A Zero Knowledge Proof framework for Cardano based on Hydra and ZK-SNARKS

Antonio Hernandez-Garduño Juan Salvador Magán Valero
Agustín Salinas

November 2023

Contents

Chapter 1

Introduction

Recently, there has been a generational leap in the blockchain industry due to new developments in the field of cryptography. This progress is making improvements in many deficient areas that were difficult to overcome in the past such as privacy and scalability. The principal technology behind these advancements is Zero-Knowledge proofs, which comprise a range of cryptographic schemes that will enable a wide array of new applications.

While ecosystems like Ethereum have made significant progress in this area, Cardano faces a technical challenge in this regard to adopt this technology. One of the key aspects of using Zero-Knowledge proofs—verifying the proof within a smart contract—is currently not feasible due to the computational limits of the network. Although a new CIP is being made to address this issue by implementing new Plutus Core primitives to perform this operation, this will be available in the next hardfork so it wouldn't be possible in the short-run. Solving this issue will allow to use this technology on-chain bringing the enhancements aforementioned. Considering the above, the objective of our project is to solve this problem and enable the verification of ZKPs. On one hand, it consists of an open source library that implements the algorithms to do such verifications on-chain, on the other, it includes an integration of this library with the Hydra protocol. This latter point is crucial because at the moment a layer 2 solution is indispensable to overcome the constraints of the network. Overall, this project aims to provide an infrastructure improvement in the ecosystem that intends to gather the first PoCs in this field on Cardano. Applications that need the use of high frequency transactions will be especially benefited by this project. In order to make our solution more illustrative we implemented a dApp of the Mastermind game, the result is a proof of concept which serves as an ideal test case for Zero-Knowledge proofs.

This document describes the process of developing this project, the technical research and the current implementation of our solution. Although there are many types of Zero-Knowledge proofs, SNARKs schemes have been found well suited for the requirements of blockchain applications. This project uses

SNARK at its core so this document will study and analyze this type of scheme. We will begin in the next two chapters with an explanation of existing SNARK schemes and elliptic curve cryptography. In the fourth chapter we discuss the theory behind the implementation of a *Weil pairing* in the context of a tower of field extensions. In the fifth chapter we will provide an overview of the libraries and tools currently employed in the development of zk-apps, along with an in-depth specification of the solutions we use in our project. Lastly, in the sixth chapter, we will provide a general explanation of the process for implementing the project.

Chapter 2

Succinct Non-Interactive Argument of Knowledge (SNARK)

Zero-Knowledge proofs are a range of cryptographic schemes that allow to demonstrate some information without revealing it. The principal idea of this technology is the replacement of information by proofs, where these proofs can either hide information or hide complexity. In this sense, a prover can send a proof to a verifier that demonstrates that it knows some secret information (witness) or that it has correctly performed certain computation. A cryptographic solution must satisfy this three properties in order to be called a Zero-Knowledge proof [1]:

- **Completeness:** Given a statement and a witness, the prover can convince the verifier.
- **Soundness:** A malicious prover cannot convince the verifier of a false statement.
- **Zero-knowledge:** The proof does not reveal anything but the truth of the statement, in particular it does not reveal the prover's witness.

SNARKs definition

Like we mentioned before there exist many types of Zero-Knowledge proofs but SNARKs have become particularly useful for blockchain applications. This cryptographic primitive stands for Succinct Non-interactive argument of knowledge [1]. Here the argument of knowledge means the involvement of verifiable knowledge or information; the succinctness means that the proof is shorter than the information to verify; and non-interactive means that the demonstration is done in one step. SNARKs have become particularly useful for blockchain applications for the following reasons:

- **Storage efficiency:** The proofs are small in terms of size (200-300 bytes) and that's why it suits the storage limitation that a blockchain can have.
- **Computation efficiency:** The verification of the proofs are fast and more efficient if we compare it to other Zero-Knowledge proof schemes such as STARKs, DARKs and Bulletproofs.
- **Non-interactive:** Contrary to other types of ZKPs, the verification is done in one step. That way just a single transaction is needed which prevents the saturation of the network.

2.1 SNARKs step by step

A SNARK scheme consists of three algorithms [2]:

1. Setup: $S(C, \lambda) = (\mathbf{pk}, \mathbf{vk})$

This first step is a preprocessing of the algorithm to be verified. In this process, two public keys are generated, a proving key pk and a verification key vk. To generate these keys, an arithmetic circuit (representing the algorithm to be proven) and a random secret parameter λ (lambda) are required. λ is crucial for ensuring the authenticity and cryptographic security of the proof. If λ is compromised, false proofs could be generated. In the case of our proof of concept, the circuit C represents the Mastermind game as a series of arithmetic operations. This representation is important as it allows for the testing of true or false computational statements within the context of the game, where zero represents true and any non-zero number represents false.

2. Proof: $P(\mathbf{pk}, x, w) = \pi$

This process consists in the generation of a proof given the setup by a prover. The proof generation function P takes as input the proving key (pk), a public argument (x), and a witness (w), which is the secret being proven. It generates a proof (π) that demonstrates that the prover knows the witness (w) and that the witness satisfies a truth condition within the program's context. In the case of the game, the "codifier" generates a proof for a clue (π) using a proving key (pk), a clue as the public value (x), and the secret combination of the guess (w).

3. Verification: $V(\mathbf{vk}, x, \pi) =$ True or False

This process consists in the verification of the proof by a verifier.The verification function V, using the verification key (vk), public parameter (x), and the proof (π), determines whether the proof is valid or not. This function aims to provide certainty that a prover knows a valid witness (w) along with a valid public parameter (x) in relation to an arithmetic circuit C (the representation of the game). The function V will return true or false depending on whether the prover can generate a proof that satisfies the aforementioned criteria.

2.2 What Types of Setup Exist?

As we'll show there exist variants of the SNARK scheme that do the setup process differently [3]. The key difference is wether the setup process requires a random parameter to ensure security or not; if it's needed, fraudulent proofs could be generated in case that these secret random parameters are compromised.

