f} u^{b_f c_f} \text{ and } C = h_1^{c_f}$$

and zero otherwise. Here $g_1$ and $h_1$ are the final values of $\mathsf{crs}_g$ and $\mathsf{crs}_h$ in the recursion which are vectors of length 1.

**Outcome**
The prover returns the proof $\pi_{\mathsf{DLInner}} = (R, \mathsf{bl}, \pi_{\mathsf{recurse},1}, \ldots, \pi_{\mathsf{recurse},\log_n}, b_f, c_f)$.

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

```
$\underline{\mathsf{DLInnerProve}(\mathsf{crs}_{\mathsf{DLInner}}, (B, C, z), ((b_1, \ldots, b_n), (c_1, \ldots, c_n)))}$
**Step 1**:
$(R_{\mathsf{DLInner}}, (\mathsf{crs}_g, \mathsf{crs}_h, u)) \leftarrow \mathsf{parse}(\mathsf{crs}_{\mathsf{DLInner}})$
$r_1, s_1, \ldots, r_n, s_n \xleftarrow{\$} \mathbb{F}$
$R \leftarrow g_1^{r_1} \cdots g_n^{r_n}$
$S \leftarrow h_1^{s_1} \cdots h_n^{s_n}$
$\mathsf{bl}_1 \leftarrow (b_1 s_1 + c_1 r_1) + \ldots + (b_n s_n + c_n r_n)$
$\mathsf{bl}_2 \leftarrow r_1 s_1 + \ldots + s_n r_n)$
$x \leftarrow \mathsf{Hash}(R, S, \mathsf{bl}_1, \mathsf{bl}_2)$
$(b_1, \ldots, b_n) \leftarrow (b_1 + x r_1, \ldots, b_n + x r_n)$
$(c_1, \ldots, c_n) \leftarrow (c_1 + x s_1, \ldots, c_n + x s_n)$

**Step 2**:
$x \leftarrow \mathsf{Hash}(x)$ $u \leftarrow u^x$

**Step 3**:
for $1 \leq j \leq \log(n)$ :
    $n \leftarrow \frac{n}{2}$
    $c_{L,b} \leftarrow \prod_{i=1}^{n} g_i^{b_{n+i}} u^{\sum_{i=1}^{n} b_{n+i} c_i}$
    $c_{L,c} \leftarrow \prod_{i=1}^{n} h_{n+i}^{c_i}$
    $c_{R,b} \leftarrow \prod_{i=1}^{n} g_{n+i}^{b_i} u^{\sum_{i=1}^{n} b_i c_{n+i}}$
    $c_{R,c} \leftarrow \prod_{i=1}^{n} h_i^{c_{n+i}}$
    $\pi_{\mathsf{recurse},j} \leftarrow (c_{L,b}, c_{L,c}, c_{R,b}, c_{R,c})$
    $x \leftarrow \mathsf{Hash}(c_{L,b}, c_{L,c}, c_{R,b}, c_{R,c})$
    $(b_1, \ldots, b_n) \leftarrow (b_1 + x b_{n+1}, \ldots, b_n + x b_{2n})$
    $(c_1, \ldots, c_n) \leftarrow (c_1 + x^{-1} c_{n+1}, \ldots, c_n + x^{-1} c_{2n})$
    $(g_1, \ldots, g_n) \leftarrow (g_1 g_{n+1}^{x^{-1}}, \ldots, g_n g_{2n}^{x^{-1}})$
    $(h_1, \ldots, h_n) \leftarrow (h_1 h_{n+1}^{x}, \ldots, h_n h_{2n}^{x})$

**Step 4**:
$b_f \leftarrow b_1$
$c_f \leftarrow c_1$
return $(R, S, \mathsf{bl}_1, \mathsf{bl}_2, \pi_{\mathsf{recurse},1}, \ldots, \pi_{\mathsf{recurse},\log(n)}, b_f, c_f)$
```

Figure 5: Proving algorithm to demonstrate that $(B, C, z) = (g_1^{b_1} \cdots g_n^{b_n}), (h_1^{c_1} \cdots h_n^{c_n}), b_1 c_1 + \cdots + b_n c_n$ for some field elements $(b_1, \ldots, b_n)$ and $(c_1, \ldots, c_n)$.

```
$\underline{\mathsf{DLInnerVerify}(\mathsf{crs}_{\mathsf{DLInner}}, (B, C, z), \pi_{\mathsf{DLInner}})}$
**Step 1**:
$(R_{\mathsf{DLInner}}, (\mathsf{crs}_g, \mathsf{crs}_h, u)) \leftarrow \mathsf{parse}(\mathsf{crs}_{\mathsf{DLInner}})$
$(R, S, \mathsf{bl}_1, \mathsf{bl}_2, \pi_{\mathsf{recurse},1}, \ldots, \pi_{\mathsf{recurse},\log_n}, b_f, c_f) \leftarrow \mathsf{parse}(\pi_{\mathsf{DLInner}})$
$x \leftarrow \mathsf{Hash}(R, S, \mathsf{bl}_1, \mathsf{bl}_2)$
$z \leftarrow z + x\mathsf{bl}_1 + x^2\mathsf{bl}_2$
$B \leftarrow BR^x$
$C \leftarrow CS^x$

**Step 2**:
$x \leftarrow \mathsf{Hash}(x)$
$u \leftarrow u^x$
$B \leftarrow Bu^{xz}$

**Step 3**:
for $1 \leq j \leq \log(n)$ :
    $n \leftarrow \frac{n}{2}$
    $(c_{L,b}, c_{L,c}, c_{R,b}, c_{R,c}) \leftarrow \mathsf{parse}(\pi_{\mathsf{recurse},j})$
    $x \leftarrow \mathsf{Hash}(c_{L,b}, c_{L,c}, c_{R,b}, c_{R,c})$
    $B \leftarrow c_{L,b}^x \cdot B \cdot c_{R,b}^{x^{-1}}$
    $C \leftarrow c_{L,c}^x \cdot C \cdot c_{R,c}^{x^{-1}}$
    $(g_1, \ldots, g_n) \leftarrow (g_1 g_{n+1}^{x^{-1}}, \ldots, g_n g_{2n}^{x^{-1}})$
    $(h_1, \ldots, h_n) \leftarrow (h_1 h_{n+1}^{x}, \ldots, h_n h_{2n}^{x})$

**Step 4**:
check $B = g_1^{b_f} u^{b_f c_f}$
check $C = h_1^{c_f}$
return 1 if both checks pass, else return 0.
```

