Anemoi: Exploiting the Link between Arithmetization-Orientation and CCZ-Equivalence

Clémence Bouvier[1,2], Pierre Briaud[1,2], Pyrros Chaidos[3], Léo Perrin[2], Vesselin Velichkov[4,5]

[1]Sorbonne University, France
[2]Inria, France
[3]National & Kapodistrian University of Athens, Greece,
[4]University of Edinburgh, Scotland
[5]Clearmatics, England

Abstract. Advanced cryptographic protocols such as Zero-knowledge (ZK) proofs of knowledge, widely used in cryptocurrency applications such as Bitcoin, Ethereum and Zcash, demand new cryptographic hash functions that are efficient not only over the binary field $\mathbb{F}_2$, but also over large fields of prime characteristic $\mathbb{F}_p$. This need has been acknowledged by the wider community and new so-called *Arithmetization-Oriented* (AO) hash functions have been proposed in response, e.g. MiMC-Hash, Rescue-Prime and Poseidon to name a few. In this paper we propose `Anemoi`: a new family of ZK-friendly AO hash functions.

The main features that set `Anemoi` apart from other such families are that 1) it is designed to be efficient within multiple proof systems (e.g. Groth16, Plonk, etc.), 2) it contains dedicated functions optimised for specific applications (namely Merkle tree hashing and general purpose hashing), 3) has competitive performance e.g. about a factor of 2 improvement over Poseidon and Rescue-Prime in terms of R1CS constraints, and a 10%-28% improvement over a highly optimized Poseidon implementation in Plonk constraints. On the theoretical side, `Anemoi` pushes further the frontier in understating the design principles that are truly entailed by arithmetization-orientation. In particular, we identify and exploit a previously unknown relationship between CCZ-equivalence and arithmetization-orientation. In addition, we propose two new standalone components that can be easily reused in new designs. One is a new S-box called `Flystel`, based on the well-studied butterfly structure, and the second is Jive – a new mode of operation, inspired by the "Latin dance" symmetric algorithms (Salsa, ChaCha and derivatives).

Keywords: Anemoi · Flystel · Jive · Arithmetization-oriented · Hash functions · CCZ-equivalence · Plonk · R1CS · Merkle tree · Zero-knowledge · Arithmetic circuits

1 Introduction

In recent years we have seen a rapid surge of interest in the practical application of an old cryptographic construction known as a zero-knowledge (ZK) proof of knowledge. It is a protocol which, when followed, allows a prover P to convince a verifier V that a certain statement x is true without revealing any additional information beyond the fact that it is verifiably correct. Such a piece of information may, for example, be that the result of a specified complex computation is 1. With a ZK protocol, V can verify that the result of this computation is correct without having to perform the computation herself. In fact, to

verify correctness V does not even need to know some of the details of the computation e.g. its intermediate values or any potentially secret inputs.

ZK proof systems have been introduced with the seminal work of Micali, Goldwasser and Rackoff back in 1989 [GMR89]. Traditionally, ZK protocols were deployed to allow the prover to keep some elements of the computation secret (e.g. a private key), but increasingly they can also be used to relieve the verifier from the necessity to perform an expensive computation for which it may not have sufficient resources (in terms of space as well as computational power). The increased interest in such protocols today is largely driven by the latest advancements in digital currencies such as Bitcoin, Ethereum, etc. In particular, ZK proofs make it possible to add privacy on the public blockchain (e.g. Zcash [BSCG+14]) and to perform off-chain computation verifiable by network nodes with significantly limited resources.

The computation performed by P and verified by V in a ZK proof, is expressed as an arithmetic circuit composed of *gates* (algebraic operations e.g. multiplication or addition) connected by *wires*. The quantities that pass over the wires and are operated on by the gates are elements of an algebraic field $\mathbb{F}_q$ for some value $q \geq 2$.

Cryptographic hash functions are fundamental to the practical application of many ZK protocols: They are often used for testing membership of some element(s) by the means of a Merkle tree. They can also be used as part of the ZK protocol itself e.g. by compressing multiple public inputs to a single hash image. The protocol then has a reduced input footprint, and the collision resistance of the hash function implies that security is not impacted. This is relevant in proof systems where the verifier's costs are proportional to the number of public inputs such as Groth16 [Gro16].

Modern cryptographic hash functions such as SHA2, SHA3 and BLAKE are designed over vector spaces of the binary binary field $\mathbb{F}_2$ (i.e. they work over bits), while ZK protocols often operate over $\mathbb{F}_q$ for a large q – that is often a prime number. Therefore the efficient execution of ZK protocols in applications such as Bitcoin and Ethereum, that aim to process millions of transactions per day, imposes the need for new hash functions designed to be natively efficient in $\mathbb{F}_q$ – the so-called *Arithmetization-Oriented* (AO) designs. Furthermore, the operation for which their efficiency is the most crucial is not their *evaluation*, but rather their *verification*. Concretely, while the cost of evaluating $y = F(x)$ given x remains important, the step with the harshest constraints is a verification: given both x and y, checking if y is indeed equal to $F(x)$ should be "efficient", where the exact meaning of "efficient" depends on the proof system considered.

The need for new arithmetization-oriented hash functions has been acknowledged by both researchers and engineers. As a result, the past couple of years have seen a surge of new proposals of hash functions that operate natively in $\mathbb{F}_q$ for q prime, and that allow an efficient verification: MiMC-Hash [AGR+16], Poseidon [GKR+21b], Rescue-Prime [AAB+20, SAD20], and ReinforcedConcrete [BGK+21] to name a few.

Constraints and Performance Metrics. There are various proof systems, each with slightly different efficiency metrics. Most systems can be used to efficiently describe the operation of an arithmetic circuit, but there exist considerable practical differences. Specifically, the exact format of the constraints accepted by the system, as well as the cost of each. In this work, we evaluate performance in two of the most-used constraint systems.

Various zero-knowledge proof systems operating on arithmetic circuits, such as the Groth16 [Gro16] ZK-SNARK [BCG+13] or Bulletproofs [BBB+18], use the R1CS (Rank-1 Constraint Satisfaction) systems. For this metric, the circuit is then described as a set of affine polynomials such that the complexity is mainly given by the number of multiplications in the resulting equation system.

We also consider Plonk [GWC19], which is both a ZK-SNARK proof system and an artithmetization contraint system. As a proof system Plonk has a universal and updateable

setup. Compared to R1CS, the Plonk arithmetization system does not directly enable zero-cost additions to produce the operands of a multiplication. However, it has been augmented with a number of performance-enhancing features, including custom gates [GW19] (as opposed to addition and multiplication only), additional wires, and lookup tables [GW20].

Our Contributions. In this work, we join the arithmetization-oriented hash function race and propose `Anemoi` – a new family of efficient and secure hash functions operating over prime and binary fields. The main features that set `Anemoi` apart from other similar designs are as follows.

1. It is designed to be **efficient within multiple proof systems** (e.g. Plonk [GWC19], Groth16 [Gro16], etc.) as opposed to being optimized for a particular proof system as e.g. ReinforcedConcrete, which is optimized specifically for Plonk [GWC19].

2. It contains **dedicated functions optimized for specific applications**, namely general purpose hashing *via* a sponge construction, and Merkle tree hashing which is handled via our new mode, Jive.

3. It has **competitive performance** compared to other designs e.g. about a factor of 2 improvement over Poseidon and Rescue-Prime in terms of R1CS constraints, and an even higher improvement in standard 3-wire plonk. In a more optimized Plonk setting we obtain 30%-40% improvement over the state of the art Poseidon optimizations of Ambrona et al. [ASTW22] using our Jive_2 mode of operation, and about 10%-28% improvement if we also extend the use of Jive_2 to Poseidon.

4. It has **limited reliance on randomness**: while dense round constants need to be generated to ensure resilience against algebraic attacks, we aim to limit our reliance on pseudo-randomly generated components in order to ease both implementation and cryptanalysis. It also limits the moldability [DP19] of the designs, which will hopefully increase the confidence in our algorithms.

5. It has a **consistent design**: all algorithms are built using very similar principles and round functions regardless of the field size or type.

6. Contains **standalone components** that can be easily reused in new AO designs. One is a new S-box called `Flystel`, based on the well-studied butterfly structure, and the second is Jive – a new mode of operation, inspired by the "Latin dance" symmetric algorithms Salsa, ChaCha and derivatives.

7. Last, but not least, `Anemoi` is based on an **improved understanding of the core foundation of arithmetization-orientation**. More precisely, we identify and exploit a previously unknown relationship between the so-called *CCZ-equivalence* (defined below) and arithmetization-orientation.

The latter insight is at the heart of the new family of functions of $\mathbb{F}_q^2$ that we called `Flystel`. They are defined over all fields, and each of them has two variants: an *open* `Flystel`, which is a high degree permutation; and a *closed* `Flystel`, which is a low-degree function. They are closely related to the *butterfly structure*, investigated in [PUB16, LTYW18, CDP17], hence our terminology of *open* and *closed* structures. The high degree permutation has the properties we would expect from a good cryptographic S-box, and the low-degree function can be efficiently verified across proof systems. Because of their CCZ-equivalence, we can reap the benefits of the peculiarities of both objects: this property allows us to use the permutation in the round function of `Anemoi`, and the function during the verification.

Outline. The structure of the paper is as follows. We begin with providing some necessary theoretical background in Section 2. Section 3 describes the two modes of operation used in `Anemoi`, namely the Sponge structure for the random oracle functionality, and the Jive mode for the Merkle compression function functionality. The `Flystel` structure is proposed and studied in Section 4, followed by the full description of the `Anemoi` family in Section 5. The security analysis of our algorithms is given in Section 6 and performance benchmarks in Section 7. The exposition concludes with Section 8. More technical details on the security analysis and performance estimates are provided in appendix, along with some reference implementations.

2 Theoretical Background

In what follows, q is an integer corresponding to the size of the field $\mathbb{F}_q$, so that $q = p$ for some prime number p or $q = 2^n$. As usual, the symbols "$+$" and "$\times$" denote respectively the addition and the multiplication over $\mathbb{F}_q$. We also let $m \geq 1$ be an integer corresponding to the number of field elements we are operating on. We denote $\langle a, b \rangle$ the usual scalar product of $a \in \mathbb{F}_q^m$ and $b \in \mathbb{F}_q^m$ which is such that $\langle a, b \rangle = \sum_{i=0}^{m-1} a_i b_i$.

Below, we consider a function $F : \mathbb{F}_q^m \to \mathbb{F}_q^m$, and recall some of the concepts behind the use and analysis of functions to design symmetric cryptographic primitives. We recall first the definition of their differential and linear properties, and then that of *CCZ-equivalence*. While the latter was seldom used in practice so far, it plays a crucial role in our work.

Differential Properties. The *Difference Distribution Table (DDT)* of function F is the two dimensional array δ_F defined by $\delta_F[a, b] = \#\{x \in \mathbb{F}_q^m | F(x + a) - F(x) = b\}$. The maximum value of $\delta_F[a, b]$ for $a \neq 0$ is the *differential uniformity* [Nyb94] of F. If $p > 2$, then the optimal value is a differential uniformity of 1, in which case the function is called *Perfect Nonlinear (PN)*. Functions with a differential uniformity of 2 are called *Almost Perfect Nonlinear (APN)*, and this is the optimal case when $p = 2$.

Linear Properties. While a general formula that works both when q is a power of two and a prime can be given, it is simpler to treat the two cases separately, especially given that the reader is probably familiar with the case of characteristic 2. If $q = 2^n$, then the Walsh transform of the component $\langle b, F \rangle : \mathbb{F}_q \to \mathbb{F}_2$ for any $b \in \mathbb{F}_q \backslash \{0\}$ is $\mathcal{W}_{\langle b,F \rangle}(a) = \sum_{x \in \mathbb{F}_2^m} (-1)^{\langle a,x \rangle + \langle b,F(x) \rangle}$.

Otherwise, when $q = p$ the Fourier transform of a function $f : \mathbb{F}_p^m \to \mathbb{F}_p$ is the function $\mathcal{W}_f : \mathbb{F}_p^m \to \mathbb{C}$ such that

$$\mathcal{W}_f(a) = \sum_{x \in \mathbb{F}_p^m} \exp\left(\frac{2\pi i \left(\langle a, x \rangle - f(x)\right)}{p}\right).$$

For a vectorial function $F : \mathbb{F}_q^m \to \mathbb{F}_q^m$, we consider the Fourier transform of each of its components, i.e. of all the linear combinations $\langle b, F \rangle$. We then investigate the value of $\mathcal{W}_{\langle b,F \rangle}(a)$. As established in [BSV07], a linear attack against F becomes possible when the squared modulus of $\mathcal{W}_{\langle b,F \rangle}(a)$ for some $a, b \in \mathbb{F}_q^m$ is high enough. Roughly speaking, the data complexity of a linear attack is around $1/|\mathcal{W}_{\langle b,F \rangle}(a)|^2$. Functions for which the maximum modulus is the lowest are called *bent*. It is well known (see for instance [CM97]) that a function mapping $\mathbb{F}_q$ to itself is PN if and only if it is bent.

CCZ-Equivalence [CCZ98]. Let $F : \mathbb{F}_q^m \to \mathbb{F}_q^m$ and $G : \mathbb{F}_q^m \to \mathbb{F}_q^m$ be two functions. They are *affine-equivalent* if there exists two affine permutation $\mu : \mathbb{F}_q^m \to \mathbb{F}_q^m$ and $\eta : \mathbb{F}_q^m \to \mathbb{F}_q^m$

such that $F = \eta \circ G \circ \mu$. This can alternatively be written using the *graphs* of these functions:

$$\Gamma_F = \underbrace{\left\{ (x, F(x)) \mid x \in \mathbb{F}_q^m \right\}}_{\text{graph of } F} = \mathcal{L}(\Gamma_G) = \left\{ \mathcal{L}\left(x, F(x)\right) \mid x \in \mathbb{F}_q^m \right\},$$

where $\mathcal{L}$ is the affine permutation defined by $\mathcal{L}(x, y) = \big(\eta(x), \mu^{-1}(y)\big)$. If we allow $\mathcal{L}$ to be any affine permutation,[1] we obtain *CCZ-equivalence*.