- **Circuit-specific Trusted Setup:** A unique random parameter is needed for each circuit, which must remain unknown to the prover.

  $S(\text{Circuit}, \text{random-data}) = (\text{Prover's Key}, \text{Verifier's Key})$

- **Universal Trusted Setup:** This is a circuit-independent secret parameter chosen only once. It comprises two phases: The first is an initialization procedure where the secret parameter is used once to generate other public parameters, and the indexing phase where these resulting public parameters, along with the circuit, are used to generate S_v and S_p.

  $S = (S_{\text{init}}, S_{\text{index}}) : S_{\text{init}}(r, \lambda) \to pp, S_{\text{index}}(pp, C) \to (S_p, S_v)$

- **Transparent Setup:** It does not utilize any secret data that depends on trust.

2.3 SNARK and Zero-Knowledge protocols comparisson

The table presents a comparison of some types of SNARKs and other Zero-Knowledge protocols based on several parameters [3]: the size of the proof, the size of a parameter S_p beyond a complexity term C, the verifier time for a common task, and whether a trusted setup is required. The first two rows are three SNARK schemes: Groth'16 and Plonk/Marlin require small proof sizes and have fast verification times, with the latter offering a universal setup. Regarding the other protocols: Bulletproofs, STARK, and DARK do not require a trusted setup, with Bulletproofs having longer verification times and STARK and DARK offering intermediate proof sizes and verification times. Groth'16 and Plonk/Marlin provide efficient proof and verification times with some trust assumptions, while Bulletproofs, STARK, and DARK prioritize trustlessness and scalability at varying levels of efficiency. This highlights the diverse approaches in cryptographic design to balance security, speed, and trust requirements.

Scheme	**Size of proof π**	**Size of S_p (beyond C)**	**Verifier time (aprox)**	**Trusted (setup?)**
Groth'16	$\approx$ 200 Bytes	$O_a(\|C\|)$	$\approx$ 3 ms	Yes/Per circuit
Plonk/Marlin	$\approx$ 400 Bytes	$O_a(\|C\|)$	$\approx$ 6 ms	Yes/Universal
Bulletproofs	$\approx$ 1.5 KB	$O_a(1)$	$\approx$ 1.5 sec	No
STARK	$\approx$ 80 KB	$O_a(\log^2 \|C\|)$	$\approx$ 10 ms	No
DARK	$\approx$ 10 KB	$O_a(1)$	$O_a(\log \|C\|)$	No

Table 2.1: Comparison of various cryptographic schemes.

2.4 Decision Making Behind Groth16

For our project, we have elected to utilize the Groth16 protocol. The rationale for this decision is: Strategically, as of the present day, Groth16 is well-known protocol in the field of cryptography and is supported by a wealth of information which facilitates its implementation. On the other hand, technically, it offers efficiency advantages that are particularly promising for a proof of concept, especially when taking into account the network constraints outlined in the introduction of this document. As ilutrated in the table 1.1, relative to other protocols, Groth16 is characterized by its minimal proof size, which conserves network storage, and it stands as the quickest among the SNARK protocols in terms of verification speed.

However, it is worth noticing one of the drawback of Groth16: the protocol needs more security measures because the setup has to be trusted and is specific to each circuit. Nonetheless, this security requirements are not prohibitive or limiting for our project. Once again, our emphasis is on efficiency to overcome any potential computational or network bottlenecks. Groth16's design is inherently compatible with our objectives, operating effectively under network limitations while still achieving a proof of concept that meets rigorous cryptographic security standards.

Chapter 3

Cryptography based on elliptic curves

Elliptic curves have been known for a long time. Their formal study in the context we know today began to take shape in the 19th century. However, it was not until the 1980's that their use was potentialized, initially in the context of the discrete logarithm. In 1985, Miller and Koblitz proposed the implementation of the Diffie-Hellman protocol in the context elliptic curve subgroups, linking two different groups. This is when the concept of *pairing* was born together with an efficient algorithm implementing it, known as the *Weil pairing*.

The *pairing* is a mathematical operation that involves two elliptic curves. It is denoted as $e : G_1 \times G_2 \to G_T$, where G_1 and G_2 are subgroups of elliptic curves and G_T is the *target* group, typically a field with its product taking the role of group multiplication.

A key properties that a pairing must satisfy are:

- Bilineality: $e(aP, bQ) = e(P, Q)^{ab}$ for all points P, Q on the elliptic curves, and integers a, b.
- Non-degeneracy: $e(P, Q) \neq 1_{G_T}$ for generator points P, Q.

3.1 Introduction to Pairing-Friendly Curves

Next, following [7], we will give a brief description of pairing-friendly-curves. This document treats the definition and properties of the BN and BLS curves in the context of pairing-based cryptography, giving details about its formulation and key characteristics for their use in security systems and cryptographic protocols.

BN (Barreto-Naehrig) curves

A BN curve [5] is classified as a pairing-friendly curve and is defined through the families of elliptic curves E and E', parameterized according to an integer t selected in a special manner.

The curve E is defined over a finite field $\mathbb{F}_p$, through an equation of the form

$$E : y^2 = x^3 + b$$

where b is a primitive element of the multiplicative group $\mathbb{F}_p^*$ of order $p - 1$.

The parameters p and r are defined as:

$$p = 36 \cdot t^4 + 36 \cdot t^3 + 24 \cdot t^2 + 6 \cdot t + 1$$
$$r = 36 \cdot t^4 + 36 \cdot t^3 + 18 \cdot t^2 + 6 \cdot t + 1$$

In the case of the BN curves, torsions of degree six can be used. If m is an element that is neither a square nor a cube in the field extension $\mathbb{F}_{p^2}$, the torsion E' of E is defined over the extended field $\mathbb{F}_{p^2}$ through the equation $E' : y^2 = x^3 + b'$ with $b' = b/m$ or $b' = b \cdot m$. Curves BN are denoted of type D in the former case, and of type M in the later. The embedding degre k is 12.