Figure 6: Verify algorithm to check that $(B, C, z) = (g_1^{b_1} \cdots g_n^{b_n}), (h_1^{c_1} \cdots h_n^{c_n}), b_1c_1 + \cdots + b_nc_n$ for some field elements $(b_1, \ldots, b_n)$ and $(c_1, \ldots, c_n)$.

## 7 Same Exponent Argument

In this section we describe the same-exponentiation argument for the relation

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

which is given formally in Figures 7 and 8. We prove security in Theorems 9.4 and **??**.

We now give an overview of our zero-knowledge argument for $R_{\mathsf{DLInner}}$.

**Step 1**
**Prover:** The prover takes as input $(R, S, T, U, r, r_t, r_u)$ and samples three blinders $\mathsf{bl}_r, \mathsf{bl}_t, \mathsf{bl}_u$. They set

$$B_t = R^{\mathsf{bl}_r} g_t^{\mathsf{bl}_t} \text{ and } B_u = S^{\mathsf{bl}_r} g_u^{\mathsf{bl}_u}$$

and computes $x = \mathsf{Hash}(R, S, T, U, B_t, B_u)$.

**Verifier:** The verifier receives as input an instance $(R, S, T, U)$ and a proof $(B_t, B_u, z_r, z_t, z_u)$. They compute $x = \mathsf{Hash}(R, S, T, U, B_t, B_u)$.

**Step 2**
**Prover:** The prover computes

$$\begin{aligned} z_r &= \mathsf{bl}_r + xr \\ z_t &= \mathsf{bl}_t + xr_t \\ z_u &= \mathsf{bl}_u + xr_u \end{aligned}$$

**Verifier:** The verifier lets $b$ equal 1 if

$$\begin{aligned} B_t T^x = R^{z_r} g_t^{z_t} \\ B_u U^x = S^{z_r} g_u^{z_u} \end{aligned}$$

and zero otherwise.

**Outcome**
The prover returns the proof $\pi_{\mathsf{sameexp}} = (B_t, B_u, z_r, z_t, z_u) \in \mathbb{G}^2 \times \mathbb{F}^3$.

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

## 8 Multiexponentiation Argument

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

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

which is given in Figures 9 and 10. We prove our construction sound and zero-knowledge in Theorems 9.5 and **??**.

```
$\underline{\mathsf{SameexpProve}(R_{\mathsf{sameexp}}, R, S, T, U, r, r_t, r_u)}$
Step 1:
$\mathsf{bl}_r, \mathsf{bl}_t, \mathsf{bl}_u \xleftarrow{\$} \mathbb{F}$
$B_t \leftarrow R^r g_t^{\mathsf{bl}_t}$
$B_u \leftarrow S^r g_u^{\mathsf{bl}_u}$
$x \leftarrow \mathsf{Hash}(R, S, T, U, B_t, B_u)$

Step 2:
$z_r \leftarrow \mathsf{bl}_r + rx$
$z_t \leftarrow \mathsf{bl}_t + r_t x$
$z_u \leftarrow \mathsf{bl}_u + r_u x$
return $(B_t, B_u, z_r, z_t, z_u)$
```

Figure 7: Proving algorithm to demonstrate that $(T, U) = (R^r g_t^{r_t}, S^r g_u^{r_u})$ for some field element $r, r_t, r_u$.

```
$\underline{\mathsf{SameexpVerify}(R_{\mathsf{sameexp}}, (R, S, T, U), \pi_{\mathsf{sameexp}})}$
Step 1:
$(B_t, B_u, z_r, z_t, z_u) \leftarrow \mathsf{parse}(\pi_{\mathsf{sameexp}})$
$x \leftarrow \mathsf{Hash}(R, S, T, U, B_t, B_u)$

Step 2:
check $B_t T^x = R^{z_r} g_t^{z_t}$
check $B_u U^x = S^{z_r} g_u^{z_u}$
return 1 if both checks pass, else return 0
```

Figure 8: Verifying algorithm to demonstrate that $(T, U) = (R^r g_t^{r_t}, S^r g_u^{r_u})$ for some field element $r, r_t, r_u$.