Definition 1 (CCZ-Equivalence)**.** Let F and G be functions of $\mathbb{F}_q^m$. We say that they are *CCZ-equivalent* if there exists an affine permutation $\mathcal{L} : (\mathbb{F}_q^m)^2 \to (\mathbb{F}_q^m)^2$ such that $\Gamma_F = \mathcal{L}(\Gamma_G)$.

An important property of CCZ-equivalence that is instrumental in our work is that it preserves the differential spectrum and the squared Walsh coefficients. In other words, all functions within the same CCZ-equivalence class share the same differential and linear properties and hence offer the same resilience against differential and linear attacks. It also means that it is sufficient to investigate these properties for a single member of a CCZ-equivalence class.

Another relevant property of CCZ-equivalence is that it does *not* preserve the degree of the function. In fact, there are known cases where a low-degree function is CCZ-equivalent to a higher-degree one. It is most notably the case of the so-called *butterfly structure*, originally introduced in [PUB16], and then further generalized in two different ways in [CDP17] and [LTYW18].

3 Modes of Operation

In advanced protocols, hash functions are used for two purposes. The first is to emulate a random oracle, in particular to return the "fingerprint" or digest of a message of arbitrary length. The idea is that this fixed length digest is simpler to sign than the full message. The second use is as a compression function within a Merkle-tree: in this case, the hash function H is used to map to input of size n to an output of size n, and the security of the higher level scheme relies on its collision resistance. While a general purpose hash function like `SHA-3` [BDPA13, Dwo15] or an arithmetization-friendly one can safely be used for both uses, for improved efficiency we chose to use a full hash function only for the random oracle case (Section 3.1). Indeed, the specific constraints of the Merkle-tree case can be satisfied more efficiently using a dedicated structure that remains permutation-based, and which we introduce in Section 3.2. SAGE implementations of both modes are provided in Appendix B.

3.1 Random Oracle: the Sponge Structure

A random oracle is essentially a theoretical function that picks each output uniformly at random while keeping track of its previous outputs in order to remain a deterministic function. The sponge construction is a convenient approach to try to emulate this behaviour. First introduced by Bertoni et al. in [BDPVA07], this method was most notably used to design SHA-3. It is also how most arithmetization-oriented hash functions have been designed, e.g. Rescue-Prime, gMiMC-Hash, Poseidon [GKR+21a], and ReinforcedConcrete. Such hash functions can easily be tweaked to be turned into eXtendable Output Function (XOF) [Dwo15] should the need arise.

[1] Starting from a given function F, applying any affine permutation of $\mathbb{F}_q^2$ to its graph is unlikely to yield the graph of another function G. Indeed, this would require that the left hand side of $\mathcal{L}(x, F(x))$ takes all the values in $\mathbb{F}_q$ as x goes through $\mathbb{F}_q$, which is a priori not the case. A mapping $\mathcal{L}$ that does yield the graph of another function is called "admissible", a concept that was extensively studied in [CP19].

The overall principle of the sponge construction is best explained by the diagram in Figure 1. In this paper, we slightly modify the original approach to operate on elements of $\mathbb{F}_q$ instead of $\mathbb{F}_2$. The main component of the structure is a permutation P operating on $\mathbb{F}_q^{r+c}$, where both r and c are non-zero integers. The *rate* r is the size of the *outer part* of the state, while c is the *capacity* and corresponds to the size of the *inner part* of the state. The digest consists of h elements of $\mathbb{F}_q$. Then, to process a message m consisting of elements of $\mathbb{F}_q$, we apply the following operations.

Padding. A basic padding works as follows: append $1 \in \mathbb{F}_q$ to the message followed by enough zeroes so that the total length is a multiple of r, and then divide the result into blocks m_0,...,$m_{\ell-1}$ of r elements.

However, with this approach, we may end up using one more call to P in the case where the length of the message was already a multiple of r. A more efficient approach is presented in [Hir16]: if the length of the message is already a multiple of r, then we do not append yet another block(s) to it. Instead, we add a constant in the capacity before squeezing. This is summarized as the addition of σ which is equal to 0 if the message length is not a multiple of r, and to 1 otherwise (see Figure 1). This variant also has the advantage of gracefully handling the case where $r = 1$.

Absorption. For each message block m_i, we add it into the outer part of the state, and then apply F on the full state.

Squeezing. We extract the r elements in the outer part of the state to generate the first r elements of the digest. If the $h > r$, then we apply F and then extract another r elements until the desired length is reached.

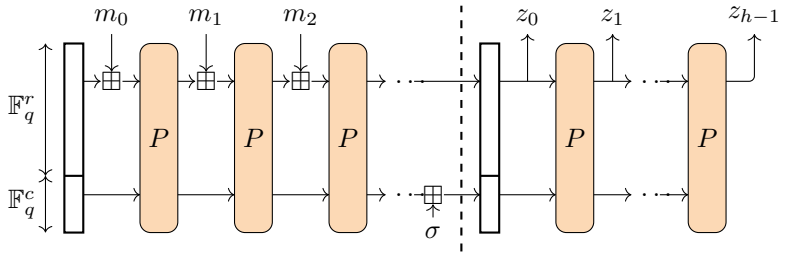


Figure 1: *Sponge construction with the modification of [Hir16].*

The security of a sponge hinges on the properties of its permutation. Informally, the only special property of the permutation should be the existence of an efficient implementation. Its differential, linear, algebraic, etc. properties should be similar to those expected from a permutation picked uniformly at random from the set of all permutations.

Following a *flat sponge claim* [BDPVA07], the designers of such an algorithm can essentially claim that any attack against it will have a complexity equivalent to at least $q^{c/2}$ calls to the permutation (provided that $h \geq c$). Thus, a flat sponge claim states that a sponge-based hash function provides $nc/2$ bits of security.

3.2 Merkle Compression Function: the Jive Mode

One of the main use cases for an arithmetization-oriented hash function is as a compression function in a Merkle tree. This case could be easily handled using a regular hashing mode, such as the sponge structure discussed above. However, due to the specifics of this use case, it is possible to use a more efficient mode.

In a Merkle tree, the elements considered are in $\mathbb{F}_q^m$, where m is chosen so that $m\lfloor \log_2 q \rfloor \geq n$, where n is the intended security level. We then need to hash two such

elements to obtain a new one. As a consequence, unlike in the usual case, the input size is fixed, and is equal to exactly twice the digest size. Given a permutation of $(\mathbb{F}_q^m)^2$, we can thus construct a suitable hash function by plugging it into the following mode.

Definition 2 (Jive)**.** Consider a permutation P defined as follows:

$$P : \begin{cases} (\mathbb{F}_q^m)^b & \to (\mathbb{F}_q^m)^b \\ (x_0, ..., x_{b-1}) & \mapsto (P_0(x_0, ..., x_{b-1}), ..., P_{b-1}(x_0, ..., x_{b-1})) \end{cases},$$

so that it operates on bm of elements of $\mathbb{F}_q$, where $P_i(x_0, \ldots, x_{b-1}) : 0 \leq i < b$ refers to the i-th element in $\mathbb{F}_q^m$ of the output $P(x_0, \ldots, x_{b-1})$ from P. The mode Jive is built from P by defining the following one way function $\mathsf{Jive}_b(P)$:

$$\mathsf{Jive}_b(P) : \begin{cases} (\mathbb{F}_q^m)^b & \to \mathbb{F}_q^m \\ (x_0, ..., x_{b-1}) & \mapsto \sum_{i=0}^{b-1} (x_i + P_i(x_0, ..., x_{b-1})) \end{cases}.$$

This approach can be seen as a permutation-based variant of the Davies-Meyer mode which, like the latter, crucially relies on a feed-forward to ensure one-wayness. Alternatively, it can be interpreted as a truncated instance of the mode used in the "Latin dance" ciphers ChaCha and Salsa [Ber08], which is also based on a public permutation combined with a feedforward. Incidentally, we called it Jive after another Latin dance.

If used inside a Merkle tree, this mode can save some computations. For example, in the case where the fan-in b is equal to 2, it would be necessary using a sponge to use a permutation operating on $(\mathbb{F}_q^m)^3$ in order to leave one vector of $\mathbb{F}_q^m$ free for the capacity. Using Jive_2 instead, we only need a permutation of $(\mathbb{F}_q^m)^2$. The trade-off of course is that, unlike a sponge-based approach, the relevance of Jive is restricted to some specific cases.

(a) Jive_2, *which maps* $(\mathbb{F}_q^m)^2$ *to* $\mathbb{F}_q^m$. **(b)** Jive_b, *which maps* $(\mathbb{F}_q^m)^b$ *to* $\mathbb{F}_q^m$.

Figure 2: *The* Jive *mode turning a permutation into a compression function.*

4 The `Flystel` Structure

The performance metrics for arithmetization-oriented algorithms differ substantially from the usual ones in symmetric cryptography. Neither the number of CPU cycles, nor the RAM consumption or the code size are the dominant factors. At the same time, pinpointing exactly what is needed for the various protocols relying on arithmetization is a difficult task as each protocol has its own subtelties. For example, Plonk offers custom gates, which are difficult to develop but can decrease the overall cost of an operation, but other proof systems might not. On the other hand, additions are essentially free for R1CS, but not for Plonk. In addition, permutations of a sequence of field elements are likely to incur cost in Plonk (in the form of copy-constraints), but are free in R1CS.

In this section, we present a family of non-linear components that provide both the cryptographic properties that we need to ensure the security of our primitives, and efficient implementations accross proof systems, which we called *open* ***Flystel***. It uses—and highlights—the connection between arithmetization-orientation and CCZ-equivalence.

4.1 On CCZ-Equivalence and Arithmetization-Orientation

In order for a function F to be arithmetization-oriented, it is necessary that verifying whether $y = F(x)$ can be done using few multiplications in a specific field (whose size is dictated by other parts of the protocol). A very straight-forward approach, and indeed the first considered, consists in using a function F which, itself, can be evaluated using small number of multiplications: both MiMC-Hash [AGR+16] and Poseidon [GKR+21b] work in this way. The downside of this approach is that the use of a low degree round function may imply a vulnerability to some attacks based on polynomial solving, often nicknamed *algebraic attacks*. As a consequence, these algorithms have to use a high number of rounds.

A first breakthrough on this topic was made by the designers of Rescue-Prime [AAB+20]. They noticed that if F is a permutation, then checking if $y = F(x)$ is equivalent to checking if $x = F^{-1}(y)$. It allows them to use both x^α and $x^{1/\alpha}$ (where $x \mapsto x^\alpha$ is a permutation of the field used) in their round function, where α is chosen so as to minimize the number of multiplication. It means that both can be verified using a (cheap) evaluation of x^α, and at the same time that the degree of the round function is very high as $1/\alpha$ is a dense integer of $\mathbb{Z}/(q-1)\mathbb{Z}$. As a consequence, much fewer rounds are needed to prevent algebraic attacks.

We go further and propose a generalization of this insight. So far, we have seen that arithmetization-orientation implies that a function or its inverse must have a particular implementation property (low number of multiplications). In fact, we claim the following:

> A subfunction is arithmetization-oriented if it is **CCZ-equivalent** to a function that can be verified efficiently.

The above should come as no surprise since a permutation and its inverse are known to be CCZ-equivalent. In that sense, this insight is a natural generalization of the one of the Rescue-Prime designers.

Exploiting this idea is simple: suppose that F and G are such that $\Gamma_F = \mathcal{L}(\Gamma_G)$, where $\mathcal{L} : (x, y) \mapsto (\mathcal{L}_L(x, y), \mathcal{L}_R(x, y))$ is an affine permutation, and where G can be efficiently verified. Then we can use F to construct an arithmetization-oriented algorithm: checking if $y = F(x)$ is equivalent to checking if $\mathcal{L}_L(x, y) = G(\mathcal{L}_R(x, y))$, which only involves G and linear functions: it is efficient.

Below, we present a first component based on this idea: the `Flystel`. Nevertheless, we hope that further research in discrete mathematics will lead to new non-linear components that are even better suited to this use case: we need more permutations with good cryptographic properties (including a high degree) that are CCZ-equivalent to functions with a low number of multiplications.

4.2 High Level View of the `Flystel` Structure

Let $Q_i : \mathbb{F}_q \to \mathbb{F}_q$ and $Q_f : \mathbb{F}_q \to \mathbb{F}_q$ be two quadratic functions, and let $E : \mathbb{F}_q \to \mathbb{F}_q$ be a permutation. Then, the ***Flystel*** is a pair of functions relying on Q_i, Q_f and E. The *open* ***Flystel*** is the permutation of $(\mathbb{F}_q)^2$ obtained using a 3-round Feistel network with Q_i, E^{-1}, and Q_f as round functions, as depicted in Figure 3a. It is denoted $\mathcal{H}$, so that $\mathcal{H}(x, y) = (u, v)$ is evaluated as follows:

1. $x \leftarrow x - Q_i(y)$,
2. $y \leftarrow y - E^{-1}(x)$,
3. $x \leftarrow x + Q_f(y)$,
4. $u \leftarrow x,\ v \leftarrow y$.

The *closed* `Flystel` is a function of $\mathbb{F}_q^2$ defined by $\mathcal{V} : (y, v) \mapsto (R_i(y, v), R_f(v, y))$, where $R_j : (y, v) \mapsto E(y - v) + Q_j(y)$ for $j \in \{i, f\}$.

(a) *Open* `Flystel`, $\mathcal{H}$. **(b)** *Closed* `Flystel`, $\mathcal{V}$.

Figure 3: *The* `Flystel` *structure (both variants are CCZ-equivalent).*

Our terminology of "open" for the permutation and "closed" for the function is based on the relationship between the `Flystel` and the butterfly structure, as detailed later. In particular, the two structures are connected by the following proposition.