BLS (Barreto-Lynn-Scott) curves

The curves BLS were proposed in 2002 [6] and are also considered pairing-friendly and are defined through the elliptic curves E and E', parameterized by an integer t. Similarly to the BN curves, these curves allow to construct optimal Ate pairings.

Elliptic curve E is defined over a finite field $\mathbb{F}_p$ according to an equation of the form $E : y^2 = x^3 + b$, and its torsion E', defined in the same manner as the BN curves, following the form $E' : y^2 = x^3 + b'$. As opposed to the BN curves, $E(\mathbb{F}_p)$ is not of prime order. Instead, its order is divisible by a big prime number parameterized by r, and is denoted as $h \cdot r$ with a cofactor h. The pairing is defined at the r-torsion points. Similarly the the BN curves, the BLS curves can be classified as of D-type or M-type.

The equations for the parameters p and r in the case of the BLS12 and BLS48 curves are defined as:

$$\text{For BLS12:} \quad p = (t-1)^2 \cdot (t^4 - t^2 + 1)/3 + t, \quad r = t^4 - t^2 + 1$$
$$\text{For BLS48:} \quad p = (t-1)^2 \cdot (t^{16} - t^8 + 1)/3 + t, \quad r = t^{16} - t^8 + 1$$

A pairing e is defined taking G_1 as a subgroup of $E(\mathbb{F}_p)$ of order r, G_2 as a subgroup of order r in $E'(\mathbb{F}_{p^2})$ for $BLS12$ and of $E'(\mathbb{F}_{p^8})$ for BLS48; and the target group G_T as a subgroup of order r in the multiplicative group $F_{p^{12}}^*$ for BLS12, and of $\mathbb{F}_{p^{48}}^*$ for BLS48.

3.2 Selection of BLS curves over BN curves

After considering the security and computational efficiency in diverse cryptographic schemes, the Barreto-Lynn-Scott (BLS) curves have revealed as more convenient in comparison with the Barret-Naehrig (BN) curves. In spite of past considerations that suggested a possible construction of BN curves that could reach a security level of 128 bits, these curves carry significative limitations when working with a bigger base field and a considerably bigger group order. The article /emphBLS12-381: New zk-SNARK Elliptic Curve Construction [8] offers more details that support the selection of the concrete curve BLS12-381 for ZK-SNARKs applications.

In what follows, we expose the key reasons that put our preference for the BLS curves over the BN ones.

Computational performance

The BN curves, being related with a larger base field ($q \approx 2^{384}$), provide a group order significantly bigger ($r \approx 2^{384}$). This situation deteriorates the performance of key cryptographic operations, like multi-exponentiation steps, rapid Fourier transforms, and other operations necessary to guarantee the efficacy of advanced cryptographic schemes, like the zk-SNARK protocols and multi-party computing.

Scalar field size

On top of the above, a larger scalar field ($\mathbb{F}_r$) associated with the BN curves carry an excess of key material, which results in more complexity and resource consumption in the implementation of cryptographic systems.

Advantages of the BLS curves

In contrast, the BLS curves provide with a more manegable group order ($r \approx 2^{256}$), in spite of having a smaller base field ($q \approx 2^{384}$). This property allows to sidestep the disadvantages in performance and usability associated with a bigger scalar field.

Based in its capacity to reach a level of security of 128 bits, current research favors the BLS curves, which provide an sweet spot between the required security and the required computational performance for a variety of cryptographic applications. This establishes the BLS curves as the most viable and effective option in comparison with the BN curves.

3.3 Conclusion

The study of Kim and Barbulescu in february of 2016 [9] presented improvements in the computation of discrete logarithms in finte fields $\mathbb{F}_{p^n}$, which impacted a fields typically used in cryptography based in pairings, like $\mathbb{F}_{p^6}$ and $\mathbb{F}_{p^{12}}$. These improvements showed the necessity of augmenting the length of bits in the BN curves.

This study resulted in the curve BN256 not being able to provide the 128 security bits with which it originally was taken for granted. This was an inflection point, where the fact that the BN384 curve augmented the number of bits eventually resulted in the selection of the BLS12-381 curve as more optimal, for the aforementioned reasons.

For our use case, and in generic blockchain related applications, we need optimal algorithms for the necessary security level. Even though the BN256 curve remained usable for a while due to the fact that its 110 bits of security were deemed sufficient for the majority of applications, new applications are seeing a slow migration the use of the BLS12-381.

It is for this reason that, in our project ,we have chosen to use the BLS12-381 elliptic curve, and focus all our efforts in translating the latest improvements in the Ate optimal pairing to the BLS curve.

Chapter 4

Tower of field extensions

Above we introduced the concept of a *pairing* between two subgroups G_1 and G_2 of elliptic curves. To understand the construction of these subgroups and appreciate their role in cryptographic schemes, includying zero knowledge proofs, it is unavoidable to delve into some basic theory.

Let $E(\mathbb{F})$ be the elliptic curve defined by the equation $y^2 = x^3 + ax + b$, with coordinates from a given field $\mathbb{F}$, that is to say,

$$E(\mathbb{F}) = \{(x, y) \in \mathbb{F} \times \mathbb{F} \mid y^2 = x^3 + ax + b\} \cup \{\mathcal{O}\}\,.$$

For us, $\mathbb{F}$ will be a finite prime field $\mathbb{F}_p$ or its extension $\mathbb{F}_{p^k}$. The symbol $\mathcal{O}$ denotes the point *at infinity* (geometrically, the point where all vertical lines meet), which plays the role of the *identity element* in the group structure of the elliptic curve.

Let $E(\mathbb{F})[r]$ be the collection of points of order r in the elliptic curve $E(\mathbb{F})$, that is to say

$$E(\mathbb{F})[r] = \{P \in E(\mathbb{F}) \mid rP = \mathcal{O}\}\,.$$

(Note that the notation rP is *not* to be understood as scalar multiplication as in $r(x, y) = (rx, ry)$, but rather as $rP = P + P + \ldots P$ (r times), where the plus sign in $P + P$ is *not* a simple vector sum but rather the group operation on the elliptic curve as defined e.g. in [10].)

