Sigmabus: Binding Sigmas in Circuits for Fast Curve Operations

George Kadianakis[1], Mary Maller[1,2], and Andrija Novakovic[3]

[1]Ethereum Foundation
[2]PQShield
[3]Geometry

asn@ethereum.org
mary.maller@ethereum.org
andrija@geometry.xyz

September 25, 2023

Abstract

This paper introduces Sigmabus, a technique designed to enhance the efficiency of zero-knowledge circuits by relocating computationally expensive operations outside the circuit. Specifically, Sigmabus focuses on moving elliptic curve group operations, typically proven with expensive non-native field arithmetic, to external computations. By leveraging Sigma protocols, elliptic curve group operations are proven outside the circuit, while additional constraints are applied to the circuit to ensure correct execution of the Sigma protocol. This approach can achieve significant performance improvements in zero-knowledge circuits. This paper presents the Sigmabus protocol along with its security proofs, and demonstrates its practical implications through various use cases.

1 Introduction

Modern zero-knowledge protocols encounter significant prover computational overhead due to the execution of elliptic curve group operations within SNARK circuits. These operations are required for recursion, aggregating proofs or verifying signatures inside the circuit. While it is possible to perform these elliptic curve group operations using non-native field arithmetic, such an approach results in field operations that are orders of magnitude slower than native field operations. To address this inefficiency, researchers have proposed various optimizations, including curve chains, curve cycles, and deferred computations in recursive settings [Wil].

In this work we aim to leverage algebraic hash functions to greatly improve efficiency for group operations. Algebraic hash functions were developed because traditional hash functions such SHA2 and SHA3 result in large prover computational overhead - considerably more so than elliptic curve group operations. Researchers and developers have invested time and resources over many years in avoiding traditional hash operations. In particular, algebraic hash functions such as MIMC, Poseidon, Rescue, and Reinforced Concrete were suggested as a cheaper alternative. Designed with the goal of being efficient to compute inside a SNARK circuit, algebraic hash functions are now so efficient that they are not only cheaper than their non-algebraic predecessors, they are also orders

of magnitude cheaper than group operations. See Table 1 for a comparison of these costs.

This paper introduces Sigmabus, a novel technique that allows circuit designers to relocate elliptic curve group operations outside the circuit. The approach involves performing a Sigma protocol [Sch91] outside the circuit and subsequently "binding" it inside the circuit. However, the message in the first round of the Sigma protocol is computed using a hash function rather than a group operation, making it cheap to bind inside the circuit. During verification, the verifier validates both the Sigma transcript and the SNARK proof, ensuring the integrity of the proof and the correctness of the relocated operations. Sigmabus is provably secure in the random oracle model assuming a binding commitment scheme and that discrete log is hard.

Sigmabus also supports arbitrary linear elliptic curve operations and allows for amortization, enabling multiple operations from the circuit to be offloaded with a single Sigma protocol invocation.

We envision that Sigmabus can be used to improve the performance of circuits that aggregate proofs, or verify signatures. It can also be used to *link* algebraic with non-algebraic commitments and hence can serve as a connector between commit-and-prove protocols.

	Non-native scalar multiplication	Poseidon hash
R1CS	$100,000$ constraints	$1,000$ constraints
Plonkish	$14,000$ constraints	220 constraints

Table 1: Constraints required for performing operations in R1CS and Plonkish

1.1 Related Work

Chase et al. [CGM16] introduced a technique that combines algebraic-based proof protocols, such as Sigma protocols, with proofs based on garbled circuits. This integration efficiently handles algebraic operations in the former and non-algebraic operations (e.g. hash functions) in the latter. The linkage between these proof systems relies on using a *private* garbling scheme to compute a one-time MAC of the witness and then proving the correctness of the MAC using a Sigma protocol. However, this technique requires garbled circuits with privacy properties, as the verifier learning the MAC value directly reveals the witness. As a result, the approach is not immediately applicable to proof systems that do not employ private garbled circuits. In contrast, Sigmabus can be utilized by any circuit-based proof system without being restricted by the need for private garbled circuits.

LegoSNARK [CFQ19] introduced a generic framework for linking different proof systems. Using the commit-and-prove paradigm [Kil90] [CLOS02], it provides a framework and generic compiler to facilitate the generic integration of proof systems and demonstrates its applicability across various use cases. Sigmabus can be seen as providing an efficient specialized LegoSNARK link between Sigma protocols and generic SNARK protocols.

GoblinPlonk [Wil] introduces a mechanism for deferring expensive operations in SNARK circuits. For instance, when encountering an expensive operation $X = xG$, the prover defers the actual computation and directly provides the final result for X. Once multiple such operations have been deferred, a specialized circuit verifies the correctness of all deferred operations in a single step. Sigmabus can be effectively utilized in a GoblinPlonk final circuit to expedite the verification process. Furthermore, when integrating Sigmabus into a larger circuit, the prover can provide the final result for X and defer its actual computation by pushing it to a Sigma protocol, following a similar approach as in GoblinPlonk.

1.2 Document Structure

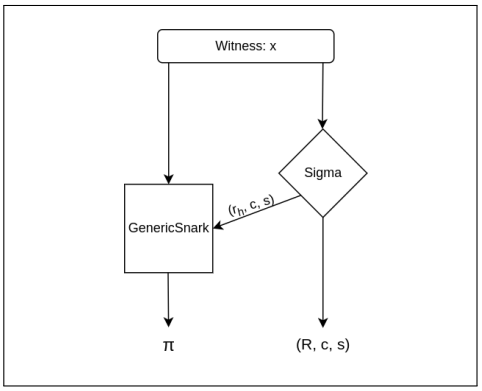
We start with Section 2, where we introduce the Sigmabus protocol by giving an informal overview of its functionality and security and follow up with Section 3 where we provide a formal description of Sigmabus. Next in Section 4, we present formal proofs for the scheme's computational knowledge soundness and honest-verifier zero-knowledge properties. In Section 5, we explore optimizations and possible extensions to enhance Sigmabus's performance within complex circuits. Moving on to Section 6, we explore various protocols where Sigmabus can be effectively utilized. We begin with Section 6.1, showcasing its application in a simple anonymous credential scheme. Next, in Section 6.2, we demonstrate its integration into a larger Plonkish proof system. Finally, Section 6.3 delves into the benefits of plugging Sigmabus into proof systems based on boolean circuits.