Proposition 1. *For a given tuple (Q_i, E, Q_f), the corresponding closed and open* `Flystel` *are CCZ-equivalent.*

Proof. Let $(u, v) = \mathcal{H}(x, y)$. Then it holds that $v = y - E^{-1}(x - Q_i(y))$, so that we can write $x = E(y - v) + Q_i(y)$. Similarly, we have that $u = Q_f(v) + E(y - v)$. Consider now the set $\Gamma_{\mathcal{H}} = \{((x, y), \mathcal{H}(x, y)), (x, y) \in \mathbb{F}_q^2\}$. By definition, we have

$$\Gamma_{\mathcal{H}} \;=\; \{((x, y), (u, v)), (x, y) \in \mathbb{F}_q^2\} \;=\; \mathcal{L}\big(\{((y, v), (x, u)), (x, y) \in \mathbb{F}_q^2\}\big)$$

where $\mathcal{L}$ is the permutation of $(\mathbb{F}_q^2)^2$ such that $\mathcal{L}((x, y), (u, v)) = ((y, v), (x, u))$, which is linear. Using the equalities we established at the beginning of this proof, we can write:

$$\begin{aligned}\mathcal{L}^{-1}(\Gamma_{\mathcal{H}}) \;&=\; \{((y, v), (x, u)), (x, y) \in \mathbb{F}_q^2\} \\ &=\; \underbrace{\{\big((y, v), (Q_i(y) + E(y - v), Q_f(v) + E(y - v))\big), (y, v) \in \mathbb{F}_q^2\}}_{\Gamma_{\mathcal{V}}} .\end{aligned}$$

We deduce that $\Gamma_{\mathcal{H}} = \mathcal{L}(\Gamma_{\mathcal{V}})$, so the two functions are CCZ-equivalent. □

This simple proposition has several crucial corollaries on which we will rely in the remainder of the paper. The first implies that it is sufficient to investigate the differential and linear properties of the closed butterfly to obtain results about the open one.

Corollary 1. *The open and closed* `Flystel` *structures have identical differential and linear properties. More precisely, the set of the values in the DDT of both functions is the same, and the set of the square of the Fourier coefficients of the components is also the same.*

The second corollary is the key reason behind the relevance of the `Flystel` structure in the airthmetization-oriented setting and is stated below.

Corollary 2. *Verifying that $(u, v) = \mathcal{H}(x, y)$ is equivalent to verifying that $(x, u) = \mathcal{V}(y, v)$.*

Indeed, Corollary 2 means that it is possible to encode the verification of the evaluation of the high degree open `Flystel` using the polynomial representation of the low degree closed `Flystel`.

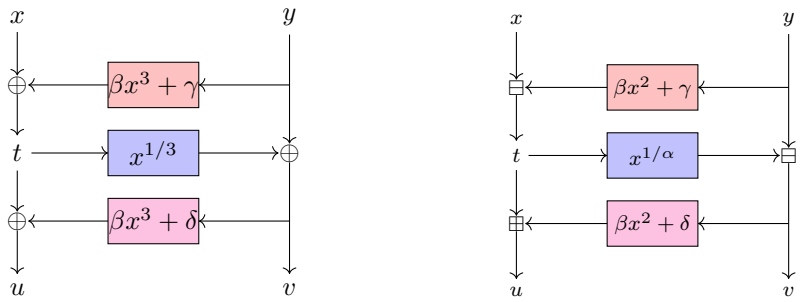
In characteristic 2, quadratic mappings correspond to different exponents than in the general case. As a consequence, when giving concrete instanciations of the `Flystel` structure, we need to treat this case separately. To highlight the difference, we call $\texttt{Flystel}_2$ the instances used in characteristic 2, and $\texttt{Flystel}_\text{p}$ the instances used in odd prime characteristic.

4.3 Characteristic 2

Let $q = 2^n$, with n odd. Furthermore, let $\alpha = 2^i + 1$ be such that $\gcd(i, n) = 1$, so that $x \mapsto x^\alpha$ is a permutation of $\mathbb{F}_q$. In this case, the $\texttt{Flystel}_2$ structure with $Q_i(x) = Q_f(x) = \beta x^\alpha$, with $\beta \neq 0$, and with $E(x) = x^\alpha$ is a degenerate generalized butterfly structure. It was studied in [LTYW18] as a generalization of the structure introduced in [PUB16], which was also refined in [CDP17]. We recall the following particular case[2] in Theorems 3, 4 and 5 of [LTYW18].

Proposition 2 ([LTYW18])**.** *Let $q = 2^n$ with n odd, $E = x \mapsto x^\alpha$, where $\alpha = 2^i + 1$ is such that $\gcd(i, n) = 1$, and $Q_i = Q_f = x \mapsto \beta x^\alpha$, where $\beta \neq 0$. Then the $\texttt{Flystel}_2$ structures defined by the functions Q_i, E, and Q_f have a differential uniformity equal to 4, a linearity equal to $2^{2n-1} - 2^n$, and an algebraic degree equal to n.*

In practice, to prevent some attacks (see Section 6.3), we instead use $Q_i(x) = \beta x^3 + \gamma$ and $Q_f(x) = \beta x^3 + \delta$, where γ and δ are constants of $\mathbb{F}_q$ such that $\gamma \neq \delta$. The resulting construction is depicted in Figure 4a.



(a) *$\texttt{Flystel}_2$ in characteristic 2.* **(b)** *$\texttt{Flystel}_\text{p}$ in odd prime characteristic.*

Figure 4: *The two variants of the open `Flystel`, mapping (x, y) to (u, v).*

4.4 Odd Characteristic

Let $q = p$. In this case, the $\texttt{Flystel}_\text{p}$ structure uses three rounds functions: $Q_i : x \mapsto \beta x^2 + \gamma$, $E : x \mapsto x^{1/\alpha}$, and $Q_f : x \mapsto \beta x^2 + \delta$, where $\beta \in \mathbb{F}_q$ is non-zero, and where γ and δ are constants of $\mathbb{F}_q$.

[2] The result of Li et al. covers all generalized butterflies, not just those corresponding to `Flystel` structures. In a `Flystel`, the first parameter (which we will denote a) is set to 1. Their results for the differential uniformity and the linearity holds only when $\beta \neq (1 + a)^\alpha$, meaning that we simply need to make sure that $\beta \neq 0$. For the algebraic degree, the condition they give in their Theorem 5 to have a degree equal to $n + 1$ degenerates into $\beta^{2^{i+1}} = \beta^{2^i+1}$, which is never the case as $i > 0$.

Differential Properties. Such structures have a low differential uniformity.

Proposition 3. *Let $q = p$ be a prime number, $E = x \mapsto x^\alpha$, where α is such that $\gcd(\alpha, p-1) = 1$, and $Q_i = x \mapsto \gamma + \beta x^2$, $Q_f = x \mapsto \delta + \beta x^2$ where $\beta \neq 0$. Then the $\texttt{Flystel}_p$ structures defined by the functions Q_i, E, and Q_f has a differential uniformity equal to $\alpha - 1$.*

Proof. Let a, b, c, d be elements of $\mathbb{F}_p$ such that $(a, b) \neq (0, 0)$. To investigate the differential uniformity of $\mathcal{V} : (y, v) \mapsto (R_i(y, v), R_f(v, y))$, we look at the number of solutions (y, v) of (1).

$$\begin{cases} R_i(y+a, v+b) - R_i(y, v) = c \\ R_f(v+b, y+a) - R_f(v, y) = d \ . \end{cases} \tag{1}$$

We have:

$$\begin{cases} R_i(y+a, v+b) - R_i(y, v) = (y + a - (v+b))^\alpha + \gamma + \beta(v+b)^2 - (y-v)^\alpha - \gamma - \beta v^2 \\ R_f(v+b, y+a) - R_f(v, y) = (v + b - (y+a))^\alpha + \delta + \beta(y+a)^2 - (v-y)^\alpha - \delta - \beta y^2 \ . \end{cases}$$

As e is odd, we deduce:

$$\begin{cases} R_i(y+a, v+b) - R_i(y, v) = (y + a - (v+b))^\alpha + \beta(v+b)^2 - (y-v)^\alpha - \beta v^2 \\ R_f(v+b, y+a) - R_f(v, y) = -(y + a - (v+b))^\alpha + \beta(y+a)^2 + (y-v)^\alpha - \beta y^2 \ . \end{cases}$$

Noting respectively ℓ_1 and ℓ_2 the rows of the system, we get:

$$\ell_1 + \ell_2 = \beta(v+b)^2 - \beta v^2 + \beta(y+a)^2 - \beta y^2 = c + d \ ,$$

which is equivalent to:

$$v = (2b)^{-1} \left(\beta^{-1}(c+d) - (2ay + a^2 + b^2) \right) \ .$$

As a consequence, we know that v can be expressed as an affine polynomial in y. So, we have

$$\ell_2 = -(y + a - (v+b))^\alpha + \beta(y+a)^2 + (y-v)^\alpha - \beta y^2$$

Recalling that v is of degree 1 in y, we have that ℓ_2 is an equation in y of degree $\alpha - 1$ (since the terms y^α cancel out), and thus at most $\alpha - 1$ solutions for y. In the end, we have at most $\alpha - 1$ solutions (y, v) for the system (since for each value of y, there is one v). □

Linear Properties. We do not have a theoretical bound on the correlation for the $\texttt{Flystel}_\texttt{p}$ structure. Nevertheless, we will argue that we do not expect any attack to come from this direction.

Let us first notice that the $\texttt{Flystel}_\texttt{p}$ is defined by the functions Q_i, E^{-1} and Q_f, where Q_i and Q_f are quadratic. Given that the function x^2 is bent (i.e. that its correlations are the lowest possible), we can argue somewhat informally that a linear trail that would activate just one of these functions should be expected to have a very low correlation.

Second, our experiments indicate that the correlation increases slowly with the field size p. In fact, we have obtained the following conjecture for the maximum value of the module of its Walsh transform.

Conjecture 1. *If $q = p$ is a prime number, then the maximum module of the Walsh transform of $\mathcal{H}$ satisfies*

$$\max_{a \in \mathbb{F}_p^m, b \in \left(\mathbb{F}_p^m\right)^*} |\mathcal{W}_{\langle b, \mathcal{H} \rangle}(a)| \ \leq \ p \log p \ .$$

While the most general case remains a conjecture at the time of writing, this results holds for small values of p ($p \leq 53$), as can be seen in Figure 5.

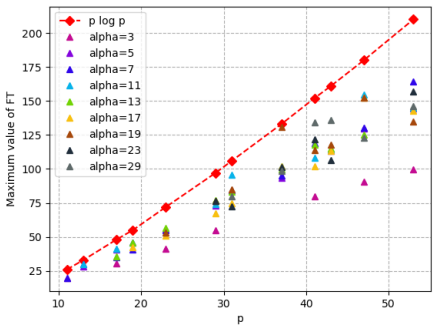


Figure 5: *The maximum value of the module of the Walsh transform of $\mathcal{H}$.*

Invariant Subset. Regardless of the characteristic, it holds that $\mathcal{H}(Q_i(x), x) = (Q_f(x), x)$. Thus, setting $Q_i = Q_f$ would mean that the `Flystel` is the identity over a subset of size q, which is why we added constant additions to ensure that $Q_i \neq Q_f$. Nevertheless, this only ensures that the open `Flystel` is a translation over the set $\{(Q_i(x), x), x \in \mathbb{F}_q\}$, which remains cryptographically weak.

While a priori undesirable, th impact of this property can be mitigated. First, the subset over which it has a simple expression is not an affine space. Second, as we show later in Section 6.3, the propagation of such patterns can be broken using the linear layer.

Degree. Given the structure of the open $\texttt{Flystel}_\texttt{p}$, its degree is lower bounded by the inverse of α modulo $q-1$, a quantity which in practice corresponds to a dense integer of $\mathbb{Z}/(q-1)\mathbb{Z}$. We deduce that one call to this permutation is sufficient to thwart all attacks that would exploit the low degree of a component, such as higher order differentials.

4.5 Implementation Aspects

For direct computation, (or witness calculation) one can simply implement the open `Flystel`. For the verification however, we also have the option to use the closed `Flystel` structure, since there is no requirement for the various verification steps to be performed in a particular order as long as consistency is enforced. In this case, the cost is of one multiplication for Q_i and Q_f, and as many as are needed to compute $x \mapsto x^\alpha$. This computation can be implemented using a technique that is slightly more subtle than the basic fast exponentiation algorithm, and instead relies on addition chains as discussed for example in [BC90]. We used `addChain` [McL21] to find the best such chains for small values of α. We listed the corresponding results in Table 1, and denote $\mathcal{C}_\alpha$ the cost of such exponentiation.

We remark that the cost of the exponentiation increases slowly, and that for example $\alpha = 17$ is less than twice as expensive as $\alpha = 5$. This fact will play an important in Section 5.3, when choosing the parameters to use in various cases.

5 Description of `Anemoi`

`Anemoi` is a family of permutations operating on $\mathbb{F}_q^{2\ell}$, for any field size q that is either a prime number or a power of two, and for positive integer ℓ. It relies on the `Flystel` component introduced in the previous section. We let $n = \lfloor \log_2 q \rfloor$ be the rounded down bitlength of a word of $\mathbb{F}_q$, so that our permutations can be thought of as operating on $2\ell n$ bits when evaluating the security they could be used to provide. Section 5.1

Table 1: *The values α for which computing $x \mapsto x^\alpha$ requires a given number of multiplications.*

# multiplications	α
2	$\{3\}$
3	$\{5\}$
4	$\{7, 9\}$
5	$\{11, 13, 15, 17\}$
6	$\{19, 21, 23, 25, 27, 33\}$
7	$\{29, 31, 35, 37, 39, 41, 43, 45, 49, 51, 65\}$
8	$\{47, 53, 55, 57, 59, 61, 63, 67, 69, 73, 75, 77, 81, 83, 85, 97, 99\}$
9	$\{71, 79, 87, 89, 91, 93, 95, 101, 103, 105, 107, 109, 111, 113, 115, 117, 119, 121, 123, 125\}$
10	$\{127\}$

presents the round function of `Anemoi` and how its parameters are chosen. In order to ease third party analysis, and to convincingly argue that we could not hide a trapdoor in our algorithm, we limit as much as possible the use of (pseudo-)randomly generated components. Thus, we present the deterministic and simple procedures that must be used to generate each component. Following the general idea outlined in [DP19], we claim that all the permutations obtained via this method are safe.

