Linear Equivalence of Block Ciphers with Partial Non-Linear Layers: Application to LowMC

Itai Dinur

Department of Computer Science, Ben-Gurion University, Israel

Abstract. LowMC is a block cipher family that is optimized for practical instantiations of multi-party computation, fully homomorphic encryption, and zero-knowledge proofs. It was designed in 2015 by Albrecht et al., and has recently become a substantial building block in several novel post-quantum cryptosystems. A large portion of LowMC instances use a relatively recent design strategy (initiated by Gèrard et al. at CHES 2013) of applying the non-linear layer to only a part of the state in each round, where the shortage of non-linear operations is partially compensated by heavy linear algebra. Since the high linear algebra complexity has been a bottleneck in several applications, one of the open questions raised by the designers was to reduce it, without introducing additional non-linear operations (or compromising security).

In this paper, we consider LowMC instances with block size n, partial non-linear layers of size $s \leq n$ and r encryption rounds. We show that when $s < n$, each LowMC instance belongs to a large class of equivalent instances. We then select a *representative instance* from this class for which encryption (and decryption) can be implemented much more efficiently than for an arbitrary instance. This yields a new encryption algorithm that is equivalent to the standard one, but reduces the evaluation time and storage of the linear layers from $r \cdot n^2$ bits to about $r \cdot n^2 - (r-1)(n-s)^2$, which is a substantial improvement for small s and a reasonable choice of r. For standard LowMC parameters, our new encryption algorithm achieves a reduction by a factor between 2 and 4, while for more extreme parameter choices (suggested by the designers) the reduction is by a factor of more than 140. Furthermore, our new encryption algorithm is applicable to all SP-networks with partial non-linear layers.

An additional unique feature of LowMC is that the linear layers of its instances are sampled at random. In the second part of the paper, we show how to reduce the sampling time and randomness complexities (i.e., the number of random bits used) by directly sampling representative instances. Finally, we formalize the notion of linear equivalence of block ciphers with partial non-linear layers and prove that the memory complexity of our encryption algorithm and the randomness complexity of our sampling algorithm are optimal.

Keywords: Block cipher, LowMC, Picnic signature algorithm, linear equivalence

1 Introduction

LowMC is a block cipher family designed by Albrecht et al. and presented at Eurocrypt 2015 [2]. The cipher is heavily optimized for practical instantiations of multi-party computation (MPC), fully homomorphic encryption (FHE), and zero-knowledge proofs. In such applications, non-linear operations incur a higher penalty in communication and computational complexity compared to linear ones. Due to its design strategy, LowMC is a popular building block in post-quantum designs that are based on MPC and zero-knowledge protocols (cf. [6, 8, 9, 12]). Most notably, it is used in the Picnic signature algorithm [7] which is a candidate in NIST's post-quantum cryptography standardization project [14].

Instances of LowMC are designed to perform well in two particular metrics that measure the complexity of non-linear operations over $GF(2)$. The first metric is multiplicative complexity (MC), which simply counts the number of multiplications (AND gates in our context) in the circuit. The second metric is the multiplicative (AND) depth of the circuit.

The relevance of each metric depends on the specific application. For example, in the context of MPC protocols, Yao's garbled circuits [21] with the free XOR technique [13] (and many of their variants) have a constant number of communication rounds. The total amount of communication depends on the MC of the circuit as each AND gate requires communication, whereas XOR operations can be performed locally. In an additional class of MPC protocols (e.g., GMW [11]), the number of communication rounds is linear in the ANDdepth of the evaluated circuit. The performance of these protocols depends on both the MC and ANDdepth of the circuit. For more details about the relevance of the metrics in various applications, refer to the LowMC paper [2].

In order to reduce the complexity of non-linear operations for a certain level of security, LowMC combines very dense linear layers over $GF(2)^n$ (where n is the block size) with simple non-linear layers containing 3×3 Sboxes of algebraic degree 2. The LowMC block cipher family includes a huge number of instances, where for each instance, the linear layer of each round is chosen independently and uniformly at random from all invertible $n \times n$ matrices. Additionally, the round keys are computed using linear transformations applied to the master key that are selected uniformly at random for each instance.

