

MPC for Group Reconstruction Circuits

Lúcas Críostóir Meier

June 9, 2022

Abstract

In this paper, we present a thing.

1 Introduction

Write the introduction

2 Background

Throughout this paper, we let \mathbb{G} denote a group of prime order q , with generators G and H . Let \mathbb{F}_q denote the scalar field associated with this group, and let $\mathbb{Z}/(q)$ denote the additive group of elements in this field.

We make heavy use of group homomorphisms throughout this paper. We let

$$\varphi(P_1, \dots, P_m) : \mathbb{A} \rightarrow \mathbb{B}$$

denote a homomorphism from \mathbb{A} to \mathbb{B} , parameterized by some public values P_1, \dots, P_m . Commonly \mathbb{A} will be a product of several groups $\mathbb{G}_1, \dots, \mathbb{G}_n$, in which case we'd write:

$$\varphi(P_1, \dots, P_m)(x_1, \dots, x_n)$$

to denote the application of φ to an element (x_1, \dots, x_n) of the product group. The public values P_i are often left implicit.

We often write products (x_1, \dots, x_n) as a single vector $\mathbf{x} \in \mathbb{A}^n$. Operations between these vectors are done element-wise, so we write $\mathbf{x} + \mathbf{y}$ for $(x_1 + y_1, \dots, x_n + y_n)$, as well as $\mathbf{x} \cdot G$ for $(x_1 \cdot G, \dots, x_n \cdot G)$.

2.1 Pedersen Commitments

2.2 Sigma Protocols

2.3 Maurer's φ -Proof

In [Mau09], Maurer generalized Schnorr's sigma protocol for knowledge of the discrete logarithm [Sch90] to a much larger class of relations. In

particular, Maurer provided a sigma protocol for proving knowledge of the pre-image of a group homomorphism φ . We denote this protocol as a “ φ -proof”, and recapitulate the scheme here.

Given a homomorphism $\varphi : \mathbb{A} \rightarrow \mathbb{B}$, and a public value $X \in \mathbb{B}$, the prover wants to demonstrate knowledge of a private value $x \in \mathbb{A}$ such that $\varphi(x) = X$. The prover does this by means of Protocol 2.1:

Protocol 2.1: φ-Proof	
Prover	Verifier
knows $x \in \mathbb{A}$	public $X \in \mathbb{B}$
$k \xleftarrow{R} \mathbb{A}$	
$K \leftarrow \varphi(k)$	
	\xrightarrow{K}
	$c \xleftarrow{R} \mathbb{Z}/(p)$
	\xleftarrow{c}
$s \leftarrow k + c \cdot x$	
	\xrightarrow{s}
	$\varphi(s) \stackrel{?}{=} K + c \cdot X$

Here, p is chosen such that $\forall B \in \mathbb{B}. p \cdot B = 0$. In practice, we'll set $p = q$, which will work perfectly for the groups we use, which are all products of \mathbb{G} or $\mathbb{Z}/(q)$.

Claim 2.1. Protocol 2.1 is a valid sigma protocol.

Completeness follows directly from the fact that φ is a homomorphism.

For the HVZK property, the simulator $\mathcal{S}(X, c)$ works by generating a random $s \xleftarrow{R} \mathbb{A}$, and then setting $K := \varphi(s) - c \cdot X$.

Finally, we prove 2-extractability. Given two verifying transcripts (K, c, s) and (K, c', s') sharing the first message, we extract a value \hat{x} satisfying $\varphi(\hat{x}) = X$ as follows:

$$\begin{aligned}
\varphi(s) - c \cdot X &= K = \varphi(s') - c' \cdot X \\
\varphi(s) - \varphi(s') &= c \cdot X - c' \cdot X \\
\frac{1}{c - c'} \cdot \varphi(s - s') &= X \\
\varphi\left(\frac{s - s'}{c - c'}\right) &= X
\end{aligned}$$

Thus, defining $\hat{x} := (s - s')/(c - c')$, we successfully extract a valid pre-image.

We conclude that the protocol is a valid sigma protocol.

■

Maurer's protocol can also work even in the case where the order of the groups are not known, but this makes the challenge generation a bit more complicated, and we don't need this functionality in this work.

2.4 UC Security and the Hybrid Model

2.5 Ideal Functionalities for Sigma Protocols

Functionality 2.1: Zero-Knowledge Functionality $\mathcal{F}(\text{ZK}, \varphi)$

A functionality \mathcal{F} for parties P_1, \dots, P_n .