We now give an overview of our zero-knowledge argument for $R_{\mathsf{multiexp}}$.

**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 $R, R_{\mathsf{bl}}, S_{\mathsf{bl}} \in \mathbb{G}$ as

$$\begin{array}{ll} R & = g_1^{r_1} \cdots g_n^{r_n} \\ T_{\mathsf{bl}} & = T_1^{r_1} \cdots T_n^{r_n} \\ U_{\mathsf{bl}} & = U_1^{r_1} \cdots U_n^{r_n} \end{array}$$

They hash $R, T_{\mathsf{bl}}, U_{\mathsf{bl}}$ obtain the field element $x$. The prover resets the inputs to equal

$$(a_1, \ldots, a_n) = (a_1 + xr_1, \ldots, a_n + xr_n)$$

**Verifier:** The verifier receives as input $(R, T_{\mathsf{bl}}, U_{\mathsf{bl}}, \pi_{\mathsf{recurse},1}, \ldots, \pi_{\mathsf{recurse},\log_n}, a_f)$. They hash $(R, T_{\mathsf{bl}}, U_{\mathsf{bl}}) \in \mathbb{G}^3$ to obtain the field element $x \in \mathbb{F}$. The verifier resets the inputs to equal

$$A = AR^x \text{ and } T = TT_{\mathsf{bl}}^x \text{ and } U = UU_{\mathsf{bl}}^x$$

**Step 2**
**Prover:** The prover now runs a recursive argument. For $1 \leq j \leq \log_2(n)$ they do the following

1. Reset $n = \frac{n}{2}$

2. Set

$$\begin{array}{ll} z_{L,t} = \prod_{i=1}^n T_{n+i}^{a_i} & z_{R,t} = \prod_{i=1}^n T_i^{a_{n+i}} \\ z_{L,u} = \prod_{i=1}^n U_{n+i}^{a_i} & z_{R,u} = \prod_{i=1}^n U_i^{a_{n+i}} \\ c_L = \prod_{i=1}^n g_{n+i}^{a_i} & c_R = \prod_{i=1}^n g_i^{a_{n+i}} \end{array}$$

   and

$$\pi_{\mathsf{recurse},j} = (z_{L,t}, z_{L,u}, z_{R,t}, z_{R,u}, c_L, c_R).$$

   Compute $x = \mathsf{Hash}(z_{L,t}, z_{L,u}, z_{R,t}, z_{R,u}, c_L, c_R)$.

3. Reset

$$\begin{array}{ll} \boldsymbol{a} = (a_1 + x^{-1}a_{n+1}, \ldots, a_n + x^{-1}a_{2n}) & \mathsf{crs}_g = (g_1 g_{n+1}^x, \ldots, g_n g_{2n}^x) \\ \boldsymbol{T} = (T_1 T_{n+1}^x, \ldots, T_n T_{2n}^x) & \boldsymbol{U} = (U_1 U_{n+1}^x, \ldots, U_n U_{2n}^x) \end{array}$$

**Verifier:** The verifier now runs a recursive argument. For $1 \leq j \leq \log_2(n)$ they do the following

1. Reset $n = \frac{n}{2}$.

2. Parse $(z_{L,t}, z_{L,u}, z_{R,t}, z_{R,u}, c_L, c_R) = \pi_{\mathsf{recurse},j}$ and compute $x = \mathsf{Hash}(z_{L,t}, z_{L,u}, z_{R,t}, z_{R,u}, c_L, c_R)$.

3. Reset

$$\begin{array}{ll} A = c_L^x \cdot A \cdot c_R^{x^{-1}} & \mathsf{crs}_g = (g_1 g_{n+1}^{x^{-1}}, \ldots, g_n g_{2n}^{x^{-1}}) \\ T = z_{L,t}^x \cdot T \cdot z_{R,t}^{x^{-1}} & \boldsymbol{T} = (T_1 T_{n+1}^x, \ldots, T_n T_{2n}^x) \\ U = z_{L,u}^x \cdot T \cdot z_{R,u}^{x^{-1}} & \boldsymbol{U} = (U_1 U_{n+1}^x, \ldots, U_n U_{2n}^x) \end{array}$$

**Step 3**
**Prover:** The prover sets

$$a_f = a_1$$

to be the final inputs from $\boldsymbol{a}$ in the recursion. Note in the final round this vector will have length 1.

**Verifier:** The verifier lets $b$ equal 1 if

$$A = g_1^{a_f} \text{ and } T = T_1^{a_f} \text{ and } U = U_1^{a_f}$$

and zero otherwise. Here $g_1$, $T_1$ and $U_1$ are the final values of $\mathsf{crs}_g$, $\boldsymbol{T}$ and $\boldsymbol{U}$ in the recursion which are vectors of length 1.

**Outcome**
The prover returns the proof $\pi_{\mathsf{multiexp}} = (R, T_{\mathsf{bl}}, U_{\mathsf{bl}}, \pi_{\mathsf{recurse},1}, \ldots, \pi_{\mathsf{recurse},\log_n}, a_f)$.

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

## 9 Security Proofs

### 9.1 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 9.1** (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 $\pi_{\mathsf{shuffle}}$ that is indistinguishable from a proof generated by an honest prover that knows the witness.

