# Improvements to the Linear Layer of LowMC: A Faster Picnic

Léo Perrin[1], Angela Promitzer[2], Sebastian Ramacher[3], and Christian Rechberger[3]

[1] INRIA, France, firstname.lastname@inria.fr
[2] angela.promitzer@gmail.com
[3] IAIK, Graz University of Technology, Austria, firstname.lastname@tugraz.at

**Abstract.** Picnic is a practical approach to digital signatures where the security is largely based on the existence of a one-way function, and the signature size strongly depends on the number of multiplications in the description of that one-way function. The highly parameterizable block cipher family LowMC has the most competitive properties with respect to this metric, and is hence a standard choice. In this paper we study various options for efficient implementations of LowMC in-depth. First, we investigate optimizations of the linear layer of LowMC independently of any implementation optimizations. By decomposing the round key computations based on the keys' effect on the S-box layer and general optimizations, we reduce runtime costs by up to 40 % and furthermore reduce the size of the LowMC matrices by around 55 % compared to the original Picnic implementation (CCS'17).
Second, we propose a Feistel structure using smaller matrices completely replacing the remaining large matrix multiplication in LowMC's linear layer. With this approach we achieve an operation count logarithmic in the blocksize, but more importantly improve over Picnic's constant-time matrix multiplication by 60 % while retaining a constant-time algorithm. Furthermore, this technique also enables us to reduce the memory requirements for the LowMC matrices by 50 %.

**Keywords:** LowMC, efficient implementation, Picnic, post-quantum digital signatures

## 1 Introduction

Light weight cryptographic primitives that only require a low number of multiplications have many applications ranging from reducing costs for countermeasures against side-channel attacks [DPVR00, GLSV14], over improving homomorphic encryption schemes [ARS+15, MJSC16, CCF+16, DSES14, NLV11] and multi-party computation [GRR+16, RSS17], to SNARKS [AGR+16]. But they also turned out to be useful to reduce signature sizes of post-quantum signature schemes based on $\Sigma$-protocols [CDG+17a] without further structured hardness assumptions. The latter in particular builds upon LowMC [ARS+15, ARS+16], a highly parameterizable block cipher, and profits not only from the low number

of multiplications, but also from the small product of multiplications and the field size. Using LowMC in this context allows further reductions to the signature size, since the LowMC parameters can be chosen suiting scenarios where an adversary can only observe one plaintext-ciphertext pair.

We focus on the use of LowMC in the post-quantum digital signature scheme Picnic [CDG+17a, CDG+17b] which is based on zero-knowledge proofs of knowledge for pre-images of one way functions instantiated using LowMC. As proof system ZKB++ is used, which is based on the "MPC-in-the-head" [IKOS07] paradigm. To compute proofs, the circuit of the one way function is decomposed into three branches where XORs can be computed locally, but ANDs require information from other branches. Signature sizes depend on the total number of AND gates used in the one way function.

From the use of LowMC in Picnic, diametral constraints merge: First, the total number of AND gates, which is a multiple of number of rounds and number of S-boxes, directly relates to the signature size and is thus desired to be kept small. Second, as one decreases the number of S-boxes, the number of rounds increases leading to a three-fold increase in the number of linear layer operations. While applications using plain LowMC could save half of the linear layer by simply pre-computing round keys for multiple encryptions and description, in Picnic the key is shared into fresh shares before each invocation of LowMC. Thus simple round key pre-computation cannot be applied to this use case.

### 1.1 Contribution

The contributions of this work can be summarized as follows. The first one is an alternative description of LowMC, while the second one is a proposal for a change of LowMC.

- We propose an alternative description of LowMC with a new structure to compute round keys. The idea here is to split the computation into linear and non-linear parts. This change allows us to replace all round key computations only affecting the linear part with exactly one matrix multiplication. The remaining non-linear parts can then be computed by much smaller round key matrices. This new description of LowMC allows us to greatly reduce the size of the LowMC matrices. In the signature use-case this optimization leads to performance improvements from 10 % for smaller block sizes to 40 % for larger block sizes. Additionally, this optimization is independent of implementation optimizations of the matrix multiplication.
- We present Fibonacci Feistel Networks (FFNs), a variant of Generalized Feistel Networks, which provide very fast diffusion. Instantiating the network with regular matrices as permutation we obtain a compact representation of a larger matrix multiplication. The obtained equivalent of a matrix multiplication algorithm with logarithmic complexity can then be used to replace the linear layer of LowMC. This technique reduces the size of the LowMC matrices again by up to 50 %.

Albeit in the practical part of the work we focus on the Picnic use-case, both contributions will also positively affect other use-cases of LowMC. The alternative description is likely to be useful for cryptanalysis purposes as well.

### 1.2 Related Work

**Efficient Implementation.** Besides the security analysis, good performance figures are an important characteristic of a useful block cipher. As such, asking for an alternative description or finding other implementation tricks which allow to improve the overall performance or make the use of the cipher viable under certain constraints in the first place, is a natural question to ask. As an example, during and after the AES competition many authors worked on fast software implementations [AL00, BS08], fast hardware implementations [SME16], but also on alternative and more efficient descriptions of the algorithm [BBF+02, BB02].

**Fast Matrix-Vector Multiplication.** Even besides cryptographic applications, fast matrix-vector multiplication over binary fields is of interest in different research areas and has seen various improvements and applications over the last decades [ADKF70, Bar06, Ber09]. Despite the improvements over the naïve matrix-vector multiplication algorithm, a runtime of $\mathcal{O}\left(nm/\log m\right)$ for $n \times m$ matrices is currently the asymptotically best achievable option. We note however, that with the advent of SIMD instruction sets, the constant factors can be significantly decreased due the fact that 128, 256 or even 512 bits can be processed simultaneously.

**Feistel Networks.** The exploration of Feistel Networks is as old as block ciphers itself. Design and analysis sof various generalizations have been explored, see e.g. [NPV17] for a recent survey. In the second part of our work be present and use a new Feistel generalization for an arbitrary number of branches with fast diffusion. The closest related work to this is due to Suzaki and Minematsu [SM10].

### 1.3 Outline

We recall LowMC and the $(2, 3)$-decomposition as used by Picnic in Section 2. In Section 3 we discuss the optimizations to LowMC's linear layer and finish with their experimental verification in Section 4. Section 5 discusses the results of the preceding section.

## 2 Preliminaries

In this section we briefly recall LowMC and the $(2, 3)$-decomposition as used by Picnic.

### 2.1 LowMC

LowMC [ARS+15, ARS+16] is a very parameterizable symmetric encryption scheme design enabling instantiation with low AND depth and low multiplicative complexity. Given any blocksize, a choice for the number of S-boxes per

round, and security expectations in terms of time and data complexity, instantiations can be found minimizing the AND depth, the number of ANDs, or the number of ANDs per encrypted bit. We recall LowMC's definition using a partial S-box and $\mathbb{F}_2$ vector space arithmetic: Let $n$ be the blocksize, $m$ be the number of S-boxes, $k$ the key size, and $r$ the number of rounds, we choose round constants $C_i \xleftarrow{R} \mathbb{F}_2^n$ for $i \in [1, r]$, full rank matrices $K_i \xleftarrow{R} \mathbb{F}_2^{n \times k}$ and regular matrices $L_i \xleftarrow{R} \mathbb{F}_2^{n \times n}$ independently during the instance generation and keep them fixed. Keys for LowMC are generated by sampling from $\mathbb{F}_2^k$ uniformly at random. LowMC consists of key whitening in the beginning and multiple rounds composed of an S-box layer, a linear layer, addition with constants and addition of the round key. Algorithm 1 gives a full description of the encryption algorithm.