On input (**prove**, **sid**, x) from P_i :

\mathcal{F} checks that **sid** has not been used by P_i before.

\mathcal{F} generates a new token π , and sets $x_\pi \leftarrow x$.

\mathcal{F} replies with (**proof**, π).

On input (**verify**, X , π):

\mathcal{F} replies with (**verify-result**, $\varphi(x_\pi) \stackrel{?}{=} X$).

2.6 Broadcast Functionalities

Functionality 2.2: Authenticated Broadcast Functionality \mathcal{C}

A functionality \mathcal{C} for parties P_1, \dots, P_n .

On receiving $(\text{broadcast-in}, \text{sid}, m)$ from P_i :
 \mathcal{C} checks that sid has not been used by P_i before.
 \mathcal{C} sends $(\text{broadcast-out}, \text{pid}_i, \text{sid}, m)$ to every party P_j .

3 Group Reconstruction Circuits

3.1 Formal Definition

3.2 Normalized Form

4 MPC Protocol for GRCs

4.1 Ideal Functionality

Functionality 4.1: GRC functionality $\mathcal{F}(\text{GRC}, \Phi, \mathbf{X}^j, \mathbf{Y}^j)$

A functionality \mathcal{F} for parties P_1, \dots, P_n .

After receiving $(\text{input}, \text{sid}, \mathbf{x}^j, \mathbf{y}^j, \boldsymbol{\alpha}^j, \mathbf{k}^j)$ from every party P_j :
 \mathcal{F} checks, for every $j \in [n]$, that:

$$\mathbf{X}^j \stackrel{?}{=} \mathbf{x}^j \cdot G$$

$$\mathbf{Y}^j \stackrel{?}{=} \mathbf{y}^j \cdot G + \boldsymbol{\alpha}^j \cdot H$$

\mathcal{F} computes, for each round $r \in [d]$:

$$\mathbf{V}_r^j := \varphi_r(\mathbf{V}_1, \dots, \mathbf{V}_{r-1})(\mathbf{x}^j, \mathbf{y}^j, \mathbf{k}^j)$$

$$\mathbf{V}_r := \sum_j \mathbf{V}_r^j$$

\mathcal{F} sends $(\text{output}, \text{sid}, \mathbf{V}_1^1, \dots, \mathbf{V}_d^n)$ to every party P_j .

4.2 Protocol

$$\psi_r(\mathbf{x}, \mathbf{y}, \boldsymbol{\alpha}, \mathbf{k}, \boldsymbol{\beta}) := (\varphi_r(\mathbf{x}, \mathbf{y}, \mathbf{j}), \mathbf{x} \cdot G, \text{Commit}(\mathbf{y}, \boldsymbol{\alpha}), \text{Commit}(\mathbf{k}, \boldsymbol{\beta}))$$

Protocol 4.1: MPC protocol for $\Phi, \mathbf{X}^j, \mathbf{Y}^j$

Each party P_j has inputs \mathbf{x}^j and \mathbf{y}^j , committed to by \mathbf{X}^j and \mathbf{Y}^j . They also have decommitments α^j for \mathbf{Y}^j . Each party P_j also has a vector \mathbf{k}^j , which honest parties will have generated randomly.

Round 0

Each party P_j generates a random vector β^j , and creates a commitment to \mathbf{k}^j with:

$$\mathbf{K}^j := \text{Commit}(\mathbf{k}^j, \beta^j)$$

P_j sends (**broadcast-in**, **sid**, \mathbf{K}^j) to the broadcast functionality \mathcal{C} .

P_j waits to receive (**broadcast-out**, **sid**, \mathbf{K}^i) for each other party i .

Round r

Each party P_j computes $\mathbf{V}_r^j := \varphi_r(\mathbf{V}_1, \dots, \mathbf{V}_{r-1})(\mathbf{x}^j, \mathbf{y}^j, \mathbf{k}^j)$.

Each party P_j sends (**prove**, **sid**, $(\mathbf{x}^j, \mathbf{y}^j, \alpha^j, \mathbf{k}^j, \beta^j)$) to $\mathcal{F}(\text{ZK}, \psi_r)$, receiving π_r^j in return.

Each party P_j sends $(\mathbf{V}_r^j, \pi_r^j)$ to every other party.

After receiving $(\mathbf{V}_r^i, \pi_r^i)$ from all other parties, P_j checks, for each i , that the proof is valid, by sending (**verify**, $(\mathbf{V}_r^i, \mathbf{X}^i, \mathbf{Y}^i, \mathbf{K}^i)$, π_r^i) to $\mathcal{F}(\text{ZK}, \psi_r)$, and aborting if the functionality returns 0.