The design strategy of LowMC attempts to offer flexibility with respect to both the MC and ANDdepth metrics. In particular, some LowMC instances minimize the MC metric by applying only a *partial non-linear layer* to the state of the cipher at each round, while the linear layers still mix the entire state. In general, this approach requires to increase the total number of rounds in the scheme in order to maintain a certain security level, but this is compensated by the reduction in the size of the non-linear layers and the total AND count is generally reduced. The global parameters of LowMC that are most relevant for this paper are (1) the block size of n bits, (2) the number of rounds r (which is determined according to the desired security level), and (3) a parameter s which denotes the domain length of each non-linear layer, namely, the number of bits on which it operates (which may be smaller than n).[1]

While LowMC's design aims to minimize the non-linear complexity of the scheme at the expense of using many linear algebra (XOR) operations, in several practical applications, XORs do not come for free and may become a bottleneck in the implementation. This phenomena was already noted and demonstrated in the original LowMC paper. More recently, it influenced the design of the Picnic signature algorithm, where the most relevant metric is the MC (the number of AND gates) that affects the signature size. In order to minimize the AND count (and the signature size), the LowMC instances used by Picnic should have a very small partial non-linear layer in each round (perhaps using only a single 3×3 Sbox). However, such an instance has a large number of rounds r and each encryption requires computation of r matrix-vector products that increase the signing and verification times. Consequently, the Picnic designers settled for non-linear layers of intermediate size in order to balance the signature size on one hand and the signing and verification times on the other.

Due to the large computational cost of LowMC's dense linear layers, one of the open problems raised by the designers was to reduce their computational cost. A first step in this direction was recently taken by Perrin et al. in [16]. The work of [16] showed that in case LowMC uses partial non-linear layers (i.e., $s < n$), the round key and round constant of each round (except for the initial key addition) can be trimmed from a size of n bits to a size of only s bits.

When considering the original LowMC design, the effect of the alternative description of [16] is relatively small, as in the original design, round keys and constants can be precomputed and hard-coded into the LowMC instance. Their total size is about $n(r + 1)$ bits which is negligible compared to the size of the linear layers which is $r \cdot n^2$ bits. On the other hand, the computation of LowMC inside the Picnic signature algorithm involves splitting the LowMC instance to 3 related instances which are evaluated with a fresh share of the key in each invocation. Therefore, it is not possible to hard-code the round keys into the LowMC instance in this specific application and the effect of the alternative description of [16] becomes more significant. Yet, even in the LowMC application in Picnic, the $r \cdot n^2$ complexity of evaluating linear layers remains the main bottleneck and it is clear that in order to achieve a significant gain in performance, this complexity has to be reduced. For this purpose, [16] also proposed to replace LowMC's randomly sampled linear matrices with a linear construction (called Fibonacci Feistel Network, or FFN) that has much more structure and can be evaluated and stored more efficiently compared to a random matrix. On the other hand, this construction modifies the design of LowMC and it is not clear whether it preserves the same level of security as the original.

Our Contribution In this paper we revisit the open problem of the LowMC designers to reduce the complexity of its linear operations, focusing on instances

[1] The LowMC specification denotes by m the number of 3×3 Sboxes in each non-linear layer and therefore $s = 3m$ in our context.

with partial non-linear layers (i.e., $s < n$). We consider a generalized LowMC construction in which the linear layers are selected uniformly at random from the set of all invertible matrices and the non-linear layers are arbitrary and applied to s bits of the internal state in each round. For such a cipher, the standard encryption algorithm requires matrices of size[2] $r \cdot n^2$ bits and performs matrix-vector products with about the same complexity. We describe a new encryption (and decryption) algorithm for any such a generalized LowMC cipher which is equivalent to the standard one, but uses matrices that require only $r \cdot n^2 - (r-1)(n-s)^2$ bits of storage and about the same linear algebra time complexity (using standard matrix-vector products[3]). Surprisingly, although the open problem of the LowMC designers involved redesigning LowMC's linear layers to reduce its linear algebra complexity (and this direction was pursued in [16]), our new encryption algorithm achieves this without changing the original design.

