A Shuffle Argument Protocol Specification
Work in Progress

The Ethereum Foundation Research Team

August 17, 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.

We implemented the scheme in Python using libff as a backend. We found that to shuffle 254 ciphertexts takes the prover in 4.8 seconds and the verifier in 0.38 seconds. It is expected that these numbers could be greatly improved with further implementation work.

This is currently a work in progress.

Contents

1 Notation **1**

2 Problem Statement **1**

3 Cryptographic Ingredients **1**
- 3.1 Hash Function 2
- 3.2 Group and Field 2
- 3.3 Randbelow 3

4 Public Coin Setup **3**

5 Construction **3**
- 5.1 Grand Product Relation 3
- 5.2 Same Exponent Relation 3
- 5.3 Multiexponentiation Relation 3
- 5.4 Overview 3

6 Grand Product Argument **5**
- 6.1 Discrete logarithm inner product relation 5
- 6.2 Grand Product Construction Overview 8

7 Security Proofs **9**
- 7.1 Correctness 9
- 7.2 Zero-Knowledge 9
- 7.3 Soundness 10

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

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

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 $1 \leq i \leq \ell$

$$T_i = R^r_{\sigma(i)} \ \wedge \ U_i = S^r_{\sigma(i)}.$$

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

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

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 Cryptographic Ingredients

We require the following cryptographic ingredients in our scheme

- A hash function that functions as a random oracle Hash.
- A group $\mathbb{G}$ in which the Decisional Diffie Hellman assumption holds.
- A random number generator $\mathsf{randbelow}$.

which we detail in this section.

3.1 Hash Function

The hash function Hash is implemented in `fiatshamir.py` and the underlying hash is SHA256. We use the `hashlib` implementation of SHA256. In order to ease integration with solidity we use the `encode_abi` function from the library `eth_abi`. This is a python equivalent of the solidity function `abi.encode` which passes an array of byte strings into a format that is accepted by SHA256. In order to ease integration with solidity we also use the `bytes_to_int` function from the library `py_ecc`. The inbuilt conversion from bytes to integers in solidity is different from the inbuilt conversion in python, and we choose to use the solidity default.

All the values we are hashing are lists of integers $v = [v_1, \ldots, v_n]$ where the maximum bit-size of each v_i element of v is 256. We hash the values one at a time. We begin with an empty byte string k. For $1 \leq i \leq n$, we convert v_i to a 32-byte binary representation using the inbuilt python function `to_bytes`. We then compute $k = \mathsf{Hash}(k, v_i)$ as $k = SHA256(encode\_abi([k, v_i]))$ Once we have the final k value, we convert back to integers using `bytes_to_int` .

```
$\underline{\mathsf{Hash}([v_1, \ldots, v_n])}$
$k = 0 \cdots 0$
for $1 \leq i \leq n$
    $k = SHA256(encode\_abi([k, to\_bytes(v_i)]))$
return $bytes\_to\_int(k)$
```

In order to hash group elements we include three integers in the list - one for each coordinate.

We store the variable `current_hash`, which is initialised to 1. Whenever we hash a new value, we insert $v_1 =$ `current_hash` to the front of the list. We update `current_hash` $= \mathsf{Hash}(v)$.

3.2 Group and Field

We can implement our shuffle argument over any group where the decisional diffie hellman assumption is assumed to be hard, or in any pairing group where the symmetric external diffie hellman assumption is assumed to be hard (this is often the case for Type 3 groups but not Type 1 groups). In the python implementation we use the BN254 curve. The add and multiply algorithms are implemented in `group_ops.py` A faster algorithm for computing multiexponentiations called `compute_multiexp` is included in `utils.py`.

The curve is built over the base field modulus

$$21888242871839275222246405745257275088696311157297823662689037894645226208583$$

and the equation for the curve is $y^2 z = x^3 + 3z^3$ or $y^2 = x^3 + 3$. The order of the curve and hence the field $\mathbb{F}_p$ is the integers modulo

$$21888242871839275222246405745257275088548364400416034343698204186575808495617.$$

3.3 Randbelow

4 Public Coin Setup

5 Construction

We begin by giving a full overview of the construction in Figures 1 and 2 which is proven secure in Theorems 7.1, 7.2 and 7.3. 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 ???.