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

```
$\underline{\mathsf{MultiexpProve}(\mathsf{crs}_{\mathsf{multiexp}}, (T, U, A, T_1, U_1, \ldots, T_n, U_n), (a_1, \ldots, a_n))}$
**Step 1**:
$(R_{\mathsf{multiexp}}, \mathsf{crs}_g) \leftarrow \mathsf{parse}(\mathsf{crs}_{\mathsf{multiexp}})$
$r_1, \ldots, r_n \xleftarrow{\$} \mathbb{F}$
$R \leftarrow g_1^{r_1} \cdots g_n^{r_n}$
$T_{\mathsf{bl}} \leftarrow T_1^{r_1} \cdots T_n^{r_n}$
$U_{\mathsf{bl}} \leftarrow U_1^{r_1} \cdots U_n^{r_n}$
$x \leftarrow \mathsf{Hash}(R, T_{\mathsf{bl}}, U_{\mathsf{bl}})$
$(a_1, \ldots, a_n) \leftarrow (a_1 + xr_1, \ldots, a_n + xr_n)$

**Step 2**:
for $1 \leq j \leq \log(n)$ :
    $n \leftarrow \frac{n}{2}$
    $z_{L,t} \leftarrow \prod_{i=1}^{n} T_{n+i}^{a_i}$
    $z_{L,u} \leftarrow \prod_{i=1}^{n} U_{n+i}^{a_i}$
    $z_{R,t} \leftarrow \prod_{i=1}^{n} T_i^{a_{n+i}}$
    $z_{R,u} \leftarrow \prod_{i=1}^{n} U_i^{a_{n+i}}$
    $c_L \leftarrow \prod_{i=1}^{n} g_{n+i}^{a_i}$
    $c_R \leftarrow \prod_{i=1}^{n} g_i^{a_{n+i}}$
    $\pi_{\mathsf{recurse},j} \leftarrow (z_{L,t}, z_{L,u}, z_{R,t}, z_{R,u}, c_L, c_R)$
    $x \leftarrow \mathsf{Hash}(z_{L,t}, z_{L,u}, z_{R,t}, z_{R,u}, c_L, c_R)$
    $(a_1, \ldots, a_n) \leftarrow (a_1 + x^{-1}a_{n+1}, \ldots, a_n + x^{-1}a_{2n})$
    $(T_1, \ldots, T_n) \leftarrow (T_1 T_n^x, \ldots, T_n T_{2n}^x)$
    $(U_1, \ldots, U_n) \leftarrow (U_1 U_n^x, \ldots, U_n U_{2n}^x)$
    $(g_1, \ldots, g_n) \leftarrow (g_1 g_{n+1}^x, \ldots, g_n g_{2n}^x)$

**Step 3**:
$a_f \leftarrow a_1$
return $(R, T_{\mathsf{bl}}, U_{\mathsf{bl}}, \pi_{\mathsf{recurse},1}, \ldots, \pi_{\mathsf{recurse},\log(n)}, a_f)$
```

Figure 9: Proving algorithm to demonstrate that $(T, U, A) = (T_1^{a_1} \cdots T_n^{a_n}), (U_1^{a_1} \cdots U_n^{a_n}), g_1^{a_1} \cdots g_n^{a_n}$ for some field elements $(a_1, \ldots, a_n)$.

```
$\underline{\mathsf{MultiexpVerify}(\mathsf{crs}_{\mathsf{multiexp}}, (T, U, A, T_1, U_1, \ldots, T_n, U_n), \pi_{\mathsf{multiexp}})}$
**Step 1**:
$(R_{\mathsf{multiexp}}, \mathsf{crs}_g) \leftarrow \mathsf{parse}(\mathsf{crs}_{\mathsf{multiexp}})$
$(R, T_{\mathsf{bl}}, U_{\mathsf{bl}}, \pi_{\mathsf{recurse},1}, \ldots, \pi_{\mathsf{recurse},\log(n)}, a_f) \leftarrow \mathsf{parse}(\pi_{\mathsf{multiexp}})$
$x \leftarrow \mathsf{Hash}(R, T_{\mathsf{bl}}, U_{\mathsf{bl}})$
$A \leftarrow AR^x$
$T \leftarrow TT_{\mathsf{bl}}^x$
$U \leftarrow UU_{\mathsf{bl}}^x$

**Step 2**:
for $1 \leq j \leq \log(n)$ :
    $n \leftarrow \frac{n}{2}$
    $(z_{L,t}, z_{L,u}, z_{R,t}, z_{R,u}, c_L, c_R) \leftarrow \mathsf{parse}(\pi_{\mathsf{recurse},j})$
    $x \leftarrow \mathsf{Hash}(z_{L,t}, z_{L,u}, z_{R,t}, z_{R,u}, c_L, c_R)$
    $A \leftarrow c_L^x \cdot A \cdot c_R^{x^{-1}}$
    $T \leftarrow z_{L,t}^x \cdot T \cdot z_{R,t}^{x^{-1}}$
    $U \leftarrow z_{L,u}^x \cdot U \cdot z_{R,u}^{x^{-1}}$
    $(g_1, \ldots, g_n) \leftarrow (g_1 g_{n+1}^x, \ldots, g_n g_{2n}^x)$
    $(T_1, \ldots, T_n) \leftarrow (T_1 T_{n+1}^x, \ldots, T_n T_{2n}^x)$
    $(U_1, \ldots, U_n) \leftarrow (U_1 U_{n+1}^x, \ldots, U_n U_{2n}^x)$

**Step 3**:
check $A = g_1^{a_f}$
check $T = T_1^{a_f}$
check $U = U_1^{a_f}$
return 1 if all checks pass, else return 0.
```

Figure 10: Verify algorithm to check that $(T, U, A) = (T_1^{a_1} \cdots T_n^{a_n}), (U_1^{a_1} \cdots U_n^{a_n}), g_1^{a_1} \cdots g_n^{a_n}$ for some field elements $(a_1, \ldots, a_n)$.