Table 1. Size of Linear Layers of LowMC Instances in Bits

n	s	r	Previous	New	Multiplicative Gain
			$r \cdot n^2$	$r \cdot n^2 - (r-1)(n-s)^2$	Previous/New
128	30	20	$2^{18.3}$	$2^{17.1}$	2.25
192	30	30	$2^{20.1}$	$2^{18.4}$	3.2
256	30	38	$2^{21.2}$	$2^{19.2}$	4.1
1024	3	901	$2^{29.8}$	$2^{22.6}$	143.8

Table 1 compares[4] the size of LowMC's linear layers in previous implementations to our new encryption algorithm for several instances. Generally, our advantage compared to the standard LowMC implementations grows as s is reduced compared to n. The first three instances are the ones used by the Picnic signature algorithm and for them we obtain a multiplicative gain of between 2.25 and 4.1. Of course, our optimization applies to all LowMC instances and not only to the ones used in Picnic. For example, the fourth instance in Table 1 is

[2] We note LowMC expands a short seed using a pseudo-random generator whose output is used to compute the matrices and generate actual instances. However, as we argue in the paper, such a short implicit description is not useful during the encryption process which needs an explicit description of the matrices to encrypt efficiently.

[3] Optimizations in matrix-vector multiplications (such as efficient implementations of the "method of the four Russians" [1]) can be applied to both the standard and to our new encryption algorithm. However, as noted in [16], such optimizations typically have a limited effect on modern computers that use SIMD instruction sets.

[4] The table does not list other parameters such as the key size and the allowed data complexity. These parameters are obviously important in general, but are not relevant for this paper.

suggested in [19], and for this extreme choice of parameters, we reduce the size of the linear layers by a factor of about 144. Note that encryption using the fourth instance is arguably impractical as it require more than 100 megabytes just to store. Our algorithm requires less than one megabyte of storage and achieves a proportional speedup in encryption time. In general, it renders as practical previously impractical LowMC instances.

Prior to this work, reducing s (in order to optimize the MC metric) while increasing r (in order to maintain the same security level for a LowMC instance) increased the linear algebra complexity proportionally to the increase in the number of rounds. One of the main consequences of this work is that such a reduction in s may actually *reduce* the linear algebra complexity[5] of the cipher since the increase in the number of rounds is compensated by the decrease in the complexity of linear algebra in each round. As a result, the value of s in the LowMC instances used by Picnic should be reconsidered, and this work may simultaneously allow a reduction in the signature size and the signing and verification times. Consequently, the gains mentioned in Table 1 are likely to become larger and the signature size will be reduced as well.

Since the new algorithm allows to reduce both the implementation size and complexity of encryption (and decryption), it is also useful for adversaries that attempt to break LowMC instances via exhaustive search. We further mention that the encryption algorithm is applicable to any SP-network with partial non-linear layers (such as[6] Zorro [10]) since it does not assume any special property of the linear or non-linear layers. Yet, if the linear layers are not selected uniformly at random, the question of whether our algorithm is more efficient compared to the standard one depends on the specific design.

After optimizing the complexity of encryption, we consider the complexity of generating a generalized LowMC instance by sampling its linear layers. We devise a new sampling algorithm that reduces this complexity[7] from about $r \cdot n^3$ to $n^3 + (r-1) \cdot s^2 \cdot n = r \cdot n^3 - (r-1) \cdot n \cdot (n^2 - s^2)$, which is a significant improvement for small s and sufficiently large r. Our sampling algorithm further reduces the number of uniform (pseudo) random bits required to sample the linear layers from about $r \cdot n^2$ to $r \cdot n^2 - (r-1)(n-s)^2$. These optimizations are useful in applications that require frequent LowMC instance generation.