5.1 Grand Product Relation

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

$$R_{\mathsf{gprod}} = \left\{ \ (A_1, \mathsf{gprod}), (a_1, \dots, a_{\ell+4}) \ \middle| \ A_1 = h_1^{a_1} \cdots h_n^{a_{\ell+4}} \wedge \mathsf{gprod} = a_1 \cdots a_\ell \ \right\}$$

5.2 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

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

5.3 Multiexponentiation Relation

The multiexponentiation relation demonstrates that given public input $(T_1, U_1, \dots, T_{\ell+4}, U_{\ell+4}) \in \mathbb{G}^{2\times(\ell+4)}$, $A \in \mathbb{G}$, and $(T, U) \in \mathbb{G}^2$ there exists $(a_1, \dots, a_{\ell+4})$ such that $A = \mathsf{compute\_multiexp}(\mathsf{crs}_h, (a_1, \dots, a_{\ell+4}))$, $T = \mathsf{compute\_multiexp}((T_1, \dots, T_{\ell+4}), (a_1, \dots, a_{\ell+4}))$, and $U = \mathsf{compute\_multiexp}((U_1, \dots, U_{\ell+4}), (a_1, \dots, a_{\ell+4}))$. In other words

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

5.4 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 correctness, zero-knowledge, and soundness in Section 7, Theorems ????.

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

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

where $\mathsf{crs}_h = (h_1, \ldots, h_{\ell+4}) \in \mathbb{G}^{\ell+4}$. 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 $a_{\ell+1}, a_{\ell+2}, a_{\ell+3}, a_{\ell+4} \overset{\$}{\leftarrow} \mathbb{F}$ randomly from $\mathbb{F}$ as masking values. They compute $A \in \mathbb{G}$ as

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

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 $AM^\alpha(h_1 \ldots h_{\ell+4})^\beta$. They compute a prove π_{gprod} that they know $(c_1, \ldots, c_{\ell+4})$ such that $A_1 = \prod_{i=1}^{\ell+4} h_i^{c_i}$ and $\prod_{i=1}^{\ell+4} c_i = \mathsf{gprod}$. In other words they run

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

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 $AM^\alpha(h_1 \ldots h_{\ell+4})^\beta$ where $\mathsf{crs}_h = (h_1, \ldots, h_{\ell+4}) \in \mathbb{G}^{\ell+4}$. 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 $\gamma_1, \delta_1, \ldots, \gamma_4, \delta_4 \in \mathbb{F}^4$. They compute $T, U \in \mathbb{G}^2$ as $T = R^r g_t^{\gamma_1 a_{\ell+1} + \ldots \gamma_4 a_{\ell+4}}$ and $U = S^r g_u^{\delta_1 a_{\ell+1} + \ldots \delta_4 a_{\ell+4}}$ 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 π_{sameexp} for the instance (R, S, T, U). In other words they run

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

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 $\gamma_1, \delta_1, \ldots, \gamma_4, \delta_4 \in \mathbb{F}^4$. 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

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

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.

6 Grand Product Argument

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

$$R_{\mathsf{gprod}} = \left\{ \ (A_1, \mathsf{gprod}), (a_1, \ldots, a_n) \ \middle| \ A_1 = h_1^{a_1} \ldots h_n^{a_n} \wedge \mathsf{gprod} = a_1 \ldots a_\ell \ \right\}$$

which is given in Figures ???. An overview of our argument is given in Section 6.2. We prove our construction correct, 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 ???

6.1 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 $(b_1, \ldots, b_{\ell+2})$ and $(c_1, \ldots, c_{\ell+2})$ such that $B = \prod_{i=1}^{\ell+2} g_i^{b_i}$ and $C = \prod_{i=1}^{\ell+2} h_i^{c_i}$. In other words

