Reinforced Concrete: Fast Hash Function for Zero Knowledge Proofs and Verifiable Computation

Mario Barbara*, Lorenzo Grassi†, Dmitry Khovratovich‡, Reinhard Lüftenegger§, Christian Rechberger¶, Markus Schofnegger‖, Roman Walch**
*§¶‖** – TU Graz, Austria
† – Radboud University, Netherlands
‡ – Dusk Network, Luxembourg
9 August 2021

***Abstract*—We propose a new hash function `Reinforced Concrete` for the proof systems that support lookup tables, concretely Plookup based on KZG commitments or FRI. It has two solid advantages over predecessors: *(a)* Table lookups instead of (big) modular reductions are much faster both in ZK and plain computations thus making verifiable computation protocols based on recursive proofs (current trend) much more efficient; *(b)* the security is no longer solely based on (high) algebraic degree but rather on more traditional AES-like components inheriting decades of public scrutiny. Our design also employs a novel and fast field-to-tables conversion, which is of independent interest and can be used in other Plookup-friendly constructions.**

The new hash function is suitable for a wide range of applications like privacy-preserving cryptocurrencies, verifiable encryption, protocols with state membership proofs, or verifiable computation. It may serve as a drop-in replacement for various prime-field hashes such as variants of MiMC, Poseidon, Pedersen hash, and others.

Keywords: Hash functions, verifiable computation, zk-snarks, finite fields.

Name	Performance		
	Zero knowledge		Plain
	R1CS	Plookup	
	(gates)		(ms)
Poseidon-BLS/BN	243	438	19
Rescue-BLS/BN	288	364	415
Rescue-Prime-BLS/BN	252	321	362
Feistel-MiMC-BLS/BN	1326	1326	34
SHA-256	27534	≈ 3000	0.37
Blake2s	21006	≈ 2000	0.22
SINSEMILLA	869*	670	131
`Reinforced Concrete`-BN/BLS	-	267	3.3
`Reinforced Concrete`-FRI	-	265	1.03

TABLE I: Hashing 512 bits of data (two field elements) with different functions. * – as Pedersen hash.

I. Introduction

The recent years have been marked as a thrive or distributed computations, in particular blockchain and cryptocurrency protocols. To speed up consensus on the result of those computations, these protocols use extensively various zero knowledge proofs and verifiable computation protocols, as those provide succinct proofs of correctness. A number of schemes from different cryptography areas, from MPC and proof systems to hash functions, are required to run those protocols. Applications of hash functions are are numerous, but in this paper we develop an instrument that targets two specific use-cases:

- **Fast and efficient set membership proofs** based on Merkle tree accumulators. Immensely popular in cryptocurrency protocols [3], [2], this case requires a hash function for the tree such that its ZK circuit has smallest number of gates, thus minimizing the proof creation time.
- **Verifiable computation based on recursive proofs**. The variant using FRI commitments employs a commit-reveal scheme with Merkle trees, and paths in these trees are later proven valid. Here we minimize both plain computation time and the number of circuit constraints.

The new hash function, named `Reinforced Concrete`, brings higher performance and improved security compared to competitors.

a) Proof systems: A number of proof systems for NP languages have appeared since 1990s, with the most efficient ones being able to compute a proof in time linear of the witness size with proof being constant or logarithmic size [48], [36], [12], [28]. In order to construct a proof for a program, it is first arithmetised as a circuit over certain prime field. The circuit gates that count towards the prover time are field multiplications and additions, though custom polynomial gates are also possible with some overhead. The most recent enrichment is Plookup [27], which enables table lookup gates. This extension is particularly interesting in our usecase since traditional cryptographic hash functions can be represented with a much smaller circuit when lookup gates are possible (cf. columns 2 and 3 of Table I), and thus enable a faster prover.

b) Existing Hash Functions in the Literature: There already exist several hash functions crafted for the first use-case with the number of circuit gates (or equivalently low-degree polynomial constraints) being the primary metric. Examples include prime-field (Feistel) MiMC versions [6], [5], FRIDAY [8], POSEIDON [31], and *Rescue* [7] (and its updated version *Rescue*-Prime [49]). All these hash functions share some common features, as the fact that the non-linear layer is instantiated via a simple power map. Focusing on POSEIDON, it is based on the HADES design strategy [32], which makes use of an uneven distribution of the S-boxes, namely, full S-box layers in the external rounds and partial S-box layers in the middle ones, in order to minimize the multiplicative complexity. The external rounds provide security against statistical attacks, while the internal rounds have the goal of increasing the degree of the permutation.

While most of them have withstood public scrutiny, two issues have been raised:

1) as designing symmetric-key primitives in this domain is relatively new, several algebraic attacks recently appeared in the literature [4], [15], [26], [39], [35];
2) The plain performance is not satisfactory (see last column of Table I), since each round of such schemes requires a finite field multiplication, which is relatively expensive (hundreds of CPU cycles) compared to bit operations utilized in traditional hash functions.

A rather recent addition to this set is SINSEMILLA [3, Sec. 5.4.1.9], which is Pedersen hash function[3, Sec. 5.4.1.7] optimized for faster elliptic-curve exponentiations using table lookups.

c) Our Contributions.: We present a new hash function `Reinforced Concrete`, in short `RC`, over $\mathbb{F}_p$ exploiting all the advantages of Plookup and suitable for both membership proofs and verifiable computation.

The permutation that instantiated `Reinforced Concrete` is based on the POSEIDON design strategy, which means, it is composed of two different rounds: the external rounds for preventing statistical attacks and the internal ones for preventing algebraic attacks. However, instead of using simple power maps as in POSEIDON and *Rescue*, we decided to use a single round function for the middle part with a complex algebraic structure.

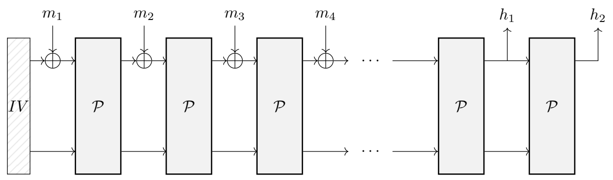


Fig. 1: A sponge hash function with a fixed-size output.

Such high-degree round function combines a layer of substitution box (such as in AES) with a field element decomposition in just a handful of small operations (or table gates in the circuit), and it admits a very simple representation when using look-up tables, as e.g. in the case of AES [25] and AES-like ciphers. As a result, the security argument we propose for preventing algebraic attacks resembles the one well known and accepted in the literature for AES and more generally AES-like ciphers, for which the algebraic attacks can attack only a tiny fraction of the rounds compared to the statistical attacks [23], [22]. Also, compared to Pedersen hash/SINSEMILLA we provide pre-image resistance in addition to collision resistance, and do not rely on hardness assumptions that are known to be weak in a world with practical quantum computers.

Besides a different security argument with respect to the one used by POSEIDON and *Rescue*, our hash function is faster in plain performance and simultaneously has fewer gates as a circuit. The performance depends on the field and we show how we can make it even better for specially crafted field sizes. Concretely, using generic prime fields (such as the scalar fields of the BLS12-381 or BN254 elliptic curves) `Reinforced Concrete` is faster by a factor of 5 compared to POSEIDON and by a factor of 125 compared to *Rescue* and 110 compared to *Rescue*-Prime. Using specially crafted fields increases these factors to 16, 348, and 285 respectively. `RC` is, thereby, only by a factor of 5 slower than Blake2s, the fastest traditional hash algorithm we benchmarked, but requires 7 times less gates when encoded into a circuit.

II. `REINFORCED CONCRETE` IN A NUTSHELL

`Reinforced Concrete`, in short `RC`, is a family of bijective transformations over a field $\mathbb{F}_p$ for a family of primes $p \gg 3$ to be used in the Sponge framework [13] (Fig. 1) in order to construct hash functions and authenticated encryption schemes. The field $\mathbb{F}_p$ is supposed to be in the order of magnitude of 2^{256}, with scalar fields of the curves BN254 [51] and BLS12-381[1] being primary instances, and a FRI-friendly prime $\widehat{p}$, crafted specially for the verifiable computation case (Section VI).

[1]https://electriccoin.co/blog/new-snark-curve/

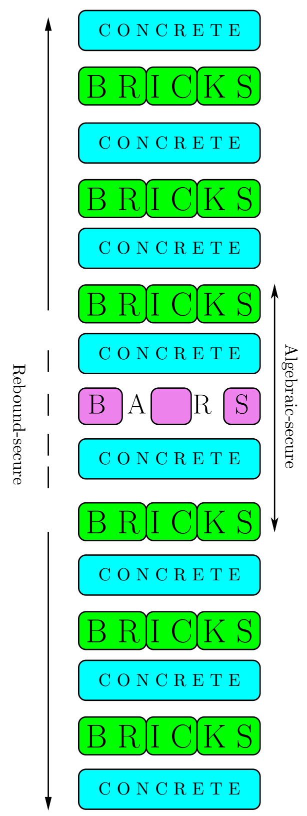


Fig. 2: The `Reinforced Concrete` permutation. The middle Br-C-B-C-Br part is secure against algebraic attacks whereas C-Br-C-Br-C-Br-C-Br-C-Br-C is secure against rebounds (more generally, statistical) attacks.

All `RC` permutations map $\mathbb{F}_p^3$ to itself, and we reserve 1 field element for the capacity in sponge, thus aiming for the 128-bit security against collision and preimage attacks for all instances. A single call to `RC` thus suffices for a 2-to-1 compression function. All the instances of `RC` are identical in security properties and offer 128-bit security against all known attacks on hash functions. The performance, however, does depend on the underlying field, and the p_{ST} instance is substantially faster.

a) Design: The `RC` design depicted at Figure 2 is a modification of a traditional word-oriented SP-network (SPN) for constructing (keyed or keyless) cryptographic permutations. The `RC` design differs from a traditional SPN in two aspects:

- the middle layer of the SP network is replaced by a special component called `Bars`. This special component effectively reinforces the permutation against cryptanalytic approaches that would cover many more rounds without `Bars`. It does not admit a low-degree polynomial description but can be implemented as a circuit with reasonable costs in ZK.
- instead of applying independent non-linear transformations on single words, `RC` uses (low-degree) non-linear layers, called `Bricks`, that additionally mix different words. It provides resistance against statistical cryptanalysis and is cheap in the zero knowledge, i.e. via gate counting.

The third component, `Concrete`, is an analog of the traditional affine layer but over $\mathbb{F}$. It ensures diffusion to make statistical or algebraic properties expand to the entire state, and is also cheap in ZK.

b) Layout: The `Bricks` and `Concrete` layers interleave exactly as in traditional SPN designs [25]. As `RC` is used in a sponge framework, the `Bricks` components at either end would bring no security against collision or preimage attacks, so we start and end with `Concrete`. The middle call to `Bricks` is replaced with `Bars`. The rationale behind putting all `Bar` into a single layer is that start-from-the-middle attacks are somewhat easier to find and thus we plan to detect them all in the design phase.

III. Security requirements and claims

Our high-level security claims, which determine the parameter selection for `RC`, are the following.

- For the sponge hash function with `RC`, we aim for a collision and preimage resistance up to 2^{128} field operations for 256-bit fields. We want to be able to instantiate a random oracle in protocols up to 2^{128} calls.
- For the authenticated encryption scheme using `RC`, we aim for confidentiality and integrity up to 2^{128} encrypted messages for 256-bit fields.
- When using the `RC` in other future schemes, we aim for a 1-element CICO security up to 2^{128} field operations. More concretely, it should be infeasible to find such x_1, x_2, y_1, y_2 such that

$$\texttt{Reinforced Concrete}(0, x_1, x_2) = (0, y_1, y_2)$$

As the properties above cannot be formally proven or reduced to assumptions, we back them up with certain requirements for the different components of our permutation. In particular, we focus on the following two classes of attacks, respectively statistical and algebraic attacks. As already mentioned in the introduction, we make use of the Hades/Poseidon design strategy in order to provide security:

- Statistical attacks (including differential, linear, rebound, truncated, impossible, MiTM, boomerang) cannot be mounted on `RC` even with the

middle component `Bricks`-`Concrete`-`Bars`-`Concrete`-`Bricks` replaced with a single `Bricks` layer up 2^{128} field operations.
- The middle component `Bricks`-`Concrete`-`Bars`-`Concrete`-`Bricks` resists invariant subspace and algebraic (e.g., Gröbner basis) attacks up to 2^{128} field operations. Due to the high degree and because we are working over prime fields, we also expect ample resistance against higher-order differential attacks (e.g., zero-sum distinguishers or cube attacks).

We give a detailed overview of statistical attack approaches in Appendix D, and we focus on algebraic attacks in Appendix G.

IV. Primary Use Cases

Our new hash function targets several usecases. Two of them are depicted at Figure 3 as well as the usage of `RC`:

- Verifiable computation and set membership protocols (see a more formal description below) are at the application level (top) and are exposed to users. The set membership protocol uses a Merkle tree as an accumulator, which uses `RC` to build the tree.
- In order to provide succinct proofs, both protocols employ some Interactive Oracle Proof framework, with Plookup [27] yielding the fastest prover when `RC` is involved.
- The IOP uses a polynomial commitment scheme, of which KZG [38] and FRI [11] are most popular. The former brings succinct proofs but requires a trusted setup. The latter uses Merkle tree for the commit-reveal process, and `RC` is used to build the tree in the verifiable computation usecase.

Now we give a more formal description of these concepts.

a) Zero Knowledge Set Membership: A typical usecase for set membership proofs is as follows. Parties $P_1, P_2, \dots, P_n$ add entries $V_1, V_2, \dots, V_k$ to some public accumulator $\mathfrak{A}$. Then at any point any party P_j can prove that $V_i \in \mathfrak{A}$. For instance, in Zcash [3] V_i are unspent transactions and $\mathfrak{A}$ is a Merkle tree over them, so that in order to spend transaction V an owner is required to provide a proof of knowledge that $V \in \mathfrak{A}$ as well as a proof of knowledge of some secret committed within V. In this usecase, `RC` would comprise the Merkle tree.

b) Verifiable Encryption.: In the same set of protocols as above, it may be legally required to accompany a transaction of d coins from Alice to Bob with $C = E_K(d, Alice, Bob)$ being the metadata encrypted on the public key K of some Inspector. The encryption should be verifiable i.e. there should be a proof that Inspector can decrypt on his own secret key. In this setting, `RC` provides an authenticated encryption scheme so that a proof of encryption correctness together with the proof that the AE symmetric key is derived from a key agreement protocol with Inspector's K.

c) Incrementally Verifiable Computation: Here we assume that there is a computational chain of functions $F_1, F_2, \dots, F_k$ applied consecutively or by some ordered graph. Starting with X, for each i Party P_i computes F_i and carries an intermediate result and a proof of correctness to the next P_{i+1} so that the last P_k provides Y and attests $X \xrightarrow{F_k \circ F_{k-1} \circ \cdots \circ F_1} Y$ being actually aware only of their own computation and the proof of correctness π_{k-1} from P_{k-1}. IVC frameworks such as Halo Infinite [20] instruct that the proof π_k asserts the correctness of F_k and that the code C_k that verifies π_{k-1} outputs a success. If the polynomial commitment scheme, used within IVC, is Merkle-tree-based (such as FRI [11]), then π_{k-1} consists of several Merkle tree openings, so that C_k makes a number of calls to the hash function that comprises the tree, for which we suggest `RC`. Therefore, a hash function is used both for commitments and proofs of openings, and both regular performance and the size of hash function circuit determine the total performance.