**Algorithm 1** LowMC encryption for key matrices $K_i \in \mathbb{F}_2^{n \times k}$ for $i \in [0, r]$, linear layer matrices $L_i \in \mathbb{F}_2^{n \times n}$ and round constants $C_i \in \mathbb{F}_2^n$ for $i \in [1, r]$.

**Require:** plaintext $p \in \mathbb{F}_2^n$ and key $y \in \mathbb{F}_2^k$
  $s \leftarrow K_0 \cdot y + p$
  **for** $i \in [1, r]$ **do**
    $s \leftarrow \text{SBOX}(s)$
    $s \leftarrow L_i \cdot s$
    $s \leftarrow C_i + s$
    $s \leftarrow K_i \cdot y + s$
  **end for**
  **return** $s$

To reduce the multiplicative complexity, the number of S-boxes applied in parallel can be reduced, leaving part of the substitution layer as the identity mapping. We also note that, that this choice has little to no influence on the efficiency of the S-box layer, since a bit-sliced implementation can process all at once[4]. The number of rounds $r$ needed to achieve the goals is then determined as a function of all these parameters.

### 2.2 (2, 3)-Decomposition of Circuits in Picnic

Circuit decomposition is a protocol for jointly computing a circuit, similar to an MPC protocol, but with greater efficiency. In a $(2, 3)$-decomposition there are three players and the protocol has 2-privacy, i.e., it remains secure even if two of the three players are corrupted. We discuss some definitions from [GMO16] and the instantiation in Picnic [CDG+17a].

[4] Given $m$ S-boxes, the bit-sliced implementation can be implemented using $3m$ bit ANDs, ORs and shifts. Thus, as long as $3m$ bit fit into a platform's registers, the $m$-fold S-box can be implemented without overhead compared to a single S-box.

**Definition 1 ((2,3)-decomposition).** *Let $f$ be a function that is computed by an $n$-gate circuit $\phi$ such that $f(x) = \phi(x) = y$. Let $k_1, k_2$, and $k_3$ be tapes of length $\kappa$ chosen uniformly at random from $\{0,1\}^\kappa$ corresponding to players $P_1, P_2$ and $P_3$, respectively. The tuple of algorithms* $(\mathsf{Share}, \mathsf{Update}, \mathsf{Output}, \mathsf{Reconstruct})$ *are defined as follows:*

$\mathsf{Share}(x, k_1, k_2, k_3)$: *: On input of the secret value $x$, outputs the initial views for each player containing the secret share $x_i$ of $x$.*

$\mathsf{Update}(\mathsf{view}_i^{(j)}, \mathsf{view}_{i+1}^{(j)}, k_i, k_{i+1})$: *On input of the views $\mathsf{view}_i^{(j)}, \mathsf{view}_{i+1}^{(j)}$ and random tapes $k_i, k_{i+1}$, compute the wire values for the next gate and returns the updated view $\mathsf{view}_i^{(j+1)}$.*

$\mathsf{Output}(\mathsf{view}_i)$: *On input of the final view $\mathsf{view}_i \equiv \mathsf{view}_i^{(n)}$, returns the output share $y_i$ of player $P_i$.*

$\mathsf{Reconstruct}(y_1, y_2, y_3)$: *On input of all output shares $y_i$, reconstructs and returns $y$.*

Correctness requires that reconstructing a $(2,3)$-decomposed evaluation of a circuit $\phi$ yields the same value as directly evaluating $\phi$ on the input value. The 2-privacy property requires that revealing the values from two shares reveals nothing about the input value.

ZKB++, a $\Sigma$-protocol for proving statements about general circuits, constructs the $(2,3)$-decomposition of a circuit as follows: Let $R$ be an arbitrary finite ring and $\phi$ a function such that $\phi : R^m \to R^\ell$ can be expressed by an $n$-gate arithmetic circuit over the ring using addition respectively multiplications by constants, and binary addition and binary multiplication gates. A $(2,3)$-decomposition of $\phi$ is then given by:

$\mathsf{Share}(x, k_1, k_2, k_3)$: Samples random $x_1, x_2 \in R^m$ from $k_1$ and $k_2$ and computes $x_3$ such that $x_1 + x_2 + x_3 = x$. Returns views containing $x_1, x_2, x_3$.

$\mathsf{Output}_i(\mathsf{view}_i^{(n)})$: Selects the $\ell$ output wires of the circuit as stored in the view $\mathsf{view}_i^{(n)}$.

$\mathsf{Reconstruct}(y_1, y_2, y_3)$: Computes $y = y_1 + y_2 + y_3$ and returns $y$.

$\mathsf{Update}_i^{(j)}(\mathsf{view}_i^{(j)}, \mathsf{view}_{i+1}^{(j)}, k_i, k_{i+1})$: Computes $P_i$'s view of the output wire of gate $g_j$ and appends it to the view. For the $k$-th wire $w_k$ where $w_k^{(i)}$ denotes $P_i$'s view, the update operation is defined as follows for the specific gate types:

- **Addition by Constant ($w_b = w_a + k$):** $w_b^{(i)} = w_a^{(i)} + k$ if $i = 1$ and $w_b^{(i)} = w_a^{(i)}$ otherwise.
- **Multiplication by Constant ($w_b = k \cdot w_a$):** $w_b^{(i)} = k \cdot w_a^{(i)}$
- **Binary Addition ($w_c = w_a + w_b$):** $w_c^{(i)} = w_a^{(i)} + w_b^{(i)}$
- **Binary Multiplication ($w_c = w_a \cdot w_b$):** $w_c^{(i)} = w_a^{(i)} \cdot w_b^{(i)} + w_a^{(i+1)} \cdot w_b^{(i)} + w_a^{(i)} \cdot w_b^{(i+1)} + R_i(c) - R_{i+1}(c)$ where $R_i(c)$ is the $c$-th output of a pseudorandom generator seeded with $k_i$.

Note that $P_i$ can compute all gate types locally with the exception of binary multiplication gates as this requires inputs from $P_{i+1}$. In other words, only outputs of binary multiplication gates need to be serialized, and thus the view size and consequentially the signature size of Picnic depend on the size of the ring $R$ and the number of multiplication gates.

## 3 Optimizing the Linear Layer

From the use of LowMC in Picnic, we obtain some constraints on the optimizations we are allowed to perform. The S-box serves as a synchronization point on the first $3m$ bits, i.e. the bits that are actually touched by the S-box. On the other $n - 3m$ bits the S-box is simply the identity map and their actual values do not matter for S-box evaluations. Thus we have to ensure that the evaluation of all AND gates stays invariant under all our optimizations. Secondly, we have to assume that the secret key – or more precisely the shares representing it – changes on every encryption.