Each party P_j then stores each \mathbf{V}_r^i as part of its output.

Claim 4.1. Provided that the discrete logarithm is hard in \mathbb{G} , Protocol 4.1 securely implements Functionality 4.1, in the hybrid model of universally composable security, given a zk functionality $\mathcal{F}(\text{ZK}, \varphi)$ (for arbitrary φ), a broadcast functionality \mathcal{C} , as well as a common reference string $(G, H) \in \mathbb{G}^2$.

Proof:

We prove this by constructing a simulator \mathcal{S} which uses the ideal functionality $\mathcal{F}(\text{GRC})$ to perfectly simulate an execution of the hybrid protocol against an adversary \mathcal{A} .

We also work in the common reference string model, where the simulator \mathcal{S} chooses the bases (G, H) for the Pedersen commitments.

We use this simulator \mathcal{S} to construct an adversary against the discrete logarithm game.

Let $\mathcal{M} \subseteq \mathcal{P}$ be the set of malicious parties, and $\mathcal{H} \subseteq \mathcal{P}$ be the set of honest parties. Naturally, we have $\mathcal{H} \cup \mathcal{M} = \mathcal{P}$ and $\mathcal{H} \cap \mathcal{M} = \emptyset$.

As an adversary against the discrete logarithm game, \mathcal{S} receives (G, H) as an instance of the discrete logarithm problem.

The simulator then proceeds as follows:

\mathcal{S} starts by setting (G, H) as the common reference string.

Round 0:

For each $j \in \mathcal{H}$, \mathcal{S} samples $\mathbf{K}^j \xleftarrow{R} \mathbb{G}$.

For each $j \in \mathcal{M}$, \mathcal{S} waits to receive $(\text{broadcast-in}, \text{sid}, \mathbf{K}^j)$.

\mathcal{S} then sends $(\text{broadcast-out}, \text{pid}_j, \text{sid}, \mathbf{K}^j)$, to all parties, for every $j \in \mathcal{P}$, emulating \mathcal{C} .

Interim:

\mathcal{S} waits to receive $(\text{prove}, \text{sid}, (\mathbf{x}^j, \mathbf{y}^j, \boldsymbol{\alpha}^j, \mathbf{k}^j, \boldsymbol{\beta}^j))$ for each malicious $j \in \mathcal{M}$, playing the role of $\mathcal{F}(\text{ZK}, \psi_1)$.

\mathcal{S} checks, for each j , that:

$$\mathbf{X}^j \stackrel{?}{=} \mathbf{x}^j \cdot G$$

$$\mathbf{Y}^j \stackrel{?}{=} \mathbf{y}^j \cdot G + \boldsymbol{\alpha}^j \cdot H$$

$$\mathbf{K}^j \stackrel{?}{=} \mathbf{k}^j \cdot G + \boldsymbol{\beta}^j \cdot H$$

otherwise, \mathcal{S} sets $\text{bad-values}_1^j \leftarrow 1$.

\mathcal{S} records the values $\mathbf{x}^j, \mathbf{y}^j, \boldsymbol{\alpha}^j, \mathbf{k}^j, \boldsymbol{\beta}^j$, for $j \in \mathcal{M}$.

Now, in the real execution against $\mathcal{F}(\text{GRC})$, with real honest parties P_i , for each $j \in \mathcal{M}$, the parties \mathcal{S} controls, \mathcal{S} sends $(\text{input}, \text{sid}, \mathbf{x}^j, \mathbf{y}^j, \boldsymbol{\alpha}^j, \mathbf{k}^j)$ to $\mathcal{F}(\text{GRC})$.

\mathcal{S} receives $(\text{output}, \text{sid}, \mathbf{V}_1^1, \dots, \mathbf{V}_d^n)$ in return, and records these values.

Round r :

For each round $r \in [d]$, \mathcal{S} proceeds as follows:

\mathcal{S} generates a new π_r^i for each $i \in \mathcal{H}$, and sends $(\mathbf{V}_r^i, \pi_r^i)$ to every malicious P_j , with $j \in \mathcal{M}$.

Unless $r = 1$, \mathcal{S} waits to receive $(\text{prove}, \text{sid}, \hat{\mathbf{x}}^j, \hat{\mathbf{y}}^j, \hat{\boldsymbol{\alpha}}^j, \hat{\mathbf{k}}^j, \hat{\boldsymbol{\beta}}^j)$ from each malicious P_j , for $j \in \mathcal{M}$, playing the role of $\mathcal{F}(\text{ZK}, \psi_r)$.