The gain in the efficiency for many LowMC instances raises the question of whether the description of the linear layers we use is optimal (i.e., minimal) or can be further compressed. Indeed, it may seem that the formula $r \cdot n^2 - (r-1)(n-s)^2 = n^2 + (r-1)(n^2 - (n-s)^2)$ obtained for the description size is suboptimal, and the formula $n^2 + (r-1) \cdot s \cdot n^2$ is more reasonable, as it is linear in s (similarly to the reduction of the length of round keys, obtained in [16]).

[5] The extent of reduction depends on the LowMC instance.

[6] Although Zorro is broken [3, 18, 20], its general design strategy remains valid

[7] Using asymptotically fast matrix multiplication and invertible matrix sampling algorithms will reduce the asymptotic complexity of both the original and our new algorithm. Nevertheless, it is not clear whether they would reduce their concrete complexity for relevant choices of parameters.

We address this question in the last part of the paper, where we prove (under two assumptions which we argue are natural) that no further optimizations that reduce the LowMC code size are possible without changing its design.

Our Techniques In order to derive our new encryption algorithm, we show that each (generalized) LowMC instance belongs to a class of equivalent instances which is very large size when $s \ll n$. We then select a representative member of the equivalence class that can be implemented efficiently using linear algebra optimizations which apply matrices with a special structure instead of random matrices (yet the full cipher remains equivalent). Next, we show how to sample such a representative more efficiently than a random member in its equivalence class. Our new sampling algorithm breaks dependencies among different parts of the linear layers in a generalized LowMC cipher, shedding light on its internal structure.

Finally, we formalize the notion of linear equivalence among generalized LowMC ciphers. This allows us to prove (based on two natural assumptions) that we correctly identified the linear equivalence classes and hence our description of the linear layers is optimal in size and we use the minimal amount of randomness to sample it. The formalization requires some care and the proof of optimality is somewhat non-standard (indeed, the claim that we prove is non-standard).

Related Work Previous works [4, 5] investigated equivalent representations of AES and other block ciphers obtained by utilizing the specific structure of their Sboxes (exploiting a property called self-affine equivalence [5]). On the other hand, our equivalent representation and encryption algorithm is independent of the non-linear layer and can be applied regardless of its specification. Yet we only deal with block ciphers with partial non-linear layers in this paper.

As mentioned above, Perrin et al. [16] obtained an alternative description of LowMC, whose first step was to exchange the order of the key and constant additions with the application of the linear layer in each round. The observation of [16] was that in case $s < n$, after reordering, the constant and key additions of consecutive rounds can be merged through the $n - s$ bits of the state that do not go through the non-linear transformation. Applying this process recursively effectively eliminates all the key and constant additions on $n - s$ bits of the state (except for the initial key addition).

Our work is related to [16], but we mention that exchanging the order of the key and constant additions is a common technique in symmetric cryptography (e.g., it is frequently used in cryptanalysis of round-reduced AES). Thus, the main novelty in [16] is in the recursive application of this reordering. On the other hand, we are not aware of previous work in the context of symmetric cryptography that obtained equivalent representations by optimizing the linear layers themselves (independently of the Sboxes). Moreover, our algorithm significantly enhances the performance of LowMC for $s \ll n$ in general (and not only when used in Picnic).

Paper Organization The rest of the paper is organized as follows. We describe some preliminaries in Section 2 and prove the basic linear algebra properties we use in Section 3. In Section 4 we describe our optimized encryption algorithm and describe our optimized algorithm for sampling the linear layers in Section 5. Finally, we prove the optimality of our description of the linear layers in Section 6.