### 3.1 Splitting the Round Key Computation

We start with the round key computation. Since the secret key is freshly shared for each LowMC evaluation in Picnic, the round keys cannot be pre-computed once during initialization. However, we can observe that due to the structure of the S-box layer for $n - 3m$ bits of the round key, which coincide with the part of the state where the S-box acts as identity map, it does not matter whether those bits are added to the state before or after the application of the S-box. Due to the linear nature of all operations involved in the computation after the S-box, we can simply change the order of adding the round key and multiplying the state with $L_i$. We modify each round as follows:

- Modify $s \leftarrow L_i \cdot s + K_i \cdot y + C_i$ to $s \leftarrow L_i \cdot (L_i^{-1} \cdot K_i \cdot y + s) + C_i$.
- Now split $L_i^{-1} \cdot K_i \cdot y$ into the lower $3m$ bits (the "non-linear part") and the upper $n - 3m$ bits (the "linear part") and move the addition of the upper $n - 3m$ bits before the S-box layer.

In the following we denote by $\rho_i^j : \mathbb{F}_2^n \to \mathbb{F}_2^{j-i}$ the map sending $n$-dimensional vectors to a vector only consisting of the $i$-th to $j$-th coordinate, i.e. $\rho_i^j(v_1, \ldots, v_n) \mapsto (v_i, \ldots, v_j)$, to simplify some notation when refering to the linear and non-linear part. In particular, we use $\rho_L = \rho_{3m+1}^n$ to identitfy the linear part and $\rho_N = \rho_1^{3m}$ for the non-linear part, respectively.

Figure 1 now demonstrates one round of LowMC with the above modifications. Observe this modification does not change the output of LowMC. Additionally, we can continue in moving the addition of the linear part to the previous round until all those additions have been moved at the start of the LowMC encryption algorithm.

After iterating this procedure to move all linear parts of the round key before the round, we end up with all additions of the linear parts of the round key before

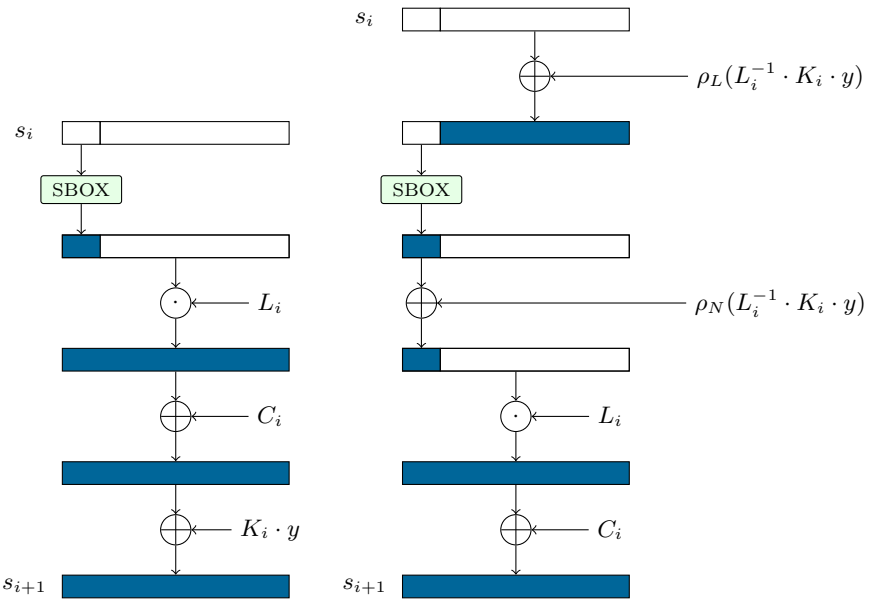



Fig. 1: One round of LowMC before (left) and after (right) the splitting of the round key

the first round and are left with a reduced round key of $3m$ bits per round. We now discuss the invovled matrices. For the computation of the linear and non-linear part, the matrix $\overline{L_i^{-1}}$ has to be computed for each LowMC round $i$. By $\overline{L_i^{-1}}$ we denote inverse of the linear layer matrix $L_i$ with the first $3m$ columns of this inverse set to 0.

The linear part can be computed by calculating the matrix $P_L$, which is defined as

$$P_L = \overline{L_1^{-1}} \cdot K_1 + \sum_{j=2}^{r} \left( \prod_{k=1}^{j} \overline{L_k^{-1}} \right) \cdot K_j$$

and then multiplying it with the master key $y$ and adding the result to the initial state $s_0$. The matrix $P_L$ can be precomputed from the LowMC matrices before any encryption and thus only the matrix multiplication with the master key $y$ and the addition to the initial state $s_0$ is required at the beginning of the encryption.

For the non-linear part we can define a similar matrix $P_{Ni}$ for round $i$, which is

$$P_{Ni} = \overline{L_i^{-1}} \cdot K_i + \sum_{j=i+1}^{r} \left( \prod_{k=i}^{j} \overline{L_k^{-1}} \right) \cdot K_j.$$

This matrix $P_{Ni}$ of the dimension $(n \times k)$ is then multiplied with the master key $y$ and the first $3m$ bits of this result are added to the state in the corresponding

LowMC round after applying the S-box function to the state. However, this still implies that this multiplication $P_{Ni} \cdot y$ is done in every LowMC round, which can be avoided by using the following structure. As only $3m$ bits of $P_{Ni} \cdot y$ are used, we can combine the first $3m$ rows of the matrix $P_{Ni}$, denoted as $\overline{P_{Ni}}^{3m}$, of all rounds $i$ to one matrix $C_N$ of dimension $(3m \cdot r \times k)$. The combined precomputed non-linear values $C_N$ can then be multiplied with the master key $y$ before the encryption, which results in a vector $v$ of dimension $(3m \cdot r \times 1)$.

$$C_N = \begin{pmatrix} \overline{P_{N1}}^{3m} \\ \vdots \\ \underbrace{\overline{P_{Nr}}^{3m}}_{k \text{ cols}} \end{pmatrix} \begin{matrix} \}3m \text{ rows} \\ \\ \}3m \text{ rows} \end{matrix}$$

Now, the $3m$ bits starting from bit $i \cdot 3m$ of the vector $v$ can be added to the non-linear part of the state in round $i$. So, although the non-linear part remains in each LowMC round the computation effort is reduced from a $(n \times k)$ matrix-vector multplication and a $n$-bit XOR to a $3m$-bit XOR in each round. The pre-comuptation of $P_L$ and $P_{Ni}$ can be done independently of the master key $y$ and before any encryption takes place. The multiplications $P_L \cdot y$ and $C_N \cdot y$ have to be done once at the start of the encryption algorithm with the specific shares of $y$ for this encryption. Algorithm 2 shows the full LowMC encryption with the reduced linear layer.