Let r be a divisor of the order of the elliptic curve $E(\mathbb{F})$. Then $E(\mathbb{F})[r]$ is a subgroup of $E(\mathbb{F})$ or order r (i.e., it has precisely r elements).

So, consider an elliptic cruve $E(\mathbb{F}_p)$ over the prime field $\mathbb{F}_p$ (the so called base field), and let n be its order (i.e., the number of points in this elliptic curve). Fix r, a prime factor of n. From the general theory of elliptic curves, there exists a minimal integer $k = k(r)$ such that $E(\mathbb{F}_{p^k})[r]$ is strictly bigger than $E(\mathbb{F}_p)[r]$. This k is referred to as the *embedding degree.* The prime factor r is referred to as the *torsion order.*

The usual setup for defining a pairing useful in cryptography, in the standard literature on elliptic curves, is to define G_1 and G_2 as follows:

$$\begin{aligned} G_1 &= E(\mathbb{F}_p)[r]\,, \\ G_2 &= \{(x,y) \in E(\mathbb{F}_{p^k})[r] \mid \pi(x,y) = p(x,y)\}\,. \end{aligned}$$

where π is the *Frobenius endomorphism* $\pi : E(\mathbb{F}_{p^k}) \longrightarrow E(\mathbb{F}_{p^k})$ defined by:

$$\begin{aligned} \pi(x,y) &= (x^p, y^p)\,, \\ \pi(\mathcal{O}) &= \mathcal{O}\,. \end{aligned}$$

In this situation, the standard Miller's algorithm [11] can then be used to implement a pairing

$$e : G_1 \times G_2 \longrightarrow \mathbb{F}^*_{q^k}$$

satisfying the *bilinearity* property

$$\begin{aligned} e(P_1 + P_2, Q) &= e(P_1, Q)e(P_2, Q)\,, \\ e(P, Q_1 + Q_2) &= e(P, Q_1)e(P, Q_2)\,. \end{aligned}$$

Our first implementation[1] of the standard Miller's algorithm and the corresponding pairing can be found in repository [12]. This repo documents a successful testing for bilinearity in the context of the toy elliptic curve BLS6_6. Crucially, this implementation assumes that the equations defining the elliptic curve of which G_1 and G_2 are subgroups are the same.

4.1 An untwist necessary for pairing

The specifications [14] for the BN128 elliptic curve define G_1 and G_2 as subgroups of the elliptic curves

$$\begin{aligned} E(\mathbb{F}_p): &\quad y^2 = x^3 + 3\,, \\ E(\mathbb{F}_{p^2}): &\quad y^2 = x^3 + 3/(u+9) \end{aligned}$$

where

$$p = 21888242871839275222246405745257275088 \ddots \\ 96311157297823662689037894645226208583$$

We note that the two equations defining $E(\mathbb{F}_p)$ and $E(\mathbb{F}_{p^2})$ are not the same, so our initial pairing implementation, mentioned in the previous section, needs to be modified or else the bilinearity property is violated.

The key is to understand that, for efficiency reasons, the implementation of the BN128 (as well as other BN and BLS elliptic curves) is based on a *tower*

[1]We developed this initial implementation as part of our participation in the Emurgo BUILD Hackathon 2023.

of field extensions, that is to say, a sequence of partial field extensions starting with the base field $\mathbb{F}_p$ culminating at the top of the tower with a field isomorphic to $\mathbb{F}_{p^k}$, where k is the embedding degree. (For the BN128, $k = 12$.)

In this context, an *untwisting* step needs to be added to the pairing algorithm, whose purpose is to homomorphicaly map G_2 to a cyclic subgroup of the elliptic curve $E(\mathbb{F}_{p^k})$ defined by the same equation as G_1. The procedure to achieve this is based on the following theorem:

Theorem 1 (Untwist)**.** *Let $\mathcal{E}_1$ and $\mathcal{E}_2$ be two elliptic curves over a field $\mathbb{F}$ defined by the equations:*

$$\mathcal{E}_1 : \quad y^2 = x^3 + b \tag{4.1}$$

$$\mathcal{E}_2 : \quad y^2 = x^3 + b/\gamma \tag{4.2}$$

where $\gamma = \beta^2 = \alpha^3$, $\alpha, \beta, \gamma \in \mathbb{F}$. Then the map

$$\begin{aligned} &\psi : \mathcal{E}_2 \longrightarrow \mathcal{E}_1 \\ &(x, y) \mapsto (\alpha x, \beta y)\,, \quad \mathcal{O} \mapsto \mathcal{O} \end{aligned}$$

is a group isomorphism[2]*.*

Proof. If $P = (x, y)$ is on $\mathcal{E}_2$ then

$$(\beta y)^2 = \gamma y^2 = \gamma(x^3 + b/\gamma) = (\alpha x)^3 + b\,,$$

so clearly $(\alpha x, \beta y)$ is on $\mathcal{E}_1$. Now, we need to show that[3]

$$\psi(P) + \psi(P) = \psi(P + P)\,,$$

that is to say,

$$(\alpha x, \beta y) + (\alpha x, \beta y) = (\alpha\tilde{x}, \beta\tilde{y})\,,$$

where

$$\begin{aligned} \tilde{x} &= m^2 - 2x\,, \\ \tilde{y} &= m(x - \tilde{x}) - y\,, \\ m &= 3x^2/2y\,. \end{aligned}$$

On the other hand, $(\alpha x, \beta y) + (\alpha x, \beta y) = (x_*, y_*)$ where

$$\begin{aligned} x_* &= m_*^2 - 2\alpha x\,, \\ y_* &= m_*(\alpha x - x_*) - \beta y\,, \\ m_* &= \frac{3\alpha^2 x^2}{2\beta y}\,. \end{aligned}$$

[2]A map $\psi : G_1 \longrightarrow G_2$ is a group homomorphism if it preserves the group structure, that is to say $\psi(e_1) = e_2$ and $\psi(gh) = \psi(g)\psi(h)$, where e_i is the identity in G_i, $i = 1, 2$. If ψ is also invertible, then we say that it is a group isomorphism.