V. Preferred IOP: Plookup

The proof systems appeared in recent years are numerous and are beyond the scope of this paper. Nevertheless, a common complexity metric of a primitive implemented within is its size as an arithmetic circuit, measured in the number of *gates*, as the prover complexity is usually an (almost) linear function of the latter. Within a single proof statement, circuits of all its components must operate in the same domain, typically prime field $\mathbb{F}_p$.

The proof systems based on polynomial commitments require the circuit wires to be encoded as a few polynomials, so that a circuit is correct if certain algebraic statement holds, with the later being proven using several commitment openings. The Plonk proof system [28] is known for relative simplicity and ability to use arbitrary polynomial commitment scheme. If KZG pairing-based commitments [38] are used then the Plonk proofs have constant size and need a so-called universal trusted setup, where the same reference string can be used for all circuits of bounded size. If FRI Merkle tree-based commitments [11] are used then the trusted setup is not needed but proofs are bigger.

Plookup [27] is an extension to the Plonk proof system [28]. Whereas Plonk uses arithmetic gates only, Plookup supports table check gates of form $(x_1, x_2, \dots, x_w) \in T_i$ where T_i is a table with w columns. The same set of commitment schemes can be

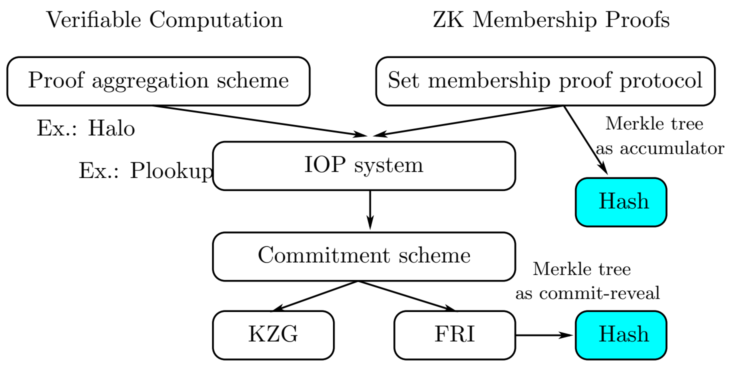


Fig. 3: `Reinforced Concrete` in set membership and verifiable computation protocols.

used, including KZG and FRI schemes. We are aware of several implementations of Plookup, though none has been claimed to be production-ready yet, so the syntax and API vary. Here we list only the main features:

- Circuit is defined over prime field $\mathbb{F}_p$ with sufficiently many (say 2^{32}) roots of unity.
- A single gate is either:
  - table check (lookup) of a w-vector of circuit variables, or
  - degree-2 equation on circuit variables, public inputs and constants with at most 1 multiplication and $w-1$ additions.

Plookup has reduced the circuit count for major hash functions. For example, there were reported SHA-256 circuits for Plookup with about 3K gates (in contrast to 27K gates for R1CS), but this number is still much higher than for recent designs like POSEIDON or *Rescue*. We refer to Table I, where the SHA-256 and Blake2s gate counts are from Hopwood's notes[2], the zkSummit5 presentations by Gabizon[3], and the `Reinforced Concrete` gate counts are estimated in Section VIII-A1.

VI. Specification

The `RC` permutation illustrated in Fig. 2, can be considered as a modified 7-round SP network, where input, output and intermediate state elements are from $\mathbb{F}_p^3$ for a prime number p. More formally,

$$\begin{aligned}&\texttt{RC} := \texttt{Concrete}^{(8)} \circ \texttt{Bricks} \circ \texttt{Concrete}^{(7)}\\ &\circ \texttt{Bricks} \circ \texttt{Concrete}^{(6)} \circ \texttt{Bricks}\\ &\circ \texttt{Concrete}^{(5)} \circ \texttt{Bars} \circ \texttt{Concrete}^{(4)}\\ &\circ \texttt{Bricks} \circ \texttt{Concrete}^{(3)} \circ \texttt{Bricks}\\ &\circ \texttt{Concrete}^{(2)} \circ \texttt{Bricks} \circ \texttt{Concrete}^{(1)}\end{aligned}$$

In the following, we refer to `Concrete` $\circ$ `Bricks` as "round".

A. *The* `Bricks` *function*

The function `Bricks` : $\mathbb{F}_p^3 \to \mathbb{F}_p^3$ is a non-linear permutation of degree $d = 5$ (with the requirement $\gcd(p-1, d) = 1$). We define `Bricks` as

$$\begin{aligned}&\texttt{Bricks}(x_1, x_2, x_3)\\ &= (x_1^d, x_2(x_1^2 + \alpha_1 x_1 + \beta_1), x_3(x_2^2 + \alpha_2 x_2 + \beta_2)),\end{aligned}$$

where $\alpha_1, \alpha_2, \beta_1, \beta_2 \in \mathbb{F}_p$ such that $\alpha_i^2 - 4\beta_i$ is not a quadratic residue modulo p. The values of $\alpha_1, \alpha_2, \beta_1, \beta_2$ are given by

- $p = p_{\text{BLS381}}$: (1,3,2,4).
- $p = p_{\text{BN254}}$: (1,3,2,4)
- $p = p_{ST}$: (1,2,3,4).

B. *The* `Concrete` *function*

The function $\texttt{Concrete}^{(j)} : \mathbb{F}_p^3 \to \mathbb{F}_p^3$ denotes the multiplication of the state by a 3×3 MDS matrix $M = \text{circ}(2, 1, 1)$ with subsequent addition of the j-th round constant vector $c^{(j)} \in \mathbb{F}_p^3$, that is

$$\texttt{Concrete}^{(j)}(x) := \begin{pmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix} \times \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + c^{(j)}.$$

Note that M is invertible and MDS for each $p \geq 3$. The elements $c_1^{(j)}, c_2^{(j)}, c_3^{(j)}$ are certain pseudo-random constants, generated using e.g. SHAKE-128 with rejection sampling.

[2]https://www.zfnd.org/zcon/0/workshop-notes/Zcon0%20Circuit%20Optimisation%20handout.pdf
[3]https://www.crowdcast.io/e/zksummit/8

C. The `Bars` *Function*

The function `Bars` $: \mathbb{F}_p^3 \to \mathbb{F}_p^3$ is defined as

$$\texttt{Bars}(x_1, x_2, x_3) = (\texttt{Bar}(x_1), \texttt{Bar}(x_2), \texttt{Bar}(x_3)).$$

The function `Bar` $: \mathbb{F}_p \to \mathbb{F}_p$ is designed to be a permutation of $\mathbb{F}_p$ coming from n smaller permutations acting *independently* on n smaller *buckets* $\mathbb{Z}_{s_1}, \ldots, \mathbb{Z}_{s_n}$, where $s_1, \ldots, s_n$ are defined for each p separately in the further text. The independence requirement is crucial for the performance of `Bar`. For this we decompose a field element $x \in \mathbb{F}_p$ into n smaller *digits* $x_1, \ldots, x_n$ with $x_i \in \mathbb{Z}_{s_i}$, and then compose it back. Overall, `Bar` $: \mathbb{F}_p \to \mathbb{F}_p$ is defined as

$$\texttt{Bar} = \texttt{Comp} \circ \texttt{SBox} \circ \texttt{Decomp}. \tag{1}$$

In the following, we define all these components. The invertibility of such a function is proved in Appendix B.

1) Decomposition and Composition: We choose the standard representation $\mathbb{F}_p = \{0, 1, \ldots, p-1\}$ for $\mathbb{F}_p$, thus identifying an element $x \in \mathbb{F}_p$ with an integer $0 \leq x \leq p-1$. Our decomposition `Decomp` $: \mathbb{F}_p \to \mathbb{Z}_{s_1} \times \ldots \times \mathbb{Z}_{s_n}$ expands $x \in \mathbb{F}_p$ as

$$x = x_1 \cdot s_2 s_3 \cdots s_n + x_2 \cdot s_3 s_4 \cdots s_n + \cdots + x_{n-1} \cdot s_n + x_n = \sum_{i=1}^{n} x_i \prod_{j>i} s_j.$$

with $0 \leq x_i < s_i$ and where the s_i are chosen such that $\prod_{i=1}^{n} s_i > p$. The digits $x_i \in \mathbb{Z}_{s_i}$ are determined similarly to ordinary base-b expansion:

$$\begin{aligned} x_n &:= x \bmod s_n, \\ x_i &:= \frac{x - \sum_{j>i} x_j \prod_{k>j} s_k}{\prod_{j>i} s_j} \bmod s_i. \end{aligned} \tag{2}$$

It follows directly from the definition in Eq. (2) that the digits x_i are *unique*. Because of the strong analogy with ordinary base-b expansion and for ease of notation in the following part, we define for $1 \leq i \leq n$ the elements

$$b_i := \prod_{j>i} s_j = s_{i+1} s_{i+2} \ldots s_n,$$

where b_n is defined by the empty product and thus $b_n := 1$. The inverse process, the composition `Comp` $: \mathbb{Z}_{s_1} \times \cdots \times \mathbb{Z}_{s_n} \to \mathbb{F}_p$ is computed as

$$\texttt{Comp}(y_1, \ldots, y_n) := \sum_{i=1}^{n} y_i b_i \bmod p. \tag{3}$$

2) `SBox`*:* Let $(v_1, v_2, \ldots, v_n) = \text{Decomp}(p-1)$ and let $p' \leq \min_{1 \leq i \leq n} v_i$. Then x_i is converted as follows:

$$y_i := S(x_i) = \begin{cases} f(x_i) & \text{if } x_i < p', \\ x_i & \text{if } x_i \geq p', \end{cases} \tag{4}$$

where f denotes a permutation of $\mathbb{Z}_{p'}$. In Lemma 3 we prove that `Bar` is indeed a permutation of $\mathbb{F}_p$. The value p' is selected for each p separately.

For the `Bar` function we choose a decomposition into $n = 27$ small S-boxes for p being the order of BLS12-381 or BN254 curves. The f function is derived from the MiMC cipher (which implicitly requires p' being prime) and its table is given in full in the Appendix.

a) BLS12-381: The prime p is given by

$$\begin{aligned} p_{\text{BLS381}} = {} & \texttt{0x73eda753299d7d483339d80809a1d80} \\ & \texttt{553bda402fffe5bfeffffffff00000001}. \end{aligned}$$

The bucket sizes

$$\begin{matrix} s_{27}, & s_{26}, & \ldots, & s_{19}, \\ s_{18}, & s_{17}, & \ldots, & s_{10}, \\ s_9, & s_8, & \ldots, & s_1, \end{matrix}$$

for the `Bar` layer are given by

693,	696,	694,	668,	679,	695,	691,	693,	700,
688,	700,	694,	701,	694,	699,	701,	701,	701,
695,	698,	697,	703,	702,	691,	688,	703,	679.

If $(v_1, \ldots, v_{27})$ denotes the decomposition of $p-1$, the largest prime p' smaller than or equal to $v = \min_{1 \leq i \leq 27} v_i$ is $p' = 659$. The values s_i were found by a variant of branch-and-bound process where we recursively determine from s_{27} to s_1 under the constraint that $s_i - v_i$ is not too large for any i.

b) BN254: The prime p is given by

$$\begin{aligned} p_{\text{BN254}} = {} & \texttt{0x30644e72e131a029b85045b68181585} \\ & \texttt{d2833e84879b9709143e1f593f0000001}. \end{aligned}$$

The bucket sizes for the `Bar` layer are given by

651,	658,	656,	666,	663,	654,	668,	677,	681,
683,	669,	681,	680,	677,	675,	668,	675,	683,
681,	683,	683,	655,	680,	683,	667,	678,	673.

If $(v_1, \ldots, v_{27})$ denotes the decomposition of $p-1$, the largest prime p' smaller than or equal to $v = \min_{1 \leq i \leq 27} v_i$ is $p' = 641$. Decomposition was found in the same way.

c) FRI prime: For FRI-based IVC we have crafted a special prime, so that the decomposition and modular reduction are extremely fast. Concretely, we found out that a 250-bit prime

$$p_{ST} = 0\text{x}3fa000\ldots001$$

admits the following representation:

$$p_{ST} = 2^{250} - 3 \cdot 2^{241} + 1 = \sum_{i=0}^{24} (2^{10} - 6) 2^{10i} + 1,$$

i.e.,

$$s_2 = s_3 = \cdots = s_{24} = 1024, \tag{5}$$

$$s_{25} = 1023, v_1 = v_2 = \cdots = v_{25} = 1018. \tag{6}$$

For this decomposition we first selected s_i to be almost all powers of two, prepared constraints that $(p-1)$ is divisible by 2^{30} for DFT, and then tried a few values for v_i until we find a prime.

D. *Sponge framework parameters*

We suggest the bijective transformation RC being used in the sponge framework [13] similarly to Poseidon [31] and Rescue [7]. The parameters are as follows:

- Rate is two $\mathbb{F}_p$ elements, capacity is one $\mathbb{F}_p$ element.
- Claimed preimage and collision security level of 128 bits.
- The padding rule consists of adding $1 \in \mathbb{F}_p$ to any input, followed by the smallest number < 2 of zeros so that the size is a multiple of 2.