**Algorithm 2** LowMC encryption with a reduced linear layer for key matrices $K_i \in \mathbb{F}_2^{n \times k}$ for $i \in [0, r]$, linear layer matrices $L_i \in \mathbb{F}_2^{n \times n}$ and round constants $C_i \in \mathbb{F}_2^n$ for $i \in [1, r]$ and the precomputed matrices $P_L$ and $C_N$.

**Require:** plaintext $p \in \mathbb{F}_2^n$ and key $y \in \mathbb{F}_2^k$
  $v \leftarrow C_N \cdot y$
  $s \leftarrow (K_0 + P_L) \cdot y + p$
  **for** $i \in [1, r]$ **do**
    $s \leftarrow \text{SBOX}(s)$
    $s \leftarrow \rho_{3 \cdot i}^{3 \cdot (i+1)}(v) + s$
    $s \leftarrow L_i \cdot s$
    $s \leftarrow C_i + s$
  **end for**
  **return** $s$

The necessary matrices for the unmodifided LowMC algorithm and LowMC with the reduced linear layer (RLL) are presented in Table 1. From the memory it can be seen, that LowMC with RLL reduces the memory consumption for the round keys from $r+1$ $(n \times k)$ matrices to 1 $(n \times k)$ and 1 $(3m \cdot r \times k)$ matrix. However, LowMC with RLL introduces an additional vector $v$ of dimension $(3m \cdot r \times 1)$ to store the non-linear part for each specific encryption.

| | **LowMC** | **LowMC with RLL** |
|---|---|---|
| Linear layer | $r$ $(n \times n)$ | $r$ $(n \times n)$ |
| Round key matrices | $(r+1)$ $(n \times k)$ | 1 $(n \times k)$ (linear part) |
| | | 1 $(3m \cdot r \times k)$ (non linear part) |
| Round constants | $r$ $(1 \times n)$ | $r$ $(1 \times n)$ |
| Additional memory | | 1 $(3m \cdot r \times 1)$ (vector $v$) |

Table 1: Necessary matrices for general LowMC and LowMC with RLL.

We note that the same modification can be done to the round constants $C_i$ (for $i \in [1, r]$). They can be moved to the beginning of the encryption in the same manner as the round key. Also, for LowMC's decryption algorithm the same modification applies.

### 3.2 Fibonacci Feistel Network

Given that the cost of storing a binary matrix operating on $n$ bits is proportional to $n^2$, it is possible to decrease the cost of a $n \times n$ matrix when it can be implemented using several $m \times m$ matrices with $m < n$. In this section, we describe a method for building large binary matrices using several smaller ones while still ensuring that every output bit may depend on every input bit.

Our main idea is to borrow techniques from block cipher design, in particular from Generalized Feistel Networks (GFN). In fact, we propose a new variant of this structure, called *Fibonacci Feistel Network (FFN)*, which provides very fast diffusion. As its name indicates, this structure uses the Fibonaci sequence $\{\phi_i\}_{i \geq 0}$ defined by the well-known induction formula:

$$\begin{cases} \phi_0 = 0 \\ \phi_1 = 1 \\ \phi_{i+1} = \phi_i + \phi_{i-1} \ , \end{cases}$$

so that $\phi_0, \phi_1, \phi_2, \phi_3, \phi_4, \phi_5 = 0, 1, 1, 2, 3, 5$. The smallest integer $i$ such that $\phi_i > b$ is denoted $i = \Lambda_\phi(b)$. For example, $\phi_8 = 21$ and $\phi_9 = 34$, so $\Lambda_\phi(32) = 9$.

**The Fibonacci-Feistel Structure.** A FFN operates on $2bw$ bits, where $w \geq 4$ and $b \geq 2$, using $R$ rounds. The round functions are different in each round although they always use the same overall structure: each round is a classical 2-branched Feistel round where the Feistel functions maps $bw$ bits to $bw$. The Feistel function used in round $i$ is denoted $F_i$ for all $0 \leq i$ and works as follows:

1. the state is divided into $b$ branches of $w$ bits,
2. each branch goes through a $w$-bit L-Box, that is, a linear function mapping $w$ bits to $w$ bits, and
3. the branches are rotated by $\phi_i$, so that $B_j \leftarrow B_{j+\phi_i}$, where $\phi_i$ is the $i$-th Fibonacci number and the sum is taken modulo $b$.

Alternatively, let us denote the $2bw$-bit internal state at round $i$ as $X^i$ and its $w$-bit branch with index $j$ as $X^i_j$. The full round function works as follows, for $j < b$:

$$X^{i+1}_{j+b} = X^i_j \ , \ X^{i+1}_j = X^i_{j+b} \oplus L^i_{j-\phi_i}\left(X^i_{j-\phi_i}\right) \ ,$$

where $j - \phi_i$ is taken modulo $b$ and where $L^i_j$ is a $w$-bit L-Box. An example is given in Figure 3 in Appendix A.

**Diffusion in a FFN.** Diffusion is very fast in such a structure. The idea of using something more sophisticated than a constant rotation for mixing the branches is not new. In [SM10], Minematsu and Suzaki proposed using complex permutations which significantly improve diffusion: for a structure operating on $k$ branches, only $2\log_2(k)$ rounds are needed to achieve full diffusion while a naïve constant rotation would need $k$ rounds. Furthermore, the gain increases as the number of rounds increases, making such methods even more appealing on larger blocks. It was for example used to design the lightweight block cipher Twine [SMMK12].

Unlike such GFN, the FFN needs a different permutation in each round. However, it works for any even number of branches, we are not restricted to powers of two. The permutations used are also much simpler and, as we explain later, they can lend themselves well to constant-time software implementations using only ANDs, XORs and rotations.

In what follows, we present two lemmas which quantify diffusion inside a FFN more formally. Lemma 1 describes how a single word diffuses through several rounds of FFN. While Lemma 2 highlights some invariant properties of such a network. These two are put together in Corollary 1 to quantify how many rounds are needed to have some form of full diffusion. The notations in Definition 2 are used throughout the remainder of this section.

**Definition 2 (Notations).** *We denote with $X^i_j$ the word with index $j$ at the input of round $i$ and we say that $x$* influences *$y$ if the expression of $y$ involves the variable $x$.*

**Lemma 1.** *Let $i$ be such that $\phi_{i+1} \leq b$. Then the word $X^0_0$ influences all words $X^i_j$ with indices $0 \leq j < \phi_{i+1}$ and $b \leq j < b + \phi_i$.*

The process behind this lemma is illustrated in Figure 2 and formally proved further below. The propagation of a word accross several rounds is illustrated in Figure 3 in Appendix A.

*Proof.* We proceed by induction.

Before round 0, i.e. at the beginning of the application of the FFN, only $X^0_0$ depends on $X^0_0$. After round 0 (where the Feistel function contains a rotation by $\phi_0 = 0$), only $X^1_0$ and $X^1_b$ depend on $X^0_0$, so that $X^1_j$ depends on $X^0_0$ if and only if $0 \leq j < \phi_2$ or $b \leq j < b + \phi_1$.