$$R_{\mathsf{DLInner}} = \left\{ \ (B, C, z), ((b_1, \ldots, b_{\ell+2}), (c_1, \ldots, c_{\ell+2})) \ \middle| \ B = g_1^{b_1} \cdots g_{\ell+2}^{b_{\ell+2}} \wedge B = h_1^{c_1} \cdots h_{\ell+2}^{c_{\ell+2}} \wedge z = b_1 c_1 + \ldots + b_{\ell+2} c_\ell\right.$$

```
$\underline{\mathsf{ShuffleProve}(\mathsf{crs}_{\mathsf{shuffle}}, R_1, S_1, T_1, U_1, \ldots, R_\ell, S_\ell, T_\ell, U_\ell, \sigma, r)}$
Step 1:
$(R_{\mathsf{shuffle}}, (\mathsf{crs}_g, \mathsf{crs}_h, u)) \leftarrow \mathsf{parse}(\mathsf{crs}_{\mathsf{shuffle}})$
$s_1, s_2, s_3, s_4 \leftarrow \mathsf{randbelow}(p)$
$M \leftarrow h_1^{\sigma(1)} \cdots h_\ell^{\sigma(\ell)} h_{\ell+1}^{s_1} \ldots h_{\ell+4}^{s_4}$
$(a_1, \ldots, a_\ell) \leftarrow \mathsf{Hash}(T_1, \ldots, T_\ell, U_1, \ldots, U_\ell, M)$

Step 2:
$a_{\ell+1}, \ldots, a_{\ell+4} \leftarrow \mathsf{randbelow}(p)$
$A \leftarrow \mathsf{compute\_multiexp}(\mathsf{crs}_h, (a_{\sigma(1)}, \ldots, a_{\sigma(\ell)}, a_{\ell+1}, \ldots, a_{\ell+4}))$
$\alpha, \beta \leftarrow \mathsf{Hash}(M, A)$

Step 3:
$\mathsf{gprod} \leftarrow (a_1 + 1 \cdot \alpha + \beta)(a_2 + 2 \cdot \alpha + \beta) \cdots (a_\ell + \ell \cdot \alpha + \beta) \mod p$
$A_1 \leftarrow A M^\alpha (h_1 \ldots h_{\ell+4})^\beta$
$\pi_{\mathsf{gprod}} \leftarrow \mathsf{Prove}\left(\mathsf{crs}_{\mathsf{gprod}}, (A_1, \mathsf{gprod}), \begin{array}{r}(a_{\sigma(1)} + \sigma(1)\alpha + \beta,\ \ldots,\ a_{\sigma(\ell)} + \sigma(\ell)\alpha + \beta, \\ a_{\ell+1} + s_1\alpha + \beta,\ a_{\ell+2} + s_2\alpha + \beta)\end{array}\right)$

Step 4:
$\gamma_1, \delta_1, \ldots, \gamma_4, \delta_4 \leftarrow \mathsf{Hash}(A)$
$R \leftarrow \mathsf{compute\_multiexp}((R_1, \ldots, R_\ell), (a_1\alpha, \ldots, a_\ell\alpha^\ell))$
$S \leftarrow \mathsf{compute\_multiexp}((S_1, \ldots, S_\ell), (a_1\alpha, \ldots, a_\ell\alpha^\ell))$
$T \leftarrow R^r g_t^{\gamma_1 a_{\ell+1} + \ldots + \gamma_4 a_{\ell+4}}$
$U \leftarrow S^r g_u^{\delta_1 a_{\ell+1} + \ldots + \delta_4 a_{\ell+4}}$
$\pi_{\mathsf{sameexp}} \leftarrow \mathsf{Prove}(R_{\mathsf{sameexp}}, (R, S, T, U), r)$

Step 5:
$\pi_{\mathsf{multiexp}} \leftarrow \mathsf{Prove}(\mathsf{crs}_{\mathsf{multiexp}}, ((T_1, U_1, \ldots, T_\ell, U_\ell, g_t^{\gamma_1}, g_u^{\delta_1}, \ldots, g_t^{\gamma_4}, g_u^{\delta_4}), A, (T, U)), (a_{\sigma(1)}, \ldots, a_{\sigma(\ell)}, a_{\ell+1}, a_{\ell+2}))$
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 σ.