2 Technical Overview

In this section we provide an informal overview both of the Sigmabus protocol and the security reasoning.

2.1 Informal Protocol Overview

Suppose that $\mathsf{cm} = \mathsf{Commit}(x)$ is a binding commitment to x and that $X = xG$. The prover's goal is to prove that cm and X contain the same value x.
Recall in a sigma protocol that verifying transcripts have the form (R, c, s) where $c = \mathsf{Hash}(X, R)$ and $R + cX = sG$. Our idea is to modify the classic protocol to include an additional terms cm and $r_h = \mathsf{Commit}(r)$. Now transcripts have the form (R, r_h, c, s) where $c = \mathsf{Hash}(X, \mathsf{cm}, R, r_h)$ and $R + cX = sG$. A formal description is given in Figure 1 using the hash function Hash as a binding commitment. Any commitment scheme can be used as long as it's efficient to open inside GenZK.



Subsequently, the prover modifies the GenZK circuit to accept public inputs $(X, \mathsf{cm}, r_h, c, s)$ and witness (x, r). Finally, the prover enhances the GenZK circuit to verify the following constraints:

$$\mathsf{cm} = Com(x) \wedge r_h = H(r) \wedge s = r + cx$$

A valid proof requires both the Sigma protocol and the GenZK proof to be valid. Notably, the GenZK circuit avoids performing any elliptic curve group operations.

2.2 Informal Security Overview

In this section, we offer an informal intuition on why Sigmabus is sound, with formal security proofs presented in subsequent sections.

Recall that the prover's objective is to convince the verifier that $X = xG$ holds within the GenZK circuit. In the protocol described above, the prover performs a Sigma protocol outside the circuit to demonstrate knowledge of x such that $X = xG$. However, this does not suffice to convince the verifier that the prover is using the correct x within the circuit.

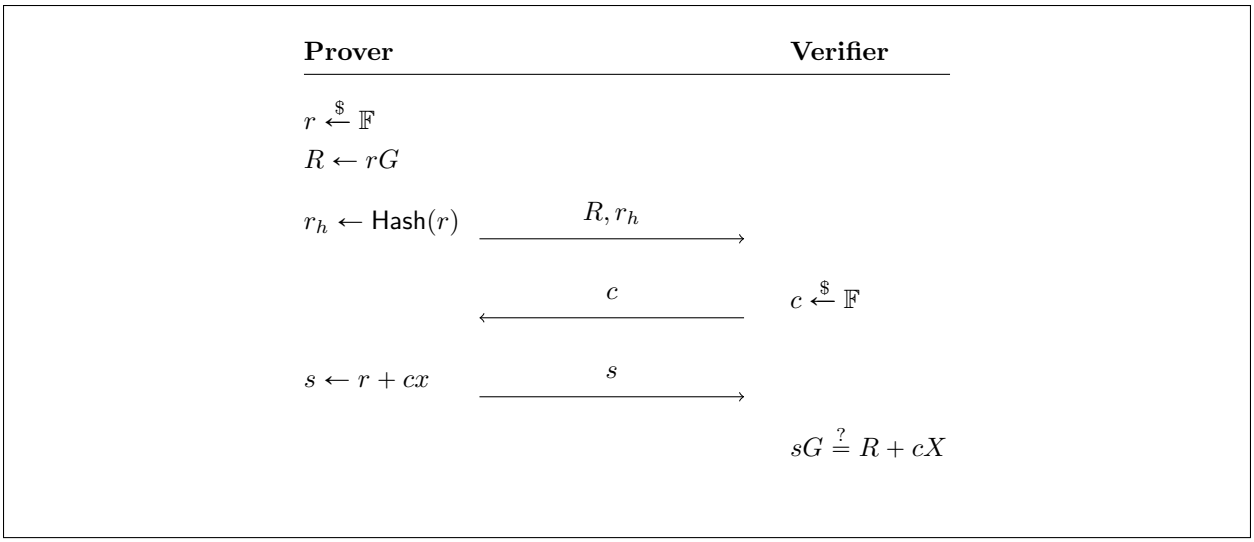


Figure 1: Modified Schnorr protocol using r_h proving $X = xG$

To address this limitation, the Sigma response value s is passed into the GenZK circuit as an instance and it is checked that $s = r' + cx'$ for $r_h = \mathsf{Commit}(r')$ and $\mathsf{cm} = \mathsf{Commit}(x')$. The probability that $r' + cx' = r + cx$ but $x \neq x'$ is negligible. Thus the verifier is convinced that the witness x is the discrete logarithm of X inside the circuit.

3 Sigmabus

Given a field element $x \in \mathbb{F}$ and a binding commitment cm to x, Sigmabus works over a generic SNARK proof system GenZK and provides an efficient way to prove the following relation:

$$\mathcal{R}_{\mathsf{sigmabus}} = \{(X, \mathsf{cm});\ (x)\ : \mathsf{cm} = \mathsf{Commit}(x) \wedge X = xG)\}$$

Sigmabus uses Sigma protocols to efficiently compute the scalar multiplication xG, and then *links the result* into GenZK without sacrificing soundness.

The commitment scheme Commit can be any binding commitment scheme, such as a SNARK-friendly hash function like Poseidon or a polynomial commitment scheme like KZG.

Sigmabus makes use of two subprotocols:

1. A modified Sigma protocol proving knowledge of x such that $X = xG$ with challenge $c = \mathsf{Hash}(X, \mathsf{cm}, R, r_h)$ and response s.

2. A SNARK subprotocol using the GenZK proof system, proving the relation $\mathcal{R}_{\mathsf{GenZK}}$ where:

   $$\mathcal{R}_{\mathsf{GenZK}} = \{(\mathsf{cm}, r_h, c, s); (x, r, o_h) : \mathsf{cm} = \mathsf{Commit}(x) \wedge r_h = \mathsf{HCommit}(r, o_h) \wedge s = r + cx\}$$

   where HCommit is a hiding commitment scheme.