VII. Lookup Tables and System of Constraints for Bar

In this section we create tables and a set of constraints such that for $x, y \in \mathbb{F}_p$ it holds $y = \texttt{Bar}(x)$ if and only if this set of constraints is satisfied. We face two challenges:

1) The S-box S_i acts on a domain of size s_i, which makes each S-box potentially unique. If we specify the behavior of each S-box separately, the table would have $\sum_i s_i$ entries, which renders it inefficient.
2) Since $\prod_i s_i > p$, there exist distinct elements $(x_1, \ldots, x_n) \neq (x'_1, \ldots, x'_n)$ in $\mathbb{Z}_{s_1} \times \ldots \mathbb{Z}_{s_n}$ that produce the same $x \in \mathbb{F}_p$, i.e., for which it holds

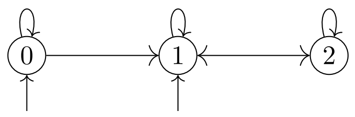
$$x = \texttt{Comp}(x_1, \ldots, x_n) = \sum_{i=1}^{n} x_i b_i \bmod p =$$

$$= \sum_{i=1}^{n} x'_i b_i \bmod p = \texttt{Comp}(x'_1, \ldots, x'_n).$$

We have to ensure that our table and set of constraints prevents this collision from happening.

We address these challenges with two additional sets of variables $(z_1, \ldots, z_n)$ and $(c_1, \ldots, c_n)$, respectively. The variable z_i encodes if $x_i < p'$ (S_i is non-linear function) or $x_i \geq p'$ (S_i is identity function) and is defined as

$$z_i := \begin{cases} 0, \text{ if } x_i < p'; \\ 1, \text{ if } x_i \geq p'. \end{cases} \tag{7}$$



$$\begin{tikzcd} \arrow[d] & & \arrow[d] & & \\ 0 \arrow[loop above] \arrow[rr] & & 1 \arrow[loop above] \arrow[rr, leftrightarrow] & & 2 \arrow[loop above] \end{tikzcd}$$

Fig. 4: Finite-state automaton $\mathcal{A}$ representing all valid sequences $c_1, c_2, \ldots, c_n$.

The purpose of variables $(c_1, \ldots, c_n)$ is to indicate if a tuple $(x_1, \ldots, x_n) \in \mathbb{Z}_{s_1} \times \ldots \times \mathbb{Z}_{s_n}$ has the property $\sum_{i=1}^{n} x_i b_i \geq p$, or not. If $\sum_{i=1}^{n} x_i b_i \geq p$, the tuple $(x_1, \ldots, x_n)$ "overflows" p and thus it is a potential candidate for a collision since by definition composition is unique for all $(x_1, \ldots, x_n)$ with $\sum_{i=1}^{n} x_i b_i < p$. With our set of constraints we need to exclude all those tuples "overflowing" p. For $(v_1, \ldots, v_n) = \texttt{Decomp}(p-1)$, we therefore define

$$c_i := \begin{cases} 0, \text{ if } x_j = v_j \text{ for all } 1 \leq j \leq i; \\ 1, \text{ if } x_i < v_i; \\ 2, \text{ if } x_i \geq v_i \text{ and } x_j \neq v_j \text{ for some } 1 \leq j \leq i; \end{cases} \tag{8}$$

By definition of c_i, only sequences $c_1, c_2, \ldots, c_n$ of length n output by the finite-state automaton $\mathcal{A}$ in Fig. 4 are allowed; they characterize all tuples $(x_1, \ldots, x_n) \in \mathbb{N}^n$ with $\sum_{i=1}^{n} x_i b_i < p$.

We create the following 4-ary tables for our set of constraints:

- Table T_2 contains all binary sequences of length 4 (Fig. 5) thus providing a means to encode all possible sequences $(z_1, \ldots, z_n)$ by concatenating as many 4-ary sequences as needed;
- Table T_3 contains all outputs of length 4 of the finite-state automaton $\mathcal{A}$ in Fig. 4. They are chained together with the last element of one 4-ary sequence matching the first element of the next 4-ary sequence to encode all possible outputs of $\mathcal{A}$ of length n, see constraints (10),(11);
- Table T_1 encodes the output of the S-Boxes $S_1, \ldots, S_n$ and indicates whether for an input to S-Box S_i the non-linear function f or the identity function is applied (Fig. 6).

We claim that $y = \texttt{Bar}(x)$ holds if and only if for $x, y \in \mathbb{F}_p$ and $(x_1, \ldots, x_n), (y_1, \ldots, y_n) \in \mathbb{N}^n$ the

$$T_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ \cdots & & & \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix},$$

Fig. 5: Lookup Table T_2.

$$T_1 = \begin{bmatrix} 0 & 0 & f(0) & 1 \\ 1 & 0 & f(1) & 1 \\ \cdots & & & \\ p'-1 & 0 & f(p'-1) & 1 \\ p' & 1 & p' & 1 \\ p'+1 & 1 & p'+1 & 1 \\ \cdots & & & \\ v_1-1 & 1 & v_1-1 & 1 \\ v_1 & 1 & v_1 & 0 \\ p' & 2 & p' & 1 \\ \cdots & & & \\ v_2-1 & 2 & v_2-1 & 1 \\ v_2 & 2 & v_2 & 0 \\ v_2 & 2 & v_2 & 2 \\ v_2+1 & 2 & v_2+1 & 2 \\ \cdots & & & \\ s_2-1 & 2 & s_2-1 & 2 \\ \cdots & & & \\ p' & n & p' & 1 \\ \cdots & & & \\ v_n-1 & n & v_n-1 & 1 \\ v_n & n & v_n & 0 \\ v_n & n & v_n & 2 \\ v_n+1 & n & v_n+1 & 2 \\ \cdots & & & \\ s_n-1 & n & s_n-1 & 2 \end{bmatrix}, \; T_3 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 2 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 2 \\ 0 & 1 & 2 & 1 \\ 0 & 1 & 2 & 2 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 2 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 2 & 2 \\ 1 & 2 & 1 & 1 \\ 1 & 2 & 1 & 2 \\ 1 & 2 & 2 & 1 \\ 1 & 2 & 2 & 2 \\ 2 & 1 & 1 & 1 \\ 2 & 1 & 1 & 2 \\ 2 & 1 & 2 & 1 \\ 2 & 1 & 2 & 2 \\ 2 & 2 & 1 & 1 \\ 2 & 2 & 1 & 2 \\ 2 & 2 & 2 & 1 \\ 2 & 2 & 2 & 2 \end{bmatrix}$$

Fig. 6: Lookup Tables T_1 and T_3.

following constraints are satisfied:

$$\begin{aligned} &\forall n \geq i \geq 1: \; (x_i, i \cdot z_i, y_i, c_i) \in T_1, && (9) \\ &\forall \lceil (n-1)/3 \rceil - 1 \geq i \geq 1: \\ &\qquad (c_{3i-2}, c_{3i-1}, c_{3i}, c_{3i+1}) \in T_3, && (10) \\ &(c_{n-3}, c_{n-2}, c_{n-1}, c_n) \in T_3, && (11) \\ &\forall \lceil n/4 \rceil - 1 \geq i \geq 1: \\ &\qquad (z_{4i-3}, z_{4i-2}, z_{4i-1}, z_{4i}) \in T_2, && (12) \\ &(z_{n-3}, z_{n-2}, z_{n-1}, z_n) \in T_2, && (13) \\ &x = \sum_{i=1}^{n} x_i b_i \bmod p, && (14) \\ &y = \sum_{i=1}^{n} y_i b_i \bmod p. && (15) \end{aligned}$$

In particular, we claim for $x \in \mathbb{F}_p$ there doesn't exist any collision in $\mathbb{Z}_{s_1} \times \ldots \mathbb{Z}_{s_n}$. I.e., there is exactly one element $(x_1, \ldots, x_n)$ in $\mathbb{Z}_{s_1} \times \ldots \mathbb{Z}_{s_n}$ with $\texttt{Comp}(x_1, \ldots, x_n) = x$. We prove these assertions in Lemma 1 and Lemma 2. As a result, the total number of lookup constraints is

$$n + \lceil (n-1)/3 \rceil + \lceil n/4 \rceil \approx n + n/3 + n/4 \approx 1.59n$$

table lookups with tables of total size $p' + \sum_i (s_i - p' + 1) + 16 + 23$.

A. Soundness and Completeness

Lemma 1. *The set of constraints* (9) – (15) *is complete, i.e., for any $x, y \in \mathbb{F}_p$ with $y = \texttt{Bar}(x)$ it is possible to construct $\{x_i, y_i, c_i, z_i : 1 \leq i \leq n\}$ that satisfy them.*

Proof. We work with the standard representation of $\mathbb{F}_p$, that is, $\mathbb{F}_p = \{0, 1, \ldots, p-1\}$. Suppose for $x, y \in \mathbb{F}_p$ it holds $y = \texttt{Bar}(x)$. Our proof works as follows:

1. We construct x_i, y_i and show that constraints (14) and (15) are satisfied;
2. we define z_i that satisfy constraints (12) and (13) regarding Table T_2;
3. we define c_i that satisfy constraints (10) and (11) regarding Table T_3;
4. we show that $(x_i, i \cdot z_i, y_i, c_i)$ satisfy the constraints (9) regarding Table T_1.

1st Step. We define $(x_1, \ldots, x_n) := \texttt{Decomp}(x)$ and $(y_1, \ldots, y_n) := \texttt{SBox}(x_1, \ldots, x_n) = (\texttt{SBox} \circ \texttt{Decomp})(x)$; then constraint (14) holds by definition of $\texttt{Decomp}$ and constraint (15) by definition of $\texttt{Bar}$, i.e.,

$$\begin{aligned} y &= (\texttt{Comp} \circ \texttt{SBox} \circ \texttt{Decomp})(x) \\ &= \texttt{Comp}\,(\texttt{SBox} \circ \texttt{Decomp}(x)) \\ &= \texttt{Comp}(y_1, \ldots, y_n) = \sum_{i=1}^{n} y_i b_i \bmod p. \end{aligned}$$

2nd Step. Let p' be according to the definition of the $\texttt{Bar}$ function, i.e., p' is the largest prime smaller than or equal to $v = \min_{1 \leq i \leq n} v_i$, where $(v_1, \ldots, v_n) = \texttt{Decomp}(p-1)$. For $1 \leq i \leq n$ we define

$$z_i := \begin{cases} 0, & \text{if } x_i < p'; \\ 1, & \text{if } x_i \geq p'; \end{cases}$$

that indicate if $x_i < p'$ or $x_i \geq p'$. The sequence $(z_1, \ldots, z_n)$ is a binary sequence of length n, where all 2^n combinations are possible: every digit x_i can be strictly smaller or greater than p'. Since T_2 contains all binary sequences of length 4, we have that the constraints (12) and (13) regarding T_2 are satisfied .

3rd Step. If $x = p - 1$, or equivalently, if $x_i = v_i$ for all $1 \leq i \leq n$, we define $c_i := 0$, for all $1 \leq i \leq n$. Thus $(c_1, \ldots, c_n) = (0, \ldots, 0)$ and the corresponding constraints (10) and (11) in Table T_3 are satisfied. If

$x < p-1$, there exists at least one index $1 \le i \le n$ with $x_i < v_i$. Let j be the minimal index with that property. We set

$$c_i := \begin{cases} 0, & \text{if } i < j; \\ 1, & \text{if } i \ge j \text{ and } x_i < v_i; \\ 2, & \text{if } i > j \text{ and } x_i \ge v_i. \end{cases}$$

Note that the case $i = j$ and $x_i \ge v_i$ cannot happen, since this would on the one hand mean $x_j \ge v_j$ and on the other hand $x_j < v_j$ (by definition of j), a contradiction. Thus, the above three cases cover all possible situations regarding i. Next, we list all subsequences of $c_1, \ldots, c_n$ that are *not* possible:

(a) $(2, \ldots)$; since $c_1 = 2$ this would mean $1 \le j < i = 1$, a contradiction.
(b) $(\ldots, 0, 2, \ldots)$; this would imply $i < j$ ($c_i = 0$) and $i + 1 > j$ ($c_{i+1} = 2$), a contradiction.
(c) $(\ldots, 1, 0, \ldots)$; a contradiction, since $i \ge j$ ($c_i = 1$) and $i + 1 < j$ ($c_{i+1} = 0$).
(d) $(\ldots, 2, 0, \ldots)$; a contradiction, since $i > j$ ($c_i = 2$) and $i + 1 < j$ ($c_{i+1} = 0$).

We explicitly note, all other subsequences are valid. In a next step, we model a finite-state automaton $\mathcal{B}$ whose outputs of length n characterize all possible sequences $(c_1, \ldots, c_n)$. Clearly, $\mathcal{B}$ has the states $0, 1, 2$ with only $0, 1$ being accepting states: due to (a) no sequence can start with 2. According to (b), (c) and (d), all possible transitions are given by

$$\{(0,0), (0,1), (1,1), (1,2), (2,1), (2,2)\}.$$

But this means, that automaton $\mathcal{B}$ is identical to automaton $\mathcal{A}$ depicted in Fig. 4. Hence we conclude, all possible sequences $(c_1, \ldots, c_n)$ of elements as defined above are precisely the outputs of length n of the finite-state automaton $\mathcal{A}$. If we divide the sequence $(c_1, \ldots, c_n)$ into chunks of 4 elements such that the last element of one chunk matches the first element of the next chunk, we see that constraints (10) and (11) regarding T_3 are satisfied.

4th Step. Constraints (9) regarding T_1 are satisfied as well: by definition of x_i, z_i, y_i, c_i we have $0 \le x_i \le s_i - 1$, $z_i \in \{0, 1\}$, $y_i = S_i(x_i)$ and $c_i \in \{0, 1, 2\}$, respectively. This means, the domains of $x_i, i \cdot z_i, y_i, c_i$ agree with the general conditions in T_1. Not all combinations are allowed, however. The following arguments show that indeed all possible 4-ary chunks $(x_i, i \cdot z_i, y_i, c_i)$ satisfy the constraints in T_1. As in the *3rd Step*, for $x = p - 1$ we define $c_i := 0$ and thus have $(x_i, i \cdot z_i, y_i, c_i) = (v_i, i, v_i, 0)$ for $1 \le i \le n$. Hence, for $x = p - 1$ the corresponding constraints (9) in Table T_1 are satisfied. Therefore let $x < p - 1$ and let again j be the minimal index with $x_i < v_i$.