[3]Here the "+" operator denotes the group sum on the corresponding elliptic curve.

Substituting, we obtain

$$\begin{aligned}
x_* &= \frac{\alpha^4}{\beta^2} m^2 - 2\alpha x = \alpha(m^2 - 2x) = \alpha\tilde{x}\,, \\
y_* &= \frac{\alpha^2}{\beta} m(\alpha x - \alpha\tilde{x}) - \beta y \\
&= \frac{\alpha^3}{\beta^2} \beta m(x - \tilde{x}) - \beta y \\
&= \beta\big(m(x - \tilde{x}) - y\big) = \beta\tilde{y}\,.
\end{aligned}$$

Therefore $x_* = \alpha\tilde{x}$, $y_* = \beta\tilde{y}$, as claimed.

Likewise, for the case of distinct points P_1, P_2 in $\mathcal{E}_2$, we need to show that $\psi(P_1) + \psi(P_2) = \psi(P_1 + P_2)$. The argument goes through in a similar fashion as above. □

4.2 Tower of field extensions for curves BN128 and BLS12-381

The embedding degree for both the BN128 and BLS12-381 elliptic curves is $k = 12$. In both cases, the extension from $\mathbb{F}_p$ to $\mathbb{F}_{p^k}$ is obtained by constructing the following "2–3–2" tower of extensions:

Step 1. $\mathbb{F}_{p^2} = \mathbb{F}_p[u]/(u^2 + 1)$

Step 2. $\mathbb{F}_{p^6} = \mathbb{F}_{p^2}[v]/(v^3 - \xi)$

Step 3. $\mathbb{F}_{p^{12}} = \mathbb{F}_{p^6}[w]/(w^2 - v)$

Here, the notation $\mathbb{F}_q[x]$ denotes the set of all polynomials in the variable x (of arbitrary degree); and the notation $\mathbb{F}_q[x]/\varphi(x)$ denotes that we "take the quotient modulo $\varphi(x)$", meaning that we declare $\varphi(x)$ to be equivalent to zero. The symbol ξ denotes an *irreducible* polynomial: for BN128 we let $\xi = u + 9$, and for BLS12-381, $\xi = u + 1$.

This tower of field extensions is implemented in Haskell as follows:

```
newtype Fp1 = Fp1  {t0 :: Integer}  deriving (Eq, Show)
data Fp2    = Fp2  {u0 :: Fp1, u1 :: Fp1}  deriving (Eq, Show)
data Fp6    = Fp6  {v0 :: Fp2, v1 :: Fp2, v2 :: Fp2}  deriving (Eq, Show)
data Fp12   = Fp12 {w0 :: Fp6, w1 :: Fp6}  deriving (Eq, Show)
```

So, `Fp2 u0 u1` represents the polynomial $u_0 + u_1 u$ with coefficients $u_0, u_1 \in \mathbb{F}_p$; `Fp6 v0 v1 v2` represents the polynomial $v_0 + v_1 v + v_2 v^2$ with coefficients $v_0, v_1, v_2 \in \mathbb{F}_{p^2}$; and `Fp12 w0 w1` represents the polynomial $w_0 + w_1 w$ with coefficients $w_0, w_1 \in \mathbb{F}_{p^6}$.

Note that `Fp1` is just a wrapper representing the integers modulo p. We declare these datatypes to belong to the `Field` class, which implements two methods and inherits those from the the `Num` class:

```
class (Num a, Eq a) => Field a where
  nrp :: a        -- Non-reducible polynomial
  inv :: a -> a  -- Multiplicative inverse
```

The equation defining both the elliptic curves $G_1 = E(\mathbb{F}_p)$ and $E(\mathbb{F}_{p^{12}})$ is given by (4.1). The equation defining $G_2 = E(\mathbb{F}_{p^2})$ is given by (4.2). Hence, a prerequisite for implementing a *pairing*

$$e : G_1 \times G_2 \longrightarrow \mathbb{F}^*_{p^{12}}$$

is to implement the *untwist*

$$\psi : G_2 \longrightarrow E(\mathbb{F}_{p^{12}})$$

where ψ is the isomorphism of theorem 1. Now, to implement this untwist function, it is necessary to know concrete instances of the α and β mentioned in theorem 1. This is provided by the following

Theorem 2 (Roots)**.** *With the notation of theorem 1, let $\alpha = v$, $\beta = vw$, and $\gamma = \xi$. Then α is a cubic root, and β is a square root, of γ.*

Proof. We need to show that $v^3 = \xi$ and $(vw)^2 = \xi$. The former is obvious from step 2 in the tower construction (since we declared $v^3 - \xi = 0$). As for the later, simply note that

$$(vw)^2 = v^2 w^2 = v^2 v = \xi\,,$$

where the second equality follows from step 3 in the tower construction (since we declared $w^2 - v = 0$). □

With this theorem in hand, it is now easy to implement the untwist function in Haskell. Note that v is represented in $\mathbb{F}_{p^6}$ by `Fp6 0 1 0`. Thus, it is represented in $\mathbb{F}_{p^{12}}$ by `Fp12 (Fp6 0 1 0) 0`. Finally, using what we said is the representation of v in $\mathbb{F}_{p^6}$, it is clear that vw is represented in $\mathbb{F}_{p^{12}}$ by `Fp12 0 (Fp6 0 1 0)`.