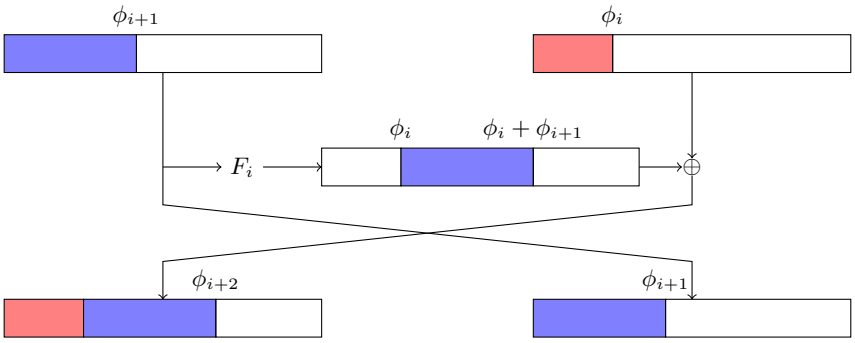



Fig. 2: How $X_0^0$ influences $X_j^i$ for increasing $i$. The parts of the internal state which depend on $X_0^0$ are colored; those that do not are in white. The Feistel function $F_i$ involves a layer of L-Boxes and a rotation by $\phi_i$.

Suppose now that $X_j^i$ depends on $X_0^0$ if and only if $0 \leq b < \phi_i$ or $b \leq j < b + \phi_{i-1}$. The round function with index $i$ maps $X^i$ to $X^{i+1}$ such that

$$\begin{cases} X_j^{i+1} = X_{j+b}^i \oplus L_{j-\phi_i}^i \left( X_{j-\phi_i}^i \right) & \text{if } j < b \\ X_j^{i+1} = X_{j-b}^i & \text{if } j \geq b \ . \end{cases}$$

If $\mathbf{j \geq b}$, then $X_j^{i+1} = X_{j-b}^i$ which, by the induction hypothesis, depends on $X_0^0$ if and only if $0 \leq j - b < \phi_i$, which is equivalent to $b \leq j < \phi_i$. Thus, the lemma holds in this case.

If $\mathbf{j < b}$, then $X_j^{i+1}$ depends on both $X_{b+j}^i$ and $X_{j-\phi_i}^i$. By the induction hypothesis, the first depends on $X_0^0$ if and only if $b \leq b + j < b + \phi_i$, which is equivalent to $0 \leq j < \phi_i$. The second depends on $X_0^0$ if and only if $0 \leq j - \phi_i < \phi_{i+1}$ or, equivalently, $\phi_i \leq j < \phi_i + \phi_{i+1} = \phi_{i+2}$. Thus, for any $j$ such that $0 \leq j < \phi_{i+2}$, $X_j^{i+1}$ depends on $X_0^0$. This dependence occurs via $X_{b+j}^i$ for $j < \phi_i$ and via $X_{j-\phi_i}^i$ for $\phi_i \leq j < \phi_{i+2}$. We deduce that the lemma holds in this case as well.

□

If we leave the details of the L-Boxes aside, a FFN is invariant under rotation of the halves of the state. This is formalized by the following lemma.

**Lemma 2.** *Let $R_k$ be a permutation of $\{0, ..., 2b-1\}$ such that*

$$R_k(j) = \begin{cases} (j+k) \mod b & \text{if } j < b, \\ b + ((j-b+k) \mod b) & \text{if } j \geq b \ , \end{cases}$$

*meaning that applying $R_k$ to the index of the branches of FFN rotates separately the left and right branches by $k$.*

*Let $P$ be one round of FFN where all L-Boxes are the same and let $(y_0, ..., y_{2b-1}) = P(x_0, ..., x_{2b-1})$. Then the following always holds:*

$$P(y_{R_k(0)}, ...y_{R_k(2b-1)}) \ = \ (y_{R_k(0)}, ...y_{R_k(2b-1)}) \ .$$

*Proof.* If the L-boxes are identical then all operations in a FFN are invariant under a word rotations applied to both halves of the internal state. □

We deduce the following Corollary which quantifies the speed of diffusion in a FFN.

**Corollary 1.** *Let $i$ be such that $b < \phi_i$. Then all output words $X_j^{i+2}$ (for all $j \in \{0, ..., 2b-1\}$) of the $i+1$-round FFN depend on all the left input words $X_k^0$ where $k < b$.*

*Proof.* Let $k \leq i-1$ be such that $\phi_k \leq b < \phi_{k+1}$. By Lemma 1, we know that after $k$ rounds, $X_0^0$ influences all words with indices $j$ if $0 \leq j < \phi_k$ or $b \leq j < b + \phi_{k-1}$. After round $k+1$, the words on the left all depend on $X_0^0$ and thus, after round $k+2 \leq i+1$, all output words depend on $X_0^0$.

Because of Lemma 2, we can generalize this dependency to all input words from the left side. We conclude that, after round $k$ all output words depend on $X_j^0$ for all $j < b$.

**Corollary 2.** *Let $i$ be such that $\phi_{i+1} \leq b$. Then each $X_j^0$ for $j < b$ influences $\phi_{i+2}$ words $X_k^i$ after $i-1$ rounds.*

*Proof.* By Lemma 1, $X_0^0$ influences words $X_j^i$ where $0 \leq j < \phi_{i+1}$ and $b \leq j < b + \phi_i$, i.e. a total of $\phi_{i+1} + \phi_i = \phi_{i+2}$ words. By Lemma 2, this property holds for any word on the left side of the input, i.e. for all $X_j^0$ with $j < b$.

**Efficient Implementation.** In this section, we describe how we can build a linear layer which can be evaluated using $\mathcal{O}(\Lambda_\phi(b) \times w)$ operations on a modern processor, where $2 \times b \times w = n$. This linear layer provides full diffusion. Of course, it uses a FFN with $2b$ branches of $w$ bits each. The core trick is in the definition and implementation of the L-Box layer composed with the rotation by $\phi_i$.

Instead of using table lookups to implement the linear layer, we can use some sort of in-place bit-sliced strategy for small branch sizes[5], as follows. Let $x$ be the $bw$-bit input of $F_i$ where $F_i$ consists in a layer of $w$-bit L-Boxes and a rotation by $\phi_i$. Then we can write

$$F_i = M^{-1} \circ \mathrm{ROT}_{\phi_i} \circ \mathcal{L}_i \circ M \ ,$$

where:

- $M$ maps the $w$-bit word with index $j$ to the bits with indices congruent to $j$ modulo $b$. More formally, $M$ is a bit permutation which maps $x_0, ...x_{bw-1}$ to $x_0, x_b, x_{2b}, ...x_{(w-1)b}, x_1, x_{b+1}..., x_{bw-1}$, i.e. each bit of the first $w$-bit word end up at positions $k \equiv 0 \pmod b$, the bits of the second $w$-bit word at positions $k \equiv 1 \pmod b$, etc.

[5] For larger branch sizes, e.g. $w \geq 32$ we can fall back on optimized matrix-vector multiplication algorithms and the branch rotation can simply be performed by index arithmetic in arrays storing the state.