Claim. *An `Anemoi` permutation of $\mathbb{F}_q^{2\ell}$ whose parameters are chosen using the method described in this section can safely be used in a mode calling for a permutation operating on $2\ell n$ bits.*

The `Anemoi` permutations are then used in Section 5.2 to construct a compression function mapping 2ℓ words to ℓ, `AnemoiJive`, which is intended to be used in Merkle trees; and `AnemoiSponge`, a general purpose hash function intended to emulate the behaviour of a random oracle. The former is based on our mode of operation, Jive, while `AnemoiSponge` is a classical sponge structure.

5.1 Round Function

The state of `Anemoi` is organized into a rectangle of elements of $\mathbb{F}_q$ of dimension $2 \times \ell$. The elements in the first row are denoted $(x_0, ..., x_{\ell-1})$, and those in the second row are $(y_0, ..., y_\ell)$ (see Figure 6a). We refer to vectors of $\mathbb{F}_q^\ell$ using the same upper-case letters, e.g. $(x_0, ..., x_{\ell-1})$ is denoted X. Subscripts correspond to indices within a vector of $\mathbb{F}_q^\ell$, and superscripts to round indices, so X^i is the top part of the state at the start of round i.

We let g be generator of the multiplicative subgroup of the field $\mathbb{F}_q$. If q is prime, then g is the smallest such generator using the usual integer ordering. Otherwise, we have that $\mathbb{F}_q = \mathbb{F}_{2^n} = \mathbb{F}_2[x]/p(x)$, where p is an irreducible polynomial of degree n, in which case we let g be one of its roots.

The round function of the members of the `Anemoi` family relies on the following operations, whose actions on the internal state are also summarized in Figure 6. The function applied during round r is denoted R_r.

Diffusion Layer $\mathcal{M}$. If $\ell > 1$, then the diffusion layer $\mathcal{M}$ operates on X and Y separately, so that

$$\mathcal{M}(X, Y) = \big(\mathcal{M}_x(X), \mathcal{M}_y(Y)\big) \ ,$$

as summarized in Figure 6b. The linear permutations $\mathcal{M}_x$ and $\mathcal{M}_y$ are closely related, but differ in order to break the column structure imposed by the non-linear layer (see below). More precisely, we impose that $\mathcal{M}_x$ is a matrix of size $\ell \times \ell$ of $\mathbb{F}_q$ with maximum diffusion, i.e. such that its branching number[3] is equal to $\ell + 1$. We then construct $\mathcal{M}_y$ as $\mathcal{M}_y = \mathcal{M}_x \circ \rho$, where ρ is a simple word permutation: $\rho(x_0, ..., x_{\ell-1}) = (x_1, ..., x_{\ell-1}, x_0)$.

[3]Recall that the branching number of a linear permutation L is the minimum over $x \neq 0$ of $\mathrm{hw}(x) + \mathrm{hw}(L(x))$, where hw counts the number of non-zero elements.

If $\ell = 1$, then there is a unique column in the internal state. In this case, to destroy some undesirable patterns at the S-box level, we still use a linear layer, except that it is applied on the vector (x_0, y_0). In this case, we use the function obtained by first applying $(x, y) \mapsto (x + gy, y)$, and then $(x, y) \mapsto (x, y + gx)$, i.e. $(x, y) \mapsto \big(x + gy, gx + (g^2 + 1)y\big)$.

While linear operations are essentially free in R1CS, additions are not as e.g. in Plonk. As a consequence, in order to decrease the cost of the matrix multiplication, it makes sense to borrow ideas from lightweight cryptography where there is substantial literature on minimizing the number of additions needed to implement MDS functions, namely from [DL18].

$\ell = 2$. We impose that $\mathcal{M}_x$ is the same linear permutation as for the case $\ell = 1$, i.e. $\mathcal{M}_x(x, y) = \big(x + gy, gx + (g^2 + 1)y\big)$.

$\ell = 3$. We use the permutation described in Figure 6 of [DL18], namely

$$\mathcal{M}_x = \begin{bmatrix} g+1 & 1 & g+1 \\ 1 & 1 & g \\ g & 1 & 1 \end{bmatrix},$$

which can be implemented using 5 additions and 2 multiplications by g. We chose this matrix rather than the 3 other candidates because it uses only 5 additions (unlike Figure 7), and has fewer symmetries than the other 2.

$\ell = 4$. We use the permutation described in Figure 8 of [DL18], namely

$$\mathcal{M}_x = \begin{bmatrix} g+1 & 1 & g^2 & g^2 \\ 1 & g+1 & g^2+g & g^2 \\ g & g & g+1 & 1 \\ g+1 & g & 1 & g+1 \end{bmatrix},$$

which can be implemented using 8 additions and 3 multiplications by g. We chose this specific matrix because it uses only 8 additions (unlike Figures 9, 11 and 12), and only one type of multiplication (unlike Figures 10 and 12).

We provide efficient implementations of all these components in Appendix B. For higher values of ℓ, we leave it to the users to choose a suitable matrix $\mathcal{M}_x$. Still, we impose that it must be MDS.

S-box Layer $\mathcal{S}$. Let $\mathcal{H}$ be an open `Flystel` operating over $\mathbb{F}_q^2$. Then we have that

$$\mathcal{S}(X, Y) = \big(\mathcal{H}(X_0, Y_0), ..., \mathcal{H}(X_{\ell-1}, Y_{\ell-1})\big),$$

as summarized in Figure 6c. A `Flystel` instance is defined by 4 parameters, regardless of whether it is a $\texttt{Flystel}_\texttt{p}$ or $\texttt{Flystel}_2$: the exponent α, the multiplier β, and the two added constants γ and δ. First, we let $\beta = g$: setting $\beta = 1$ would lead to the invariant space (Section 6.3) having equation (x^2, x), which we deem too simple; g is then the most natural non-trivial constant. Furthermore, in order to break the symmetry of the `Flystel`, we impose that $\gamma \neq \delta$. We thus let $\gamma = 0$ and $\delta = g^{-1}$ as this value is both different from 1 and g while retaining a simple definition.

All that remains is to choose the exponent α. If $q = 2^n$, then we let $\alpha = 3$: we have to use a Gold exponent (i.e. of the shape $2^k + 1$), and 3 always works since n is odd. Otherwise, when q is prime, the process is a bit more involved as a higher value allows using fewer rounds to thwart Gröbner-basis-based attacks, but is also more expensive. Users should use the value of α that yields the most efficient algorithm according to their metrics.

Constant Additions $\mathcal{A}$. We let $x_j \leftarrow x_j + c_j^i$ and $y_j \leftarrow y_j + d_j^i$, where $c_j^i \in \mathbb{F}_q$ and $d_j^i \in \mathbb{F}_p$ are round constants that depend on both the position (index j) and the round (index i). The aim is to increase the complexity of the algebraic expression of multiple rounds of the primitive and to prevent the appearance of patterns that an attacker could leverage in their attack.

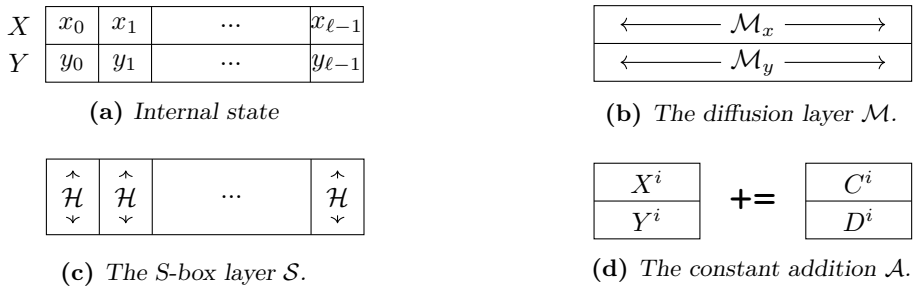
They are derived using the digits of π using the following procedure. We let

$$(\pi_0, \pi_1) = (1415926535897932384626433832795028841971693993751058209749445923078164062862089986280348253421170679, 8214808651328230664709384460955058223172535940812848111745028410270193852110555964462294895493038196)$$

be the first and second blocks of 100 digits of π. We derive the round constants c_j^i and d_j^i by applying an open `Flystel` with the same parameters as in the round function on the pair (π_0^i, π_1^j), so that

$$\begin{cases} c_j^i &= g(\pi_0^i)^2 + \left(\pi_0^i + \pi_1^j\right)^\alpha \\ d_j^i &= g(\pi_1^j)^2 + \left(\pi_0^i + \pi_1^j\right)^\alpha + g^{-1} \ , \end{cases}$$

where the computations are done in $\mathbb{F}_q$. When $q = 2^n$, π_0 and π_1 are cast to field elements using the usual mapping sending $\sum_{k=0}^{n-1} x_i 2^i$ to $\sum_{k=0}^{n-1} x_i g^i$, where $(x_0, ..., x_{n-1})$ is the binary representation of x modulo 2^n.



(a) *Internal state*

(b) *The diffusion layer $\mathcal{M}$.*

(c) *The S-box layer $\mathcal{S}$.*

(d) *The constant addition $\mathcal{A}$.*

Figure 6: *The internal state of* `Anemoi` *and its basic operations.*

These operations correspond to a classical SPN structure. A complete description of one round of `Anemoi` is given in Figure 7 (both as a diagram and as an algorithm), where Q is the quadratic operation of the field considered, i.e. $x \mapsto x^3$ if $q = 2^n$, $x \mapsto x^2$ otherwise.

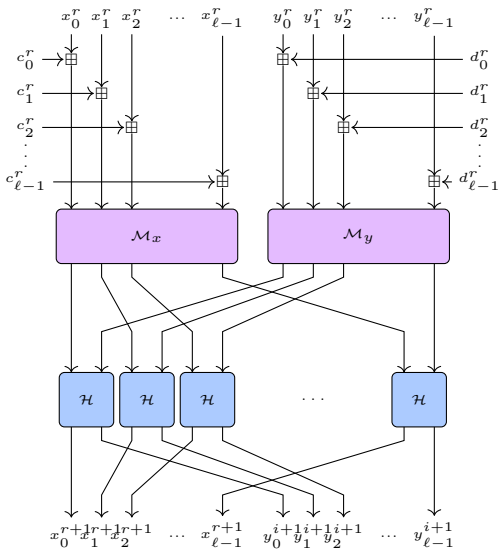
5.2 Higher Level Algorithms

`Anemoi`. The `Anemoi` permutation iterates n_r rounds of the round function described in Figure 7, followed by a call to the linear layer $\mathcal{M}$:

$$\texttt{Anemoi}_{q,\alpha,\ell} \ = \ \mathcal{M} \circ \mathsf{R}_{n_r-1} \circ ... \circ \mathsf{R}_0$$

In symmetric cryptography, we usually *remove* outer linear layers, e.g. in the AES. That is because, in the case of a block cipher, those could simply be peeled off for free by an adversary. In the case of a sponge construction however, the adversary only controls a part of the state, namely the outer part (the rate). Thus, starting/finishing with a diffusion layer ensures that this control is spread across the full state in a way which is not aligned with the non-linear layer. A similar goal could be achieved using *indirect injection*, as is done in Esch [BBC+20].

The number of rounds n_r is computed using the following rule that is derived from our security analysis in Section 6. Let s be the required security level, and (q, ℓ, α) be the



(a) *(Assuming $\ell > 1$)*

▷ Constant addition
for $i \in \{0, ..., \ell - 1\}$ **do**
  $x_i \leftarrow x_i + c_i^r$
  $y_i \leftarrow y_i + d_i^r$
end for
▷ Linear layer $\mathcal{M}$
if $\ell > 1$ **then**
  $X \leftarrow \mathcal{M}_x(X)$
  $Y \leftarrow \mathcal{M}_x \circ \rho(Y)$
else
  $(x_0, y_0) \leftarrow \mathcal{M}_x(x_0, y_0)$
end if
▷ S-box layer $\mathcal{H}$
for $i \in \{0, ..., \ell - 1\}$ **do**
  $x_i \leftarrow x_i - gQ(y_i)$
  $y_i \leftarrow y_i - x_i^{1/\alpha}$
  $x_i \leftarrow x_i + gQ(y) + g^{-1}$
end for
return (X, Y)

(b)

Figure 7: R_r, *the r-th round of* `Anemoi`*, applied on the state* $(X, Y) \in \mathbb{F}_q^\ell \times \mathbb{F}_q^\ell$*, where* $X = (x_0, ..., x_{\ell-1})$ *and* $Y = (y_0, ..., y_{\ell-1})$.

Table 2: *Number of Rounds of* `Anemoi`.

α	3	5	7	11	13	17
$\ell = 1$	19	19	18	18	17	16
$\ell = 2$	12	12	11	11	11	10
$\ell = 3$	10	10	10	10	10	10
$\ell = 4$	10	10	10	10	10	10

(a) *When* $s = 128$.

α	3	5	7	11	13	17
$\ell = 1$	35	35	34	34	33	32
$\ell = 2$	20	20	19	19	19	18
$\ell = 3$	15	15	15	15	15	14
$\ell = 4$	14	14	13	13	13	13

(b) *When* $s = 256$.

parameters imposed by the use case. Then the number of rounds n_r is the smallest value satisfying both of the following conditions:

$$n_r \geq 10, \text{ and}$$

$$n_r \geq \underbrace{1 + \ell}_{\text{security margin}} + \underbrace{\min\left\{ r \in \mathbb{N} \;\middle|\; \binom{2\ell r + \alpha + 1 + 2 \cdot (\ell r - 2)}{2\ell r}^2 \geq 2^s \right\}}_{\text{to prevent algebraic attacks, see Section 6.4}} \tag{2}$$

We derived the number of rounds needed for various values of ℓ and α, both for a security level of 128 bits (Table 2a), and of 256 bits (Table 2b). Note that the values of the digest size h and of the state size $2\ell n = 2\ell \log_2(q)$ must be coherent with the desired security level.