- For $0 \le x_i < p'$, we have $z_i = 0$, $i \cdot z_i = 0$, $y_i = S(x_i) = f(x_i)$ and $c_i = 1$ (since $x_i < p' \le v_i$) by construction of x_i, z_i, y_i and c_i, respectively. Thus the first p' constraints in T_1 are satisfied.
- For $p' \le x_i = v_i$ two cases can happen: if $i < j$, then $c_i = 0$; if $i > j$, then $c_i = 2$. In both cases the corresponding 4-ary chunk $x_i, i \cdot z_i = i, y_i = x_i, c_i \in \{0, 2\}$ is contained in T_1. We note, the case $x_i = v_i$ and $i = j$ cannot happen due to the definition of j.
- For $p' \le x_i < v_i$, we have $z_i = 1$, $i \cdot z_i = i$, $y_i = S(x_i) = x_i$ and $c_i = 1$ (since $x_i < v_i$). Thus the corresponding $v_i - p'$ constraints in T_1 are satisfied.
- For $v_i + 1 \le x_i \le s_i - 1$ it holds $z_i = 1$, $i \cdot z_i = i$, $y_i = S(x) = x_i$ and $c_i = 2$, which shows that the corresponding $s_i - v_i - 1$ constraints in T_1 are fulfilled.

Specifically, for $i = 1$ there is no entry $(x_1, i \cdot z_1, y_1, 2)$ in T_1, therefore we have to argue that this case cannot happen; this is clear, however, since we have already shown that automaton $\mathcal{B}$, which represents all valid sequences $(c_1, \ldots, c_n)$, guarantees $c_1 \in \{0, 1\}$. □

Lemma 2. *The set of constraints* (9)–(15) *is sound, i.e., for any $x, y \in \mathbb{F}_p$ and any $\{x_i, y_i, z_i, c_i \in \mathbb{N} : 1 \le i \le n\}$ that satisfy them all it holds $y = \texttt{Bar}(x)$.*

Proof. We work with the standard representation of $\mathbb{F}_p$. For $\mathcal{R} := \mathbb{Z}_{s_1} \times \ldots \times \mathbb{Z}_{s_n}$ let

$$\mathcal{R}_{<p} := \{(z_1, \ldots, z_n) \in \mathcal{R} : \sum_{i=1}^{n} z_i b_i < p\}.$$

Our proof consists of the following parts:

1) Show that $(x_1, \ldots, x_n)$ is a valid decomposition of x, i.e., $(x_1, \ldots, x_n) = \texttt{Decomp}(x)$.
2) Show that for all $1 \le i \le n$ we have $y_i = S_i(x_i)$ according to (4) and deduce $(y_1, \ldots, y_n) = (\texttt{SBox} \circ \texttt{Decomp})(x)$.
3) Use the above two facts and deduce $y = \texttt{Bar}(x)$.

1st Step. Let $(x'_1, \ldots, x'_n) := \texttt{Decomp}(x)$ and $\hat{x} := \sum_{i=1}^{n} x_i b_i$. Suppose $\hat{x} < p$, or in other words $(x_1, \ldots, x_n) \in \mathcal{R}_{<p}$. Then by (14) we have $\hat{x} = \hat{x} \bmod p = \sum_{i=1}^{n} x_i b_i \bmod p = x < p$, and thus

$$\begin{aligned} \texttt{Decomp}(x) &= \texttt{Decomp}\left(\sum_{i=1}^{n} x_i b_i \bmod p\right) \\ &= (\texttt{Decomp} \circ \texttt{Comp})(x_1, \ldots, x_n) = (x_1, \ldots, x_n). \end{aligned}$$

The last equality uses the fact, that Decomp and Comp are inverse to each other on $\mathcal{R}_{<p}$ and $\mathbb{F}_p$; we proved this in more detail in Lemma 3.

We show that the case $\hat{x} \geq p$ leads to a contradiction. For this, suppose $\hat{x} \geq p$. This implies that there exists $1 \leq k \leq n$ with

$$x_i = v_i \text{ for all } 1 \leq i < k \text{ and } x_k > v_k.$$

Note that $k > 1$ as $x_1 \leq v_1$ by Table T_1 (constraint (9)). Also, by constraint (9) it holds $c_i \in \{0, 2\}$ for all $1 \leq i < k$ and in particular $c_1 = 0$. Therefore, constraints (10) and (11) regarding Table T_3 ensure that actually all $c_i = 0$ for $1 \leq i < k$ since there is no sequence with $(\ldots, 0, 2, \ldots)$ in T_3. Therefore, again by constraints (10) and (11), we have that $c_k \in \{0, 1\}$. By constraint (9) this is only possible if $x_k \leq v_k$. A contradiction.

2nd Step. Let $1 \leq i \leq n$. We show $y_i = S(x_i)$. By constraints (12) and (13) it holds $z_i \in \{0, 1\}$. If $z_i = 0$ then $i \cdot z_i = 0$ and by constraint (9) we have $x_i < p'$ and $y_i = f(x_i)$. If $z_i = 1$, we have $i \cdot z_i = i > 1$, and again by constraint (9) it holds $x_i \geq p'$ and $y_i = x_i$. Altogether we get that $y_i = S_i(x_i)$ and thus

$$\begin{aligned}(y_1, \ldots, y_n) &= \texttt{SBox}(x_1, \ldots, x_n) \\ &\overset{\text{Part1}}{=} \texttt{SBox}(\texttt{Decomp}(x)) = (\texttt{SBox} \circ \texttt{Decomp})(x).\end{aligned} \tag{16}$$

3rd Step. For the last part we use the definition of Bar, Part 2, the definition of Comp and constraint (15), which yields

$$\begin{aligned}\texttt{Bar}(x) &\overset{(1)}{=} (\texttt{Comp} \circ \texttt{SBox} \circ \texttt{Decomp})(x) \\ &= \texttt{Comp}(\texttt{SBox} \circ \texttt{Decomp}(x)) \\ &\overset{\text{Part 2}}{=} \texttt{Comp}(y_1, \ldots, y_n) \\ &\overset{(3)}{=} \sum_{i=1}^{n} y_i b_i \bmod p \overset{(15)}{=} y. \qquad \square\end{aligned}$$

VIII. Performance

In this section we consider performance of plain and zero knowledge implementations of RC. As the application, we consider a single call to permutation RC, which corresponds to hashing of two $\mathbb{F}$ elements, or computing one node of a Merkle tree.

A. Proof System Performance

1) Gate count:

a) RC: We can make a gate estimate for the Plookup proof system with 4-ary addition and 4-ary table for the BLS/BN primes.

- Bricks: 7 gates per round;
- Concrete: 1 gate per element, 3 per round.
- Bars: 68 gates per element, 204 per round
  - decomposition: 13 add gates
  - composition: 13 add gates
  - table: 42 gates.

Total: $7 \cdot 6 + 3 \cdot 7 + 204 = 267$ gates to process two $\mathbb{F}_p$ elements of data. The p_{ST} case uses only 25 s_i so the total number of gates is 265.

b) Poseidon: Poseidon-128 [31] with 2 inputs, which needs 438 gates for the same setting: each full round needs 9 quadratic gates and 3 addition gates, whereas each partial round needs 3 quadratic and 3 addition gates. Total count is $12 \cdot 8 + 57 \cdot 6 = 438$.

c) Rescue: Rescue with 2 inputs requires 16 full founds, which together utilize 268 quadratic gates. In addition, each (out of 16) round carries two matrix multiplications, i.e. 6 4-ary additions. The total Plonk/Plookup gate count is then 364. *Rescue*-Prime, a new variant of Rescue, requires only 14 rounds and, thus, is 12% cheaper.

d) Sinsemilla: SINSEMILLA is parameterized by k that determines the lookup table length 2^k and the same number of EC generators $P_0, P_1, \ldots, P_{2^k-1}$. A hash of tk-bit $M = (M_1, M_2, \ldots, M_t), t < 254$ is defined as

$$H(M) = (Q + \sum_{i \leq t} [2^{t-i}] P_{M_i})_x,$$

where Q is some EC point, $+$ is EC addition, $[a]$ is the EC scalar multiplication by a, $()_x$ is the x-coordinate of the curve.

The SINSEMILLA authors provide only a gate count tailored to a concrete IVC framework Halo2 (which is certainly possible for RC as well). In order to compare apples to apples, we take their system[4] of $5t$ quadratic equations and a single t-ary addition of message decomposition. Accounting for 4-ary addition gates, we obtain that SINSEMILLA needs $7t + t/2$ addition gates, $5t$ multiplication gates, and t lookup gates. For $k = 10$ and $t = 51$ we obtain 510-bit message input, for which the total gate count is about $370+250+50 = 670$ regular Plookup gates.

B. Plain Implementation Performance

We implemented RC in pure Rust using the ff_ce library[5] for field operations. Further, we re-implemented POSEIDON and *Rescue* with a statesize of 3 words and Feistel-MiMC using ff_ce to compare them to RC in a fair setting. We further compare RC to pure Rust implementations of traditional hash algorithms[6], and

[4]https://zcash.github.io/orchard/design/circuit/gadgets/sinsemilla.html

[5]https://docs.rs/ff_ce/0.13.1/ff_ce/

[6]https://github.com/RustCrypto/hashes

compare it to SINSEMILLA using an implementation found in the Zcash/Orchard repository on Github, and to a Pedersen Hash implementation from librustzcash[7]. We benchmark input sizes of at least 512-bit (i.e., two field elements). We, thus, benchmark one permutation call for all symmetric hash functions, except for Feistel-MiMC for which we require two. All benchmarks where obtained on a Linux Desktop PC with an Intel i7-4790 CPU (3.9 GHz) and 16 GB RAM using stable Rust version 1.53 and the `target-cpu=native` flag. The resulting benchmarks can be found in Table II, code to reproduce them is publicly available [1].

TABLE II: Plain performance comparison of different hash functions in Rust.

Hashing algorithm		BN	BLS	ST
	ns	ns	ns	ns
`Reinforced Concrete`	-	3 284	3 262	1 032
Concrete Layer	-	44.8	44.4	36.8
Bricks Layer	-	146.0	157.0	93.4
Bars Layer	-	1 982	1 980	214.1
POSEIDON	-	19 464	18 564	17 643
Rescue	-	415 230	446 980	359 510
Rescue-Prime	-	362 870	391 560	294 660
Feistel-MiMC	-	33 800	35 847	28 594
SINSEMILLA	131 460	-	-	-
Pedersen Hash	39 807	-	-	-
SHA-256	366.5	-	-	-
Blake2b	245.1	-	-	-
Blake2s	219.5	-	-	-
SHA3-256	392.3	-	-	-

As Table II shows, the plain performance of RC highly depends on the choice of the prime field, more specifically, how elements can be decomposed. The bars layer for p_{ST} can be evaluated by using only one biginteger division[8], whereas a generic decomposition, i.e., for p_{BN254} and p_{BLS12}, requires significantly more. The result is a runtime difference by a factor of 3 for the total hashing time. Compared to the previous state-of-the-art one can observe that RC is significantly faster. More concretely, RC is faster than the previously fastest hash function over finite fields (i.e., POSEIDON) by a factor of 5 for p_{BN254} and p_{BLS12}, and by a factor 12 for the p_{ST} prime field. The SINSEMILLA hash algorithm, which also leverages lookup tables for a faster plain evaluation, is thereby slower than RC by a factor of up to 125, while the traditional Pedersen Hash is only slower by a factor of 37. Compared to fast binary hash function, RC is only slower by a factor of 5 than Blake2s, the fastest benchmarked hashing algorithm. Blake2s in turn however requires 7 times more Plookup gates than RC.

Acknowledgements. We express our special thanks to Ferdinand Sauer for valuable discussion on Gröbner basis computations and for his assistance in conducting the practical experiments. We thank Alex Vlasov (Matter Labs) for his modular math optimizations and comments on the earlier version of the paper.

Lorenzo Grassi is supported by the European Research Council under the ERC advanced grant agreement under grant ERC-2017-ADG Nr. 788980 ESCADA. Roman Walch is supported by the "DDAI" COMET Module within the COMET – Competence Centers for Excellent Technologies Programme, funded by the Austrian Federal Ministry for Transport, Innovation and Technology (bmvit), the Austrian Federal Ministry for Digital and Economic Affairs (bmdw), the Austrian Research Promotion Agency (FFG), the province of Styria (SFG) and partners from industry and academia. The COMET Programme is managed by FFG.

REFERENCES

[1] Hash functions for zero-knowledge applications zoo. https://extgit.iaik.tugraz.at/krypto/zkfriendlyhashzoo (Aug 2021), IAIK, Graz University of Technology

[2] Tornado cash privacy solution version 1.4 (2021), https://tornado.cash/Tornado.cash_whitepaper_v1.4.pdf

[3] ZCash protocol specification (2021, 19th June), https://github.com/zcash/zips/blob/master/protocol/protocol.pdf

[4] Albrecht, M.R., Cid, C., Grassi, L., Khovratovich, D., Lüftenegger, R., Rechberger, C., Schofnegger, M.: Algebraic Cryptanalysis of STARK-Friendly Designs: Application to MARVELlous and MiMC. In: ASIACRYPT. LNCS, vol. 11923, pp. 371–397 (2019)

[5] Albrecht, M.R., Grassi, L., Perrin, L., Ramacher, S., Rechberger, C., Rotaru, D., Roy, A., Schofnegger, M.: Feistel Structures for MPC, and More. In: ESORICS. LNCS, vol. 11736, pp. 151–171. Springer (2019)

[6] Albrecht, M.R., Grassi, L., Rechberger, C., Roy, A., Tiessen, T.: MiMC: Efficient Encryption and Cryptographic Hashing with Minimal Multiplicative Complexity. In: ASIACRYPT 2016. LNCS, vol. 10031, pp. 191–219 (2016)

[7] Aly, A., Ashur, T., Ben-Sasson, E., Dhooghe, S., Szepieniec, A.: Design of Symmetric-Key Primitives for Advanced Cryptographic Protocols. Cryptology ePrint Archive, Report 2019/426 (2019), https://eprint.iacr.org/2019/426

[8] Ashur, T., Dhooghe, S.: MARVELlous: a STARK-Friendly Family of Cryptographic Primitives. Cryptology ePrint Archive, Report 2018/1098 (2018)

[9] Baignères, T., Stern, J., Vaudenay, S.: Linear Cryptanalysis of Non Binary Ciphers. In: Adams, C.M., Miri, A., Wiener, M.J. (eds.) Selected Areas in Cryptography – SAC 2007. LNCS, vol. 4876, pp. 184–211. Springer (2007)

[10] Bardet, M., Faugere, J., Salvy, B., Yang, B.: Asymptotic behaviour of the index of regularity of quadratic semi-regular polynomial systems. In: The Effective Methods in Algebraic Geometry Conference (MEGA). pp. 1–14 (2005)

[11] Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Fast reed-solomon interactive oracle proofs of proximity. In: ICALP. LIPIcs, vol. 107, pp. 14:1–14:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2018)

[6] https://github.com/zcash/orchard, slightly modified to actually use the lookup tables in plain evaluation.