We note that Sigmabus can prove arbitrary linear transformations, instead of just scalar multiplications, using a Sigma protocol that proves knowledge of a vector commited with a Pedersen commitment. For simplicity, in this report, we focus on scalar multiplications.

A formal description of Sigmabus is provided in Figure 2.

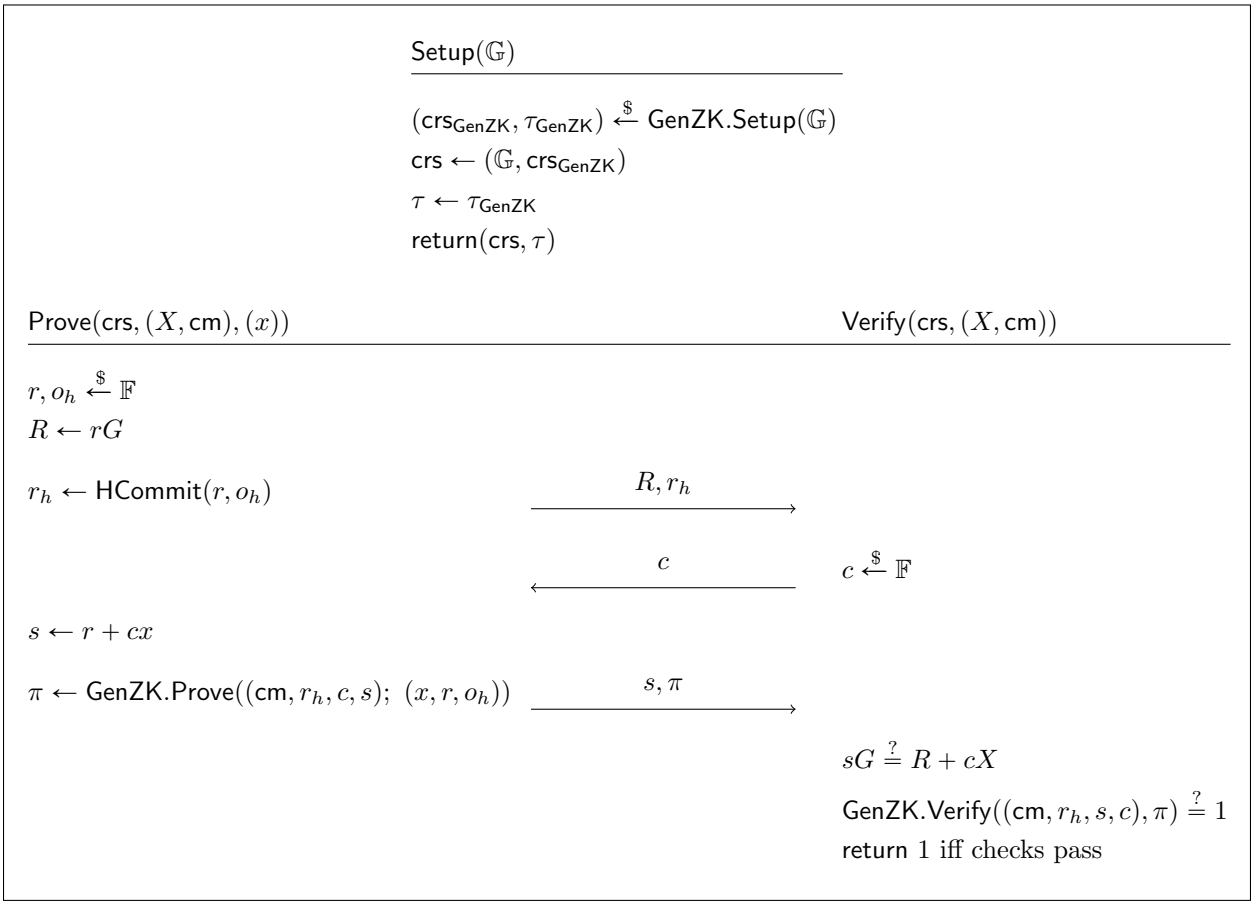


Figure 2: Protocol description of Sigmabus. Sigmabus proves that $(X, \mathsf{cm}) \in \mathcal{R}_{\mathsf{sigmabus}}$ and uses a zero-knowledge proving scheme for $\mathcal{R}_{\mathsf{GenZK}}$ as a subroutine.

4 Security

4.1 Computational Knowledge Soundness

Theorem 4.1 (Sigmabus argument is computational knowledge-sound)**.** *If GenZK is knowledge-sound, and the commitment scheme* Commit *is binding, then the Sigmabus argument described in Figure 2 is computational knowledge-sound.*

Proof. We design an extractor $\mathcal{X}_{\mathsf{sigmabus}}$ such that for any adversary $\mathcal{A}$ that convinces the verifier, the extractor with overwhelming probability returns x such that $X = xG$ and $\mathsf{cm} = \mathsf{Commit}(x)$. This implies

$$((X, \mathsf{cm}); (x)) \in \mathcal{R}_{\mathsf{sigmabus}}$$

Given a verifying Sigmabus transcript (R, r_h, c, s, π), we run the GenZK extractor on π recovering (x, r, o_h). The extractor $\mathcal{X}_{\mathsf{sigmabus}}$ outputs x as the witness. We now demonstrate that the extracted witness x is valid with high probability.

To demonstrate that x is a valid witness, we first account for the bad cases where our extractor can fail to return a valid witness. We do so by describing two reductions:

- Let $\mathcal{B}_1$ be a reduction against the binding property of the commitment scheme Commit. First, by continuously rewinding the adversary, $\mathcal{B}_1$ obtains two valid $\mathsf{Sigmabus}$ transcripts

$$(X, \mathsf{cm}), (R, r_h, c_0, s_0, \pi_0)$$
$$(X, \mathsf{cm}), (R, r_h, c_1, s_1, \pi_1)$$

  where $c_0 \neq c_1$. By the general forking lemma [BN06] $\mathcal{B}_1$ can obtain these two transcripts in polynomial time. Let $\mathcal{C}$ be the adversary that runs $\mathcal{A}$ to receive $(X, \mathsf{cm}), (R, r_h, c_0, s_0, \pi_0)$ and returns $(\mathsf{cm}, r_h, c, s), \pi$. Then $\mathcal{B}_1$ runs $\mathcal{C}$ with and then the GenZK extractor on π_b for $b = 0, 1$, recovering $(x_b, r_b, o_{h,b})$.