```
$\underline{\mathsf{VerifyProve}(\mathsf{crs}_{\mathsf{shuffle}}, R_1, S_1, T_1, U_1, \ldots, R_\ell, S_\ell, T_\ell, U_\ell, \pi_{\mathsf{shuffle}})}$
**Step 1**:
$(R_{\mathsf{shuffle}}, (\mathsf{crs}_g, \mathsf{crs}_h, u)) \leftarrow \mathsf{parse}(\mathsf{crs}_{\mathsf{shuffle}})$
$(M, A, T, U, \pi_{\mathsf{gprod}}, \pi_{\mathsf{sameexp}}, \pi_{\mathsf{multiexp}}) \leftarrow \mathsf{parse}(\pi_{\mathsf{shuffle}})$
$(a_1, \ldots, a_\ell) \leftarrow \mathsf{Hash}(T_1, \ldots, T_\ell, U_1, \ldots, U_\ell, M)$

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

**Step 3**:
$\mathsf{gprod} \leftarrow (a_1 + 1 \cdot \alpha + \beta)(a_2 + 2 \cdot \alpha + \beta) \cdots (a_\ell + \ell \cdot \alpha + \beta) \mod p$
$A_1 \leftarrow AM^{\alpha}(h_1 \ldots h_n)^{\beta}$
$b_1 \leftarrow \mathsf{Verify}(\mathsf{crs}_{\mathsf{gprod}}, (A_1, \mathsf{gprod}), \pi_{\mathsf{gprod}})$

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

**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.

6.2 Grand Product Construction Overview

Step 1:
Prover: The prover takes as input $A_1 \in \mathbb{G}$, $\mathsf{gprod} \in \mathbb{F}$, $(a_1, \ldots, a_{\ell+2}) \in \mathbb{F}^{\ell+2}$ such that $A_1 = \prod_{i=1}^{\ell+2} h_i^{a_i}$ and $\mathsf{gprod} = \prod_{i=1}^{\ell} a_i$. They set

$$(b_1, \ldots, b_\ell) = (1, a_2, a_2 a_3, a_2 a_3 a_4, \ldots, a_2 \cdots a_\ell)$$

and select 2 random values $b_{\ell+1}, \ldots, b_{\ell+2} \xleftarrow{\$} \mathbb{F}$ from the field. They compute

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

and hash $A_1, \mathsf{gprod}, B, \mathsf{bl} \in \mathbb{G} \times \mathbb{F}$ 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 .... . They hash A_1, gprod along with $B, \mathsf{bl} \in \mathbb{G} \times \mathbb{F}$ to obtain the randomness $x \in \mathbb{F}$.

Step 2:
Prover: The prover computes the commitment

$$C = A_1 (h_1 \cdots h_\ell)^{-x^{-1}}.$$

They then reset the generators

$$(h_1, \ldots, h_n) = (h_2^{x^{-1}}, h_3^{x^{-2}}, \ldots, h_\ell^{x^{\ell-1}}, h_1^{x^\ell}, h_{\ell+1}^{x^{-\ell-1}}, \ldots h_n^{x^{-\ell-1}}).$$

and the corresponding opening to C as

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

Verifier: The verifier computes the commitment

$$C = A_1 (h_1 \cdots h_\ell)^{-x^{-1}}.$$

They then reset the generators

$$(h_1, \ldots, h_n) = (h_2^{x^{-1}}, h_3^{x^{-2}}, \ldots, h_\ell^{x^{\ell-1}}, h_1^{x^\ell}, h_{\ell+1}^{x^{-\ell-1}}, \ldots h_n^{x^{-\ell-1}}).$$

Remark: Observe that

$$\begin{aligned} < (b_1, \ldots, b_n), (c_1, \ldots, c_n) >= \; & (b_1 a_2 x - b_1) + (b_2 a_3 x^2 - b_2 x) + \ldots + (b_{\ell-1} a_\ell x^{\ell-1} - b_{\ell-1} x^{\ell-2}) \\ & + (b_\ell a_1 x^\ell - b_\ell x^{\ell-1}) + (b_{\ell+1} a_{\ell+1} + \ldots + b_n a_n) x^{\ell+1}. \end{aligned}$$

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

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

Step 3:
Prover: The prover computes a prove π_{gpInner} that they know $(b_1, \ldots, b_{\ell+2})$ and $(c_1, \ldots, c_{\ell+2})$ such that $B = \prod_{i=1}^{\ell+2} g_i^{b_i}$ and $C = \prod_{i=1}^{\ell+2} h_i^{c_i}$ and $< (b_1, \ldots, b_{\ell+2}), ((c_1, \ldots, c_{\ell+2})) >= \mathsf{gprod}x^{\ell} + \mathsf{bl}x^{\ell+1} - 1$. In other words they run

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

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.