[7] https://github.com/zcash/librustzcash

[8] We implemented divisions using precomputed reciprocals for all prime fields.

[12] Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., Ward, N.P.: Aurora: Transparent succinct arguments for R1CS. In: EUROCRYPT (1). Lecture Notes in Computer Science, vol. 11476, pp. 103–128. Springer (2019)

[13] Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: On the Indifferentiability of the Sponge Construction. In: EUROCRYPT 2008. LNCS, vol. 4965, pp. 181–197 (2008)

[14] Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: On alignment in Keccak. https://keccak.team/files/KeccakAlignment.pdf

[15] Beyne, T., Canteaut, A., Dinur, I., Eichlseder, M., Leander, G., Leurent, G., Naya-Plasencia, M., Perrin, L., Sasaki, Y., Todo, Y., Wiemer, F.: Out of Oddity - New Cryptanalytic Techniques Against Symmetric Primitives Optimized for Integrity Proof Systems. In: CRYPTO. LNCS, vol. 12172, pp. 299–328 (2020)

[16] Biham, E., Biryukov, A., Shamir, A.: Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials. In: EUROCRYPT. LNCS, vol. 1592, pp. 12–23. Springer (1999)

[17] Biham, E., Shamir, A.: Differential Cryptanalysis of DES-like Cryptosystems. In: Advances in Cryptology - CRYPTO 1990. LNCS, vol. 537, pp. 2–21. Springer (1990)

[18] Biham, E., Shamir, A.: Differential Cryptanalysis of the Data Encryption Standard. Springer (1993)

[19] Bogdanov, A., Wang, M.: Zero Correlation Linear Cryptanalysis with Reduced Data Complexity. In: FSE. LNCS, vol. 7549, pp. 29–48. Springer (2012)

[20] Boneh, D., Drake, J., Fisch, B., Gabizon, A.: Halo infinite: Recursive zk-snarks from any additive polynomial commitment scheme. Tech. rep., Cryptology ePrint Archive, Report 2020/1536 (2020)

[21] Boura, C., Canteaut, A., De Cannière, C.: Higher-Order Differential Properties of Keccak and *Luffa*. In: FSE 2011. LNCS, vol. 6733, pp. 252–269 (2011)

[22] Cid, C., Leurent, G.: An Analysis of the XSL Algorithm. In: Advances in Cryptology - ASIACRYPT 2005. LNCS, vol. 3788, pp. 333–352. Springer (2005)

[23] Courtois, N.T., Pieprzyk, J.: Cryptanalysis of Block Ciphers with Overdefined Systems of Equations. In: ASIACRYPT. LNCS, vol. 2501, pp. 267–287. Springer (2002)

[24] Daemen, J., Knudsen, L.R., Rijmen, V.: The Block Cipher Square. In: FSE. LNCS, vol. 1267, pp. 149–165 (1997)

[25] Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Information Security and Cryptography, Springer (2002)

[26] Eichlseder, M., Grassi, L., Lüftenegger, R., Øygarden, M., Rechberger, C., Schofnegger, M., Wang, Q.: An algebraic attack on ciphers with low-degree round functions: Application to full mimc. In: ASIACRYPT. LNCS, vol. 12491, pp. 477–506. Springer (2020)

[27] Gabizon, A., Williamson, Z.J.: plookup: A simplified polynomial protocol for lookup tables. IACR Cryptol. ePrint Arch. **2020**, 315 (2020)

[28] Gabizon, A., Williamson, Z.J., Ciobotaru, O.: Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. IACR Cryptol. ePrint Arch. **2019**, 953 (2019)

[29] Genovese, G.: Improving the algorithms of berlekamp and niederreiter for factoring polynomials over finite fields. J. Symb. Comput. **42**(1-2), 159–177 (2007)

[30] Grassi, L.: Mixture Differential Cryptanalysis: a New Approach to Distinguishers and Attacks on round-reduced AES. IACR Trans. Symmetric Cryptol. **2018**(2), 133–160 (2018)

[31] Grassi, L., Khovratovich, D., Roy, A., Rechberger, C., Schofnegger, M.: Poseidon: A new hash function for zero-knowledge proof systems. Usenix Security 2021 (2021)

[32] Grassi, L., Lüftenegger, R., Rechberger, C., Rotaru, D., Schofnegger, M.: On a generalization of substitution-permutation networks: The HADES design strategy. In: EUROCRYPT (2). Lecture Notes in Computer Science, vol. 12106, pp. 674–704. Springer (2020)

[33] Grassi, L., Rechberger, C., Rønjom, S.: Subspace Trail Cryptanalysis and its Applications to AES. IACR Trans. Symmetric Cryptol. **2016**(2), 192–225 (2016)

[34] Grassi, L., Rechberger, C., Rønjom, S.: A New Structural-Differential Property of 5-Round AES. In: EUROCRYPT. LNCS, vol. 10211, pp. 289–317 (2017)

[35] Grassi, L., Rechberger, C., Schofnegger, M.: Proving Resistance Against Infinitely Long Subspace Trails: How to Choose the Linear Layer. IACR Trans. Symmetric Cryptol. **2021**(2) (2021)

[36] Groth, J.: On the size of pairing-based non-interactive arguments. In: EUROCRYPT (2). Lecture Notes in Computer Science, vol. 9666, pp. 305–326. Springer (2016)

[37] Jakobsen, T., Knudsen, L.R.: The Interpolation Attack on Block Ciphers. In: FSE 1997. LNCS, vol. 1267, pp. 28–40 (1997)

[38] Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: ASIACRYPT. Lecture Notes in Computer Science, vol. 6477, pp. 177–194. Springer (2010)

[39] Keller, N., Rosemarin, A.: Mind the Middle Layer: The HADES Design Strategy Revisited. In: EUROCRYPT. LNCS, vol. 12697, pp. 35–63. Springer (2021)

[40] Knudsen, L.R.: Truncated and Higher Order Differentials. In: FSE 1994. LNCS, vol. 1008, pp. 196–211 (1994)

[41] Knudsen, L.R.: DEAL – A 128-bit Block Cipher. (1998)

[42] Lai, X.: Higher Order Derivatives and Differential Cryptanalysis. In: Communications and Cryptography: Two Sides of One Tapestry. pp. 227–233. Springer US (1994)

[43] Lamberger, M., Mendel, F., Rechberger, C., Rijmen, V., Schläffer, M.: Rebound Distinguishers: Results on the Full Whirlpool Compression Function. In: ASIACRYPT 2009. LNCS, vol. 5912, pp. 126–143 (2009)

[44] Leander, G., Abdelraheem, M.A., AlKhzaimi, H., Zenner, E.: A Cryptanalysis of PRINTcipher: The Invariant Subspace Attack. In: CRYPTO. LNCS, vol. 6841, pp. 206–221 (2011)

[45] Leander, G., Minaud, B., Rønjom, S.: A Generic Approach to Invariant Subspace Attacks: Cryptanalysis of Robin, iSCREAM and Zorro. In: EUROCRYPT. LNCS, vol. 9056, pp. 254–283 (2015)

[46] Matsui, M.: Linear Cryptanalysis Method for DES Cipher. In: EUROCRYPT 1993. LNCS, vol. 765, pp. 386–397 (1993)

[47] Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: The Rebound Attack: Cryptanalysis of Reduced Whirlpool and Grøstl. In: FSE 2009. LNCS, vol. 5665, pp. 260–276 (2009)

[48] Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: Nearly practical verifiable computation. In: IEEE Symposium on Security and Privacy. pp. 238–252. IEEE Computer Society (2013)

[49] Szepieniec, A., Ashur, T., Dhooghe, S.: Rescue-prime: a standard specification (sok). IACR Cryptol. ePrint Arch. **2020**, 1143 (2020)

[50] Wagner, D.A.: The Boomerang Attack. In: FSE 1999. LNCS, vol. 1636, pp. 156–170 (1999)

[51] Wood, G., et al.: Ethereum: A secure decentralised generalised transaction ledger. ethereum project yellow paper.(2014) (2014)

Appendix

A. *Bijectivity of* `Bricks`

Given $\alpha_1, \alpha_2, \beta_1, \beta_2 \in \mathbb{F}_p$ such that

$$\alpha_i^2 - 4 \cdot \beta_i \text{ is a } \textbf{non}\text{-quadratic residue mod } p,$$

for each $i \in \{1, 2\}$, the generalized `Bricks` function is defined as follows:

$$\begin{aligned}&\texttt{Bricks}(x_1, x_2, x_3)\\&\quad = (x_1^d, x_2(x_1^2 + \alpha_1 x_1 + \beta_1), x_3(x_2^2 + \alpha_2 x_2 + \beta_2)).\end{aligned}$$

This function is invertible. Indeed, given $\texttt{Bricks}(x_1, x_2, x_3) = (y_1, y_2, y_3)$, we have

$$x_1 = y_1^{1/d}, \quad x_2 = \frac{y_2}{(x_1^2 + \alpha_1 \cdot x_1 + \beta_1)}, \\ x_3 = \frac{y_3}{(x_2^2 + \alpha_2 \cdot x_2 + \beta_2)},$$

where

(1) $x \mapsto x^d$ is invertible due to the assumption on d,

(2) $z^2 + \alpha_i \cdot z + \beta_i \neq 0$ for each $z \in \mathbb{F}_p$ due to the definition of α_i, β_i. In particular, the only possible solutions of this equation would be

$$z_\pm = \left(-\alpha_i \pm \sqrt{\alpha_i^2 - 4 \cdot \beta_i}\right)/2,$$

which do not exist due to the fact that $\alpha_i^2 - 4 \cdot \beta_i$ is not a square.

B. Bijectivity of `Bar`

Lemma 3. *The function* `Bar` *permutes* $\mathbb{F}_p$.

Proof. We work with the standard representations of $\mathbb{F}_p$ and $\mathbb{Z}_{s_1}, \ldots, \mathbb{Z}_{s_n}$. For $\mathcal{R} := \mathbb{Z}_{s_1} \times \ldots \times \mathbb{Z}_{s_n}$ let

$$\mathcal{R}_{<p} := \{(z_1, \ldots, z_n) \in \mathcal{R} : \sum_{i=1}^{n} z_i b_i < p\}.$$

The idea of the proof reads as follows: we show that

1) `Decomp` is injective and $\texttt{Decomp}(\mathbb{F}_p) \subseteq \mathcal{R}_{<p}$;
2) $\texttt{SBox}(\mathcal{R}_{<p}) \subseteq \mathcal{R}_{<p}$ and deduce that `SBox` permutes $\mathcal{R}_{<p}$;
3) `Comp` is injective on $\mathcal{R}_{<p}$.

With these statements, it follows at once that the function $\texttt{Bar} : \mathbb{F}_p \to \mathbb{F}_p$ given by

$$\texttt{Bar} = \texttt{Comp} \circ \texttt{SBox} \circ \texttt{Decomp}$$

is injective and hence surjective as well. In particular, we see that `Decomp` and `Comp` are indeed inverse functions over $\mathcal{R}_{<p}$ and $\mathbb{F}_p$.

Ad (1), (3): the statement $\texttt{Decomp}(\mathbb{F}_p) \subseteq \mathcal{R}_{<p}$ is a direct consequence of the definition of `Decomp`. For the injectivity of `Decomp` we show that it has a left inverse function on $\mathcal{R}_{<p}$ which is precisely given by `Comp` restricted to $\mathcal{R}_{<p}$. Indeed, for $x \in \mathbb{F}_p$ it holds

$$(\texttt{Comp} \circ \texttt{Decomp})(x) = \texttt{Comp}(x_1, \ldots, x_n) \\ = \sum_{i=1}^{n} x_i b_i \bmod p = \sum_{i=1}^{n} x_i b_i = x.$$

The second equality is just the definition of `Comp`, the third equality uses the fact that $\texttt{Decomp}(\mathbb{F}_p) \subseteq \mathcal{R}_{<p}$, and the fourth equality is true by definition of `Decomp`. Similarly, we obtain for $(z_1, \ldots, z_n) \in \mathcal{R}_{<p}$

$$(\texttt{Decomp} \circ \texttt{Comp})(z_1, \ldots, z_n) \\ = \texttt{Decomp}(\sum_{i=1}^{n} z_i b_i \bmod p) \\ = \texttt{Decomp}(\sum_{i=1}^{n} z_i b_i) = (z_1, \ldots, z_n)$$

and hence that `Comp` restricted to $\mathcal{R}_{<p}$ has the left inverse `Decomp`.