  Consider the bad event E_1 where $((R, r_h, c_0, s_0), (x_0, r_0, o_{h,0})) \notin \mathcal{R}_{\mathsf{GenZK}}$. This happens with probability $\mathsf{Adv}^{\mathsf{GenZK\text{-}ksound}}_{\mathcal{C},\mathsf{Ext}}(1^\lambda)$. Similar, the bad event E_2 where $((R, r_h, c_1, s_1), (x_1, r_1, o_{h,1})) \notin \mathcal{R}_{\mathsf{GenZK}}$ happens with probability $\mathsf{Adv}^{\mathsf{GenZK\text{-}ksound}}_{\mathcal{C},\mathsf{Ext}}(1^\lambda)$.

  When neither E_1 nor E_2 occur and the extracted $x_0, x_1, r_0, r_1, o_{h,0}, o_{h,1}$ satisfy:

$$\begin{aligned} \mathsf{Commit}(x_0) = \mathsf{Commit}(x_1) &= \mathsf{cm} \\ \mathsf{HCommit}(r_0, o_{h,0}) = \mathsf{HCommit}(r_1, o_{h,1}) &= r_h \\ r_0 &= s_0 - c_0 x_0 \pmod p \\ r_1 &= s_1 - c_1 x_1 \pmod p \end{aligned} \tag{1}$$

  If $x_0 \neq x_1$ then reduction $\mathcal{B}_1$ returns cm, x_0, x_1 thus breaks the binding of Commit. Else it aborts.

- Let $\mathcal{B}_2$ be a reduction against the binding property of the commitment scheme $\mathsf{HCommit}$. Then $\mathcal{B}_2$ behaves identically to B_1 except to extract $x_0, x_1, r_0, r_1, o_{h,0}, o_{h,1}$. Then if $r_0 \neq r_1$then reduction $\mathcal{B}_2$ returns $r_h, r_0, o_{h,0}, r_1, o_{h,1}$ thus breaks the binding of $\mathsf{HCommit}$.

Observe that with probability $1 - \mathsf{Adv}^{\mathsf{binding}}_{\mathcal{B}_1}(\lambda) - 2\mathsf{Adv}^{\mathsf{GenZK\text{-}ksound}}_{\mathcal{C},\mathsf{Ext}}(\lambda)$ the reduction $\mathcal{B}_1$ fails (and neither E_1 nor E_2 occur). With probability $1 - \mathsf{Adv}^{\mathsf{binding}}_{\mathcal{B}_2}(\lambda) - 2\mathsf{Adv}^{\mathsf{GenZK\text{-}ksound}}_{\mathcal{C},\mathsf{Ext}}(\lambda)$ the reduction $\mathcal{B}_2$ fails (and neither E_1 nor E_2 occur).

When both reductions fail, we have that $r_0 = r_1$ and $x_0 = x_1$, and from the equations in (1) we get:

$$s_0 - c_0 x_0 = s_1 - c_1 x_0 \pmod p$$

thus

$$x_0 = \frac{s_0 - s_1}{c_0 - c_1} \pmod p \tag{2}$$

Also, since the two transcripts are valid, we have from Schnorr that:

$$R = s_0 G - c_0 X = s_1 G - c_1 X$$

which means that

$$xG = \frac{s_0 - s_1}{c_0 - c_1} G$$

and hence

$$x = \frac{s_0 - s_1}{c_0 - c_1} \pmod{p} \tag{3}$$

By Equation (2) and Equation (3) we have that $x = x_0$ and that x_0 is the discrete logarithm of X. Hence the extracted witness x_0 satisfies $x_0 G = X$ and $\mathsf{Commit}(x_0) = \mathsf{cm}$ (from the GenZK soundness), it is a valid witness for the sigmabus relation. Thus

$$\mathsf{Adv}^{\text{sigmabus-ksound}}_{\mathcal{A},\mathsf{Ext}} \leqslant \mathsf{Adv}^{\text{Commit-binding}}_{\mathcal{B}_1}(\lambda) + \mathsf{Adv}^{\text{HCommit-binding}}_{\mathcal{B}_2}(\lambda) + 4\mathsf{Adv}^{\text{GenZK-ksound}}_{\mathcal{C},\mathsf{Ext}}(\lambda)$$

□

4.2 Computational Honest-Verifier Zero Knowledge

Theorem 4.2 (Sigmabus argument is computational honest-verifier zero-knowledge)**.** *If GenZK is zero-knowledge and if the commitment scheme* HCommit *is hiding then the Sigmabus argument described in Figure 2 is computationally honest-verifier zero-knowledge.*

Proof. We design a simulator that takes as input a common reference string, trapdoor, and instance, and outputs a simulated proof transcript where it can choose the verification responses. The simulator behaves as follows.

$$\begin{array}{l}\underline{\mathsf{Simulate}(\mathsf{crs}, \tau, (X, \mathsf{cm}))} \\ c, s, r, o_h \xleftarrow{\$} \mathbb{F} \\ R \leftarrow sG - cX \\ r_h \leftarrow \mathsf{HCommit}(r, o_h) \\ \pi \leftarrow \mathsf{GenZK.Simulate}(\mathsf{crs}_{\mathsf{GenZK}}, \tau_{\mathsf{GenZK}}, (\mathsf{cm}, r_h, c, s)) \\ \mathsf{return}(R, r_h, c, s, \pi)\end{array}$$

We must show that it is difficult for an adversary to distinguish transcripts output by Simulate from genuine transcripts generated by an honest prover and verifier.