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. □

**Theorem 9.2** (Grand product argument is zero-knowledge)**.** *If the discrete logarithm inner product argument is zero-knowledge, then the grand-product argument described in Figures 3 and 4 is zero-knowledge.*

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

$$A_1, \mathsf{gprod}$$

and outputs a proof $\pi_{\mathsf{shuffle}}$ that is indistinguishable from a proof generated by an honest prover that knows the witness.

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

1. In the first step they sample $B \xleftarrow{\$} \mathbb{G}$ and $\mathsf{bl} \xleftarrow{\$} \mathbb{F}$. They compute $x = \mathsf{Hash}(T_1, \ldots, T_\ell, U_1, \ldots, U_\ell, M)$.
2. In the second step they compute the commitment $C = A_1(g_1 \cdots g_\ell)^{-x^{-1}}$ and the generators
$$(h_1, \ldots, h_{\ell+4}) = (g_2^{x^{-1}}, g_3^{x^{-2}}, \ldots, g_\ell^{x^{-(\ell-1)}}, g_1^{x^{-\ell}}, g_{\ell+1}^{x^{-(\ell+1)}}, \ldots, g_{\ell+4}^{x^{-(\ell+1)}})$$
3. In the third step they compute
$$\pi_{\mathsf{DLInner}} = \mathsf{Simulate}((\mathsf{crs}_g, \mathsf{crs}_h, u), (B, C, (\mathsf{gprod}x^\ell + \mathsf{bl}x^{\ell+1} - 1))).$$
They return $(R, \mathsf{bl}, \pi_{\mathsf{DLInner}})$.

Now we must argue that the simulated proof is indistinguishable from the real proof. To do this first note that $\pi_{\mathsf{DLInner}}$ is a zero-knowledge proofs and thus the simulated proofs are indistinguishable from the real proofs. The provers value $R, \mathsf{bl}$ is randomised by $b_1, \ldots, b_4$ and thus is indistinguishable from the simulators random $R, \mathsf{bl}$. This completes the proof. □

**Theorem 9.3** (Discrete log inner product argument is zero-knowledge)**.** *The discrete logarithm inner product argument described in Figures 5 and 6 is zero-knowledge in the random oracle model.*

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

$$(B, C, z)$$

and outputs a proof $\pi_{\mathsf{DLInner}}$ 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 $b_1, c_1, \ldots, b_n, c_n, \mathsf{bl}_1, \mathsf{bl}_2, x \xleftarrow{\$} \mathbb{F}$ such that $\sum_{i=1}^{n} b_i c_i = z + x\mathsf{bl}_1 + x^2\mathsf{bl}_2$. They compute

   $$R = (g_1^{b_1} \cdots g_n^{b_n} B^{-1})^{x^{-1}} \text{ and } S = (h_1^{c_1} \cdots h_n^{c_n} C^{-1})^{x^{-1}}$$

   They program $\mathsf{Hash}(R, S, \mathsf{bl}_1, \mathsf{bl}_2)$ to equal $x$.

2. In the third step through to the fourth step they behave exactly as the honest prover with respect to the inputs $\boldsymbol{b}$ and $\boldsymbol{c}$.

Now we must argue that the simulated proof is indistinguishable from the real proof and that the simulator doesn't abort. First observe that $R$, $S$ 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 values and simulators values $R$, $\mathsf{bl}_1$, $\mathsf{bl}_2$ $c_{L,b}$ and $c_{R,b}$ for $1 \leq j \leq n$ are randomly distributed due to the randomises $r_1, \ldots, r_n$ for the prover and $b_1, \ldots, b_n$ for the simulator. The value $b_f$ is the unique value then satisfying the verifiers equation for both the prover and simulator. Third observe that the provers values and simulators values $S$, $c_{L,c}$ and $c_{R,c}$ for $1 \leq j \leq n$ are randomly distributed due to the randomises $s_1, \ldots, s_n$ for the prover and $c_1, \ldots, c_n$ for the simulator. The value $c_f$ is the unique value then satisfying the verifiers equation for both the prover and simulator. This completes the proof. □

**Theorem 9.4** (Same exponent argument is zero-knowledge)**.** *The same exponent argument described in Figures 7 and 8 is zero-knowledge in the random oracle model.*

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

$$(R, S, T, U)$$

and outputs a proof $\pi_{\mathsf{sameexp}}$ 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 chooses $z_r, z_t, z_u, x \xleftarrow{\$} \mathbb{F}$. They set $B_t = T^{-x} R^{z_r} g_t^{z_t}$ and $B_u = U^{-x} S^{z_r} g_u^{z_u}$. They program the random oracle to return $x$ on input $B_t$ and $B_u$ and return $(B_t, B_u, z_r, z_t, z_u)$.