So, for the BN128 curve, the Haskell implementation of the untwist is:

```
untwist :: EllipticCurve Fp2 -> EllipticCurve Fp12
untwist Infty       = Infty
untwist (EC x2 y2) = EC x12 y12
  where
    cubicRoot = Fp6 0 1 0
    x12 = (Fp12 (Fp6 x2 0 0) 0) * (Fp12 cubicRoot 0)
    y12 = (Fp12 (Fp6 y2 0 0) 0) * (Fp12 0 cubicRoot)
```

Here the `EllipticCurve` datatype is used to represent points in affine coordinates and is declared as

```
data EllipticCurve a = EC {ax :: a, ay :: a} | Infty
  deriving (Eq, Show)
```

For the BN12-381 curve, the γ in theorem 1 corresponds to ξ^{-1}, so the untwist is defined sligtly different, only needing to change in the last two lines:

```
    x12 = (Fp12 (Fp6 x2 0 0) 0) * inv (Fp12 cubicRoot 0)
    y12 = (Fp12 (Fp6 y2 0 0) 0) * inv (Fp12 0 cubicRoot)
```

4.3 Haskell implementation

Using the theory discussed in this chapter, we have implemented a version of Miller's algorithm compatible with both the BN128 and the BLS12-381 elliptic curves, and which validates the *json* outputs produced by **snarkjs** [15].

This Haskell implementation can be found in our repository [13].

Testing

Confidence in our code is bolstered by the following testing:

1. Property based testing of our pairing algorithm using *Quickcheck*.
2. Successfully tested validation of proof output produced by *snarkjs*.

Chapter 5

Zero-Knowledge programming: The state of art and ecosystem

The development of zero-knowledge applications often implies a solid understanding of mathematical concepts. In this sense, creating applications can be more complex to engineer than non-zero-knowledge applications, even for trivial examples. Looking at the state of the art of how zero-knowledge applications are built today, there are various ways to abstract this complexity and bridge any potential mathematical gaps. The main problem that a zero-knowledge developer must consider concerns the methods by which the setup and proofs are generated. There are two main approaches to deal with these processes: ASIC and CPU approach [16].

5.1 ASIC Approach

This method involves creating circuits that represent a specific algorithm, like an Application-Specific Integrated Circuit (ASIC), the circuit design is designed to address a precise problem.

A circuit is an arithmetical representation of an algorithm to be proved. It consists of inputs, outputs, and a set of computational gates. The inputs are variables and constants, and the gates perform arithmetic operations like modulo p addition, subtraction, multiplication and division under a finite field modulo a prime p .This circuit form a Directed Acyclic Graph (DAG), which can be further conceptualized as a polynomial expression. Thus in the ASIC approach the developers have to represent the algorithm of an application as an arithmetic circuit which are later used to generate the proof [3].

This standard method for creating zero-knowledge applications (zk-Apps) is computationally efficient and optimizes the processing of zk-SNARKs. Nonetheless, it demands from developers a higher proficiency in mathematics to effectively engineer solutions using this pa radigm. Therefore, an application using this method can achieve high levels of efficiency when generating the proof. However, the creation of the circuit is not a straightforward process for developers, and more measures must be taken to ensure that the circuit represents the algorithm correctly.Several well-known solutions use this methodology, offering varying degrees of abstraction to accommodate different developer skill sets.

5.1.1 Circom 2

Circom is a low-level programming language and toolset written in Rust, designed for creating arithmetic circuits and specifically designed for use with zk-SNARKs. With the Circom language, a developer can represent an algorithm as a circuit, which is then compiled and used to generate proofs. The compiler outputs the circuit representation as constraints and files in formats suitable for computing proof verification. The Circom compiler and its ecosystem of tools allow you to create, test, and generate zero-knowledge proofs for your circuits [17].

5.1.2 Noir

Noir is a Rust-based domain specific language (DSL) created to facilitate the development of zk-SNARKs. It was was designed with the intent to make the development of privacy-preserving cryptographic protocols more accessible and to abstract the complexity associated with zk-SNARKs, making it easier for developers to implement them. It provides a high-level abstraction over the low-level arithmetic circuits that are traditionally used to create zk-SNARKs, allowing developers to write in a more conventional programming style [18].

5.2 CPU Approach: Zero-Knowledge Virtual Machines

The CPU is characterized as a method where the circuit is a general purpose circuit for any algorithm just like a central processing unit. Instead of creating a circuit for each program a complete virtual machine can be implemented as a circuit. This virtual machine can represent a particuar set of instructions of some architecture and the components needed.

The CPU model is characterized by its use of a general-purpose circuit, which is capable of executing any algorithm, mirroring the versatility of a traditional central processing unit. Rather than designing a distinct circuit for each program, it's possible to implement a comprehensive virtual machine that process any algorithm. This encapsulated virtual machine is designed to process a specific set of instructions, reflecting the architecture of the system it's intended to emulate, along with all the necessary computational components. This approach streamlines the process, allowing for a single, adaptable circuit to accommodate a broad spectrum of programs and tasks. This zero-knowledge virtual machine accomplish two tasks:

1. Evaluation of the programs. This means that the zkVM outputs the result of a program given certain inputs. 2. Generate a proof of the program. This imply the generation of verifiable execution given the program.

This general-purpose method facilitates the evaluation and proof of programs compatible with the designated instruction set and architecture. It offers a trade-off between versatility and efficiency: while the proof generation process take a longer time than ASIC approach solutions, the need to design the proof for each specific program is eliminated. This trade-off enhances the accessibility of developing zero-knowledge applications by omitting the time-consuming and skill-intensive process of circuit design. Consequently, it simplifies the overall development procedure, making the creation of zk-Applications more straightforward for developers.

This zkVMs represent a specific set of instructions or primitives which can be used as an common low level interface for multiple languages. On the other hand, some zkVMs can support well-known languages such as eg. Rust, where any Rust based program can be executed and proved in the zkVM. This has the benefit of leveraging whole ecosystems because any Rust library can be converted in a verfiable program.

At the moment the are different zk-Virtual-Machines that implent various architectures.