Game 0: This is the standard zero-knowledge game with respect to the simulator Simulate.

$$\begin{array}{l}\underline{\mathsf{Game}^{\text{sigmabus-zk}}\ \text{MAIN}(\mathbb{G})} \\ b \xleftarrow{\$} \{0,1\} \\ (\mathsf{crs}, \tau) \xleftarrow{\$} \mathsf{Setup}(\mathbb{G}) \\ b' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_b}(\mathsf{crs}) \\ \text{return } b = b'\end{array}$$

$$\begin{array}{l}\underline{\mathcal{O}_0((X, \mathsf{cm}), (x))} \\ \mathsf{assert}\ ((X, \mathsf{cm}), x) \in \mathcal{R}_{\mathsf{sigmabus}} \\ \mathsf{return}\ \langle \mathsf{Prove}(\mathsf{crs}, (X, \mathsf{cm}), x), \mathsf{Verify}(\mathsf{crs}, (X, \mathsf{cm}))\rangle\end{array} \qquad \begin{array}{l}\underline{\mathcal{O}_1((X, \mathsf{cm}), (x))} \\ \mathsf{assert}\ ((X, \mathsf{cm}), x) \in \mathcal{R}_{\mathsf{sigmabus}} \\ \mathsf{return}\ \mathsf{Simulate}(\mathsf{crs}, \tau, (X, \mathsf{cm}))\end{array}$$

Game 1: The first game Game_1 behaves identically to Game_0 except that the simulation oracle generates r_h using the witness $s - cx$. See Figure 3. We define a reduction $\mathcal{B}_1$ against the hiding of HCommit and show that

$$\mathsf{Adv}^{\mathsf{Game}_0}_{\mathcal{A}} \leqslant 2\mathsf{Adv}^{\text{hiding}}_{\mathcal{B}_1} + \mathsf{Adv}^{\mathsf{Game}_1}_{\mathcal{A}}$$

The reduction $\mathcal{B}_1$ behaves as follows. It receives as input a commitment $\mathsf{HCommit}$.

Choose $b \xleftarrow{\$} \{0,1\}$, $(\mathsf{crs}, \tau) \xleftarrow{\$} \mathsf{Setup}(\mathbb{G})$, and run $b' \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_b}(\mathsf{crs})$. When $\mathcal{A}$ queries $\mathcal{O}_b((X, \mathsf{cm}), x)$ assert that $((X, \mathsf{cm}), x) \in \mathsf{sigmabus}$.

- If $b = 0$, return $\langle \mathsf{Prove}(\mathsf{crs}, (X, \mathsf{cm}), x), \mathsf{Verify}(\mathsf{crs}, (X, \mathsf{cm})) \rangle$.
- Else if $b = 1$, choose $c, r, s \xleftarrow{\$} \mathbb{F}$, $R \leftarrow sG - cX$. Query $\mathcal{O}_{\mathsf{HCommit}}(r, s - cx)$ to get r_h. Run $\pi \leftarrow \mathsf{GenZK.Simulate}(\mathsf{crs}_{\mathsf{GenZK}}, \tau_{\mathsf{GenZK}}, (\mathsf{cm}, r_h, c, s))$ and return (R, r_h, c, s, π)

When Adv returns b', then $\mathcal{B}_1$ returns 0 if $b = b'$ and 1 if $b \neq b'$.

In the case that $b = 0$ we have that this is a perfect simulation of both Game_0 and Game_1. If however $b = 1$, when $\mathsf{HCommit}$ commits to r this is a perfect simulation of Game_0. When $\mathsf{HCommit}$ commits to $s - cx$ this is a perfect simulation of Game_1.

We now verify the reduction claim by computing the advantage of $\mathcal{B}_1$ in the hiding game. For convenience, let h be the internal coin of the hiding challenger: let $h \leftarrow 0$ if $\mathcal{O}_{\mathsf{HCommit}}(r, s - cx)$ commits to r and $h \leftarrow 1$ if it commits to $s - cx$.

We start by computing the probability that B_1 wins the hiding game:

$$\begin{aligned}\Pr[\mathcal{B}_1 \text{ wins hiding game}] &= \Pr[\mathcal{B}_1(\mathsf{crs}) = 0 \mid h = 0] \Pr[h = 0] + \Pr[\mathcal{B}_1(\mathsf{crs}) = 1 \mid h = 1] \Pr[h = 1] \\ &= \frac{1}{2} \left(\Pr[\mathcal{B}_1(\mathsf{crs}) = 0 \mid h = 0] + \Pr[\mathcal{B}_1(\mathsf{crs}) = 1 \mid h = 1] \right)\end{aligned}$$

Now we compute the conditional probabilities from above:

$$\begin{aligned}\Pr[\mathcal{B}_1(\mathsf{crs}) = 0 \mid h = 0] &= \Pr[\mathcal{A}(\mathsf{crs}) = 0 \mid \mathsf{Game}_{0,b} = 0] \Pr[\mathsf{Game}_{0,b} = 0] \\ &\quad + \Pr[\mathcal{A}(\mathsf{crs}) = 1 \mid \mathsf{Game}_{0,b} = 1] \Pr[\mathsf{Game}_{0,b} = 1] \\ &= \frac{1}{2} \left(\Pr[\mathcal{A}(\mathsf{crs}) = 0 \mid \mathsf{Game}_{0,b} = 0] + \Pr[\mathcal{A}(\mathsf{crs}) = 1 \mid \mathsf{Game}_{0,b} = 1] \right)\end{aligned}$$

$$\begin{aligned}\Pr[\mathcal{B}_1(\mathsf{crs}) = 1 \mid h = 1] &= 1 - \Pr[\mathcal{B}_1(\mathsf{crs}) = 0 \mid h = 1] \\ &= 1 - \frac{1}{2} \left(\Pr[\mathcal{A}(\mathsf{crs}) = 0 \mid \mathsf{Game}_{1,b} = 0] + \Pr[\mathcal{A}(\mathsf{crs}) = 1 \mid \mathsf{Game}_{1,b} = 1] \right)\end{aligned}$$