First observe that $B_t, B_u$ are randomised and thus with overwhelming probability the oracle will not have already been programmed at this point. Second we see that the values $B_t, B_u, z_r$ are perfectly randomised for both prover and simulator. Given $B_t, B_u, z_r$ the values $z_t$ and $z_u$ are uniquely determined by the verifier. Thus the prover and simulator values are sampled from the same distribution. □

**Theorem 9.5** (Multiexponentiation argument is zero-knowledge)**.** *The same exponent argument described in Figures 9 and 10 is zero-knowledge in the random oracle model.*

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

$$(T, U, A, T_1, U_1, \ldots, T_n, U_n)$$

and outputs a proof $\pi_{\mathsf{multiexp}}$ 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 $\mathsf{Simulate}$ samples $a_1, \ldots, a_n x \xleftarrow{\$} \mathbb{F}$ and computes

$$\begin{aligned} A' &= g_1^{a_1} \cdots g_n^{a_n} \\ T' &= T_1^{a_1} \cdots T_n^{a_n} \\ U' &= U_1^{a_1} \cdots U_n^{a_n} \end{aligned}$$

They set

$$\begin{aligned} R &= (A'A^{-1})^{-x} \\ T_{\mathsf{bl}} &= (T'T^{-1})^{-x} \\ U_{\mathsf{bl}} &= (U'U^{-1})^{-x} \end{aligned}$$

program $\mathsf{Hash}(R, T_{\mathsf{bl}}, U_{\mathsf{bl}})$ to equal $x$. In the remaining steps they behave exactly as the honest prover with respect to the inputs $\boldsymbol{a}$.

Now we must argue that the simulated proof is indistinguishable from the real proof and that the simulator doesn't abort. First observe that $R$, $T_{\mathsf{bl}}$ and $U_{\mathsf{bl}}$ 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 values and simulators values $R$, $T_{\mathsf{bl}}$, $U_{\mathsf{bl}}$ , $z_{L,t}$, $z_{L,u}$, $z_{R,t}$, $z_{R,u}$, $c_L$, and $c_R$ for $1 \leq j \leq n$ are randomly distributed due to the randomises $r_1, \ldots, r_n$ for the prover and $a_1, \ldots, a_n$ for the simulator. The value $a_f$ is the unique value then satisfying the verifiers equation for both the prover and simulator. This completes the proof. $\square$

## 9.2 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 9.6** (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 $\sigma$ 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 $r_{\mathsf{coin}}$. 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 $r_{\mathsf{coin}}$. 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 $r_{\mathsf{coin}}$.

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 $\sigma$ 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 $(\sigma, r)$ is such that $(T_i, U_i) = (R_{\sigma(i)}^r, S_{\sigma(i)}^r)$ for all $1 \leq i \leq \ell$ then return $(\sigma, r)$. 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 $\alpha$,

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

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. $\square$

**Theorem 9.7** (Grand Product Argument is sound)**.** *If the discrete logarithm inner product argument is knowledge-sound, and the discrete logarithm assumption holds, then the shuffle argument described in Figures 3 and 4 is knowledge-sound.*

*Proof.* We design an extractor $\mathcal{X}_{\mathsf{gprod}}$ 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 some $(a_1, \ldots, a_n) \in \mathbb{F}$ such that

$$((A, \mathsf{gprod}), (a_1, \ldots, a_n)) \in R_{\mathsf{gprod}}.$$

By the knowledge-soundness of the discrete log inner product argument there exists an extractors $\mathcal{X}_{\mathsf{DLInner}}$ such that if $\mathcal{A}$ returns verifying $\pi_{\mathsf{DLInner}}$ then they return valid witnesses for $R_{\mathsf{DLInner}}$ with overwhelming probability.

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

1. Run $\mathcal{A}$ to obtain $B, \mathsf{bl}, \pi_{\mathsf{DLInner}}$ that the verifier accepts. Compute $x = \mathsf{Hash}(B, \mathsf{bl})$ and run $\mathcal{X}_{\mathsf{DLInner}}$ to return $(b_1, \ldots, b_n)$ and $(c_1, \ldots, c_n)$ such that

$$\begin{aligned} B &= g_1^{b_1} \cdots g_n^{b_n} \\ C &= h_1^{c_1} \cdots h_n^{c_n} \quad = g_2^{c_1 x^{-1}} g_3^{c_2 x^{-2}} \cdots g_\ell^{c_\ell x^{-(\ell-1)}} g_1^{c_1 x^{-\ell}} g_{\ell+1}^{x^{-(\ell+1)} c_{\ell+1}} \cdots g_{\ell+4}^{x^{-(\ell+1)} c_{\ell+4}} \end{aligned}$$

   Set

$$(a_1, \ldots, a_n) = ((c_\ell + x^{\ell-1}) x^{-\ell}, (c_1 + 1) x^{-1}, \ldots, (c_{\ell-1} + x^{\ell-2}) x^{-(\ell-1)} c_{\ell+1} x^{-(\ell+1)}, \ldots, c_{\ell+4} x^{-(\ell+1)})$$

2. Program $y = \mathsf{Hash}(B, \mathsf{bl})$ and run $\mathcal{A}$ and $\mathcal{X}_{\mathsf{DLInner}}$ again to return $(b_1', \ldots, b_n')$ and $(c_1', \ldots, c_n')$. If $(b_1, \ldots, b_n') \neq (b_1, \ldots, b_n)$ return a collision in $\mathsf{crs}_g$. If $(c_1', \ldots, c_n') \neq (a_2 y - 1, a_3 y^2 - y, \ldots, a_\ell y^{\ell-1} - y^{\ell-2}, a_1 y^\ell - y^{\ell-1}, a_{\ell+1} y^{\ell+1}, \ldots, a_{\ell+4} y^{\ell+1})$ return a collision in $\mathsf{crs}_g$.