Ad (2): Since `SBox` is the parallel application of n smaller bijections it is clearly injective. The only assertion to prove is hence the inclusion $\texttt{SBox}(\mathcal{R}_{<p}) \subseteq \mathcal{R}_{<p}$. Let $(x_1, x_2, \ldots, x_n) \in \mathcal{R}_{<p}$ and let $(y_1, \ldots, y_n) = (S(x_1), \ldots, S(x_n))$ denote the image under `SBox`. Now recall that $v = \min_i v_i$ where $(v_1, v_2, \ldots, v_n) = \texttt{Decomp}(p-1)$, and let m be the smallest index such that $x_m < v$. If there is no such m, then all S-boxes S are identity functions and the assertion holds. If such an m exists, then for all $i < m$ we have $y_i = S(x_i) = x_i$ by the definition of the S_i. Moreover, for $i = m$ we have $y_m = S(x_m) < v \leq v_m$. For the remaining part we highlight the following property of our decomposition (which has an analogous counterpart in ordinary base-b expansion): for every $1 \leq k \leq n$ it holds

$$\sum_{i=k+1}^{n} (s_i - 1) b_i = \sum_{i=k+1}^{n} (s_i - 1) \prod_{l>i} s_l \\ = \sum_{i=k+1}^{n} \left(\prod_{l>i-1} s_l - \prod_{l>i} s_l \right) = \prod_{l>k} s_l - 1 = b_k - 1.$$

Informally speaking, this translates to the statement "the sum of the maximal values of the first $l = n - k$ *least* significant positions equals the value of the next greater significant position minus 1". We use this fact and deduce

$$\sum_{i=1}^{n} y_i b_i = \sum_{i=1}^{m-1} y_i b_i + y_m b_m + \underbrace{\sum_{i=m+1}^{n} y_i b_i}_{<b_m} \\ < \sum_{i=1}^{m-1} x_i b_i + (y_m + 1) b_m \leq \sum_{i=1}^{m-1} x_i b_i + v_m b_m \\ \leq \sum_{i=1}^{m-1} v_i b_i + v_m b_m \leq p - 1.$$

Hence, $\texttt{SBox}(x_1, \ldots, x_n) \in \mathcal{R}_{<p}$ which implies that `SBox` permutes $\mathcal{R}_{<p}$. The second last inequality uses the property that for $u \in \mathbb{F}_p$ with $u \leq p-1$, the decomposi-

tions $(u_1, \dots, u_n)$ and $(v_1, \dots, v_n) = \texttt{Decomp}(p-1) \in \mathcal{R}$ satisfy for any $1 \leq k \leq n$ the inequality

$$\sum_{i=1}^{k} u_i b_i \leq \sum_{i=1}^{k} v_i b_i.$$

In other words, "if u is smaller than or equal to v, the sum of the values of any first k *most* significant digits of u is smaller than or equal to the corresponding sum for v." For $u = v$, the statement is obvious. For $u \neq v$, there is at least one index $1 \leq i \leq n$ with $u_i < v_i$; let t denote the minimal index with this property. If $k < t$, then $\sum_{i=1}^{k} u_i b_i = \sum_{i=1}^{k} v_i b_i$ by definition of t. If $k \geq t$ then

$$\begin{aligned}\sum_{i=1}^{k} u_i b_i &= \sum_{i=1}^{t-1} u_i b_i + u_t b_t + \sum_{i=t+1}^{k} u_i b_i \\ &< \sum_{i=1}^{t-1} u_i b_i + (u_t + 1) b_t \leq \sum_{i=1}^{t-1} v_i b_i + v_t b_t \\ &\leq \sum_{i=1}^{k} v_i b_i.\end{aligned}$$

$\square$

C. The $\texttt{SBox}$ function

In Eq. (4), $f : \mathbb{F}_{p'} \to \mathbb{F}_{p'}$ denotes the non-identity part of each S-box S_i. Since S_i shall be a permutation of $\mathbb{Z}_{s_i}$, we also need f to be a permutation of $\mathbb{F}_{p'}$. In particular, when f is represented as a univariate polynomial over $\mathbb{F}_{p'}$ it needs to have a high degree and a dense polynomial description (i.e., many non-zero coefficients). Other properties (e.g., high nonlinearity) are not needed in this context, because security against the corresponding attacks is already achieved using the Bricks layer (through large-word operations). We apply the following technique to choose the function f.

1) We choose the smallest $d \in \mathbb{N}$ such that d is prime, $d = 2^n - 1$ for some $n \in \mathbb{N}$, and $\gcd(d, p' - 1) = 1$. The last requirement ensures that the resulting polynomial is a permutation polynomial over $\mathbb{F}_{p'}$.
2) we compute the r-fold composition

$$f(X) := (f_r \circ f_{r-1} \circ \cdots \circ f_1)(X) \in \mathbb{F}_{p'}[X],$$

where $f_i(X) := (X + c_i)^d$ for random $c_i \in \mathbb{F}_{p'}$.

In the second step, we set $r = 2 \lceil \log_d(p') \rceil$, and we want to reach a degree of $p'-2$ and $p'-1$ non-zero coefficients. If either of these conditions is not fulfilled, we sample a new set of r constants $c_1, c_2, \dots, c_r$ and apply the above function f again until the resulting polynomial is dense and of maximum degree. In our experiments, both conditions are fulfilled after only a small number of trials.

We note that the final representation of f is similar to the polynomial representation of the MiMC permutation [6], where the key is set to a known constant.

We practically evaluated the algebraic properties of the resulting S-box S_i when embedded in $\mathbb{F}_2^{n'}$, where $n' := \lceil \log_2(p') \rceil$. As expected, in our experiments we observed that the algebraic degree of S_i is n' (note that S_i embedded in $\mathbb{F}_2^{n'}$ is not a permutation).[9]

For the sake of completeness, the full S-box definition is given in auxiliary files.

In this section, we analyze the security of our design against known attacks on bijective transformations relevant in the hash function and encryption settings.

D. Statistical Attacks

Firstly, we show that our design is secure against statistical attacks, including the differential one and its variants, the linear attack and the rebound attack. In order to achieve this goal, we make use of the same strategy originally proposed for HADESMiMC and POSEIDON. That is, we make use only of the Bricks and of the Concrete components in order to guarantee security against statistical attack. In particular, here we consider a variant of the RC permutation denoted by RC′ in which the middle component Bricks-Concrete-Bars-Concrete-Bricks is replaced with a single Bricks, i.e.,

$$\begin{aligned}\texttt{RC}' := \texttt{Concrete}^{(8)} \circ \texttt{Bricks} \circ \texttt{Concrete}^{(7)} \\ \circ \texttt{Bricks} \circ \texttt{Concrete}^{(6)} \circ \texttt{Bricks} \circ \texttt{Concrete}^{(3)} \\ \circ \texttt{Bricks} \circ \texttt{Concrete}^{(2)} \circ \texttt{Bricks} \circ \texttt{Concrete}^{(1)}.\end{aligned}$$

We claim that if a sponge hash function instantiated with RC′ is secure against the statistical attacks proposed in this section, then it is also secure if it is instantiated with the full RC permutation RC instead. This is a reasonable assumption, since RC exhibits the same structure, but increases the number of nonlinear components.

1) Differential Cryptanalysis: Differential cryptanalysis [17], [18] and its variations are the most widely used techniques to analyze symmetric-key primitives. Given pairs of inputs with some fixed input differences, differential cryptanalysis considers the probability distribution of the corresponding output differences produced by the cryptographic primitive. Let $\delta_I, \delta_O \in \mathbb{F}_p^t$ be respectively the input and the output differences through a function F over $\mathbb{F}_p^t$. The differential probability (DP) of having

[9]The algebraic degree refers to the maximum degree of all component functions.

a certain output difference δ_O given a particular input difference δ_I is equal to

$$\text{Prob}(\delta_I \to \delta_O) = \frac{|\{x \in \mathbb{F}_p^t \mid F(x+\delta_I) - F(x) = \delta_O\}|}{p^t}.$$

As our design is an iterated scheme, a cryptanalyst searches for ordered sequences of differences over any number of rounds that are called differential characteristics/trails. Assuming the independence of the rounds, the DP of a differential trail is the product of the DPs of its one-round differences. We claim that the security against differential attacks is achieved if every differential characteristic has a probability smaller than p^{-2}. This is due to the fact that many characteristics can be used together in order to set up the attack, which means that a probability of p^{-1} may not be sufficient to provide security.

To show that our scheme is secure against this attack, we start by considering the maximum differential probability (DP$_{\text{max}}$) of each component of the `Bar`. As it is well known,

$$\text{DP}_{\text{max}}(x \mapsto x^d) = (d-1)/p.$$

Instead, let us consider the other component.

Lemma 4. *Let $\alpha, \beta \in \mathbb{F}_p \setminus \{0\}$ such that $\alpha^2 - 4\beta$ is not a square modulo p. Let $F : \mathbb{F}_p^2 \to \mathbb{F}_p$ be defined as*

$$F(x,y) = x(y^2 + \alpha y + \beta).$$

For each input difference $\delta_I = (\delta_{I,x}, \delta_{I,y}) \in \mathbb{F}_p^2 \setminus \{(0,0)\}$ and output difference $\delta_O \in \mathbb{F}_p$, we have that

$$\textit{Prob}(\delta_I \to \delta_O) \leq \begin{cases} \frac{2}{p} & \textit{if } \delta_{I,y} = 0 \\ & \textit{or } \delta_{I,x} = \delta_O = 0, \\ \frac{p-1}{p^2} < \frac{1}{p} & \textit{otherwise.} \end{cases}$$

In particular, if $\delta_{I,y} = 0$, then δ_O cannot be equal to zero.

Proof. Given $\delta_I = (\delta_{I,x}, \delta_{I,y})$ and δ_O, we look for the number of solutions (x,y) for

$$\begin{aligned}&(\delta_{I,x} + x)\left((\delta_{I,y} + y)^2 + \alpha(\delta_{I,y} + y) + \beta\right) - x(y^2 + \alpha y + \beta) \\ &= \delta_{I,x}\left(\delta_{I,y}^2 + \delta_{I,y}(2y + \alpha) + (y^2 + \alpha y + \beta)\right) \\ &\quad + x\delta_{I,y}\big(\delta_{I,y} + (2y + \alpha)\big) = \Delta_O.\end{aligned}$$

We analyze separately the following cases:

1) If $\delta_{I,y} = 0$ and $\delta_{I,x} \neq 0$, we have $\delta_{I,x}\left(y^2 + \alpha y + \beta\right) = \delta_O$. Since $y^2 + \alpha y + \beta \neq 0$,

$$y^2 + \alpha \cdot y + \beta - (\delta_O/\delta_{I,x}) = 0$$

can have at most 2 solutions y (for each x). If $\delta_O = 0$, no solution exists.

2) If $\delta_{I,x} = 0$ and $\delta_{I,y} \neq 0$, we have $x\delta_{I,y}\left(\delta_{I,y} + (2y + \alpha)\right) = \delta_O$. If $\delta_O = 0$, then this equality is satisfied by $x = 0$ or $\delta_{I,y} + (2y+\alpha) = 0$, for a total of $2p - 1 \leq 2p$ solutions. If $\delta_O \neq 0$ and $y \neq -(\alpha + \delta_{I,y})/2$, the solutions are given by

$$x = \frac{\delta_O}{\delta_{I,y} \cdot (\delta_{I,y} + (2y + \alpha))}.$$

As a result, $(\delta_{I,x}, \delta_{I,y}) = (0, \delta) \mapsto \delta_O$ with probability at most $(p-1)/p^2 \leq 1/p$. This result holds also for $\delta_O = 0$.

3) If $\delta_{I,x} \neq 0$ and $\delta_{I,y} \neq 0$, the solutions are given by

$$x = \frac{\delta_O - \delta_{I,x} \cdot \left(\delta_{I,y}^2 + \delta_{I,y} \cdot (2y + \alpha) + (y^2 + \alpha \cdot y + \beta)\right)}{\delta_{I,y} \cdot (\delta_{I,y} + (2y + \alpha))}$$

if $y \neq -(\alpha + \delta_{I,y})/2$. As before, $(\delta_{I,x}, \delta_{I,y}) \mapsto \delta_O$ holds with probability at most $(p-1)/p^2 \leq 1/p$.

□

Here we show that the best differential characteristic over two rounds has probability at most

$$\frac{4(d-1)^2}{p^4} \ll p^{-3}.$$

Roughly speaking, this is due to the facts that

- at least four words are active (due to the branch number of the matrix that defines the linear layer),
- each active word affects the overall probability by a factor proportional to p^{-1}.

Examples of differential characteristics that achieve a probability of $\approx p^{-4}$ are the following.

1) The third word at the input of the first round is active, while all words at the input of the second round are active, i.e., $\begin{pmatrix}0\\0\\\delta_1\end{pmatrix} \xrightarrow{\texttt{Br.}(\cdot)} \begin{pmatrix}0\\0\\\delta_2\end{pmatrix} \xrightarrow{\texttt{Conc.}(\cdot)} \begin{pmatrix}\delta_2\\\delta_2\\2\delta_2\end{pmatrix} \xrightarrow{\texttt{Br.}(\cdot)} \begin{pmatrix}\delta_3\\\delta_4\\\delta_5\end{pmatrix}$ for fixed differences $\delta_1, \ldots, \delta_5 \in \mathbb{F}_p$. Note that if δ_2 is not fixed, then the probability increases by a factor p (but it is still much smaller than p^{-2});
2) At the input of both rounds, the second and the third words are active, i.e., $\begin{pmatrix}0\\\delta_1\\\delta_2\end{pmatrix} \xrightarrow{\texttt{Br.}(\cdot)} \begin{pmatrix}0\\\delta_3\\\delta_4\end{pmatrix} \xrightarrow{\texttt{Conc.}(\cdot)} \begin{pmatrix}\delta_3+\delta_4\\2\delta_3+\delta_4\\\delta_3+2\delta_4\end{pmatrix} \xrightarrow{\texttt{Br.}(\cdot)} \begin{pmatrix}\delta_5\\\delta_6\\\delta_7\end{pmatrix}$ for fixed differences $\delta_1, \ldots, \delta_7 \in \mathbb{F}_p$ such that $\delta_3 + \delta_4 = \delta_5 = 0$. Note that if δ_3 is not fixed, then the probability increases by a factor p (but it is still much smaller than p^{-2});
3) The first word at the input of the first round is active, while the second and the third words at the input of the second round are active, i.e., $\begin{pmatrix}\delta_1\\0\\0\end{pmatrix} \xrightarrow{\texttt{Br.}(\cdot)} \begin{pmatrix}\delta_2\\\delta_3\\0\end{pmatrix} \xrightarrow{\texttt{Conc.}(\cdot)} \begin{pmatrix}2\delta_2+\delta_3\\\delta_2+2\delta_3\\\delta_2+\delta_3\end{pmatrix} \xrightarrow{\texttt{Br.}(\cdot)} \begin{pmatrix}\delta_4\\\delta_5\\\delta_6\end{pmatrix}$ for

fixed differences $\delta_1, \ldots, \delta_6 \in \mathbb{F}_p$ such that $2 \cdot \delta_2 + \delta_3 = \delta_4 = 0$. Note that if δ_2 is not fixed, then the probability increases by a factor p (but it is still much smaller than p^{-2}).

Note that this last case is consistent with the branch number of the matrix. Indeed, note that if the first word is active at the input of Bricks, then the two first words in output are active. This means that the number of active input and output words of the matrix is four.

If the difference in the first words is non-zero in both rounds, then the probability of the differential characteristic is much smaller than p^{-4}, since at least other three words (for a total of five active words) are active at the input of the Bricks layer (in order to satisfy the branch number of the matrix, and due to the definition of the Bricks layer).

As a result, two (consecutive) rounds are sufficient to provide security against differential attacks.

2) Truncated and Impossible Differential Attacks: Truncated differential cryptanalysis [40] is a variant of classical differential cryptanalysis, in which the attacker can specify only part of the difference between pairs of texts. Impossible differential cryptanalysis was introduced by Biham et al. [16] and Knudsen [41]. It exploits differentials that occur with probability zero.