Finally, we bring the above together to calculate the advantage of $\mathcal{B}_1$ in the hiding game:

$$
\begin{aligned}
\mathsf{Adv}^{\mathsf{hiding}}_{\mathcal{B}_1} &= |1 - 2\Pr[\mathcal{B}_1 \text{ wins hiding game}]| \\
&= |1 - \Pr[\mathcal{B}_1(\mathsf{crs}) = 0 \mid h = 0] - \Pr[\mathcal{B}_1(\mathsf{crs}) = 1 \mid h = 1]| \\
&= |1 - \frac{1}{2}\left(\Pr[\mathcal{A}(\mathsf{crs}) = 0 \mid \mathsf{Game}_{0,b} = 0] + \Pr[\mathcal{A}(\mathsf{crs}) = 1 \mid \mathsf{Game}_{0,b} = 1]\right) \\
&\quad - \left(1 - \frac{1}{2}\left(\Pr[\mathcal{A}(\mathsf{crs}) = 0 \mid \mathsf{Game}_{1,b} = 0] + \Pr[\mathcal{A}(\mathsf{crs}) = 1 \mid \mathsf{Game}_{1,b} = 1]\right)\right) \\
&= \frac{1}{2}|\Pr[\mathcal{A}(\mathsf{crs}) = 0 \mid \mathsf{Game}_{1,b} = 0] + \Pr[\mathcal{A}(\mathsf{crs}) = 1 \mid \mathsf{Game}_{1,b} = 1] \\
&\quad - \Pr[\mathcal{A}(\mathsf{crs}) = 0 \mid \mathsf{Game}_{0,b} = 0] - \Pr[\mathcal{A}(\mathsf{crs}) = 1 \mid \mathsf{Game}_{0,b} = 1]| \\
&= \frac{1}{2}|1 - \Pr[\mathcal{A}(\mathsf{crs}) = 1 \mid \mathsf{Game}_{1,b} = 0] + \Pr[\mathcal{A}(\mathsf{crs}) = 1 \mid \mathsf{Game}_{1,b} = 1] \\
&\quad - 1 + \Pr[\mathcal{A}(\mathsf{crs}) = 1 \mid \mathsf{Game}_{0,b} = 0] - \Pr[\mathcal{A}(\mathsf{crs}) = 1 \mid \mathsf{Game}_{0,b} = 1]| \\
&= \frac{1}{2}|\left(\Pr[\mathcal{A}(\mathsf{crs}) = 1 \mid \mathsf{Game}_{0,b} = 0] - \Pr[\mathcal{A}(\mathsf{crs}) = 1 \mid \mathsf{Game}_{0,b} = 1]\right) \\
&\quad - \left(\Pr[\mathcal{A}(\mathsf{crs}) = 1 \mid \mathsf{Game}_{1,b} = 0] - \Pr[\mathcal{A}(\mathsf{crs}) = 1 \mid \mathsf{Game}_{1,b} = 1]\right)|
\end{aligned}
$$

Denote $a_0 = 2\mathsf{Adv}^{\mathsf{hiding}}_{\mathcal{B}_1}$, $a_1 = \left(\Pr[\mathcal{A}(\mathsf{crs}) = 1 \mid \mathsf{Game}_{0,b} = 0] - \Pr[\mathcal{A}(\mathsf{crs}) = 1 \mid \mathsf{Game}_{0,b} = 1]\right)$ and $a_2 = \left(\Pr[\mathcal{A}(\mathsf{crs}) = 1 \mid \mathsf{Game}_{1,b} = 0] - \Pr[\mathcal{A}(\mathsf{crs}) = 1 \mid \mathsf{Game}_{1,b} = 1]\right)$. We have that $a_0 \geqslant 0$.

- If $a_0 = |a_1 - a_2| = a_1 - a_2$ then $a_1 = a_0 + a_2 \leqslant |a_0| + |a_2|$.
- If $a_0 = |a_1 - a_2| = a_2 - a_1$ then $a_1 = a_2 - a_0 \leqslant |a_0| + |a_2|$.

Hence

$$
\begin{aligned}
& |\Pr[\mathcal{A}(\mathsf{crs}) = 1 \mid \mathsf{Game}_{0,b} = 0] - \Pr[\mathcal{A}(\mathsf{crs}) = 1 \mid \mathsf{Game}_{0,b} = 1]| \\
\leqslant\ & 2\mathsf{Adv}^{\mathsf{hiding}}_{\mathcal{B}_1} + |\Pr[\mathcal{A}(\mathsf{crs}) = 1 \mid \mathsf{Game}_{1,b} = 0] - \Pr[\mathcal{A}(\mathsf{crs}) = 1 \mid \mathsf{Game}_{1,b} = 1]| \\
\Rightarrow\ & \mathsf{Adv}^{\mathsf{Game}_0}_{\mathcal{A}} \leqslant 2\mathsf{Adv}^{\mathsf{hiding}}_{\mathcal{B}_1} + \mathsf{Adv}^{\mathsf{Game}_1}_{\mathcal{A}}
\end{aligned}
$$

as required.

$\underline{\mathcal{O}_1((X, \mathsf{cm}), (x))}$
$\mathsf{assert}\ ((X, \mathsf{cm}), x) \in \mathcal{R}_{\mathsf{sigmabus}}$
$c, s, o_h \xleftarrow{\$} \mathbb{F}$
$R \leftarrow sG - cX$
$r_h \leftarrow \mathsf{HCommit}(s - cx, o_h)$
$\pi \leftarrow \mathsf{GenZK.Simulate}(\mathsf{crs}_{\mathsf{GenZK}}, \tau_{\mathsf{GenZK}}, (\mathsf{cm}, r_h, c, s))$
$\mathsf{return}(R, r_h, c, s, \pi)$

Figure 3: Simulated proving oracle $\mathcal{O}_1$ in Game_1. The differences with Game_0 are highlighted

Game 2: The second game Game_2 behaves identically to Game_1 except that the simulation oracle generates π using the real prover. See Figure 4. We define a reduction $\mathcal{B}_2$ against the zero-knowledge of GenZK and show that

