A Shuffle Argument Protocol Specification
Work in Progress

The Ethereum Foundation Research Team

July 14, 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

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

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

2 Problem Statement

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

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

a shuffler computes a second set of El-Gamal ciphertexts

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

and proves in zero knowledge that there exists a permutation

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

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

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

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

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

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

3 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

3.2 Group and Field

3.3 Randbelow

4 Construction

We begin by giving a full overview of the construction in Figures 1 and 2 which is proven secure in Theorem **??**. 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 $(a_1, \ldots, a_{\ell+2})$ such that $A = \mathsf{compute\_multiexp}(\mathsf{crs}_h, (a_1, \ldots, a_{\ell+2}))$ and $\mathsf{gprod} = \prod_{i=1}^{\ell} a_i$. In other words

$$R_{\mathsf{gprod}} = \left\{ \ (A_1, \mathsf{gprod}), (a_1, \ldots, a_{\ell+2}) \ \middle| \ A_1 = h_1^{a_1} \cdots h_n^{a_{\ell+2}} \wedge \mathsf{gprod} = a_1 \cdots a_\ell \ \right\}$$

4.2 Same Exponent Relation

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

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

4.3 Multiexponentiation Relation

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

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

4.4 Overview

Step 1: The prover sends $(T_1, U_1), \ldots, (T_\ell, U_\ell))$ to the verifier.

Step 2: The verifier samples $a_1, \ldots, a_\ell \xleftarrow{\$} \mathbb{F}$ randomly from $\mathbb{F}$ and returns these values to the prover.

Step 3: The prover samples $a_{\ell+1}, \ldots, a_n \xleftarrow{\$} \mathbb{F}$ randomly from $\mathbb{F}$ as $n - \ell$ masking values. The prover sets

$$(b_1, \ldots, b_n) = (a_{\sigma(1)}, \ldots, a_{\sigma(\ell)}, a_{\ell+1}, \ldots, a_n)$$

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

Step 1:

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

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

Step 2:

$a_{\ell+1}, a_{\ell+1} \leftarrow \mathsf{randbelow}(p)$

$A \leftarrow \mathsf{compute\_multiexp}(\mathsf{crs}_h, (a_{\sigma(1)}, \ldots, a_{\sigma(\ell)}, a_{\ell+1}, a_{\ell+2}))$

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

Step 3:

$\mathsf{gprod} \leftarrow (a_1 - \alpha)(a_2 - \alpha) \cdots (a_\ell - \alpha) \mod p$

$A_1 \leftarrow A(h_1 \ldots h_n)^{-\alpha}$

$\pi_{\mathsf{gprod}} \leftarrow \mathsf{Prove}(R_{\mathsf{gprod}}, (A_1, \mathsf{gprod}), (a_{\sigma(1)}, \ldots, a_{\sigma(\ell)}, a_{\ell+1}, a_{\ell+2}))$

Step 4:

$R \leftarrow \mathsf{compute\_multiexp}((R_1, \ldots, R_\ell), (a_1, \ldots, a_\ell))$

$S \leftarrow \mathsf{compute\_multiexp}((S_1, \ldots, S_\ell), (a_1, \ldots, a_\ell))$

$T \leftarrow R^r$

$U \leftarrow S^r$

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

Step 5:

$\pi_{\mathsf{multiexp}} \leftarrow \mathsf{Prove}(R_{\mathsf{multiexp}}, ((T_1, U_1, \ldots, T_\ell, U_\ell), A, (T, U)), (a_{\sigma(1)}, \ldots, a_{\sigma(\ell)}, a_{\ell+1}, a_{\ell+2}))$

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

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

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

Step 3:
$\mathsf{gprod} \leftarrow (a_1 - \alpha)(a_2 - \alpha) \cdots (a_\ell - \alpha) \mod p$
$A_1 \leftarrow A(h_1 \ldots h_n)^{-\alpha}$
$b_1 \leftarrow \mathsf{Verify}(R_{\mathsf{gprod}}, (A_1, \mathsf{gprod}), \pi_{\mathsf{gprod}})$

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

Step 5:
$b_3 \leftarrow \mathsf{Verify}(R_{\mathsf{multiexp}}, ((T_1, U_1, \ldots, T_\ell, U_\ell), A, (T, U)), \pi_{\mathsf{multiexp}})$
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.

and computes

$$A = h_1^{b_1} \dots h_n^{b_n}$$

and sends A to the verifier.

Step 4: The verifier samples $\alpha \xleftarrow{\$} \mathbb{F}$ randomly from $\mathbb{F}$. The verifier sets $\mathsf{gprod} = \prod_{i=0}^{\ell}(a_i - \alpha)$ and

$$A_1 = A(h_1 \dots h_n)^{-\alpha}$$

and returns α to the prover.