Working over a single round, we have that

$$\begin{pmatrix} 0 \\ 0 \\ \Delta_1 \end{pmatrix} \xrightarrow{\texttt{Bricks}(\cdot)} \begin{pmatrix} 0 \\ 0 \\ \Delta_2 \end{pmatrix} \xrightarrow{\texttt{Concrete}(\cdot)} \begin{pmatrix} \Delta_2 \\ \Delta_2 \\ 2 \cdot \Delta_2 \end{pmatrix}$$

and

$$\begin{pmatrix} \Delta_1 \\ 0 \\ 0 \end{pmatrix} \xrightarrow{\texttt{Bricks}(\cdot)} \begin{pmatrix} \Delta_2 \\ \Delta_3 \\ 0 \end{pmatrix} \xrightarrow{\texttt{Concrete}(\cdot)} \begin{pmatrix} 2 \cdot \Delta_2 + \Delta_3 \\ \Delta_2 + 2 \cdot \Delta_3 \\ \Delta_2 + \Delta_3 \end{pmatrix}$$

with probability 1 for (unknown) differences $\Delta_1, \Delta_2, \Delta_3 \in \mathbb{F}_p$ (the case in which the middle word is active is analogous). In a similar way, if we activate the second and the third words in input, we have

$$\begin{pmatrix} 0 \\ \Delta_1 \\ \Delta_2 \end{pmatrix} \xrightarrow{\texttt{Bricks}(\cdot)} \begin{pmatrix} 0 \\ \Delta_3 \\ \Delta_4 \end{pmatrix} \xrightarrow{\texttt{Concrete}(\cdot)} \begin{pmatrix} \Delta_3 + \Delta_4 \\ 2 \cdot \Delta_3 + \Delta_4 \\ \Delta_3 + 2 \cdot \Delta_4 \end{pmatrix}$$

with probability 1 for (unknown) differences $\Delta_1, \ldots, \Delta_4 \in \mathbb{F}_p$. If the two active words are in a different position in the input, then no truncated differential with probability 1 exists.

Note that in both these cases, we we have a linear relation among the output differences. Such linear relation is then broken/lost after the next Bricks layer. The only way to extend them is that one output word is equal to zero. However, this happens with with probability $1/p$, exactly as in the case in which the outputs are generated by a pseudo-random permutation (besides the fact that p is our security level). Due to this fact and since the Concrete layer is defined via the multiplication with a MDS matrix, it is not possible to extend the truncated differentials just given over more rounds (even when working with a nonzero probability $\in (1/p, 1)$). See also the analysis given in the previous section for the case of differential cryptanalysis in which the middle differences are not fixed.

At the same time, it is possible to set up an impossible differential over two rounds, since

$$\begin{pmatrix} 0 \\ 0 \\ \Delta_1 \end{pmatrix} \xrightarrow{\texttt{Conc.oBr.}(\cdot)} \begin{pmatrix} \Delta_2 \\ \Delta_2 \\ 2\Delta_2 \end{pmatrix} \neq \begin{pmatrix} 0 \\ 0 \\ \Delta_3 \end{pmatrix} \xrightarrow{\texttt{Conc.oBr.}(\cdot)} \begin{pmatrix} \Delta_4 \\ \Delta_4 \\ 2\Delta_4 \end{pmatrix}$$

holds with probability 0 for (unknown) differences $\Delta_1, \ldots, \Delta_4 \in \mathbb{F}_p$. It follows that three rounds are sufficient to provide security against truncated and impossibledifferential attacks.

3) Meet-in-the-Middle and Boomerang Attacks: Meet-in-the-Middle and boomerang [50] distinguishers (and their variants) rely on chaining two good differential/linear trails. Due to the differential/linear analysis just proposed, we claim that our analyzed scheme RC′ with six rounds (composed of Bricks layers) is secure against these attacks.

4) Rebound Attacks: Rebound attacks were first presented in [43], [47]. The goal of this attack is to find two (input, output) pairs such that the two inputs satisfy a certain (truncated) input difference and the corresponding outputs satisfy a certain (truncated) output difference. The rebound attack consists of two phases, called *inbound* and *outbound* phase. According to these phases, the internal permutation of the hash function is split into three subparts. Let $P : \mathbb{F}_p^t \to \mathbb{F}_p^t$ be the target permutation, then $P = P_{fw} \circ P_{in} \circ P_{bw}$. The inbound phase is in the middle of the permutation and the two outbound phases are next to the inbound part. In this inbound part, the attacker tries to cover a middle part in the construction separately, which would otherwise be expensive in a classical differential attack. Having found input and output differences such that this part is covered in the inbound phase, the attacker now extends the trail in both directions in the outbound phase.

Here we show that RC′ (namely, the 6-round variant of the RC permutation in which the middle component Middle is replaced with a single Bricks layer) is secure against the rebound attack.

a) Inbound Phase: From Appendix D2 we know that there exist truncated differentials with a probability of 1 over a single round. However, these cannot be extended over more rounds, not even when considering probabilities between $1/p$ and 1. Hence, using an insideout approach, the attacker can cover two rounds in the inbound phase.

In order to apply the outbound phase and due to the truncated differential trails that we found, it is desirable that the difference in at least one word of the trail found by the inbound phase is equal to zero. Again, this cannot be achieved with a probability larger than $1/p$. Hence, we claim that the attacker cannot cover three (or more) rounds in the inbound phase.

b) Outbound Phase: In order to extend the trails found in the inbound phase, we make use of the results regarding the truncated differentials presented before. Since one round can always be covered with a truncated differential characteristic of probability 1, the attacker can skip two rounds (one in each direction).

c) Conclusion: Due to the analysis just proposed, we claim that $\texttt{RC}'$ instantiated with six rounds is secure against the rebound attack. Since the hash sponge function instantiated with this weaker permutation is secure with respect to this attack, the same result holds when considering the original permutation RC.

5) Linear and Zero-Correlation Cryptanalysis: In the case of Boolean functions, linear cryptanalysis [46] searches for a linear combination of input, output and (if present) key bits that is unbalanced, i.e., biased towards 0 or towards 1. In the $\mathbb{F}_p$ case, linear cryptanalysis [9] consists in the search of a linear combination of input, output, and (if present) key words that is unbalanced, i.e., biased towards an element of $\mathbb{F}_p$ with probability higher than $1/|\mathbb{F}_p| = 1/p$. Linear attacks pose no threat to our design instantiated with the same number of rounds previously defined for classical differential cryptanalysis.

Similar to impossible differential attack, zero-correlation attacks are a variant of linear attacks that exploit linear hulls with a zero correlation [19]. In general, those linear hulls are found by a miss-in-the-middle approach. E.g., the approach is to combine two trails that propagate some deterministic properties in order to ensure that the property cannot be fulfilled. Due to our security analysis against linear and differential cryptanalysis and since our analyzed scheme $\texttt{RC}'$ has four Bricks layers, we claim that finding impossible differentials or zero-correlation linear hulls is infeasible.

6) Square/Integral & Mixture Differential Attacks: Integral cryptanalysis is an attack first applied on SQUARE [24] and is particularly efficient against block ciphers based on strong-aligned SPN schemes [14], as AES and AES-like schemes. It is based on the analysis the propagation of sums of values. In the case of our scheme, only one round can be covered with such an attack, e.g.[10] $\begin{pmatrix} C \\ C \\ A \end{pmatrix} \xrightarrow{\texttt{Bricks}(\cdot)} \begin{pmatrix} C \\ C \\ A \end{pmatrix} \xrightarrow{\texttt{Concrete}(\cdot)} \begin{pmatrix} A \\ A \\ A \end{pmatrix} \xrightarrow{\texttt{Bricks}(\cdot)} \begin{pmatrix} ? \\ ? \\ ? \end{pmatrix}$, since both Bricks and Concrete mix the components of the state.

Other distinguishers that are particular efficient against strong-aligned schemes are the "multiple-of-n" one [34] and the mixture differential cryptanalysis [30]. By appropriate choices of a number of input pairs (related by particular linear/differential relations), it is possible to make sure that the number of times that the difference of the resulting output pairs lie in a particular subspace is always a multiple of n. Since both Bricks and Concrete mix the components of the state, we claim that these attacks pose no threat to our design.

E. Invariant Subspace Attack and Fixed Points

1) Invariant Subspaces: Following [33], we say that a subspace $\mathcal{S} \subseteq \mathbb{F}_p^t$ is invariant for a function F over $\mathbb{F}_p^t$ if and only if for each $a \in \mathbb{F}_p^t$ there exists $b \in \mathbb{F}_p^t$ such that

$$F(\mathcal{S} + a) = \mathcal{S} + b.$$

For completeness, we mention that this definition is a slightly different from the one proposed in [44], [45], which is based on the existence of weak keys.

Here we analyze the security of our scheme against this attack, since recent proposals have shown vulnerabilities [15], [35]. We start with the Bars layer. Since Bars operates independently on each input word, we have the following:

- All subspaces in which only a single word is active (e.g., $\mathcal{S} = \langle(0, 1, 0)\rangle$ are invariant through it. In other words, if the difference in one word is equal to zero, it remains equal to zero after Bars.
- The subspaces of the form $\langle(1, 1, 0)\rangle$, or $\langle(1, 0, 1)\rangle$, or $\langle(0, 1, 1)\rangle$, or $\langle(1, 1, 1)\rangle$ are invariant since the same function f defined in Eq. (4) is applied on each word.

At the same time, note that the subspaces of the form e.g. $\langle(1, a, 0)\rangle$ for fixed $a \in \mathbb{F}_p \setminus \{0, 1\}$ cannot be invariant due to the fact that the initial linear relation is destroyed by the function SBox.

We point out that there are invariant *affine* subspaces even if no word is fully active. In particular, remember that $\texttt{Bars} = \texttt{Comp} \circ \texttt{SBox} \circ \texttt{Decomp}$, where

- both $\texttt{Comp} : \mathbb{Z}_{s_1} \times \ldots \times \mathbb{Z}_{s_n} \rightarrow \mathbb{F}_p$ and $\texttt{Decomp} : \mathbb{F}_p \rightarrow \mathbb{Z}_{s_1} \times \ldots \times \mathbb{Z}_{s_n}$ are linear operations that works at word level, where

[10]We use the standard notation $A, C, B, ?$ to denote respectively an active word, a constant one, a balanced one, and an unknown one. We recall that an active word is also balanced.

`Comp`$(x) = (x_1, \ldots, x_n) \in \mathbb{Z}_{s_1} \times \ldots \times \mathbb{Z}_{s_n}$ and where $\sum_{i=1}^{n} x_i \cdot b_i = x$ for given b_i, and

- `SBox` operates independently on each x_i.

Hence, the affine subspace $\mathcal{I}$ defined as

$$\mathcal{I} := \left\{ \sum_i x_i \cdot b_i \in \mathbb{F}_p \;\middle|\; \forall x_1 \in \mathbb{Z}_{s_1} \text{ and } x_2, \ldots, x_n \text{ fixed} \right\}$$

is an invariant affine subspace through `Bars` (note that the values of x_i for $i \geq 2$ change, but this would only change the coset and not the subspace itself). Other invariant affine subspaces can be defined similarly.

Due to the analysis just proposed, there is no invariant subspace for `Bricks`. This means that our scheme is secure against the invariant subspace attack.

2) Fixed Points: For completeness, we also discuss the case of fixed points. We say that a function F over $\mathbb{F}_p^t$ has a fixed point $x \in \mathbb{F}_p^t$ if $F(x) = x$.

The only fixed points for `Bricks` are $(0,0,0)$, $(\pm 1, 0, 0)$ and $(\pm\sqrt{-1}, 0, 0)$. Indeed:

- the only fixed points for $x \mapsto x^5$ are the ones that satisfy $x \cdot (x^4 - 1) = 0$, that is $\{0, \pm 1, \pm\sqrt{-1}\}$. Note that -1 is a quadratic residue modulo p if and only if $p = 1 \mod 4$, which is exactly the case of p_{BLS381} and p_{BN254}.
- the only fixed points for $x \mapsto x \cdot (y^2 + \alpha \cdot y + \beta)$ for a given fixed $y \in \mathbb{F}_p^3$ are *(1)* $\{(0, y) \in \mathbb{F}_p^2 \mid \forall y \in \mathbb{F}_p\}$ and *(2)* $\{(x, y) \in \mathbb{F}_p^2 \mid \forall x \in \mathbb{F}_p$ and $y \in \mathbb{F}_p$ s.t. $(y^2 + \alpha \cdot y + \beta) = 1\}$. Since this second condition is never satisfied for $y \in \{0, \pm 1, \pm\sqrt{-1}\}$ (that is, the fixed points of the first component), it follows that the only fixed points are the ones given before.

In the case of `Bar`, there are several fixed points for each S-box S_i as defined in Eq. (4). In particular, the input x of S_i remains unchanged if $x \geq p'$. Since there are n independent S-boxes S_i for each one of the three words, it follows that the number of fixed points for `Bar` are

$$\left(\prod_{i=1}^{n} (s_i - p') \right)^3,$$

over p^3. As a concrete example, when using $p_{\text{BLS381}} \approx 2^{256}$, the probability for a random point to be a fixed point is

$$\left(\frac{2^{134.54}}{2^{256}} \right)^3 \approx 2^{-364.4}.$$

Recall that the `Bar` layer plays no role in our security arguments for `RC` regarding statistical attacks. When considering algebraic attacks on the middle layer, we have not found a way to exploit these fixed points in attacks on the middle part of `RC`. Since the fixed point property is not described by a low-degree equation, we expect that, for instance, finding a solution to the CICO problem with `Bar` inputs being fixed points is much higher than without these restrictions.

F. Gröbner Basis Cryptanalysis

Gröbner Basis Cryptanalysis usually proceeds in two stages: first, one models the (cryptographic) permutation as a system of equations with unknown parameters as variables. Subsequently, a Gröbner basis for the (zero-dimensional) ideal defined by the polynomials describing the equation system is computed. In practice, the second step is divided into a triad of computations, namely

(1) Compute a Gröbner basis for the (zero-dimensional) ideal with respect to a fast term ordering, usually *degrevlex*;
(2) convert the *degrevlex*-Gröbner basis into a *lex*-Gröbner basis using the FGLM algorithm;
(3) factor the univariate polynomial in the *lex*-Gröbner basis and determine the solutions for the corresponding variable. Back-substitute those solutions, if needed, to determine solutions for other variables.