`AnemoiSponge`. This function is a "regular" hash function, in the sense that it should be able to process messages of arbitrary length. We therefore rely on a sponge construction with Hirose's domain separation (as detailed in Section 3), where r words are used as the rate, c are used as the capacity, and where the permutation is the `Anemoi` instance

Table 3: *Parameters for some instances of* `Anemoi`.

Curve	$\log_2 q$	s	α	g	Function	ℓ	n_r
BLS12-381	381	190	5	2	`AnemoiJive`	1	19
					`AnemoiSponge`	2	20
BN-254	254	127	5	3	`AnemoiJive`	1	19
					`AnemoiSponge`	2	20

operating on $\mathbb{F}_q^{r+c}$. Using a hermetic sponge claim, the security level against all attacks is then expected to be $nc/2$ bits. Note that the inner workings of `Anemoi` imply that $r + c$ must be even.

`AnemoiJive`. We can construct a compression function mapping b-to-1 vectors of $\mathbb{F}_q^v$ elements, using Jive_b and an `Anemoi` instance operating on bv elements of $\mathbb{F}_q$. The only constraint is that bv must be even.

5.3 Specific Instances

In this section, we present some examples of functions in the `Anemoi` family that are defined over different fields and aim for different APIs (both `AnemoiSponge` and `AnemoiJive`). We focus on the fields used by the elliptic curves BLS12-381 and BN-254. The parameters needed are summarized in Table 3.

`AnemoiJive`. Let q be the prime used either by the curve BLS12-381 or BN-254. `AnemoiJive`-BLS12-381 and `AnemoiJive`-BN-254 are Merkle Compression functions mapping two elements of $\mathbb{F}_q$ to a unique one. They work using the following components.

S-box. In this case, $\mathcal{H}$ uses the parameters g and α given in Table 3.

Linear layer. As $\ell = 1$, we use small linear permutations of $\mathbb{F}_q^2$ as defined in Section 5.1. For BLS12-381 and BN-254, these are respectively

$$\mathcal{M}_x^{\text{BLS12-381}} = \begin{bmatrix} 1 & 2 \\ 2 & 5 \end{bmatrix} \quad \text{and} \quad \mathcal{M}_x^{\text{BN-254}} = \begin{bmatrix} 1 & 3 \\ 3 & 10 \end{bmatrix}. \tag{3}$$

Round Constants. These are generated as described in Section 5.1.

Round r is then defined as $\mathsf{R}_r : (x, y) \mapsto \mathcal{H} \circ \mathcal{M}(x + c_r, y + d_r)$, and we define the compression functions as follows. Let (x, y) be the input, and P be the `Anemoi` instance defined by $P := \mathcal{M} \circ \mathsf{R}_{18} \circ ... \circ \mathsf{R}_0$. Then `AnemoiJive`$(x, y)$ is evaluated as follows:

1. $(u, v) = P(x, y)$,
2. return $x + y + u + v$.

Security Claims. The best way to find collisions in `AnemoiJive`-BLS12-381 (respectively `AnemoiJive`-BN-254) is to rely on a generic collision search. Since the output is an element of $\mathbb{F}_q$, this is expected to require about respectively 2^{190} or 2^{127} calls to the function.

`AnemoiSponge`. Let q be the prime used either by the curve BLS12-381 or BN-254. `AnemoiSponge`-BLS12-381 and `AnemoiSponge`-BN-254 are hash functions mapping a sequence $\{x_i\}_{0 \leq i < m}$ of elements of $\mathbb{F}_q$ to an element of $\mathbb{F}_q$, where ℓ is a positive integer. It is constructed using a sponge which relies on `Anemoi` as the permutation. We aim to provide about 256 bits of security. As $\log_2(q) \approx 381$ in one case and 254 in the other, we need to have a capacity of 2 elements of $\mathbb{F}_q$. We then pick an identical rate, so that $r = c = 2$, and thus $\ell = 2$. The subcomponents are the same as above, except that the

linear layer is applied in parallel on (x_0, x_1) and (y_0, y_1), and that $\mathcal{M}_y$ is equal to $\mathcal{M}_x \circ \rho$ with $\rho(y_0, y_1) = (y_1, y_0)$. Unlike for `AnemoiJive`, we aim here to provide about 256 bits of security. As a consequence, we need to pick the value of n_r from Table 2b: it is 20 in both cases.

Security Claims. We claim that `AnemoiSponge`-BLS12-381 provide 256 bits of security against all attacks, and that `AnemoiSponge`-BN254 provides 254.

6 Security Analysis

The security of our high level algorithms is reduced to the security of their inner permutation, namely the `Anemoi` family. In this section, we argue that the latter has sufficient security level.

6.1 Differential and Linear Attacks

In this part, we argue that differential and linear attacks can be prevented by the `Flystel` construction, thanks to the differential and linear properties of the scheme as presented in Section 5.

Differential attacks exploit the probability distribution of a given non-zero input difference leading to a given output difference after a certain number of rounds. As established in Proposition 2 for the $\texttt{Flystel}_2$ and in Proposition 3 for the $\texttt{Flystel}_\texttt{p}$, the differential uniformity of a `Flystel` is low (namely 4 in the former case and $(\alpha - 1)$ in the latter). As a consequence, the probability of any transition of the form $\mathcal{H}(x + a, y + b) - \mathcal{H}(x, y) = (c, d)$ is small: it is upper bounded by $(\alpha - 1)/q^2$. Given that q is typically bigger than 2^{63}, we only need to activate 3 S-boxes to obtain more than 128 bits of security, and 5 for 256 bits.

A similar arguments holds for linear attacks. As for the differential uniformity, the correlation increases slowly with q according to Conjecture 1. As a consequence, it is again sufficient to activate a few S-boxes to prevent the existence of high correlation linear trails.

For both attacks, the activation of many S-boxes is further helped by our use of MDS diffusion matrices. The structure $\mathcal{M}$, based on two parallel MDS matrices $\mathcal{M}_x$ and $\mathcal{M}_y$, ensures that at least $\ell + 1$ S-boxes are active in every pair of rounds.

6.2 Integral Attacks

A classical integral attack tracks the evolution of simple patterns through the rounds. Consider a function of $\mathbb{F}_q^\ell$. As explained in [BS01, BS10], a multiset of elements of $\mathbb{F}_q^\ell$ can have a word be saturated (i.e. this word takes all possible values exactly once), be constant, have a sum equal to zero, or not yield any specific pattern. These patterns are denoted "$*$", "C", "0", and "?" respectively. For example, through an S-box layer, $(*, C, ..., C)$ is mapped to $(*, C..., C)$, while the application of an MDS matrix maps $(*, C, ..., C)$ to $(*, *, ..., *)$.

In our case, such attacks do not pose a significant threat. First, the open `Flystel` is a 3-round Feistel network where the center round function is a permutation, so that the only integral pattern is of the form $(*, C) \rightsquigarrow (?, *)$. As a consequence, patterns at the word level cannot be propagated over two full rounds since we would need to consider open `Flystel` instances where one of the inputs has the ? pattern. Patterns at the open `Flystel` level are a bit more promising, i.e. saturating a full column using q^2 queries would lead to having fully saturated columns after one round, a patterned destroyed by the following linear layer (see [BS01, BS10] for a more thorough treatment of such generic integral attacks against SPNs).

As shown in [BCD+20], a new direction can be used in $\mathbb{F}_p$: instead of saturating a word of $\mathbb{F}_q$, it is possible instead to saturate a multiplicative subgroup. Against some algorithms like gMiMC-Hash, this approach is promising as the diffusion is slow and the only non-linear operations are monomials—under which subgroups are stable. In our case, subgroups will not be stable through an open $\texttt{Flystel}_\texttt{p}$ call because of its three internal addition/subtractions and constant additions.

In binary fields, primitives with low algebraic degree are potentially vulnerable to higher order differential cryptanalysis [Knu95], which are themselves closely related to integral attacks. The open $\texttt{Flystel}_2$ is an efficient counter-measure against such attacks since open butterflies operating on $(\mathbb{F}_{2^n})^2$ are known to have an algebraic degree equal to n (see Proposition 2). As shown in [BCD+20], a low degree can also be leveraged in the case where q is prime. Still, a similar argument will hold: the degree of $x \mapsto x^{1/\alpha}$ is too high to allow any meaningfull pattern to emerge.

6.3 Invariant Subspaces

Remember that, regardless of the characteristic, it always holds that $\mathcal{H}(Q_i(y), y) = (Q_f(y), y)$. For each `Flystel` instance in the round function (i.e., for each column in the state), the probability that an input is in this set is equal to $1/q$. As this pattern is non-linear, we deem it unlikely that it is preserved by the combination of the constant addition and the linear layer with a probability higher than chance, meaning that this pattern will be activated in inner rounds with a negligible probability.

That being said, it is a pattern that can be used to simplify the equations modeling a call to `Anemoi` during an algebraic attack: if an attacker has some degrees of freedom, then forcing the emergence of such a pattern within some `Flystel` instances is the best strategy to simplify these equations.

6.4 Algebraic Attacks

In this section, we evaluate the security of `Anemoi` with respect to Gröbner basis attacks. Since we are mainly interested in a minimal condition of the number of rounds to reach a security of 2^s bits, we allow ourselves to *underestimate* the real complexity in several places, out of caution. In our experiments, for practical reasons, we restrict ourselves to $\ell = 1$ for both even and odd characteristics. We focus on the following version of the CICO (Constrained Input Constrained Output) problem:

Definition 3. Let $P : \mathbb{F}_q^2 \to \mathbb{F}_q^2$ be a permutation. The CICO problem consists in finding $(y_{\text{in}}, y_{\text{out}}) \in \mathbb{F}_q^2$ such that $P(0, y_{\text{in}}) = (0, y_{\text{out}})$.

Solving method.

Clearly, there are plenty of ways to model CICO as an algebraic system. In this paper, we restrict ourselves to the possibly most promising one for `Anemoi` which is by introducing equations and variables at each round. Such an approach has already been taken to study similar arithmetization-oriented primitives [DGGK21, BSGL20, GØSW22]. More precisely, for $0 \leq j \leq n_r - 1$, we define f_j and g_j by

$$(x_{j+1}, y_{j+1}) = \mathsf{R}_j(x_j, y_j) \Leftrightarrow \begin{cases} f_j := f_j(x_j, y_j, x_{j+1}, y_{j+1}) & = 0 \\ g_j := g(x_j, y_j, x_{j+1}, y_{j+1}) & = 0, \end{cases}$$

where R_j is the round function and where f and g are closely related to the verification equations. Also, let $\mathcal{F} := \{f_0, g_0, f_1, g_1, \ldots, f_{n_r-1}, g_{n_r-1}\}$ and let $\mathcal{F}_{\mathsf{CICO}} := \mathcal{F} \cup \{x_0, x_{n_r}\}$. This system can be seen as a system of $2n_r$ equations in $2n_r$ variables. To solve it, we apply the standard *zero-dimensional* strategy:

1. compute a Gröbner basis $\mathcal{G}_{\text{drl}}$ for a DRL ordering [Lou94, Definition 1.4.3],

2. then compute a new Gröbner basis $\mathcal{G}_{\text{lex}}$ for the LEX ordering by using the FGLM algorithm [FGLM93] on $\mathcal{G}_{\text{drl}}$.

We will analyze both steps. For Step 1, the running time of Gröbner basis algorithms such as F4 [Fau99] or F5 [Fau02] is usually estimated by evaluating the *solving degree* of the system denoted by d_{solv}. This degree can be informally defined as the maximal degree of a polynomial which occurs during the Gröbner basis computation. Once d_{solv} is known, a generic estimate for the cost of F4/F5 is

$$\mathcal{O}\left(\binom{d_{\text{solv}} + n_v}{n_v}^{\omega}\right) \tag{4}$$

field operations, where n_v is our number of variables and where $2 \leq \omega \leq 3$ is a linear algebra constant. We stress that this estimation is heuristic and it is an upper bound that does not take into account the structure or the sparsity of the given Macaulay matrices. In particular, to use it as a guidance, we will adopt the conservative $\omega = 2$ for the linear algebra constant. Regarding Step 2, the complexity of FGLM is in $\mathcal{O}\left(n_r \cdot \deg\left(I_{\mathsf{CICO}}\right)^{\omega}\right)$, where $I_{\mathsf{CICO}} := \langle \mathcal{F}_{\mathsf{CICO}} \rangle$ is the ideal generated by the system and where $\deg\left(I_{\mathsf{CICO}}\right)$ is the *degree* of this ideal.

Definition 4. Let $I \subset \mathbb{F}_q[x_0, \ldots, x_{n_r}, y_0, \ldots, y_{n_r}]$ be a zero-dimensional ideal. The degree $\deg(I)$ is defined as the dimension of $\mathbb{F}_q[x_0, \ldots, x_{n_r}, y_0, \ldots, y_{n_r}]/I$.

There exists a sparse variant of FGLM [FM11], but the key quantity to grasp its complexity is still $\deg\left(I_{\mathsf{CICO}}\right)$. Also, for some AO primitives, the cost of this step was even estimated by $\mathcal{O}\left(n_r \cdot \deg\left(I_{\mathsf{CICO}}\right)\right)$, see for instance [BSGL20, Appendix, p.12]. We are not aware of the techniques employed to derive this result; in particular, this bound might underestimate the real cost in the case of Anemoi. Still, we will use it as a conservative lower bound to derive our parameters. Let us now estimate $d_{\text{solv}}\left(I_{\mathsf{CICO}}\right)$ and $\deg\left(I_{\mathsf{CICO}}\right)$ in the case of Anemoi when $\ell = 1$.