– $\mathcal{L}_i$ is parametrized by $w$ words of size $bw$ denoted $\ell_j^i$. It maps a $bw$-bit word $y$ to a value equal to

$$\mathcal{L}_i(y) = \bigoplus_{j=0}^{w-1} \ell_j^i \wedge (y \lll (j \times b)) \ .$$

Thus, the output bit of $\mathcal{L}_i$ with index $k$ is a linear combination of $w$ variables $y_k, y_{b+k}, y_{2b+k}, ..., y_{(w-1)b+k}$. The $w$ output variables with indices $k \equiv a \pmod b$ depend only on the $w$ input variables with indices $k \equiv a \pmod b$: $\mathcal{L}_i$ is effectively an L-Box layer except that it operates on slices of the state defined by $k \pmod b$ rather than $\lfloor k/b \rfloor$.

Basically, $M$ changes the representation of the words and $\mathcal{L}_i$ evaluates the L-Box layer on this alternative representation. In fact, we can simplify the functions $M$ in each round by applying $M$ to each half of the input of the whole FFN and $M^{-1}$ to each half of its output. If the only aim is full diffusion we can simply remove them.

In the end, evaluating a linear FFN with full diffusion on $2b$ branches with $w$-bit words requires $\Lambda_\phi(b) + 1$ rounds, each of which requires

– $w$ rotations applied word-wide on a $bw$-bit word by $\phi_i + b \times k$ for all $k \in \{0, ..., w-1\}$,
– $w$ ANDs applied word-wide on a $bw$-bit word,
– $w$ XORs applied word-wide on a $bw$-bit word to combine the $\ell_j^i \wedge (x \lll bk + \phi_i)$ with the other branch.

Let us consider a practical, namely a 256-bit linear layer to be used by a smaller-block variant of LowMC. For $n = 256 = 2 \times 32 \times 4$, we can implement a bijective linear layer with full diffusion using $1 + \Lambda_\phi(32) = 10$ rounds by composing a 9-round FFN with another round before it. The role of this first round is to ensure that the left input of the FFN depends on both the left and right words of the input of the structure. In this way, at the end of the FFN, all output words depend on all input words. The rotation used in this first round can be chosen freely; we suggest using $b/2 = 16$.

Evaluating such a linear layer requires 10 rounds during which we perform:

– 4 copies of the left 128-bit half into 4 different 128-bit words;
– a rotation of each of these four 128-bit words by the following number of bits: $\phi_{i-1}, 32 + \phi_{i-1}, 64 + \phi_{i-1}, 96 + \phi_{i-1}$; where we set $\phi_{-1} = 16$ and where $\phi_i$ for $i \geq 0$ is the usual Fibonacci sequence;
– a AND of each of these four 128-bit words by 128-bit values derived from the expression of $L_j^i$;
– a XOR of each of these four 128-bit words into the right half of the internal state; and
– a swap of the left and right words.

In total, we need 40 copies, 40 rotations, 40 XORs, 40 ANDs, and 10 swaps where all operations are on 128-bit words. An example of $256 \times 256$ binary matrix which can be implemented in this fashion is provided in Appendix B.

## 4 Experimental Verification

This section covers the experimental verifications of both suggested optimizations.

### 4.1 Reduced Linear Layer

We start with the evaluation of reduced linear layer introduced in Section 3.1. We implemented the proposed optimizations on top of the Picnic implementations available on GitHub[6] and based the matrix pre-computations on the LowMC reference implementation[7].

We performed the benchmarks on an Intel Core i7-4790 running Ubuntu 17.04 and Raspberry Pi 3 Model B running openSUSE Leap 42.2. All measurements were repeated 1000 times and averaged and are presented both as milliseconds and number of cycles. We only benchmarked the Fiat-Shamir transformed variant of Picnic, since the Unruh transformed variant does not perform any additional rounds of the proof system. Hence improvements to LowMC encryption apply to Picnic-FS and Picnic-UR in the same way. For the benchmarks we use the instances listed in Table 2 and denote as Picnic-$n$ the instance for blocksize $n$ benchmarked for Picnic-FS.

| **Blocksize** | **S-boxes** | **Keysize** | **Rounds** |
|---|---|---|---|
| $n$ | $m$ | $k$ | $r$ |
| 128 | 10 | 128 | 20 |
| 192 | 10 | 192 | 30 |
| 256 | 10 | 256 | 38 |

Table 2: Parameters for LowMC targeting various security levels ($n/2$) as used by Picnic [CDG+17b]. All parameters are computed for data complexity $d = 1$.

| | with RLL | | without RLL | | Perf. gain | |
|---|---|---|---|---|---|---|
| Parameters | Sign | Verify | Sign | Verify | Sign | Verify |
| Picnic-128 | 1.95 | 1.37 | 2.16 | 1.49 | 10% | 8% |
| (cycles) | 7021464.80 | 4914566.23 | 7791425.37 | 5357557.45 | | |
| Picnic-192 | 6.84 | 4.78 | 10.51 | 7.45 | 35% | 36% |
| (cycles) | 24609965.13 | 17224381.00 | 37846564.92 | 26821911.32 | | |
| Picnic-256 | 14.13 | 9.71 | 25.23 | 17.47 | 44% | 44% |
| (cycles) | 50872097.81 | 34969117.91 | 90827770.05 | 62885085.88 | | |

Table 3: Benchmarks without and with RLL on Intel Core i7.

[6] https://github.com/IAIK/Picnic
[7] https://github.com/LowMC/lowmc

From Table 3 we can observe a small 10 % improvement for the 128 bit case and up to 40 % for signing with the larger instances. The results on the Raspberry Pi 3 are presented in Table 4 and show an even larger improvement of 40 to 50 % for signing and verifying with all instances. We also note that the numbers without reduced linear layers also show improvements over the numbers published in [CDG+17a]. Those improvements are obtained via general memory usage optimizations based on better data locality and better cache usage.

| | with RLL | | without RLL | | Perf. gain | |
|---|---|---|---|---|---|---|
| Parameters | Sign | Verify | Sign | Verify | Sign | Verify |
| Picnic-128 | 10.36 | 7.12 | 17.08 | 11.61 | 39% | 39% |
| (cycles) | 37295940.40 | 25615307.22 | 61497049.90 | 41804895.48 | | |
| Picnic-192 | 39.94 | 26.89 | 72.05 | 48.06 | 45% | 44% |
| (cycles) | 143772804.12 | 96819499.16 | 259362245.44 | 172999619.37 | | |
| Picnic-256 | 84.27 | 56.45 | 172.86 | 115.42 | 51% | 51% |
| (cycles) | 303377247.02 | 203205361.38 | 622302669.39 | 415507521.20 | | |

Table 4: Benchmarks without and with RLL on Raspberry Pi 3.