5.2.1 Risc-Zero VM

The RISC Zero zero-knowledge virtual machine (zkVM) lets you prove correct execution of arbitrary Rust code. Utilizing the RISC architecture (Reduced Instruction Set Computer) it enables secure and private computations through the generation of zero-knowledge proofs, specifically zk-SNARKs. By allowing users to build zero-knowledge applications that leverage existing Rust packages, the RISC Zero zkVM makes it quick and easy to build verifiable software applications. The virtual machine streamlines the production of these proofs to foster practicality in diverse applications. It is built to assist developers with tools that simplify the end-to-end process of developing privacy-preserving applications, from creation to deployment [19].

5.2.2 Aleo snarkVM

The Aleo SnarkVM is a core component of the Aleo platform, which focuses on providing a development framework for private applications using zero-knowledge proofs. The SnarkVM facilitates the creation and verification of these proofs, enabling developers to build applications that execute transactions confidentially. It incorporates a tailored design to support the execution of Aleo programs while maintaining a secure environment that adheres to privacy standards. The SnarkVM's architecture is built to be compatible with the specific requirements of Aleo's privacy-centric blockchain, aiming to provide an efficient and user-friendly experience for developers interested in creating secure, scalable, and private digital applications. It stands as a fundamental building block for the Aleo platform, contributing to the platform's goal of blending blockchain functionality with stringent data protection [20].

5.3 Zero-Knowledge Tools in Cardano

The Cardano ecosystem is currently in the nascent stages of integrating Zero-Knowledge proof tools. Among these emerging tools is the ZKFold Symbolic language, which is in development and takes inspiration from the ASIC model. ZKFold Symbolic is a high-level functional programming language, derived from a Haskell subset, designed for crafting zero-knowledge smart contracts [21]. This language simplifies the creation of smart contracts by allowing them to be compiled into arithmetic circuits suitable for various Zero-Knowledge Proofs, supported fully by Haskell Language Server (HLS). Its syntax bears resemblance to PlutusTx, and it's developed to eliminate the need for in-depth Zero-Knowledge Proof expertise, as well as removing execution unit and script size constraints. It enables developers to articulate contract logic clearly without resorting to cumbersome optimizations. ZKFold Symbolic is particularly beneficial for scalable decentralized applications, offering reasonable network fees for substantial transactions, including those with extensive data inputs and outputs. It also simplifies the development of complex smart contracts, from on-chain voting mechanisms to interoperability bridges between different blockchains. The initial release of ZkFold Symbolic is planned to feature four main components: an arithmetic circuit compiler, a Cardano smart contract library, an on-chain verification script for ZkFold Symbolic contracts, and additional utility code necessary for generating Zero-Knowledge Proofs.

Chapter 6

Implementation proposal for Modulo-P

The goal of Modulo-P is to bring zero-knowledge proofs to the Cardano ecosystem. Implementing zero-knowledge proofs in smart contracts is somewhat incompatible with the requirements of a blockchain, at least in a general-use blockchain.

Esta circunstancia hace que muchas cadenas tenga que desarrollar en su funcionalidad interna funciones criptográficas de pareamiento como ha hecho Ethereum y como está realizando Cardano en la versión de Plutus V3. De hecho, muchos estudios apuntan a que la escalabilidad en las cadenas de bloques se potenciará por el aspecto sucinto de las pruebas de conocimiento cero.

This circumstance leads many blockchains to have to develop cryptographic pairing functions in their internal functionality, as Ethereum has done and as Cardano is doing in the Plutus V3 version. In fact, many studies suggest that scalability in blockchains will be enhanced by the succinct nature of zero-knowledge proofs.

Simultaneously, efforts will be made, to the extent possible, to implement the algorithms in Plutus. The goal of this project is to establish a standard framework for executing zero-knowledge proofs on Cardano and to serve as an educational resource to promote the use of zero-knowledge proofs.

6.1 Implementation

For the Catalyst project, Modulo-P aims to achieve the following deliverables as part of the development milestones:

- The implementation of the BLS6-6 curve in the Sagemath mathematical language as a didactic example of zero-knowledge proof.
- The implementation of the BLS12-381 curve in Plutus will help us under-

stand if Plutus is sufficient as a general-purpose language for conducting zero-knowledge proof algorithms.

- If the Plutus algorithm fulfills its purpose, versions will be developed in Aiken as a reference for the current state of the art in Plutus code optimization.
- In the event of not achieving the implementation in Plutus, an adjustment will be made in Hydra to have the possibility of running Plutus V3, and we will carry out zero-knowledge proof tests with the new cryptographic primitives of the new version.
- Development of tools that enable the integration of results from other zero-knowledge tools from various ecosystems, such as Ethereum, including SnarkJS, Circom, etc...
- Demonstration and publication of the code in a Mastermind game repository using the aforementioned technologies.

6.2 Using SnarkJS proofs with Modulo-P pairing algorithm

We demonstrated in our repository [13] how a proof generated in SnarkJS can be verified in Haskell using our algorithm. The files proof.json and verification_key.json contain a proof for the example circuit from the SnarkJS tutorial of the repository [15]. Next, we apply our Miller's Algorithm to validate the Groth16 equality:

```
eAB :: Fp12
eAB = pairing piA piB

eAlphBeta :: Fp12
eAlphBeta = pairing vkAlpha vkBeta

vkI :: G1
vkI = vkI0 <> ecExp vkI1 168932

eIGamma :: Fp12
eIGamma = pairing vkI vkGamma

eCDelta :: Fp12
eCDelta = pairing piC vkDelta


lhs :: Fp12
lhs = eAlphBeta * eIGamma * eCDelta
```

```
proof :: Bool
proof = lhs == eAB
```

When we execute the proof in the cabal repl, we can verify that the equality is true. This confirms that the use of SnarkJS is compatible with Plutus and can be employed in a Cardano smart contract.

6.3 Proof of Concept: A Dapp with ZK-Verifiable Contracts on Hydra.

As part of our proof of concept, we developed a Dapp based on the game Mastermind. Mastermind is a two-player game where one party provides hints about a secret code, and the other player tries to guess the code to win. This game dynamics proved ideal for applying zero-knowledge proofs because, in each round, the hints need to be demonstrated without being revealed. To achieve this, we implemented the Groth 16 protocol on the Cardano network. This protocol utilizes a ZK-Snark proof, which is verified on-chain through a smart contract. Initially, we attempted to generate transactions on the mainnet but encountered limitations due to the resource constraints mentioned previously. This justified the subsequent use of the Hydra protocol to process computationally intensive transactions. This milestone was a key breakthrough in our project and, in our opinion, opens up promising possibilities for the development of complex Dapps.

