AOHashes

AO privacy-friendly primitives testing over Dusk-Plonk ZK library

Filippo Merlo

March 2025

Contents

Chapter 1

Introduction

In few years, the interest in the cryptographic community has grown a lot towards the Zero-Knowledge (ZK) proofs. In fact, the goal of ZK proofs is to provide a way to prove the correctness of a computation without revealing any information about the data involved in the computation itself. Among the types of ZK proofs, there are the non-interactive ones, which are used in environments where prover and verifier are not able to communicate with each other, like in blockchains. In this paper, the ZK system used for the proofs is the PLONK system, which belongs to the SNARK[1] family, and has its own structure for the computation of the proofs based initially on arithmetic circuits, which are translated into constraints ones and finally evaluated as polynomials. This structure allows the use of more powerful gates, while maintaining the computational complexity low. However, having an internal structure like this means that old cryptographic primitives are not suitable and adapted for these computations, leading to huge losses in terms of performance and efficiency. Therefore, this strong interested in ZK has shaped the development of new cryptographic primitives towards the so-called Arithmetization-Oriented (AO) ones, aiming to provide better and more efficient solutions that could exploit as much as possible the power of ZK proofs, by maintaining a complexity as low as possible from a constraints' perspective (i.e. additions and multiplications). For this reason we wrote this paper with the aim of shading some light on the different performances of the latest and most promising AO primitives proposed in these last years.

[1] Succinct Non-interactive ARguments of Knowledge

Chapter 2

Overview

Talk about AO
Curve chosen BLS12-381

2.1 Primitives

The primitives that have been implemented and tested in this project are:

- `GMiMC`
- `Poseidon`
- `Rescue`
- `Rescue-Prime`
- `Griffin`
- `Anemoi`
- `Arion`

We can split them into two categories: in the *first category* we found those that have been designed for maintaining the **degree of polynomials** as **low** as possible, while having a high number of rounds to achieve a minimum level of bits security, which are `GMiMC`, `Poseidon`, `Rescue` and `Rescue-Prime`, while in the *second* one has been used the opposite strategy, i.e. achieving efficiency maintaining a **low number of rounds**, but increasing exponentially the polynomial degree with the introduction of multiplicative inverses, and these primitives are `Griffin`, `Anemoi` and `Arion`.

2.1.1 GMiMC

The first function implemented in this project is GMiMC[1], a hash function based on unbalanced Feistel networks with a low multiplicative complexity that suits ZK-SNARK applications. [1].

2.1.2 Poseidon

Poseidon is a **sponge function** based on the hashing designed strategy called `HADES`, a round function composed of both *full* and *partial* rounds. The decision to opt to this division is to lighten the computational cost of `HADES`, allowing it to maintain a minimum level of performance even with a high number of rounds. The only difference between a partial and a full round is just that the S-Box[2] is applied only to **one** element of the state, while on the

[1] Generalized MiMC
[2] Substitution Box

other hand is applied to each element of the state.

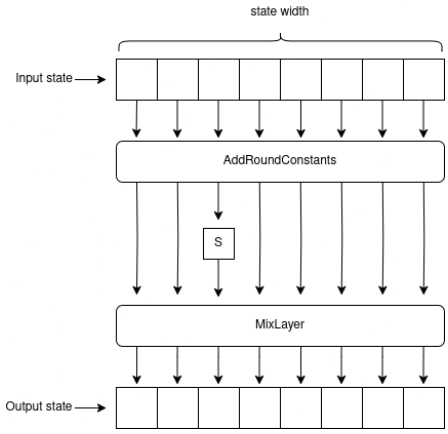
NOTE:
It is important to notice that `HADES` can be adjusted to use only full rounds and achieving in this way a greater level of security. However, the opposite, i.e. using only partial rounds, is strongly discouraged because to achieve a minimum level of security a minimal amount of full rounds are still required, where the exact value depends on the size of the state.

A single round is composed of the following concatenation of functions:

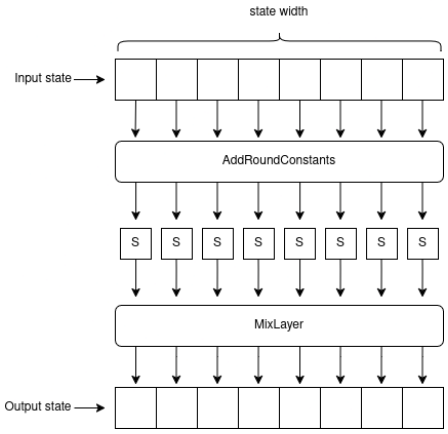
- *AddRoundConstant*: a constant that depends on the round is added to each value of state;
- *S-Box*: the input is exponentiated to an exponent α where $\alpha \geq 3$ and also α is co-prime with $p - 1$. This function is applied singularly to each/one element of the state, depending on the type of round;
- *MixLayer*: matrix-vector multiplication between a fixed matrix and the state, where the matrix is a square MDS[3] matrix chosen such that no subspace trail with inactive/active S-Boxes can be set up for more than $t - 1$ rounds, with t the size of the state.

The following (figure 2.1) are the visual representations of the internal structure of the `HADES` function:

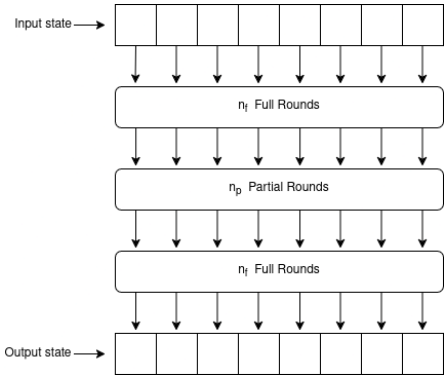
[3]Maximum Distance Separable



(a) `HADES` partial round



(b) `HADES` full round



(c) Complete `HADES` function

Figure 2.1

In our case, because working over the BLS12-381 curve, some commonly used instantiations of Poseidon are shown here in Table 2.1 :

t state width	3	5
n_f full rounds	8	8
n_p partial rounds	57	60

Table 2.1: Posedon instances

2.1.3 Rescue

2.1.4 Rescue-Prime

2.1.5 Griffin

2.1.6 Anemoi

2.1.7 Arion

2.2 Plonk

Plonk and plonk arithmetic

2.3 Implementation

All the primitives mentioned above have been implemented completely in Rust language

2.3.1 Execution

2.3.2 Dusk-Plonk

Chapter 3

Tests

3.1 Environment

3.2 Constraints

3.3 Plain execution

3.4 Proof generation

3.5 Proof verification

Chapter 4

Conclusion

Chapter 5

References

[1] Martin R. Albrecht. *Feistel Structures for MPC and More*. URL: https://eprint.iacr.org/2019/397.

[2] *GMiMC homepage*. URL: https://hadesmimc.github.io/gmimc/.