6.4.1 Characteristic 2.

When $q = 2^n$ for some odd n, recall that $\alpha = 3$, $Q_i(x) = \beta x^3 + \gamma$ and $Q_f(x) = \beta x^3 + \delta$. The system $\mathcal{F}_{\mathsf{CICO}}$ then contains $2n_r$ cubic polynomials in $2(n_r + 1) - 2 = 2n_r$ variables. For such a system, one usually relies on the Macaulay bound for an upper bound on d_{solv}, namely $2n_r(3-1) + 1 = 4n_r + 1$. This bound would be tight if the system of homogeneous parts of highest degree was *regular*. However, our experiments indicate that $\mathcal{F}_{\mathsf{CICO}}$ does not behave as such, see Appendix A.1. Therefore, we chose to extrapolate these data to find a lower bound on d_{solv}. As this method might lead to inaccuracies for a higher number of rounds, our bound is voluntarily very coarse.

Conjecture 1. *For $n_r \geq 2$, the maximal degree $d_{solv}\left(I_{\mathsf{CICO}}\right)$ which occurs while computing a* DRL*-Gröbner basis for I_{CICO} is such that $d_{solv}\left(I_{\mathsf{CICO}}\right) \geq 3n_r$.*

Then, using this generic estimate, we evaluate the complexity of Step 1. using Equation (4), with $n_v = 2n_r$. For Step 2 we have to evaluate $\deg\left(I_{\mathsf{CICO}}\right)$. Using the generic Bezout bound for a system of $2n_r$ cubic equations, we obtain

$$\deg\left(I_{\mathsf{CICO}}\right) \leq 3^{2n_r} \ . \tag{5}$$

Also, from experiments it seems that this bound is tight as we always have $\deg\left(I_{\mathsf{CICO}}\right) = 3^{2n_r}$. Therefore, we estimate the complexity of Step 2 to be $\mathcal{O}(n_r \cdot 3^{2n_r})$ with $\omega = 1$.

6.4.2 Odd characteristic.

When $q > 2$ is an odd prime, the equations f_j and g_j are affine of degree α, but it has already been noted in the proof of Proposition 3 that there is a quadratic relation h_j at each round, namely $h_j := f_j - g_j$ for $0 \leq j \leq n_r - 1$. It is also clear that taking $\{f_j, h_j\}$ instead of $\{f_j, g_j\}$ as a generating set for each round does not change the final ideal I_{CICO} but it better captures the specificity of the system. In contrast to the even characteristic case, an important remark is that the equations have a part of degree 2 due to the expressions of Q_i and Q_j and a degree α part due to the $x \mapsto x^\alpha$ permutation. This feature seems to make the analysis of d_{solv} slightly more complicated, for instance it is not encompassed in a standard Hilbert series which is a common tool to estimate d_{solv}. Experimentally, the behaviour of $\mathcal{F}_{\mathsf{CICO}}$ was clearly not the one of a generic system especially when α grows. Our experiments as well as further explanations are provided in Appendix A.2, and from them we also propose

Conjecture 2. *For $n_r \geq 2$, the maximal degree $d_{solv}(I_{\mathsf{CICO}})$ which occurs while computing a* DRL*-Gröbner basis for I_{CICO} is such that $d_{solv}(I_{\mathsf{CICO}}) \geq \alpha + 1 + 2(n_r - 2)$.*

Plugging this degree into Equation (4), we get the complexity estimate that we rely on to derive the number of rounds needed in `Anemoi`, i.e. Equation (2). From Conjecture 2 we can then derive a lower bound for the cost of Step 1 in the same way as in even characteristic. For Step 2, we have $\deg(I_{\mathsf{CICO}}) \leq 2^{n_r}\alpha^{n_r}$ using the Bezout bound. However, it turns out that the observed degree is much smaller:

Conjecture 3 (Degree, odd characteristic)**.** *We have $deg(I_{\mathsf{CICO}}) \leq (\alpha + 2)^{n_r}$.*

We are able to prove Conjecture 3 by hand for $n_r = 1$, and an investigation of the general case is left for future work. Actually, even by adopting the Bezout bound instead of Conjecture 3 as well as $\omega = 2$, a very rough upper-bound for Step 2 is $\mathcal{O}(n_r^2 \cdot 2^{2n_r} \cdot \alpha^{2n_r})$, and this is already quite below the cost of Step 1.

Several columns.

When $\ell > 1$ the number of equations and variables is naturally multiplied by ℓ and thus experiments were extremely difficult to conduct. We generalize our formulae to $\ell > 1$ by replacing n_r by $\ell \cdot n_r$ everywhere, whichis natural when looking at the expressions of the Macaulay bound and the Bezout bound. In fact, the bounds given for Rescue in [BSGL20] exhibit this extra ℓ factor.

7 Benchmarks

In this section, we compare various instances of Rescue-Prime, Poseidon and `Anemoi` with respect to R1CS (Section 7.1) and Plonk (Section 7.2). For the last one, we will also propose a comparison with ReinforcedConcrete.

To do so, we need to set the parameters. Then, let $\mathbb{F}_q$, where $q = p$, be a prime field, and let m be the number of field elements we operate on ($m = 2\ell$ for `Anemoi`). Besides, let s denote the security level in bits, n_r the number of rounds, and $\mathcal{C}_\alpha$ the cost of an exponentiation $x \mapsto x^\alpha$, as defined in Table 1.

Rescue-Prime requires $1.5 \cdot \max\{5, \lceil (s+2)/4m \rceil\}$ rounds when $\alpha = 3$ and $1.5 \cdot \max\{5, \lceil (s+3)/5.5m \rceil\}$ rounds when $\alpha = 5$ (see [AAB+19, SAD20]). Poseidon has $n_r = \mathsf{RF} + \mathsf{RP}$ rounds. While the bound is a complex expression, in our setting and for the safety margin recommended by the authors, it holds that $\mathsf{RF} = 8$, and that RP must be higher than (or equal to) $1.075 \cdot (\lceil \log_\alpha(2) \cdot \min\{s, \log_2(p)\} \rceil + \lceil \log_a m \rceil - \mathsf{RF})$.

While we also consider $\alpha = 17$ as a good exponent (the cost of an exponentiation for $\alpha = 17$ is not so far from an exponentiation for $\alpha = 5$), we will compare here only instances

Table 4: *Number of rounds for each hash function considered.*

s	$\log_2 q$	m	Rescue'	Poseidon	Anemoi
128	192	8	8	$(8+84)$	10
	256	6	9	$(8+84)$	10
	384	4	14	$(8+84)$	12
256	192	8	14	$(8+128)$	14
	256	6	17	$(8+171)$	15
	384	4	26	$(8+171)$	20

(a) *when* $\alpha = 3$.

s	$\log_2 q$	m	Rescue'	Poseidon	Anemoi
128	192	8	8	$(8+57)$	10
	256	6	8	$(8+57)$	10
	384	4	9	$(8+56)$	12
256	192	8	9	$(8+86)$	14
	256	6	12	$(8+117)$	15
	384	4	18	$(8+116)$	20

(b) *when* $\alpha = 5$.

Table 5: *Constraints comparison of several hash functions for R1CS.*

s	$\log_2 q$	m	Rescue'	Poseidon	Anemoi
128	192	8	256	296	**160**
	256	6	216	264	**120**
	384	4	224	232	**96**
256	192	8	448	384	**224**
	256	6	408	438	**180**
	384	4	416	406	**160**

(a) *when* $\alpha = 3$.

s	$\log_2 q$	m	Rescue'	Poseidon	Anemoi
128	192	8	384	363	**200**
	256	6	288	315	**150**
	384	4	216	264	**120**
256	192	8	432	450	**280**
	256	6	432	495	**225**
	384	4	432	444	**200**

(b) *when* $\alpha = 5$.

with $\alpha = 3$ and $\alpha = 5$, as previously proposed in the other designs. In the following, we will then consider the instances of Table 4.

7.1 R1CS Systems

We first estimate the number of constraints for R1CS. Using the *closed* `Flystel` of Figure 3b, we obtain the following verification equations for the S-Box:

$$\begin{cases} (v-y)^\alpha + \beta y^2 + \gamma - x = 0 \\ (v-y)^\alpha + \beta v^2 + \delta - u = 0 \end{cases} \tag{6}$$

Then, evaluating one S-Box costs $\mathcal{C}_\alpha$ constraints to obtain $(v-y)^\alpha$, and 1 constraint for each of the two quadratics. For Rescue-Prime and Poseidon, each S-Box costs $\mathcal{C}_\alpha$ constraints. As a consequence, when using Rescue-Prime, Poseidon and `Anemoi` as hash functions in sponge mode, the number of constraints is respectively $\mathcal{C}_\alpha \cdot 2m \cdot n_r$, $\mathcal{C}_\alpha \cdot (m\mathsf{RF} + \mathsf{RP})$, and $(\mathcal{C}_\alpha + 2) \cdot (\frac{m}{2} \cdot n_r)$.

Then we compare the number of constraints for those three schemes in Table 5. As we can see, the `Anemoi` permutations are consistently much more efficient than both Poseidon and Rescue-Prime by about a factor 2.

7.2 Plonk

Plonk [GWC19] uses a different, more complex arithmetization than R1CS. In standard Plonk, a constraint is of the form:

$$q_{L,i} \cdot a_i + q_{R,i} \cdot b_i + q_{O,i} \cdot c_i + q_{M,i} \cdot a_i \cdot b_i + q_{C,i} = 0$$

We refer to the $q_{j,i}$ values as gate selectors, as they can be used to "select" the functionality implemented by each constraint. The a_i, b_i, c_i values are called *wire values* and can be thought of as intermediate values in a calculation. Plonk uses an internal permutation argument to enforce consistency amongst wires, allowing us to assign a variable to multiple wires: the permutation argument forces the assigned values to match. For ease of exposition,

Table 6: *Constraints comparison of several hash functions for Plonk.*

s	$\log_2 q$	m	Rescue'	Poseidon	Anemoi
128	192	8	1152	5448	**520**
	256	6	756	3024	**330**
	384	4	560	1336	**216**
256	192	8	2016	8000	**728**
	256	6	1428	5808	**495**
	384	4	1040	2554	**360**

(a) *when* $\alpha = 3$.

s	$\log_2 q$	m	Rescue'	Poseidon	Anemoi
128	192	8	1280	4003	**560**
	256	6	768	2265	**360**
	384	4	432	1032	**240**
256	192	8	1440	5714	**784**
	256	6	1152	4245	**540**
	384	4	864	1932	**784**

(b) *when* $\alpha = 5$.

we will consider rounds to be shifted so that constant additions and linear operations come after the S-box.

As for R1CS, we again investigate Equation (6). In standard Plonk, evaluating one S-Box costs 1 constaint to derive $w = y - v$ and $\mathcal{C}_\alpha$ constraints to obtain $w^\alpha = (y - v)^\alpha$. We also need 1 constraint for each of the two quadratics, and 1 each for the sums on x, u. Concretely, for $\alpha = 5$:

1. $w = y - v$
2. $w_2 = w.w$
3. $w_4 = w_2.w_2$
4. $w_5 = w_4.w$
5. $y_2 = \beta.y.y$
6. $x = w_5 + y_2 + \gamma$
7. $v_2 = \beta.v.v$
8. $u = w_5 + v_2 + \delta$

While one would like to combine the last two pairs of constraints, standard Plonk does not provide enough wires: we need one wire to hold w_5, two wires to produce the quadratic term via the two inputs of the multiplication gate, and a fourth wire to hold u or x respectively. However, given a fourth wire or a dedicated single wire squaring gate, we can save 2 constraints on the above. Nevertheless, the total cost for the S-box layer with 3 wires is $(\mathcal{C}_\alpha + 5)\frac{m}{2}$.

The constant additions can be folded into the linear layer and can thus be disregarded. For $m > 2$, the linear layer itself consists of 2 separate matrix-vector multiplications, each producing $\frac{m}{2}$ sums of $\frac{m}{2}$ terms, requiring $m \cdot (\frac{m}{2} - 1)$ constraints. For $m = 2$, the linear layer is different, and we only require 2 constraints, which is especially relevant for the Jive$_2$ mode of operation.

Poseidon uses simpler S-Boxes, each costing $\mathcal{C}_\alpha$ constraints. Full rounds use m S-boxes whereas partial ones use only one. The linear layer costs $m \cdot (m - 1)$ constraints for all rounds.

Rescue-Prime uses m standard and m inverted S-Boxes, each costing $\mathcal{C}_\alpha$. Each round also utilizes 2 independent linear layers each costing $m \cdot (m - 1)$ constraints for all rounds. We then compare the number of constraints for those three schemes in Table 6. As we can see, again, `Anemoi` is consistently ahead of the competition with a significant margin.

7.2.1 Plonk Optimizations.

One of the more fruitful, but also challenging aspects of Plonk is its ability to extend the expressive power of the constraints at a reasonable cost. In the above analysis, the cost of the linear layer dominates that of the S-Boxes. This is particularly impactful for Poseidon, as the efficiency benefit of its partial rounds is negated. The recent work of Ambrona et al. [ASTW22] presents a set of generic and tailor-made optimizations for Plonk which are applicable to Poseidon.

Two of the more powerful (but also costly) optimizations involve adding new wires in the constraint system and also including terms of higher degree. For example, by adding a fourth wire, we can handle additions of many terms more efficiently: we still need one wire for the new total and one for the old one, but we can now consume two terms per constraint. Such an addition alone can almost halve the constraint cost of the linear layers. Adding a higher degree term, could allow one to directly produce x^α in a single constraint, instead of $\mathcal{C}_\alpha$. While very powerful, these additions incur a considerable cost to the prover's computation.

A computationally cheaper addition, is to add selectors so that one constraint can address the wire values of the *next* constraint. While this limits the next constraint (and may even necessitate leaving it blank) it can be quite powerful in practice: one addition using six wires can consume 5 terms ($a + b + c + d + e - T = 0$), while two additions on 3 wires can only consume 3: $(a + b - c = 0), (c + d - T = 0)$. Finally, we may add terms that do not increase the maximum degree of the constraint system for a lower cost, than those than do. In our case, we are interested in a quadratic gate so that we can calculate v^2 and t^2 without requiring the use of a second wire.