3. If $(b_1, \ldots, b_\ell) \neq (1, a_2, a_2a_3, \ldots, a_2 \cdots a_\ell)$ then abort. If $a_1 \cdots a_\ell \neq \mathsf{gprod}$ then abort. Return $(a_1, \ldots, a_n)$.

**Extractor runs in polynomial time:** This follows from the fact that $\mathcal{X}_{\mathsf{DLInner}}$ runs in polynomial time.

**Extractor does not abort:** The extrator does not abort in Steps 1 or 2 because the extractor $\mathcal{X}_{\mathsf{DLInner}}$ succeeds. Furthermore $(a_2y-1, a_3y^2-y, \ldots, a_\ell y^{\ell-1}-y^{\ell-2}, a_1y^\ell-y^{\ell-1}, a_{\ell+1}y^{\ell+1}, \ldots, a_{\ell+4}y^{\ell+1})$ is a valid discrete logarithm for $A_1(g_1 \ldots g_\ell)^{-x^{-1}}$ with respect to the basis $(h_1, \ldots, h_n) = g_2^{y^{-1}}, \ldots, g_{\ell+4}^{y^{-(\ell+1)}}$.

In Step 3 we have that

$$\begin{aligned} <(b_1, \ldots, b_n), (c'_1, \ldots, c'_n)>= \; & (b_1a_2y - b_1) + (b_2a_3y^2 - b_2y) + \ldots + (b_{\ell-1}a_\ell y^{\ell-1} - b_{\ell-1}y^{\ell-2}) \\ & +(b_\ell a_1 y^\ell - b_\ell y^{\ell-1}) + (b_{\ell+1}a_{\ell+1} + \ldots + b_n a_n)y^{\ell+1}. \\ & = -b_1 + (a_2b_1 - b_2)y + \ldots + (a_\ell b_{\ell-1} - b_\ell)y^{\ell-1} + a_1b_\ell y^\ell \\ & +(b_{\ell+1}a_{\ell+1} + \ldots + b_na_n)y^{\ell+1} \\ & = -1 + \mathsf{gprod}y^\ell + \mathsf{bl}y^{\ell+1} \end{aligned}$$

at a random point $y$ and by the Schwartz-Zippel Lemma the following holds with overwhelming probability

$$\begin{array}{ll} b_1 = 1 & \\ b_1a_2 - b_2 = 0 & \Rightarrow a_2 = b_2 \\ b_2a_3 - b_3 = 0 & \Rightarrow b_3 = a_2a_3 \\ \vdots & \\ b_{\ell-1}a_\ell - b_\ell = 0 & \Rightarrow b_\ell = a_2 \cdots a_\ell \\ b_\ell a_1 = \mathsf{gprod} & \Rightarrow \mathsf{gprod} = a_1 \cdots a_\ell \\ b_{\ell+1}a_{\ell+1} + \ldots + b_na_n \neq \mathsf{bl} & \end{array}$$

Thus $\mathcal{X}_{\mathsf{gprod}}$ does not abort. □

To prove the following theorem we take advantage of a theorem for proving the security of generalised inner product arguments by Bünz et al [BMMV19] (Theorem 5.4).

**Theorem 9.8** (Discrete Logarithm Inner Product Argument is sound)**.** *The discrete logarithm inner product argument described in Figures 5 and 6 is knowledge-sound in the random oracle model under the discrete logarithm assumption.*

*Proof.* First observe that by the discrete-logarithm assumption

$$(B, C) = (g_1^{b_1} \cdots g_n^{b_n} u^z, g_1^{c_1} \cdots g_n^{c_n})$$

is a binding inner product commitment to $(b_1, \ldots, b_n, c_1, \ldots, c_n, z) \in \mathbb{F}^{2n+1}$. Also observe that the recursive inner product argument in Step 3 is a direct instantiation of the generalised inner product argument of Bünz et al. Thus by [BMMV19], Theorem 5.4, there exists an extractor $\mathcal{X}_{\mathsf{GIPA}}$ that outputs $(\beta_1, \ldots, \beta_n, \gamma_1, \ldots, \gamma_n, z)$ such that

$$\begin{aligned} B &= g_1^{\beta_1} \cdots g_n^{\beta_n} u^z \\ C &= g_1^{\gamma_1} \cdots g_n^{\gamma_n} \\ z &= \beta_1\gamma_1 + \cdots + \beta_n\gamma_n \end{aligned}$$

in Step 3.

We now specify an extractor $\mathcal{X}_{\mathsf{DLInner}}$ that works as follows.

1. Run $\mathcal{A}$ to obtain $R, S$ that the verifier accepts. Compute $x_1 = \mathsf{Hash}(R, S, \mathsf{bl}_1, \mathsf{bl}_2)$ and $y_1 = \mathsf{Hash}(x_1)$ and run $\mathcal{X}_{\mathsf{GIPA}}$ to return $(\alpha_1, \ldots, \alpha_n)$ and $(\beta_1, \ldots, \beta_n)$ and $z_1$.