To give one concrete example for the memory savings discussed in Section 3.1, we calculate the memory requirements for parameter set Picnic-256. For the round key calculations the general LowMC algorithm requires $38 + 1$ matrices of dimension $256 \times 256$, which correspond to 312 KB. LowMC with reduced linear layer only uses one $256 \times 256$, and one $1140 \times 256$ matrix as well as one 1140-bit vector, which yields a memory consumption of $43,625$ KB. Hence, in the parameter set Picnic-256 we achieve a reduction of the memory cost for the round key by more than 85%. Taking the linear layer and the round constants into account we still save more than 43% of the memory. However, in the practical Picnic implementation the memory saving is even higher and reaches around 55%. This effect occurs because every matrix stores additional meta information about the matrix itself and padding and as the number of matrices is greatly reduced in LowMC with RLL, also a lot of meta information and padding can be ommited.

Therefore, we can conclude that our alternative description provides both a significant performance boost and also saves a large amount of memory required to represent LowMC.

### 4.2 Fibonacci Feistel Network

We also implemented the proposed Feistel network from Section 3.2 and benchmarked it against an implementation of a constant-time matrix multiplication available in the Picnic project on GitHub. All measurements were repeated 1000000 times and averaged and are presented as number of cycles.

Performance and memory consumption of a 256-bit blocksize Feistel network and different branch sizes are shown in Table 5. The constant-time implementa-

tion has a performance of 2302 cycles for a $256 \times 256$ matrix-vector multiplication. As Table 5 shows, the Feistel network performs 20% better than the constant-time implementation if 16 bit branches are used. However, using 64-bit branches increases the performance gain to around 60 %.
The constant-time algorithm has a memory consumption of 8192 bytes because the whole $256 \times 256$ matrix has to be stored. The Feistel network for 64 bit branches only uses $\frac{1}{4}$ of this memory because the $256 \times 256$ matrix is instead represented by smaller $64 \times 64$ matrices.

| **Branch size** | **Rounds** | **Cycles** | **Memory (Bytes)** |
|---|---|---|---|
| 4 | 10 | 4745 | 640 |
| 8 | 8 | 2965 | 1024 |
| 16 | 6 | 1897 | 1536 |
| 32 | 4 | 2203 | 2048 |
| 64 | 2 | 837 | 2048 |

Table 5: Benchmarking of Feistel network with different branch sizes on Intel Core i7.

## 5 Discussion

The results we presented in Section 4 let us presume that both optimizations yield even better results for larger blocksizes, e.g. larger instances as used by LowMCHash-256 [AGR+16]. Note that for the reduced linear layer the number of S-boxes is essential for the performance gain from using a reduced linear layer. If the number of S-boxes stays the same but the blocksize is increased, removing the multiplication of the round key matrix with the master key has a higher impact the larger the blocksize is. If $3m$ is almost as large as $n$, then we expect no performance gain because the matrix $C_N$ is almost as big as all round key matrices and the multiplication with the master key is then not much faster than multiplying each round key separately.

Also the proposed Feistel network will result in even better performance gains if the blocksize is increased. Table 6 shows that the performance gain for blocksizes. While the cost for the constant-time multiplication quadruples if the blocksize doubles, the Feisel network less than triples the cost for large blocksizes. Also, the larger the blocksize the slower the cost grows with the Fibonacci structure.

From an implementation point of view the branch size for very large blocksizes, e.g. 1024 bit or morge, can be extended to 128-bit, 256-bit or even 512 bit using the SSE2 and AVX2 and AVX512 instruction sets. This can be an interesting approach for very large blocksizes, because the number of branches and rounds can be reduced making the algorithm faster.

| Blocksize | Rounds | Branches | XOR operations in Feistel network | XOR operations with constant-time multiplication |
|---|---|---|---|---|
| 256 | 2 | 2 | 256 | 1024 |
| 512 | 4 | 4 | 1024 | 4096 |
| 1024 | 6 | 8 | 3072 | 16384 |
| 2048 | 8 | 16 | 8192 | 65536 |

Table 6: Development of the FFN performance for larger blocksizes.

Besides the runtime improvements, reducing the linear layer might also have consequences on the cryptoanalysis of LowMC. As this optimization leads to an equivalent description of LowMC, only smaller round keys are added in each round while keeping the number of rounds the same. The cryptoanalysis of LowMC based on Fibanocci Feistel networks might also need to be adapted, as the Feistel network is only able to represent some regular matrices, but not all as sampled by LowMC in the instance generation.

**Acknowledgments.** We thank Tyge Tiessen for interesting ideas and discussions on optimizing LowMC's linear layer. S. Ramacher, and C. Rechberger have been supported by H2020 project Prismacloud, grant agreement n°644962. C. Rechberger has additionally been supported by EU H2020 project PQCRYPTO, grant agreement n°645622.

## References

ADKF70. V. Arlazarov, E. Dinic, M. Kronrod, and I. Faradzev. On economical construction of the transitive closure of a directed graph. *Soviet Math Dokl.*, 1970.

AGR+16. Martin R. Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In *ASIACRYPT*, pages 191–219, 2016.

AL00. Kazumaro Aoki and Helger Lipmaa. Fast implementations of AES candidates. In *AES Candidate Conference*, pages 106–120, 2000.

ARS+15. Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In *EUROCRYPT*, 2015.

ARS+16. Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. *IACR Cryptology ePrint Archive*, 2016:687, 2016.

Bar06. Gregory V. Bard. Accelerating cryptanalysis with the method of four russians. *IACR Cryptology ePrint Archive*, 2006:251, 2006.

BB02. Elad Barkan and Eli Biham. *In How Many Ways Can You Write Rijndael?*, pages 160–175. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

BBF+02. Guido Bertoni, Luca Breveglieri, Pasqualina Fragneto, Marco Macchetti, and Stefano Marchesin. Efficient software implementation of AES on 32-bit platforms. In *CHES*, volume 2523, pages 159–171, 2002.

Ber09. Daniel J. Bernstein. Optimizing linear maps modulo 2. 2009. https://binary.cr.yp.to/linearmod2.html.

BS08. Daniel J. Bernstein and Peter Schwabe. New AES software speed records. In *INDOCRYPT*, volume 5365 of *Lecture Notes in Computer Science*, pages 322–336. Springer, 2008.

CCF+16. Anne Canteaut, Sergiu Carpov, Caroline Fontaine, Tancrède Lepoint, María Naya-Plasencia, Pascal Paillier, and Renaud Sirdey. Stream ciphers: A practical solution for efficient homomorphic-ciphertext compression. In *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, pages 313–333, 2016.

CDG+17a. Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In *CCS*, pages 1825–1842. ACM, 2017.

CDG+17b. Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. The Picnic Signature Algorithm Specification, 2017. https://github.com/Microsoft/Picnic/blob/master/spec.pdf.

DPVR00. Joan Daemen, Michaël Peeters, Gilles Van Assche, and Vincent Rijmen. Nessie Proposal: NOEKEON, 2000. http://gro.noekeon.org/Noekeon-spec.pdf.

DSES14. Yarkin Doröz, Aria Shahverdi, Thomas Eisenbarth, and Berk Sunar. Toward practical homomorphic evaluation of block ciphers using prince. In *FC Workshops BITCOIN and WAHC*, pages 208–220, 2014.

GLSV14. Vincent Grosso, Gaëtan Leurent, François-Xavier Standaert, and Kerem Varici. Ls-designs: Bitslice encryption for efficient masked software implementations. In *FSE*, pages 18–37, 2014.