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Game}_1} \leqslant 2\mathsf{Adv}_{\mathcal{B}_2}^{\mathsf{GenZK\text{-}zk}} + \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Game}_2}$$

Let $\mathcal{A}$ be an adversary against Game_0. Then the reduction $\mathcal{B}_2$ behaves as follows. Upon receiving $\mathsf{crs}_{\mathsf{GenZK}}$ it samples $b \xleftarrow{\$} \{0,1\}$, sets $\mathsf{crs} \leftarrow (\mathbb{G}, \mathsf{crs}_{\mathsf{GenZK}})$ and runs $\mathcal{A}^{\mathcal{O}_b}(\mathsf{crs})$. When $\mathcal{A}$ queries $\mathcal{O}_b((X, \mathsf{cm}), x)$ then $\mathcal{B}_2$ asserts that $((X, \mathsf{cm}), x) \in \mathcal{R}_{\mathsf{sigmabus}}$. If yes then it responds as follows:

- If $b = 0$, return $\langle \mathsf{Prove}(\mathsf{crs}, (X, \mathsf{cm}), x), \mathsf{Verify}(\mathsf{crs}, (X, \mathsf{cm})) \rangle$.
- Else if $b = 1$, choose $c, r, s, o_h \xleftarrow{\$} \mathbb{F}$, $R \leftarrow sG - cX$. Set $r_h = \mathsf{HCommit}(s - cx, o_h)$. Query $\pi \leftarrow \mathcal{O}^{\mathsf{GenZK}}((\mathsf{cm}, r_h, c, s), (x, s - cx, o_h))$ and return (R, r_h, c, s, π)

In the case that $b = 0$ we have that this is a perfect simulation of both Game_0 and Game_1. If however $b = 1$, when $\mathcal{O}^{\mathsf{GenZK}}$ is the proving oracle is a perfect simulation of Game_1. Else this is a perfect simulation of Game_2.

We derive the reduction claim using a similar argument to the one of Game_1.

$\underline{\mathcal{O}_1((X, \mathsf{cm}), (x))}$
$\mathsf{assert}\ ((X, \mathsf{cm}), x) \in \mathcal{R}_{\mathsf{sigmabus}}$
$c, s, o_h \xleftarrow{\$} \mathbb{F}$
$R \leftarrow sG - cX$
$r_h \leftarrow \mathsf{HCommit}(s - cx, o_h)$
$\pi \leftarrow \mathsf{GenZK.Prove}(\mathsf{crs}_{\mathsf{GenZK}}, (\mathsf{cm}, r_h, c, s), (x, s - cx, o_h))$
$\mathsf{return}(R, r_h, c, s, \pi)$

Figure 4: Simulated proving oracle $\mathcal{O}_1$ in Game_2. The differences with Game_1 are highlighted

We now see that the transcripts given in Game_2 between the two oracles are identically distributed. Indeed, R is distributed uniformly at random due to s. Then s is the unique value satisfying the verifiers equation. The adversary cannot see the order in which values are sampled. Thus Game_2 is statistically impossible. □

5 Optimizations and Future Extensions

5.1 Improving communication complexity for big linear transformations

While the previous sections showcased Sigmabus with a simple scalar multiplication, we noted that Sigmabus can also handle arbitrary linear transformations. For instance, a multi-scalar multiplication can be defined as $\phi(\vec{x}) : \vec{X} \to x_1G_1 + x_2G_2 + x_3G_3 + \ldots + x_nG_n$. In this case, the Sigma protocol's communication complexity grows linearly with the size of the vector $\vec{x}$, contributing to the overall proof size of Sigmabus. Future improvements could leverage Compressed Σ-protocols [AC20] to amortize the communication complexity. This optimization could reduce the communication complexity from linear to polylogarithmic in the input size.

5.2 Amortizing multiple transformations

In a circuit that computes multiple linear operations, such as $X_1 = x_1 G \wedge X_2 = x_2 G \wedge X_3 = x_3 G$, a naive approach with Sigmabus would involve performing three separate Schnorr proofs for each X_i. We can amortize the operation by conducting a random linear combination of all the claims and verifying them within a single Schnorr proof. This amortization strategy, specifically tailored for PLONK, is demonstrated in Section 6.2.2.

6 Applications

6.1 Anonymous Credentials

Consider a simple protocol dlhash where given an algebraic commitment X and a hash-based commitment x_h, the prover demonstrates knowledge of x such that $X = xG$ and $x_h = H(x)$. Similar protocols find applications in anonymous credential systems [KMNC] or privacy-preserving signature verification [CGM16].

In this context, we instantiate Sigmabus with $\mathsf{Commit}(x) : x \to \mathsf{Hash}(x)$ for a hash function Hash, resulting in the following relation:

$$\mathcal{R}_{\mathsf{dlhash}} = \{(X, x_h);\ (x)\ : x_h = \mathsf{Hash}(x) \wedge X = xG\}$$

The construction makes use of a SNARK protocol inner as a subprotocol, with the following relation:

$$\mathcal{R}_{\mathsf{inner}} = \{(x_h, r_h, s, c); (x, r) : x_h = \mathsf{Hash}(x) \wedge r_h = \mathsf{Hash}(r) \wedge s = r + cx\}$$

By incorporating Sigmabus into this protocol, we significantly reduce the number of constraints by replacing the in-circuit scalar multiplication with an algebraic hash function invocation. Moreover, the dlhash protocol can be flexibly adapted to accommodate different types of discrete-log-based commitments and various kinds of Sigma protocols.

We also note that when the Fiat-Shamir heuristic is applied in dlhash, the prover can avoid sending r_h to the verifier. Instead the prover can utilize the Fiat-Shamir challenge c, and show that c was computed correctly inside the SNARK, effectively using c as a commitment to r.