7 Security Proofs

We first prove the correctness of our argument schemes, i.e. that an honest prover always convinces an honest verifier.

7.1 Correctness

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

7.2 Zero-Knowledge

Theorem 7.2 (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.*

7.3 Soundness

Theorem 7.3 (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 $h_1, \ldots, h_{\ell+2}$ 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 1 obtain $A \in \mathbb{G}$. Compute $\alpha_1, \beta_1 = \mathsf{Hash}(M, A)$.

2. Using the grand-product extractor $\mathcal{X}_{\mathsf{gprod}}$, extract $d_1, \ldots, d_{\ell+2}$ such that $AM^{\alpha_1}(h_1 \ldots h_{\ell+2})^{\beta_1} = h_1^{d_1} \cdots h_{\ell+2}^{d_{\ell+2}}$. Sample $\alpha_2, \beta_2 \xleftarrow{\$} \mathbb{F}$ and program $\mathsf{Hash}(M, A) = (\alpha_2, \beta_2)$. 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 $AM^{\alpha_1}(h_1 \ldots h_{\ell+2})^{\beta_1} = h_1^{d'_1} \cdots h_{\ell+2}^{d'_{\ell+2}}$. Abort if $\mathcal{X}_{\mathsf{gprod}}$ fails.

3. Set $m_1, \ldots, m_{\ell+2} = \frac{(d_1 - \beta_1) - (d'_1 - \beta_2)}{\alpha_1 - \alpha_2}, \ldots, \frac{(d_{\ell+2} - \beta_1) - (d'_{\ell+2} - \beta_2)}{\alpha_1 - \alpha_2}$ and $c_1, \ldots, c_{\ell+2} = (d_1 - m_1\alpha_1, \ldots, d_{\ell+2} - m_{\ell+2}\alpha_{\ell+2})$. Abort if $A \neq \prod_{i=1}^{\ell+2} h_i^{c_i}$ or $M \neq \prod_{i=1}^{\ell+2} h_i^{m_i}$.

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

5. Using the grand-product extractor $\mathcal{X}_{\mathsf{gprod}}$, extract $d_1, \ldots, d_{\ell+2}$ such that $AM^{\alpha_3}(h_1 \ldots h_{\ell+2})^{\beta_3} = h_1^{d_1} \cdots h_{\ell+2}^{d_{\ell+2}}$ 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 $(d_1 + \alpha m_1 + \beta, \ldots, d_{\ell+2} + \alpha m_{\ell+2} + \alpha) \neq (c_1, \ldots, c_{\ell+2})$ 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 such that $T = (\prod_{i=1}^{\ell} R_i^{a_i})^r$ and $U = (\prod_{i=1}^{\ell} S_i^{a_i})^r$. Abort if $\mathcal{X}_{\mathsf{sameexp}}$ fails.

8. Using the multi exponentiation extractor $\mathcal{X}_{\mathsf{multiexp}}$, extract $(d_1, \ldots, d_{\ell+2})$ such that $A = h_1^{d_1} \cdots h_{\ell+2}^{d_{\ell+2}}$, $T = \prod_{i=1}^{\ell} T_i^{d_i}$ and $U = \prod_{i=1}^{\ell} U_i^{d_i}$. Abort if $\mathcal{X}_{\mathsf{multiexp}}$ fails. If $(d_1, \ldots, d_{\ell+2}) \neq (c_1, \ldots, c_{\ell+2})$ return the discrete log relation $(d_1 + \alpha, \ldots, d_{\ell+2} + \alpha), (c_1, \ldots, c_{\ell+2})$.

9. If (σ, r) is such that $(T_i, U_i) = (R_{\sigma(i)}^r, S_{\sigma(i)}^r)$ for all $1 \leq i \leq \ell + 2$ 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,

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

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 $(c_1, \ldots, c_{\ell+2}) = (a_{\sigma(1)}, \ldots, a_{\sigma(\ell)}, a_{\ell+1}, a_{\ell+2})$ we have that

$$\prod_{i=1}^{\ell} T_i^{a_{\sigma(i)}} = \prod_{i=1}^{\ell} R_i^{ra_i} \ \wedge \ \prod_{i=1}^{\ell} U_i^{a_{\sigma(i)}} = \prod_{i=1}^{\ell} S_i^{ra_i}.$$

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, this happens with negligible probability. □

References

[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.