GMO16. Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. Zkboo: Faster zero-knowledge for boolean circuits. In *USENIX*, pages 1069–1083, 2016.

GRR+16. Lorenzo Grassi, Christian Rechberger, Dragos Rotaru, Peter Scholl, and Nigel P. Smart. Mpc-friendly symmetric key primitives. In *CCS*, pages 430–443, 2016.

IKOS07. Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In *STOC*, pages 21–30, 2007.

MJSC16. Pierrick Méaux, Anthony Journault, François-Xavier Standaert, and Claude Carlet. Towards stream ciphers for efficient FHE with low-noise ciphertexts. In *EUROCRYPT*, pages 311–343, 2016.

NLV11. Michael Naehrig, Kristin E. Lauter, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In *CCSW*, pages 113–124, 2011.

NPV17. Valérie Nachef, Jacques Patarin, and Emmanuel Volte. *Feistel Ciphers - Security Proofs and Cryptanalysis*. Springer, 2017.

RSS17. Dragos Rotaru, Nigel P. Smart, and Martijn Stam. Modes of operation suitable for computing on encrypted data. *IACR Trans. Symmetric Cryptol.*, 2017(3):294–324, 2017.

SM10. Tomoyasu Suzaki and Kazuhiko Minematsu. Improving the generalized feistel. In *FSE*, volume 6147, 2010.

SME16. Shady Mohamed Soliman, Baher Magdy, and Mohamed A. Abd El-Ghany. Efficient implementation of the AES algorithm for security applications. In *SoCC*, pages 206–210. IEEE, 2016.

SMMK12. Tomoyasu Suzaki, Kazuhiko Minematsu, Sumio Morioka, and Eita Kobayashi. TWINE: A lightweight block cipher for multiple platforms. In Lars R. Knudsen and Huapeng Wu, editors, *Selected Areas in Cryptography*, volume 7707 of *Lecture Notes in Computer Science*, pages 339–354. Springer, 2012.

## A An Example of Fibonacci-Feistel Network

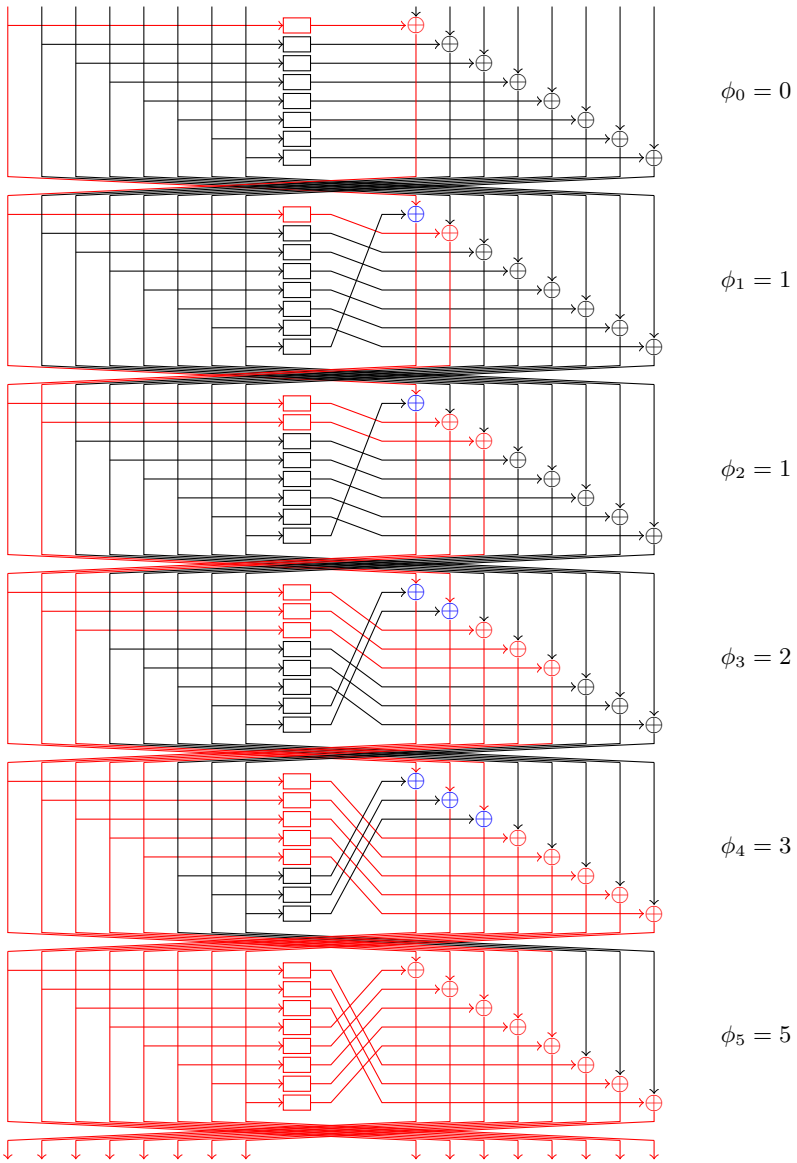


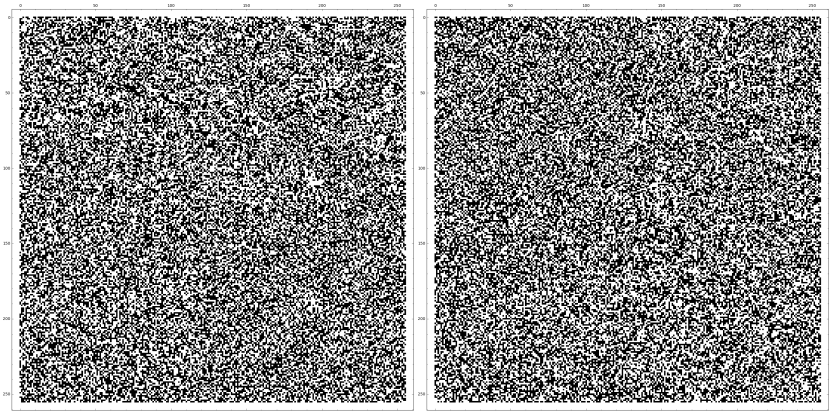

Fig. 3: 6 rounds of the FFN structure for $b = 8$. The rectangles correspond to distinct L-Box calls. A branch is red if its value depends on the left-most word of the input.

## B Example of Binary Matrices Corresponding to a FFN

The matrix presented in Figure 4 has the following properties:

- It has full rank.
- $h = 33585 \approx 0.51 \times 2^{16}$ ofs its coefficients are equal to 1.
- It can be evaluated using a 10-round FFN with 32 independent and random 4-bit linear permutations used as L-boxes in each round. A new L-box layer is used for each round.

Its inverse has similar properties and is also depicted in Figure 4.



(a) The matrix $M$. (b) The inverse matrix $M^{-1}$.

Fig. 4: A matrix $M$ and its inverse $M^{-1}$ corresponding to a 10-round 256-bit FFN with $b = 32, w = 4$. Black means 1, white means 0.