2. Program $\mathsf{Hash}()$ to return $x_2$ on input $(R, S, \mathsf{bl}_1, \mathsf{bl}_2)$ and $y_1$ on input $x_2$. Run $\mathcal{A}$ and $\mathcal{X}_{\mathsf{GIPA}}$ again to return $(\gamma_1, \ldots, \gamma_n)$ and $(\delta_1, \ldots, \delta_n)$ and $z_2$.

3. Set
$$\begin{aligned} (r_1, \ldots, r_n, r_u) &= ((\alpha_1 - \gamma_1)(x_1 - x_2)^{-1}, \ldots, (\alpha_n - \gamma_n)(x_1 - x_2)^{-1}, (z_1 - z_2)(x_1 - x_2)^{-1}) \\ (s_1, \ldots, s_n) &= ((\beta_1 - \delta_1)(x_1 - x_2)^{-1}, \ldots, (\beta_n - \delta_n)(x_1 - x_2)^{-1}) \end{aligned}$$
Abort if $\boldsymbol{r}$ or $\boldsymbol{s}$ is not a valid discrete logarithm of $R$, $S$ with respect to the basis $(g_1, \ldots, g_n)$.

4. Set
$$\begin{aligned} (b_1, \ldots, b_n, b_u) &= (\alpha_1 - x_1 r_1, \ldots, \alpha_n - x_1 r_n, z_1 - x_1 r_u) \\ (c_1, \ldots, c_n) &= (\beta_1 - x_1 s_1, \ldots, \beta_n - x_1 s_n) \end{aligned}$$
and abort if $\boldsymbol{b}$ or $\boldsymbol{c}$ is not a valid discrete logarithm of $B, C$ with respect to the basis $(g_1, \ldots, g_n, u_n)$.

5. Program $\mathsf{Hash}()$ to return $x_3$ on input $(R, S, \mathsf{bl}_1, \mathsf{bl}_2)$ and $y_3$ on input $x_3$. Run $\mathcal{A}$ and $\mathcal{X}_{\mathsf{GIPA}}$ again to return $(\alpha'_1, \ldots, \alpha'_n)$ and $(\beta'_1, \ldots, \beta'_n)$ and $z_3$. If
$$(b_1 + x_3 r_1, \ldots, b_n + x_3 r_n, b_u + x_3 r_u + (z + x_3 \mathsf{bl}_1 + x_3^2 \mathsf{bl}_2) y_3) \neq (\alpha'_1, \ldots, \alpha'_n, y_3 z_3)$$
return a collision in $(\mathsf{crs}_g, u)$. Abort if $y_3 z_3 = b_u + x_3 r_u + (z + x_3 \mathsf{bl}_1 + x_3^2 \mathsf{bl}_2) y_3$ and $b_u, r_u \neq 0$.

6. If $(c_1 + x_3 s_1, \ldots, c_n + x_3 s_n) \neq (\beta'_1, \ldots, \beta'_n)$ return a collision in $\mathsf{crs}_g$. If $b_1 c_1 + \cdots + b_n c_n \neq z$ then abort. Return $(b_1, \ldots, b_n), (c_1, \ldots, c_n)$.

**Extractor runs in polynomial time:** This follows from the fact that $\mathcal{X}_{\mathsf{GIPA}}$ runs in polynomial time.

**Extractor does not abort:** The extrator does not abort in Steps 1, 2, 3, or 4 because the extractor $\mathcal{X}_{\mathsf{GIPA}}$ succeeds and because $x_1 \neq x_2$.

In Step 4 we have that
$$B R^{x_1} u^{(z + x_1 \mathsf{bl}_1 + x_1^2 \mathsf{bl}_2) y_1} = g_1^{\alpha_1} \cdots g_n^{\alpha_n} u^{y_1 z_1}$$
so the discrete logarithm is correct. A similar argument holds for $C$. Hence $\mathcal{X}_{\mathsf{DLInner}}$ does not abort.

In Step 5 we have that
$$(b_u + X r_u) + (z + X \mathsf{bl}_1 + X^2 \mathsf{bl}_2) Y = z_3(X) Y$$
evaluates to 0 at randomly chosen $x_3, y_3$ or $\mathcal{X}_{\mathsf{DLInner}}$ returns a collision. By the Scwartz-Zippel lemma this happens with negligible probability unless the polynomial is equal to 0. Hence with overwhelming probability and $b_u = r_u = 0$ and the extractor does not abort.

In Step 6 we have that

$$((b_1 + r_1X, \dots, b_n + r_nX) \cdot (c_1 + s_1X, \dots, c_n + s_nX)) = z + \mathsf{bl}_1X + \mathsf{bl}_2X^2$$

evaluates to 0 at randomly chosen $x_3$. By the Scwartz-Zippel lemma this happens with negligible probability unless the polynomial is equal to 0. In the latter case we get that

$$(b_1c_1 + \cdots + b_nc_n) - z_3$$

and the extractor outputs a valid witness. □

## 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.

[BMMV19] Benedikt Bünz, Mary Maller, Pratyush Mishra, and Noah Vesely. Proofs for inner pairing products and applications. *IACR Cryptol. ePrint Arch.*, 2019:1177, 2019.