\mathcal{S} then checks, for each j , that:

$$\mathbf{X}^j \stackrel{?}{=} \hat{\mathbf{x}}^j \cdot G$$

$$\mathbf{Y}^j \stackrel{?}{=} \hat{\mathbf{y}}^j \cdot G + \hat{\boldsymbol{\alpha}}^j \cdot H$$

$$\mathbf{K}^j \stackrel{?}{=} \hat{\mathbf{k}}^j \cdot G + \hat{\boldsymbol{\beta}}^j \cdot H$$

and sets $\text{bad-values}_r^j \leftarrow 1$ otherwise.

The first check implies that $\hat{\mathbf{x}}^j = \mathbf{x}^j$. If it holds that $\hat{\mathbf{y}}^j \neq \mathbf{y}^j$ or $\hat{\mathbf{k}}^j \neq \mathbf{k}^j$,

then \mathcal{S} has found a value h such that $h \cdot G = H$, as shown in [reference previous section](#), and \mathcal{S} aborts, returning h .

(Including when $r = 1$) \mathcal{S} generates a new π_r^j , and returns (proof, π_r^j) , playing the role of $\mathcal{F}(\text{ZK}, \psi_r)$.

Concurrently, \mathcal{S} plays the role of $\mathcal{F}(\text{ZK}, \psi_r)$, responding to $(\text{verify}, (\hat{\mathbf{V}}_r^i, \hat{\mathbf{X}}^i, \hat{\mathbf{Y}}^i, \hat{\mathbf{K}}^i), \pi)$ queries. \mathcal{S} checks that there exists some $j \in \mathcal{P}$ such that $\pi_r^j = \pi$. \mathcal{S} then returns:

$$\hat{\mathbf{V}}_r^i \stackrel{?}{=} \mathbf{V}_r^i \wedge \hat{\mathbf{X}}^i \stackrel{?}{=} \mathbf{X}^i \wedge \hat{\mathbf{Y}}^i \stackrel{?}{=} \mathbf{Y}^i \wedge \hat{\mathbf{K}}^i \stackrel{?}{=} \mathbf{K}^i \wedge \text{bad-values}_r^j \neq 1$$

\mathcal{S} then waits to receive $(\hat{\mathbf{V}}^j, \hat{\pi}_r^j)$ for every malicious party P_j , with $j \in \mathcal{M}$. \mathcal{S} then checks if the query $(\text{verify}, (\hat{\mathbf{V}}_r^j, \mathbf{X}^j, \mathbf{Y}^j, \mathbf{K}^j), \hat{\pi}_r^j)$ would yield 1, according to the logic in the section above. (If $\hat{\pi}_r^j$ doesn't match anything, the check is considered to fail). If this check fails, then \mathcal{S} simulates every honest P_i aborting, with $i \in \mathcal{H}$, to abort, as if they'd seen an invalid proof themselves.

This concludes the simulation.

If \mathcal{S} aborts with a value h , then they've successfully solved an instance of the discrete logarithm problem. Under our assumption that this problem is hard, this happens with negligible probability.

We argue that if \mathcal{S} does not abort in this way, then the simulation is perfect. For the first round, because pedersen commitments are perfectly hiding, sampling a random \mathbf{K}^j has an identical distribution as an honest party generating a pedersen commitment. For the rest of the protocol, all of our checks are equivalent to those made by honest parties. This is because the \mathbf{V}_j^i values are necessarily computed correctly, and use the inputs provided by the parties the adversary \mathcal{A} controls.

Because our simulator \mathcal{S} is perfect, and doesn't rewind the adversary \mathcal{A} , we conclude that our protocol satisfies universally composable security, in the hybrid model.

□

4.3 Security Analysis

4.4 Practical Considerations

5 Applications

6 Limitations and Further Work

7 Conclusion

References

- [Mau09] Ueli Maurer. Unifying Zero-Knowledge Proofs of Knowledge. In *AFRICACRYPT 2009*, volume 5580 of *LNCS*, pages 272–286. Springer, Berlin, Heidelberg, 2009.
- [Sch90] C. P. Schnorr. Efficient Identification and Signatures for Smart Cards. In *CRYPTO 1989*, volume 435 of *LNCS*, pages 239–252, New York, NY, 1990. Springer.