It is clear from the above that an exhaustive comparison of optimization options is beyond the scope of this work. At the same time, real-world usage implies that a reasonable set of optimizations have been applied before deployment. For this reason, we perform a minimal comparison between: Poseidon as optimized by Ambrona et al., the ReinforcedConcrete [BGK+21] Hash function which was built with Plonk optimizations in mind, and `Anemoi`. As Poseidon and ReinforcedConcrete are sponge based we use $s = 128, \alpha = 5$ and $m = 3$ to represent popular deployment choices, while we set $m = 2$ for Anemoi, using the Jive_2 mode. To facilitate comparison we also extrapolate a Jive_2 version of Poseidon with the optimizations of Ambrona et al., as well as ReinforcedConcrete.

We use one of the constraint systems used by Ambrona et al. [ASTW22]: a 3-wire constraint system with a x^5, as well as selectors for the next constraint wires:

$$q_L.a + q_R.b + q_O.c + q_M.a.b + q_5.c^5 + q_{L'}.a' + q_{R'}.b' + q_{O'}.c'$$

At a base level, the relations we need to express one $\texttt{AnemoiJive}_2$ round are:

1. $y - v - w = 0$
2. $w^5 + \beta yy + \gamma - x = 0$
3. $w^5 + \beta vv + \delta - u = 0$
4. $\tilde{u} - u - v - \rho = 0$
5. $\tilde{v} - \tilde{u} - v - \kappa = 0$

Where $\tilde{u}, \tilde{v}$ are the values of u, v after the linear layer and ρ, κ are round constants. We can save one constraint by calculating $\tilde{u}$ directly and eliminating u. We also need to make sure that the relations fit into the available wires, and make sure that the last constraint leaves the "next constraint" wires free, so that each set of round constraints can be followed by any constraint without restriction. To accomplish this, we also need to perform some reordering. Setting $\rho' = \rho + \delta$, the end result is:

1. $w^5 + \beta yy + \gamma - x = 0$, where: $(a, b, c) = (y, y, w)$ and $(a', b,' c') = (x\_, \_)$
2. $y - v - w = 0$, where: $(a, b, c) = (x, y, w)$ and $(a', b,' c') = (v, \_, \_)$
3. $w^5 + \beta vv + \rho' + v - \tilde{u} = 0$, where: $(a, b, c) = (v, v, w)$ and $(a', b,' c') = (\tilde{u}, \_, \_)$
4. $\tilde{v} - \tilde{u} - v - \kappa$, where: $(a, b, c) = (\tilde{u}, \tilde{v}, v)$ and $(a', b,' c') = (\_, \_, \_)$

Thus, we are able to perform one `AnemoiJive` round in 4 constraints, 2 additional constraints to account for the initial linear layer, and 1 extra constraint for the final Jive_2 addition (using the "next" wires). Using the calculations of [ASTW22], the prover cost for

Table 7: *Constraints comparison of several hash functions for Plonk. We fix $s = 128$, and prime field sizes of $256, 384$.*

	m	Constraints
Poseidon	3	110
	2	88
ReinforcedConcrete	3	378
	2	236
AnemoiJive	2	**79**

(a) *With 3 wires.*

	m	Constraints
Poseidon	3	98
	2	82
ReinforcedConcrete	3	267
	2	174
AnemoiJive	2	**60**

(b) *With 4 wires.*

each constraint increases by a factor between of at most 40% (usually smaller, as the cost is often dominated by exponentiation costs which scale more favourably), compared to standard 3-wire Plonk. Thus, even under the most conservative estimate the optimized version outperforms the standard one. With four wires, we are able to also eliminate w, by replacing it with $y - v$, reducing each round to 3 constraints, and need 1 fewer constraint for the final Jive_2 sum.

We summarize our findings in Table 7. We extrapolate the $m = 2$ costs for Poseidon and ReinforcedConcrete by assuming a Jive_2 mode of operation is feasible at no additional overhead or increase in rounds.We note while that the costs between Poseidon and Anemoi are directly comparable as they use the same features (namely x^5 and "next constraint" selectors), ReinforcedConcrete leverages lookup tables [BGK+21, GW20] instead. We do note that by [ASTW22, Table 2], the additional cost (compared to standard Plonk) for these features is between 22% and 40% for 3 wires and between 10% and 33% for 4 wires. The lower range represents the impact to prover exponentiations and the higher range represents the impact to prover FFTs.

8 Conclusion

We have made several contributions towards both the theoretical understanding and the practical use of arithmetization-oriented hash functions. Our main contribution is of course Anemoi, a family of permutations that are efficient across various arithmetization methods, yielding gains from 10% up to more than 50% depending on the context, over existing designs. Furthermore, in order to be able to design its main component, the Flystel structure, we had to first identify the link between arithmetization-orientation and CCZ-equivalence. We hope that functions such as the Flystel itself as well as similar ones will be studied by mathematicians as we believe those to be of independent interest.

Finally, we provided a new simple mode, Jive_b, which adds to the growing list of permutation-based mode of operations to provided b-to-1 compression functions of particular relevance in Merkle trees. It allows us to further improve upon the state-of-the-art, so that AnemoiJive requires only 60 Plonk constraints in total (when 4 wires are used), compared to the best sponge-based instance of Poseidon which requires 98 Plonk constraints.

Acknowledgements

Thanks to Markulf Kohlweiss, Antoine Rondelet and Duncan Tebbs for proof-reading a draft version of the paper and for providing insightful comments and suggestions. Thanks to Duncan Tebbs for providing an independent estimation of the Flystel circuit cost in terms of R1CS constraints.

References

[AAB+19] Abdelrahaman Aly, Tomer Ashur, Eli Ben-Sasson, Siemen Dhooghe, and Alan Szepieniec. Design of symmetric-key primitives for advanced cryptographic protocols. Cryptology ePrint Archive, Report 2019/426, 2019. https://eprint.iacr.org/2019/426.

[AAB+20] Abdelrahaman Aly, Tomer Ashur, Eli Ben-Sasson, Siemen Dhooghe, and Alan Szepieniec. Design of symmetric-key primitives for advanced cryptographic protocols. *IACR Trans. Symm. Cryptol.*, 2020(3):1–45, 2020.

[AGR+16] Martin R. Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. MiMC: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 191–219, Hanoi, Vietnam, December 4–8, 2016. Springer, Heidelberg, Germany.

[ASTW22] Miguel Ambrona, Anne-Laure Schmitt, Raphael R. Toledo, and Danny Willems. New optimization techniques for plonk's arithmetization. Cryptology ePrint Archive, Paper 2022/462, 2022. https://eprint.iacr.org/2022/462.

[Bar04] Magali Bardet. *Étude des systèmes algébriques surdéterminés. Applications aux codes correcteurs et à la cryptographie.* Theses, Université Pierre et Marie Curie - Paris VI, December 2004.

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy*, pages 315–334, San Francisco, CA, USA, May 21–23, 2018. IEEE Computer Society Press.

[BBC+20] Christof Beierle, Alex Biryukov, Luan Cardoso dos Santos, Johann Großschädl, Léo Perrin, Aleksei Udovenko, Vesselin Velichkov, and Qingju Wang. Lightweight AEAD and hashing using the Sparkle permutation family. *IACR Trans. Symm. Cryptol.*, 2020(S1):208–261, 2020.

[BC90] Jurjen N. Bos and Matthijs J. Coster. Addition chain heuristics. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 400–407, Santa Barbara, CA, USA, August 20–24, 1990. Springer, Heidelberg, Germany.

[BCD+20] Tim Beyne, Anne Canteaut, Itai Dinur, Maria Eichlseder, Gregor Leander, Gaëtan Leurent, María Naya-Plasencia, Léo Perrin, Yu Sasaki, Yosuke Todo, and Friedrich Wiemer. Out of oddity - new cryptanalytic techniques against symmetric primitives optimized for integrity proof systems. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 299–328, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Heidelberg, Germany.

[BCG+13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 90–108, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany.

[BDPA13] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*,

volume 7881 of *LNCS*, pages 313–314, Athens, Greece, May 26–30, 2013. Springer, Heidelberg, Germany.

[BDPVA07] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Sponge functions. In *ECRYPT hash workshop*, number 9. Citeseer, 2007.

[Ber08] Daniel J Bernstein. The Salsa20 family of stream ciphers. In *New stream cipher designs*, pages 84–97. Springer, 2008.

[BGK+21] Mario Barbara, Lorenzo Grassi, Dmitry Khovratovich, Reinhard Lueftenegger, Christian Rechberger, Markus Schofnegger, and Roman Walch. Reinforced concrete: Fast hash function for zero knowledge proofs and verifiable computation. Cryptology ePrint Archive, Report 2021/1038, 2021. https://eprint.iacr.org/2021/1038.

[BS01] Alex Biryukov and Adi Shamir. Structural cryptanalysis of SASAS. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 394–405, Innsbruck, Austria, May 6–10, 2001. Springer, Heidelberg, Germany.

[BS10] Alex Biryukov and Adi Shamir. Structural cryptanalysis of SASAS. *Journal of Cryptology*, 23(4):505–518, October 2010.

[BSCG+14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. Cryptology ePrint Archive, Report 2014/349, 2014. https://ia.cr/2014/349.

[BSGL20] Eli Ben-Sasson, Lior Goldberg, and David Levit. Stark friendly hash – survey and recommendation. Cryptology ePrint Archive, Report 2020/948, 2020. https://ia.cr/2020/948.

[BSV07] Thomas Baignères, Jacques Stern, and Serge Vaudenay. Linear cryptanalysis of non binary ciphers. In Carlisle M. Adams, Ali Miri, and Michael J. Wiener, editors, *SAC 2007*, volume 4876 of *LNCS*, pages 184–211, Ottawa, Canada, August 16–17, 2007. Springer, Heidelberg, Germany.

[CCZ98] Claude Carlet, Pascale Charpin, and Victor Zinoviev. Codes, bent functions and permutations suitable for DES-like cryptosystems. *Designs, Codes and Cryptography*, 15(2):125–156, 1998.

[CDP17] A. Canteaut, S. Duval, and L. Perrin. A generalisation of Dillon's APN permutation with the best known differential and nonlinear properties for all fields of size 2^{4k+2}. *IEEE Transactions on Information Theory*, 63(11):7575–7591, Nov 2017.

[CLO07] David Cox, John Little, and Donal O'Shea. Ideals, varieties, and algorithms. an introduction to computational algebraic geometry and commutative algebra. 2007.

[CM97] Robert S Coulter and Rex W Matthews. Bent polynomials over finite fields. *Bulletin of the Australian Mathematical Society*, 56(3):429–437, 1997.

[CP19] Anne Canteaut and Léo Perrin. On CCZ-equivalence, extended-affine equivalence, and function twisting. *Finite Fields and Their Applications*, 56:209–246, 2019.

[DGGK21] Christoph Dobraunig, Lorenzo Grassi, Anna Guinet, and Daniël Kuijsters. Ciminion: Symmetric encryption based on toffoli-gates over large finite fields. Springer-Verlag, 2021.

[DL18] Sébastien Duval and Gaëtan Leurent. MDS matrices with lightweight circuits. *IACR Trans. Symm. Cryptol.*, 2018(2):48–78, 2018.

[DP19] Orr Dunkelman and Léo Perrin. Adapting rigidity to symmetric cryptography: Towards" unswerving" designs. In *Proceedings of the 5th ACM Workshop on Security Standardisation Research Workshop*, pages 69–80, 2019.

[Dwo15] Morris Dworkin. Sha-3 standard: Permutation-based hash and extendable-output functions, 2015-08-04 2015.

[Fau99] Jean-Charles Faugére. A new efficient algorithm for computing gröbner bases (f4). *Journal of Pure and Applied Algebra*, 139(1):61–88, 1999.

[Fau02] Jean Charles Faugère. A new efficient algorithm for computing gröbner bases without reduction to zero (f5). In *Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation*, ISSAC '02, pages 75–83, New York, NY, USA, 2002. Association for Computing Machinery.

[FGLM93] J.C. Faugère, P. Gianni, D. Lazard, and T. Mora. Efficient computation of zero-dimensional gröbner bases by change of ordering. *Journal of Symbolic Computation*, 16(4):329–344, 1993.

[FM11] Jean-Charles Faugère and Chenqi Mou. Fast algorithm for change of ordering of zero-dimensional Gröbner bases with sparse multiplication matrices. In *ISSAC 2011 - International Symposium on Symbolic and Algebraic Computation*, pages 115–122, San Jose, United States, June 2011. ACM.

[GKR+21a] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for Zero-Knowledge proof systems. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 519–535. USENIX Association, August 2021.

[GKR+21b] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2020*, pages 1–17. USENIX Association, August 2021.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[GØSW22] Lorenzo Grassi, Morten Øygarden, Markus Schofnegger, and Roman Walch. From farfalle to megafono via ciminion: The prf hydra for mpc applications. Cryptology ePrint Archive, Report 2022/342, 2022. https://ia.cr/2022/342.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany.

[GW19] Ariel Gabizon and Zachary J. Williamson. Proposal: The turbo-plonk program syntax for specifying snark programs. Available online at https://docs.zkproof.org/pages/standards/accepted-workshop3/proposal-turbo_plonk.pdf, 2019.

[GW20] Ariel Gabizon and Zachary J. Williamson. plookup: A simplified polynomial protocol for lookup tables. Cryptology ePrint Archive, Report 2020/315, 2020. https://eprint.iacr.org/2020/315.

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019. https://eprint.iacr.org/2019/953.

[Hir16] Shoichi Hirose. Sequential hashing with minimum padding. In *NIST Workshop on Lightweight Cryptography 2016*. National Institute of Standards and Technology (NIST), 2016.

[Knu95] Lars R. Knudsen. Truncated and higher order differentials. In Bart Preneel, editor, *FSE'94*, volume 1008 of *LNCS*, pages 196–211, Leuven, Belgium, December 14–16, 1995. Springer, Heidelberg, Germany.

[Lou94] W.W.A.P. Loustaunau. *An Introduction to Grobner Bases*. American Mathematical Soc., 1994.

[LTYW18] Yongqiang Li, Shizhu Tian, Yuyin Yu, and Mingsheng Wang. On the generalization of butterfly structure. *IACR Trans. Symm. Cryptol.*, 2018(1):160–179, 2018.