6.1.1 Hiding Commitment

Observe that in $\mathcal{R}_{\mathsf{inner}}$ we are treating $\mathsf{Hash}(r)$ as a hiding commitment. This is an abuse of the term. Even if $\mathsf{Hash}()$ were a perfect random oracle, one could distinguish $\mathsf{Hash}(r_1)$ from $\mathsf{Hash}(r_2)$ given r_1 and r_2. However, in Sigmabus, if the adversary ever queried $\mathsf{Hash}(r)$ then one could build a discrete logarithm solving reduction. This reduction observes queries r_i to $\mathsf{Hash}()$, computes $x_i = \frac{s - r_i}{c}$ for the adveraries proof (R, r_h, s, π), and returns x_i if $X = x_i G$. The function $\mathsf{Hash}()$ does meet the necessary hiding requirements against adversaries that cannot query Hash on r.

6.2 Plonkish

In this section we show how Sigmabus can be integrated into a $\mathcal{P}\mathfrak{lonK}$ circuit.

Consider a $\mathcal{P}\mathfrak{lonK}$ circuit with witness column $\vec{w}$ and its low degree extension $w(x)$. In $\mathcal{P}\mathfrak{lonK}$ circuits, we construct gates and selectors to check constraints over specific positions in the witness.

Given witness polynomial $w(x)$, we can check that $\vec{w}[i]$ satisfies constraint $\mathcal{C}$ by constrainting that $L_i(X)(\mathcal{C}(f(X)) = 0$, where $L_i(x)$ is the Lagrange polynomial of index i.

In the following two sections we show how Sigmabus can be used to efficiently compute elliptic curve operations in $\mathcal{P}\mathfrak{lonK}$ circuits:

6.2.1 $\mathcal{P}\mathfrak{lonK}$ circuit with a single scalar multiplication

For this section, our circuit computes $H = xG$ where x is in cell i of the witness column $\vec{w}$. This can be seen as the following Plonkish constraint: $L_i(X)(w(X) \cdot G - H) = 0$

Here is an informal description of how Sigmabus can be used in this context:

1. Prover sends H and $Com(w)$, where H is the intended result and $Com(w)$ is a KZG commitment to the witness polynomial
2. Prover initiates the Sigmabus protocol (see Figure 2) by sending R and r_h
3. Verifier sends Sigmabus challenge c
4. Prover sends the Sigmabus response s
5. Prover modifies the Plonkish circuit to demonstrate knowledge of r such that $r_h = H(r)$ and also adds the following constraint: $L_i(X)(s - c + r \cdot w(X))$

Note that in the above protocol, the prover had to send the scalar multiplication result H to the verifier. This would break zero-knowledge if H is part of the circuit's trace. In those cases, the protocol can be modified to compute a hiding Pedersen commitment so that H is never revealed directly.

6.2.2 $\mathcal{P}\mathfrak{lonK}$ circuit with a multiple scalar multiplications

In this section, we consider a $\mathcal{P}\mathfrak{lonK}$ circuit that wants to compute multiple scalar multiplications: $H_1 = x_1G \wedge H_2 = x_2G \wedge H_3 = x_3G$

This translates to the following in vector notation:

$$\vec{w_1}[i_1] \cdot G = H_1 \wedge \vec{w_2}[i_2] \cdot G = H_2 \wedge \vec{w_3}[i_3] \cdot G = H_3$$

Naively, we would need to conduct the Sigmabus protocol three times, but we now demonstrate how to amortize the operation.

Here is how this can be done:

1. Prover sends $H_1, H_2, H_3, Com(w_1), Com(w_2), Com(w_3)$
2. Prover sends R and r_h
3. Verifier sends (c, γ) where $\gamma \xleftarrow{\$} \mathbb{F}$
4. Prover sends s of Schnorr for H where $H = H_1 + \gamma H_2 + \gamma^2 H_3$
5. Prover modifies the Plonkish circuit to demonstrate knowledge of r such that $r_h = H(r)$ and also adds the following constraint:

$$L_i(X)(s - c + r * (w_1(X) + \gamma w_2(X) + \gamma^2 w_3(X)))$$

6.3 Linking algebraic with non-algebraic commitments

Consider a protocol where a party aims to demonstrate knowledge of a secret value x that has been committed using a Pedersen commitment $X = xG$ and a non-algebraic hash $x_h = \mathsf{SHA3}(x)$

Similar to the approach of [CGM16], we can use a proof system over boolean fields like [GMO16] (where SHA3 can be efficiently computed) and offload the algebraic operations to a Sigma protocol using Sigmabus.

A similar technique can be used to *link* Pedersen commitments with Merkle commitments.

7 Acknowledgements

Special thanks to Michele Orrù, Nicolas Mohnblatt and to Mark Simkin for valuable inputs and discussions.

References

[AC20] Thomas Attema and Ronald Cramer. Compressed σ-protocol theory and practical application to plug & play secure algorithmics. Cryptology ePrint Archive, Paper 2020/152, 2020. https://eprint.iacr.org/2020/152.

[BN06] Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, October 30 - November 3, 2006*, pages 390–399. ACM, 2006.

[CFQ19] Matteo Campanelli, Dario Fiore, and Anaïs Querol. Legosnark: Modular design and composition of succinct zero-knowledge proofs. *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019.

[CGM16] Melissa Chase, Chaya Ganesh, and Payman Mohassel. Efficient zero-knowledge proof of algebraic and non-algebraic statements with applications to privacy preserving credentials. Cryptology ePrint Archive, Paper 2016/583, 2016. https://eprint.iacr.org/2016/583.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. Cryptology ePrint Archive, Paper 2002/140, 2002. https://eprint.iacr.org/2002/140.

[GMO16] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. Zkboo: Faster zero-knowledge for boolean circuits. Cryptology ePrint Archive, Paper 2016/163, 2016. https://eprint.iacr.org/2016/163.

[Kil90] Joe Kilian. *Uses of Randomness in Algorithms and Protocols*. MIT Press, Cambridge, MA, USA, 1990.

[KMNC] George Kadianakis, Mary Maller, Andrija Novakovic, and Suphanat Chunhapanya. Proof of Validator: A simple anonymous credential scheme for Ethereum's DHT.

[Sch91] C. P. Schnorr. Efficient signature generation by smart cards. *J. Cryptol.*, 4(3):161–174, jan 1991.

[Wil] Zac Williamson. Goblin plonk: lazy recursive proof composition.