6.4 The Mastermind Game

Originally, Mastermind is a two-player board game that involves guessing a secret combination of colors. On one side, there is the “code maker” who begins the game by generating a secret combination, which the “code breaker” must guess within a certain number of attempts to win. In each incorrect attempt by the “code breaker”, the “code maker” provides a clue: a black peg for each element of the combination that is the correct color and in the correct position, and a white peg for each element that is the correct color but in the wrong position [22]. For example, if the secret combination is RRVA (red-red-green-blue) and the “code breaker” tries to guess with ”MRVR” (purple-red-green-red), the clue given by the “code maker” would be:

- Two black pegs (indicating that the red and green pegs are both the correct color and in the correct positions).
- One white peg (indicating that the final red peg is the correct color but in the wrong position).
- No pegs for the purple peg, as it is neither the correct color nor in the correct position.

Lately, in our adaptation as a Dapp, the “code breake” wins if they guess the secret combination within a certain number of attempts. It has become a game of gambling, where if the “code breaker” wins, the smart contract allows them to claim the Ada (cryptocurrency), and if they lose or abandon the game after a certain time, the “code maker” can claim the Ada. It is worth noting that in the game design, both the number of colors to guess and the number of attempts before losing can be arbitrary. The crucial factor for fair gameplay is that the clues should not include any traps or misleading information. This is where the technology of zero-knowledge proofs comes into play. It ensures that the clues provided by the “code maker” are honest without revealing the secret combination. It is essential to explain the significance of this in order to understand the functionality and scope of the project. By utilizing zero-knowledge proofs, the integrity of the game is upheld, allowing for a fair and secure gaming experience where the secrecy of the secret combination is maintained

Bibliography

[1] Jens Groth. *On the Size of Pairing-based Non-interactive Arguments.* Cryptology ePrint Archive, Paper 2016/260, 2016. `https://eprint.iacr.org/2016/260`.

[2] Consensys. *Introduction to zk-SNARKs.* Consensys Blog, March 27, 2017. `https://consensys.io/blog/introduction-to-zk-snarks`.

[3] Dan Boneh. *ZK Whiteboard Sessions - Module One: What is a SNARK?.* [Video]. YouTube. Published on 20/087/22. `https://www.youtube.com/watch?v=h-94UhJLeck`.

[4] Mina protocol. *ZK Tech You Should Know — Part 1: SNARKs & STARKs.* [Video]. YouTube. Published on 21/10/2022. `https://www.youtube.com/watch?v=li0n-n4lqfA`.

[5] Barreto, P. and M. Naehrig, *Pairing-Friendly Elliptic Curves of Prime Order*, DOI 10.1007/11693383_22, Selected Areas in Cryptography pp. 19-331, 2006, https://doi.org/10.1007/11693383_22.

[6] Barreto, P., Lynn, B., and M. Scott, *Constructing Elliptic Curves with Prescribed Embedding Degrees*, DOI 10.1007/3-540-36413-7_19, Security in Communication Networks pp. 257-267, 2003, https://doi.org/10.1007/3-540-36413-7_19.

[7] Sakemi, Y., Kobayashi, T., Saito, T., Wahby, R. (Eds.), *Pairing-Friendly Curves*, Internet-Draft draft-irtf-cfrg-pairing-friendly-curves-11, November 6, 2022. DOI: 10.17487/RFC0000, Internet Engineering Task Force, Informational, Expires: May 10, 2023. Available at: https://www.ietf.org/archive/id/draft-irtf-cfrg-pairing-friendly-curves-11.txt.

[8] Bowe, Sean, BLS12-381: New zk-SNARK Elliptic Curve Construction (2017).

[9] Taechan Kim and Razvan Barbulescu, *Extended Tower Number Field Sieve: A New Complexity for the Medium Prime Case*, Cryptology ePrint Archive, Paper 2015/1027 (2015).

[10] Hoffsten, J., J. Pipher & J. H. Silverman, *An Introduction to Mathematical Cryptography*, second edition, Springer (2014).

[11] Richter, M., *The MoonMath Manual to zk-SNARKs*, v. 1.1.1, Least Authority.

[12] https://github.com/Modulo-P/miller-first-study

[13] https://github.com/Modulo-P/groth-pairing

[14] https://github.com/ethereum/EIPs/blob/master/EIPS/eip-197.md

[15] https://github.com/iden3/snarkjs

[16] grjte. *ZK Whiteboard Sessions – Module Seven: Zero Knowledge Virtual Machines (zkVM) with grjte*. [Video]. YouTube. URL: https://www.youtube.com/watch?v=GRFPGJW0hic.

[17] Circom 2 Documentation. *Circom 2 Documentation.* Available at https://docs.circom.io.

[18] Aztec Labs. *Introducing Noir: The Universal Language of Zero Knowledge.* Medium, [Oct 6, 2022]. URL: https://medium.com/aztec-protocol/introducing-noir-the-universal-language-of-zero-knowledge-ff43f38d86d9.

[19] RISC Zero Developer Docs. *Introduction.* Available at https://dev.risczero.com/api.

[20] Collin Chin. *Aleo instructions and snarkVM.* Welcome to Aleo Documentation, Last updated on Sep 21, 2023. Available at https://developer.aleo.org/aleo.

[21] zkFold. *Zero Knowledge Smart Contracts and zk-Rollups on Cardano.* Available at https://zkfold.io.

[22] Mordecai Meirowitz. *Mastermind (board game).* Wikipedia, The Free Encyclopedia. Years active 1970 to present, Board game genre. URL: https://en.wikipedia.org/wiki/Mastermind_(board_game).