[McL21] Michael B. McLoughlin. addchain: Cryptographic addition chain generation in go. Repository https://github.com/mmcloughlin/addchain, October 2021.

[Nyb94] Kaisa Nyberg. Differentially uniform mappings for cryptography. In Tor Helleseth, editor, *EUROCRYPT'93*, volume 765 of *LNCS*, pages 55–64, Lofthus, Norway, May 23–27, 1994. Springer, Heidelberg, Germany.

[PUB16] Léo Perrin, Aleksei Udovenko, and Alex Biryukov. Cryptanalysis of a theorem: Decomposing the only known solution to the big APN problem. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 93–122, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany.

[SAD20] Alan Szepieniec, Tomer Ashur, and Siemen Dhooghe. Rescue-prime: a standard specification (SoK). Cryptology ePrint Archive, Report 2020/1143, 2020. https://eprint.iacr.org/2020/1143.

Table 8: *Gröbner basis computation on the $\mathcal{F}_{\mathsf{CICO}}$ system with $\ell = 1$ over $\mathbb{F}_{2^{15}}$.*

n_r	$d_{\text{solv}}(\mathcal{F}_{\mathsf{CICO}})$	Macaulay bound	Step degrees	Total F4 time (s)
2	8	9	$3 \to 8$	0.009
3	12	13	$3 \to 12$	0.510
4	15	17	$3 \to 15$	11.580
5	18	21	$3 \to 18$	344.050
6	21	25	$3 \to 21$	14807.639

A Details on Algebraic Attacks

We now provide more technical content related to the Gröbner basis attack on `Anemoi`. Regardless of the value of q we have

$$(x_{j+1}, y_{j+1}) := \mathcal{H}(\mathcal{M}_x(x_j, y_j)[0] + c_j, \mathcal{M}_x(x_j, y_j)[1] + d_j),$$

where $\mathcal{M}_x$ is the linear layer and where $(c_j, d_j) \in \mathbb{F}_q^2$ are round constants.

A.1 Characteristic 2

For even characteristic we chose $Q_i(x) = \beta x^3 + \gamma$ and $Q_f(x) = \beta x^3 + \delta$ for $\gamma \neq \delta$, so that

$$(u, v) = \mathcal{H}(x, y) \Leftrightarrow \begin{cases} (v-y)^3 + \beta y^3 + \gamma - x &= 0 \\ (v-y)^3 + \beta v^3 + \delta - u &= 0. \end{cases}$$

Assuming a linear layer of the form $\mathcal{M}_x : (x, y) \mapsto (x + \alpha y, \alpha x + (\alpha^2 + 1)y)$ where α is a primitive element of $\mathbb{F}_q = \mathbb{F}_{2^n}$, the cubic equations at hand are

$$\begin{cases} f_j &:= (y_{j+1} - \alpha x_j - (\alpha^2+1)y_j - d_j)^3 + \beta(\alpha x_j + (\alpha^2+1)y_j + d_j)^3 - \alpha y_j - x_j - c_j \\ g_j &:= (y_{j+1} - \alpha x_j - (\alpha^2+1)y_j - d_j)^3 + \beta y_{j+1}^3 - x_{j+1}. \end{cases}$$

Experiments for Conjecture 1.

We compared the behaviour of Magma's F4 algorithm on the $\mathcal{F}_{\mathsf{CICO}}$ system for various values of n_r. In Table 8, "$d_{\text{solv}}(\mathcal{F}_{\mathsf{CICO}})$" stands for the maximal degree which occurs while computing the Gröbner basis and "Macaulay bound" is equal to $4n_r + 1$. The F4 algorithm consists in a sequence of steps, each of these steps considering all pairs of polynomials having minimal degree and treating them at the same time. Column "Step degrees" lists the degree of these steps and may provide more information than just the solving degree alone. More precisely, the "$x \to y$" indication means that the working degree of F4 increases by 1 at each step from degree x to degree y and that no *degree fall polynomials* occur, which can be considered as the expected pattern in this case. On the one hand, we see that the $\mathcal{F}_{\mathsf{CICO}}$ system seems to behave as such by looking at the sequence of step degrees. On the other hand, the solving degree grows more slowly than the Macaulay bound which is the expected bound for a random cubic system in $2n_r$ equations and more than $2n_r$ variables. Finally, the lower bound of $3n_r$ that we give in Conjecture 1 seems quite conservative regarding our results.

A.2 Odd characteristic

In this section, we explain why the analysis may be more complicated for odd q and we give the results of our Magma experiments to support Conjecture 2. This part is a bit more involved and we refer the reader to [CLO07, Bar04] for some details on Gröbner basis computation. We may consider a DRL ordering such that the largest variables are

the y_i's and for which $y_{n_r} > \cdots > y_0 > x_{n_r-1} > \cdots > x_1 > x_{n_r} > x_0$. This choice is quite natural since x_0 and x_{n_r} are the fixed variables in CICO. Similarly to Appendix A.1, the expressions for f_j and g_j can be obtained from the verification equations

$$(v-y)^\alpha + \beta y^2 + \gamma - x = 0$$
$$(v-y)^\alpha + \beta v^2 + \delta - u = 0.$$

These are polynomials of degree α with leading monomial y_{j+1}^α, and their difference $h_j := g_j - f_j$ is quadratic with leading monomial y_{j+1}^2. By Buchberger's First Criterion, this implies that polynomial pairs involving equations from different rounds $i \neq j$ do not need to be treated in the first step of F4 since the leading terms are coprime. In particular, this first step only selects the pairs $\{h_j, f_j\}$ and therefore it may be relevant to analyze what happens for only one round:

Lemma 1. *A Gröbner basis for $\{h_j, f_j\}$ can be found in degree $\alpha+1$ and the set of leading terms in the reduced Gröbner basis is*

$$\zeta_j := \{y_{j+1}^2, y_j^\alpha\} \cup \{y_{j+1} x_{j+1}^u y_j^{\alpha-1-u},\ 0 \leq u \leq \alpha - 3\}.$$

In particular, there is one quadratic leading term and $\alpha - 1$ leading terms of degree α. The latter come from degree falls from degree $\alpha + 1$ to degree α.

Experiments for Conjecture 2.

We proceed in the same way as in Appendix A.1 to derive the following Table 9 but we do not include the Macaulay bound. Instead, we compare the behaviour of Magma's F4 on $\mathcal{F}$ and $\mathcal{F}_{\mathsf{CICO}}$ for various values of α and n_r to grasp the effect of the fixed variables. Compared to Table 8, we may also bracket the maximal degree of a polynomial in the reduced Gröbner basis in columns "$d_{\mathrm{solv}}(\mathcal{F})$" and "$d_{\mathrm{solv}}(\mathcal{F}_{\mathsf{CICO}})$".

A first observation is that the results for $\mathcal{F}$ and $\mathcal{F}_{\mathsf{CICO}}$ are the same when $\alpha = 3$. This might be a consequence of Lemma 1. Indeed, the leading terms in the system do not depend on the x_j variables in this case, and therefore fixing $x_0 = 0$ and $x_{n_r} = 0$ does not seem to affect the analysis. Moreover, the behaviour of the Gröbner basis algorithm seems quite close to the one on a regular system: the sequence of step degrees increases steadily until we reach the maximal degree. Also, there are no degree falls apart from the ones associated to the *plateau* at $\alpha + 1 = 4$ which are once again a consequence of Lemma 1. All seems to happen as if there were no x_j variables, so among $\{h_j, f_j\}$ one would only keep f_j since h_j expresses x_{j+1} in terms of larger variables. Also, note that the observed value $2n_r + 1$ for the maximal degree indeed corresponds to the Macaulay bound $n_r(3-1)+1$ for $\{f_0, \ldots, f_{n_r-1}\}$ assuming that it is regular.

When α grows, the behaviour of $\mathcal{F}_{\mathsf{CICO}}$ starts to deviate from the one of $\mathcal{F}$. The unusual behaviour of the computation may also be seen by looking at the sequence of step degrees which is quite erratic. In particular, many degree fall polynomials at degree larger than $\alpha + 1$ occur and they imply why the solving degree remains quite low. We have not been able to analyze these degree falls at high degree. Overall, an explanation only based on simple algebraic considerations seems out of reach since it would probably be valid for any value of α while the observed results depend a lot on α. Still, it is reasonable to believe that the increased sparsity of the system for large α comes into play.

Regarding the experimental lower bound of Conjecture 2, it would be tempting to suggest an increase of d_{solv} larger than 2 at each round when α is higher, for instance 3 for $\alpha = 11$ and more generally λ_α for α where $\alpha \mapsto \lambda_\alpha$ slowly increases. However, looking at the case $\alpha = 9$ between rounds 2 and 3 and rounds 3 and 4 should not give us confidence regarding a constant increase, and also there are no theoretical arguments (for instance Hilbert series based ones) to support such a claim.

Table 9: *Gröbner basis computation on $\mathcal{F}$ and $\mathcal{F}_{\mathsf{CICO}}$ for $3 \leq \alpha \leq 11$ and for various number of rounds (odd characteristic).*

α	n_r	$d_{\text{solv}}(\mathcal{F})$	$d_{\text{solv}}(\mathcal{F}_{\mathsf{CICO}})$	Step degrees $\mathcal{F}_{\mathsf{CICO}}$
3	2	5	5(5)	3,4,4,5
	3	7	7(7)	3,4,4,5,6,7
	4	9	9(9)	3,4,4,5,6,7,8,9
	5	11	11(11)	3,4,4,5,6,7,8,9,10,11
	6	13	13(13)	3,4,4,5,6,7,8,9,10,11,12,13
5	2	10(10)	7(6)	5,6,6,6,6,7,6,6
	3	15(15)	8(8)	5,6,6,6,6,7,8,8,8,8,8,8
	4	18(18)	10(10)	5,6,6,6,6,7,8,8,8,9,9,9,9,10,10,10,10
	5		12(12)	5,6,6,6,6,7,8,8,8,9,9,9,9,10,9,10,9,10,10,11,11,11,11,12,12,12
7	2	14(14)	8(7)	7,8,8,8,8,8,8,8,8,7,7
	3		10(9)	7,8,8,8,8,8,8,8,9,10,10,10,10,10,10,10,10
	4		12(12)	7,8,8,8,8,8,8,8,9,10,10,10,10,11,11,11,12,11,12,12,12,12,12,12,12
9	2	18(18)	10(9)	
	3		13(11)	
	4		15(?)	
11	2		12(10)	
	3		15(13)	
	4		18(?)	

B Reference Implementation

A full reference implementation of `Anemoi`, including `AnemoiJive` and `AnemoiSponge`, is provided in our GitHub[4] repository. It contains various routines to evaluate these functions and to generate the corresponding systems of equations as well. Nevertheless, we include some snippets from this implementation below.

The following SAGE functions evaluate the linear layers that are used in `Anemoi` depending on the number ℓ of columns.

```
def M_2(x_input, b):
    x = x_input[:]
    x[0] += b*x[1]
    x[1] += b*x[0]
    return x

def M_3(x_input, b):
    """Figure 6 of [DL18]."""
    x = x_input[:]
    t = x[0] + b*x[2]
    x[2] += x[1]
    x[2] += b*x[0]
    x[0] = t + x[2]
    x[1] += t
    return x


def M_4(x_input, b):
    """Figure 8 of [DL18]."""
    x = x_input[:]
    x[0] += x[1]
    x[2] += x[3]
    x[3] += b*x[0]
    x[1]  = b*(x[1] + x[2])
    x[0] += x[1]
    x[2] += b*x[3]
    x[1] += x[2]
    x[3] += x[0]
    return x
```

The following function computes the number of rounds using our heuristic.

```
def get_n_rounds(s, l, alpha):
    """Returns the number of rounds needed in Anemoi (based on the
    complexity of algebraic attacks).

    """
    r = 0
    complexity = 0
    while complexity < 2**s:
        r += 1
        complexity = binomial(
            2*l*r + alpha + 1 + 2*(l*r-2),
            2*l*r
        )**2
    r += l+1 # security margin
    if r > 10:
        return r
    else:
        return 10
```

[4] https://github.com/vesselinux/anemoi-hash/

Finally, the two modes in which `Anemoi` can be plugged are implemented by the following function. They both take an input `P` which must implement a permutation. Concretely, it must be such that calling `P(x)` on a list `x` of elements of the relevant field returns a list of elements of the same field of the same size.

```
def jive(P, b, _x):
    """Returns an output b times smaller than _x using the Jive mode of
    operation and the permutation P.

    """
    if b < 2:
        raise Exception("b must be at least equal to 2")
    if P.input_size() % b != 0:
        raise Exception("b must divide the input size!")
    x = _x[:]
    u = P(x)
    compressed = []
    c = P.input_size()/b # length of the compressed output
    for i in range(0, c):
        compressed.append(sum(x[i+c*j] + u[i+c*j]
                              for j in range(0, b)))
    return compressed

def sponge_hash(P, r, h, _x):
    """Uses Hirose's variant of the sponge construction to hash the
    message x using the permutation P with rate r, outputting a digest
    of size h.

    """
    x = _x[:]
    if P.input_size() <= r:
        raise Exception("rate must be strictly smaller than state size!")
    # message padding (and domain separator computation)
    if len(x) % r == 0 and len(x) != 0:
        sigma = 1
    else:
        sigma = 0
        x += [1]
        x += (len(x) % r)*[0]
    padded_x = [[x[pos+i] for i in range(0, r)]
                for pos in range(0, len(x), r)]
    # absorption phase
    internal_state = [0] * P.input_size()
    for pos in range(0, len(padded_x)):
        for i in range(0, r):
            internal_state[i] += padded_x[pos][i]
        internal_state = P(internal_state)
        if pos == len(padded_x)-1:
            # adding sigma if it is the last block
            internal_state[-1] += sigma
    # squeezing
    digest = []
    pos = 0
    while len(digest) < h:
        digest.append(internal_state[pos])
        pos += 1
        if pos == r:
            pos = 0
            internal_state = P(internal_state)
    return digest
```