Each of the above three steps comes with its own complexity estimate. Under the assumption of a semi-regular input system $f_1, \ldots, f_k$ in l variables with degrees $d_1, \ldots, d_m$, it is well-known that the Hilbert series of the ideal corresponding to the input system is related to its Gröbner basis, see [10]. The first index with non-positive coefficient of the expression

$$S_{k,l}(z) = \frac{\prod_{i=1}^{k} (1 - z)^{d_i}}{(1 - z)^l}$$

is the degree of regularity d_{reg} and it is an upper bound for the highest degree element in a Gröbner basis with respect to a graded ordering. Thus, d_{reg} helps to establish the following upper bound for the complexity C (counting finite field operations) of computing a Gröbner basis of a semi-regular input system:

$$C_{GB}(l, d_{reg}) \in \mathcal{O}\left(\binom{l + d_{\text{reg}}}{l}^{\omega} \right), \tag{17}$$

where ω denotes the linear algebra constant. The terms hidden by $\mathcal{O}(\cdot)$ are relatively small, that's why for our analysis we drop the $\mathcal{O}(\cdot)$ and use the expression directly. Our security analysis consists of three steps:

1) We present a system of algebraic equations for the `Bar` layer and count its contribution to the degree of regularity d_{reg} of a primitive that contains `Bar`.
2) We run a series of attacks on the three-layer version `Concrete` $\circ$ `Bar` $\circ$ `Concrete` with a much smaller prime p and argue that already for this

weakened version the attack complexity is as high as $(C_{GB}(l, d_{reg}/3))^{1/2}$.

3) The `Middle` part based on the same small prime can not be attacked (the GB computation does not finish within reasonable time) so we expect that attacking this part is significantly more expensive.

a) Algebraic Representation of `Bar`*:* For an algebraic model of `Bar`, we "embed" $\mathbb{Z}_{s_i}$ in $\mathbb{F}_p$ for all $1 \leq i \leq n$. This embedding is not an embedding in the strict mathematical sense of a structure preserving injective map. Instead, given the respective standard representations of $\mathbb{Z}_{s_i}$ and $\mathbb{F}_p$, we treat the elements $0, 1, \ldots, s_i - 1 \in \mathbb{Z}_{s_i}$ as elements in $\mathbb{F}_p$. As a result, we suggest the following system of $2n+2$ equations over $\mathbb{F}_p$ in the $2n+2$ variables $x, y, x_1, \ldots, x_n, y_1, \ldots, y_n \in \mathbb{F}_p$ to model the `Bar` function:

$$\begin{cases} x = x_1 b_1 + x_2 b_2 + \cdots + x_n b_n \\ 0 = p_{s_i}(x_i), \quad 1 \leq i \leq n \\ y_i = L_i(x_i), \quad 1 \leq i \leq n \\ y = y_1 b_1 + y_2 b_2 + \cdots + y_n b_n \end{cases},$$

where

- $p_{s_i}(x_i) := \prod_{k=0}^{s_i-1}(x_i - k)$ is a polynomial of degree s_i that vanishes at $\{0, 1, \ldots, s_i - 1\}$; p_{s_i} ensures that $x_i \in \mathbb{Z}_{s_i}$;
- $L_i(x_i)$ is the interpolation polynomial of degree $s_i - 1$ for S-box S_i ("embedded" in $\mathbb{F}_p$), i.e.

$$L_i(x_i) := \sum_{0 \leq k \leq s_i - 1} S_i(k) \prod_{\substack{0 \leq j \leq s_i - 1 \\ j \neq k}} \frac{x_i - j}{k - j}.$$

Under regularity assumptions, the entire system has an expected degree of regularity

$$\begin{aligned} d_{\text{reg}}^{\texttt{Bar}} &= 1 + \sum_{i=1}^{n}(s_i - 1) + \sum_{i=1}^{n}(s_i - 2) \\ &= 1 - n + 2\sum_{i=1}^{n}(s_i - 1) \approx 2n\sqrt[n]{p}. \end{aligned}$$

For our BN, BLS, and FRI cases we have $d_{\text{reg}}^{\texttt{Bar}} > 2^{14}$.

b) Algebraic Representation of `Concrete` $\circ$ `Bar` $\circ$ `Concrete`*:* In this part we argue that already the computational cost of the *first* step (i.e., computing a *degrevlex*-Gröbner basis) of a *truncated* version (in essence, `Concrete` $\circ$ `Bar` $\circ$ `Concrete` in the CICO-setting) of `RC` far exceeds the 128-bit security requirement. Our arguments are based on empirical observations on small-scale instances of this truncated version.

We model the composition `Concrete` $\circ$ `Bar` $\circ$ `Concrete` in the CICO-setting[11] and suggest the following system of $6n+8$ equations in $6n+6$ variables as algebraic model:

p	47	61	71	97	109	127	131
v	5	7	7	7	7	7	7
s_1, s_2	7,7	8,8	8,9	10,10	10,11	11,12	11,12
d_{reg}	28	33	35	43	45	50	50
d_{mag}	13	13	13	15	16	19	18
$d_{\text{reg}} : d_{\text{mag}}$	2.2	2.5	2.7	2.9	2.8	2.6	2.8
M (GiB)	0.07	0.1	0.46	0.46	1.44	2.66	2.13
log T (cycles)	37	38	41	42	43	43	43
$C_{\text{bit}} : 2$	28	28	28	30	31	34	33

TABLE III: Results of Gröbner basis computations on small-scale instances of `Concrete` $\circ$ `Bar` $\circ$ `Concrete` in the CICO-setting for various primes p and decompositions into $n = 2$ buckets $\mathbb{Z}_{s_1}, \ldots, \mathbb{Z}_{s_n}$. The table contains the degree of regularity d_{reg} under the assumption that the input system is semi-regular, the timings of the Gröbner basis computations T in cycles, the memory usage M in Giga-byte and the complexity estimate C_{bit} in bits divided by 2.

$$\text{CBC}_{cico} = \begin{cases} y_1 = \texttt{Bar}(x_1) \\ y_2 = \texttt{Bar}(x_2) \\ y_3 = \texttt{Bar}(x_3) \\ 0 = \texttt{Concrete}^{-1}(x_1, x_2, x_3)[1] \\ 0 = \texttt{Concrete}(y_1, y_2, y_3)[1] \end{cases},$$

Here, `Concrete`$(\,\cdot\,, \,\cdot\,, \,\cdot\,)[i]$ denotes the i-th word of the state (for $1 \leq i \leq 3$) and n describes the number of *buckets* $\mathbb{Z}_{s_1}, \ldots, \mathbb{Z}_{s_n}$ in the decomposition `Decomp`. The variables x_1, x_2, x_3 and y_1, y_2, y_3, respectively, denote the input and output to `Bars`.

c) Discussion of Practical Results: In our practical experiments we computed Gröbner bases of small-scale instances of CBC_{cico} for various primes p and decompositions into $n = 2$ buckets.[12] Table III we present the results of our experiments. Instead of taking d_{reg} for establishing the complexity estimates, we computed Gröbner basis of several small-scale instances and observed the maximum degree d_{mag} reached during these computations using the CAS Magma. Subsituting d_{reg} with d_{mag} in above expression, results in our complexity estimate C. We use $\omega = 2$ and, furthermore, we take $C_{\text{bit}} := \log_2(C)$ to write down the complexity estimates in Table III. Note that $C_{\text{bit}}/2 \sim \log_2 T - 10$.

Our practical findings can be summarized as follows: *(i)* the maximum degree d_{mag} reached during the Gröbner basis computations is roughly 3 times smaller than the theoretical estimate for the maximum degree d_{reg}; *(ii)*

[11]See Appendix H1 for further details.

[12]We conducted our experiments on a machine with Intel® Core™ i5-8265U CPU @ 1.60GHz (8 cores) and 8GB RAM under 64-bit Ubuntu 21.04 using Magma V2.26-2.

using the empirical value d_{mag} for establishing complexity estimates, we observed that our practical experiments run about as fast as the square root of the complexity estimates. This yields an estimate for attacking the CBC layer with Groebner basis: $(C_{GB}(l = 150, d_{reg}/3 = 2^{12.5})^{1/2} > 2^{12.5 \cdot 150} = 2^{1875}$ which far exceeds the security level.

G. Algebraic Attacks

Here, we show that our design is secure against other algebraic attacks, including interpolation as well as higher-order differential distinguishers (we highlight that we do not claim security against zero-sum partitions). To achieve this goal, we argue that all mentioned methods cannot penetrate `Middle`. In particular, this implies that the full permutation provides security with respect to above mentioned cryptanalytical methods.

To rule out algebraic attacks, we introduce the following parameters:

- d_B is the degree of the `Bar` transformation as an operation over $\mathbb{F}_p$.
- d_S is the maximum degree of the component functions of the `Bricks` layer as an operation over $\mathbb{F}_p$.

1) Interpolation Analysis: In its basic form, interpolation analysis aims at constructing the polynomial representation of a given (cryptographic) function [37]. To provide resistance against interpolation, a function must exhibit maximal degree (or a degree close to its maximum) and a dense polynomial representation (i.e., a description with many non-zero coefficients).

The total degree of one word of the permutation `RC` over $\mathbb{F}_p$ is $d_B \cdot d_S^6$. It is enough to require $d_B > 2^{127}$ for 128-bit security.

A heuristic argument that $d_B > 2^{127}$ is that the we define `Bars` on at least p'^{27} points in a nonlinear way. This accounts to at least 2^{251} points, so the degree should exceed 2^{251}. We also computed the degree d_B for small-scale instances of `Bar` with $f(x) = x^{-1}$ as internal function f for the small S-Boxes $S_1, \ldots, S_n$ and $n = 2, 3$. For every instance we tested, the degree of `Bar` was maximal, i.e. $p - 2$, with almost all coefficients of the polynomial being non-zero. Extrapolating this trend and since $\log_2(p) \approx 256$ for the full-scale permutation `RC`, we conclude that above requirement is far exceeded.

2) Higher-Order Differential Attack and Zero-Sum Distinguishers: Given a vectorial Boolean function $\mathcal{F}$ over $\mathbb{F}_2^n$ of degree d, the higher-order differential attack [42], [40] exploits the fact that

$$\sum_{x \in \mathcal{V}+v} x = \sum_{x \in \mathcal{V}+v} F(x) = 0$$

for each affine subspace $\mathcal{V} + v \subseteq \mathbb{F}_2^n$ of dimension strictly bigger than d (that is, $\dim(\mathcal{V}) \geq d + 1$). The corresponding attack in the case of a prime field $\mathbb{F}_p$ has been recently proposed by Beyne et al. [15]. Since this result is related to the degree of the polynomial that describes the permutation, we claim that the security against the interpolation attack implies security against this attack as well.

A possible variant of higher-order sum in the case of permutations is the zero-sum partition distinguisher [21]. Here we explicitly state that *we do not make claims about the security of our scheme against zero-sum partitions.* This choice is motivated by the gap present in the literature between the number of rounds of the internal permutation that can be covered by a zero-sum partition and by the number of rounds in the corresponding sponge hash function that can be broken e.g. via a preimage or a collision attack.

H. Every Building Block is Necessary

1) The Necessity of `Bars`: We first focus on a design which excludes the `Bars` layer, and we show that a much higher number of rounds is needed to provide security. In order to do this, we use a Gröbner basis approach. Further, as our permutation is used in a Sponge setting, we consider the CICO (constrained input, constrained output) problem. More specifically, our goal is to find $t - k$ variables such that the first k words of both the input and the output of `RC` are zero. For a good hash transformation, we expect this to take a workload of p^k operations when working over $\mathbb{F}_p$.

In more detail, we want that

$$x = 0\| \cdots \|0\|x_{k+1}\| \cdots \|x_t$$

is the input of the function (where $\cdot\|\cdot$ denotes the concatenation) and

$$y = \texttt{RC}(x) = 0\| \cdots \|0\|y_{k+1}\| \cdots \|y_t$$

is the output, and our goal is to find $x_{k+1}, \cdots, x_t$.

Focusing on our function with $t = 3$ and using a single element of approximately 256 bits for the capacity (for a 128-bit security level), let us consider $k = 1$. We then have 2 variables and only one equation. This system is underdetermined, but we can arbitrarily fix one of the variables. In the end, we arrive at a single equation in a single variable.

a) Full-Round Equations.: Using this straightforward approach, we note that our `Bricks` layer has a degree of 3. Without further considering the density of the resulting polynomials, our final goal is to find the roots of a univariate equation of degree 3^r, where r is the number of rounds. Since this cost is approximately an element in $\mathcal{O}(d^3)$ [29], we want that

$$3^{3r} \geq p \implies r \geq \log_{27}(p).$$

For example, if $p \approx 2^{256}$, this results in $r \geq 54$, which is much larger than what our current proposal needs.

b) Intermediate Variables and Equations.: Another possible approach is to keep the degrees low by introducing additional variables. In order to do this, we introduce 3 new variables in each round and we arrive at a system of degree-3 equations.

Let us again assume that we use r rounds. Then, we introduce $3(r-1)$ new variables and equations. In the end, we arrive at $n_e = 3(r-1)+1$ degree-3 equations and the same number of variables n_v (one additional equation for the final zero element, and the original variable x_t at the beginning). Generically, the complexity of solving such a system is then in

$$\mathcal{O}\left(\binom{D_{\text{reg}} + n_v}{n_v}^{\omega}\right),$$

where we set $\omega = 2$ and where

$$D_{\text{reg}} = 1 + \sum_{i=1}^{n_e} 2 = 1 + 2 \cdot (3(r-1)+1).$$

In this case, we want that

$$\binom{1 + 6(r-1) + 2 + 3(r-1) + 1}{3(r-1)+1}^2 = \binom{9r-5}{3r-2}^2 \geq p,$$

which results in $r \geq 33$ for $p \approx 2^{256}$. Note that we are not exploiting the density and general structure of the polynomials. Indeed, when using equations which cover single rounds, we can assume that they do not exhibit strong pseudo-random properties, which means that the above estimation is actually a pessimistic one (from the attacker's perspective). However, this is sufficient to show the efficiency of our current proposal, since any faster attack would only further increase the number of rounds needed for security in the design without the `Bars` layer.

2) The Necessity of `Concrete`: Without the `Concrete` layer, we would have a weaker diffusion over the 3 words. In particular, note that the `Bricks` layer does not provide any mixing in the first word. Hence, when omitting the `Concrete` layer, the subspaces $\langle(0,1,0),(0,0,1)\rangle$ and $\langle(0,0,1)\rangle$ are invariant through the whole permutation, independent of the number of rounds.

3) The Necessity of `Bricks`: Without the `Bricks` layer, an attacker could work with a system of equations over the smaller fields of the `Bars` layer. Moreover, the in-word diffusion (i.e., the diffusion in a single word) would only happen in the `Bars` layer, which is weak. Further, we need the `Bricks` layer for statistical arguments, since e.g. in a rebound attack both outbound phases would be linear otherwise (when considering the `Bars` layer in the inbound phase).