Step 5: The prover runs a zero-knowledge grand-product argument that multiplying the first ℓ elements of A_1 together yields gprod. In other words the prover demonstrates knowledge of the witness $(b_1 - \alpha, \dots, b_n - \alpha)$ such that $(A_1, \mathsf{gprod}) \in L_{\mathsf{gprod}}$ where

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

The prover returns the grand-product proof to the verifier.

Step 6: The verifier checks that the provers grandproduct argument verifies for (A_1, gprod) and returns 0 if not. The verifier computes

$$R = \prod_{i=1}^{\ell} R_i^{a_i}$$

and

$$S = \prod_{i=1}^{\ell} S_i^{a_i}$$

Step 7: The prover computes $T = R^r$ and $U = S^r$ and runs a zero-knowledge same exponent argument that know r such that $T = R^r$ and $U = S^r$. In other words they prove in zero-knowledge that $(R, S, T, U) \in L_{\mathsf{sameexp}}$ where

$$R_{\mathsf{sameexp}} = \left\{ \ (g_1, g_2, y_1, y_2), \mathsf{exp} \ \middle| \ y_1 = g_1^{\mathsf{exp}} \ \wedge \ y_2 = g_2^{\mathsf{exp}} \ \right\}$$

This can be achieved with a sigma protocol. The prover returns T, U and the same-exp proof to the verifier.

Step 8: The verifier checks that the provers same-exp argument verifies for (R, S, T, U) and returns 0 if not.

Step 9: The prover runs a zero-knowledge multi-exponentiation argument that

$$T = \prod_{i=1}^{\ell} T_i^{b_i} \ \wedge \ U = \prod_{i=1}^{\ell} U_i^{b_i}$$

for $(b_1, \dots, b_n)$ the discrete log of A. In other words they prove in zero-knowledge that they know $(b_1, \dots, b_n)$ such that $((T_1, \dots, T_\ell, U_1, \dots, U_\ell), A, T, U) \in L_{\mathsf{multiexp}}$ where

$$R_{\mathsf{multiexp}} = \left\{ \ ((T_1, \dots, T_\ell, U_1, \dots, U_\ell), A, T, U), (b_1, \dots, b_n) \ \middle| \ A = \textstyle\prod_{i=1}^{n} h_i^{b_i} \ \wedge \ T = \prod_{i=1}^{\ell} T_i^{b_i} \ \wedge U = \prod_{i=1}^{\ell} U_i^{b_i} \ \right\}$$

The prover returns the multi-exp proof to the verifier.

Step 10: The verifier checks that the provers same-exp argument verifies for (R, S, T, U) and returns 0 if not.

4.5 Grand Product Argument

We require a zero knowledge argument for the relation

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

Step 1: The prover first selects $n - \ell$ random values $b_{\ell+1}, \ldots, b_n \xleftarrow{\$} \mathbb{F}$ from the field. The prover sets

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

and computes

$$B = \prod_{i=1}^{n} g_i^{b_i} \ \wedge \mathsf{bl} = \sum_{i=\ell+1}^{n} a_i b_i.$$

The prover returns (B, bl) to the verifier.

Step 2: The verifier samples $x \xleftarrow{\$} \mathbb{F}$ randomly from the field. They compute the commitment

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

The verifier resets the generators

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

The verifier returns x to the prover.

Observe now that C is a commitment to

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

under the rescaled generators. Observe further that

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

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

$$\begin{array}{ll} -b_1 = -1 & \Rightarrow b_1 = 1 \\ b_1a_2 - b_2 = 0 & \Rightarrow a_2 = b_2 \\ b_2a_3 - b_3 = 0 & \Rightarrow b_3 = a_2a_3 \\ \quad \vdots & \\ b_{\ell-1}a_\ell - b_\ell = 0 & \Rightarrow b_\ell = a_2 \cdots a_\ell \\ b_\ell a_1 = \mathsf{gprod} & \Rightarrow \mathsf{gprod} = a_1 \cdots a_\ell \\ b_{\ell+1}a_{\ell+1} + \ldots + b_n a_n = \mathsf{bl} & \Rightarrow \text{N/A} \end{array}$$

Step 3: The prover will shortly run an inner product argument and they do not want to reveal any information in the process. Thus the prover samples $(r_1, \ldots, r_n) \xleftarrow{\$} \mathbb{F}$ and sets

$$\mathsf{bl}_2 =< (b_1, \ldots, b_n), (r_1, \ldots, r_n) >, \wedge R = h_1^{r_1} \ldots h_n^{r_n}.$$

The prover sends (R, bl_2) to the verifier.

Step 5: The verifier samples $y \xleftarrow{\$} \mathbb{F}$. They set $C = CR^y$ and return c to the prover.

Step 6: The prover sets

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

The prover runs an inner product argument like the one in bulletproofs to demonstrate that the inner product of B with C is equal to $\mathsf{gprod} x^\ell + \mathsf{bl} x^{\ell+1} - 1 + \mathsf{bl}_2 y$.

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.