2 Preliminaries

2.1 Notation

Given a string of bits $x \in \{0,1\}^n$, denote by $x[|d]$ its d most significant bits (MSBs) and by $x[d|]$ its d least significant bits (LSBs). Given strings x, y, denote by $x\|y$ their concatenation. Given a matrix A, denote by $A[*, i]$ its i'th column, by $A[*, d|]$ its first d columns and by $A[*, |d]$ its last d columns. Given two matrices $A \in GF(2)^{d_1 \times d_2}$ and $B \in GF(2)^{d_1 \times d_3}$ denote by $A\|B \in GF(2)^{d_1 \times (d_2+d_3)}$ their concatenation. Denote by I_d the identity matrix with dimensions $d \times d$.

Throughout this paper, addition $x + y$ between bit strings $x, y \in \{0,1\}^n$ is performed bit-wise over $GF(2)^n$ (i.e., by XORing them).

2.2 Generalized LowMC Ciphers

We study generalized LowMC (GLMC) ciphers where the block size is n bits and each non-linear layer operates on $s \leq n$ bits of the state.

Each instance is characterized by a number of rounds r, rounds keys K_i for $i \in \{0, \dots, r\}$ and round constants C_i, for $i \in \{1, \dots, r\}$. The cipher consists of r (partial) invertible non-linear layers $S_i : \{0,1\}^s \to \{0,1\}^s$ and r invertible linear layers $L_i \in GF(2)^{n \times n}$ for $i \in \{1, \dots, r\}$.

A GLMC instance is generated by choosing each L_i independently and uniformly at random among all invertible $n \times n$ matrices.[8] However, we note that the main encryption algorithm we devise in Section 4 is applicable regardless of the way that the linear layers are chosen. We do not place any restriction on how the round keys and constants are generated. Moreover, we do not restrict the invertible non-linear layers.

The encryption procedure manipulates n-bit words that represent GLMC states, while breaking them down according to their s LSBs (which we call "part 0 of the state") and $n - s$ MSBs (which we call "part 1 of the state"). To simplify our notation, given any n-bit string x, we denote $x^{(0)} = x[s|]$ and $x^{(1)} = x[|n - s]$.

The basic GLMC encryption procedure is given in Algorithm 1. Decryption is performed by applying the inverse operations to a ciphertext.

[8] Alternatively, they can be selected in a pseudo-random way from a short seed, as in LowMC.

```
Input: $x_0$
Output: $x_{r+1}$
begin
    $x_1 \leftarrow x_0 + K_0 + C_0$
    for $i \in \{1, 2, \ldots, r\}$ do
        $y_i \leftarrow S_i(x_i^{(0)}) \| x_i^{(1)}$
        $t_{i+1} \leftarrow L_i(y_i)$
        $x_{i+1} \leftarrow x_i + K_i + C_i$
    end
    Output $x_{r+1}$
end
```

Algorithm 1: Basic Encryption

```
Input: $x_0$
Output: $x_{r+1}$
begin
    $x_1 \leftarrow x_0 + K_0$
    for $i \in \{1, 2, \ldots, r\}$ do
        $y_i \leftarrow S_i(x_i^{(0)}) \| x_i^{(1)}$
        $x_{i+1} \leftarrow L_i(y_i)$
    end
    Output $x_{r+1}$
end
```

Algorithm 2: Simplified Encryption

Simplification We follow the simplification of the cipher proposed in [16]. Consider round i of Algorithm 1 and note that the order of the application of L_i and the addition of $K_i + C_i$ can be exchanged by defining $K'_i = L_i^{-1}(K_i + C_i)$ and obtaining $L_i(y_i + K'_i) = L_i(y_i) + L_i(K'_i) = L_i(y_i) + K_i + C_i = x_{i+1}$ as required. Next, as S_i is only applied to $x_i^{(0)}$, the addition of $K_i'^{(1)}$ can be pushed back to the beginning of the round. The modified round now consists of the operations

$$\begin{aligned} x_i^{(1)} &\leftarrow x_i^{(1)} + K_i'^{(1)} \\ y_i &\leftarrow S_i(x_i^{(0)}) \| x_i^{(1)} \\ y_i^{(0)} &\leftarrow y_i^{(0)} + K_i'^{(0)} \\ x_{i+1} &\leftarrow L_i(y_i). \end{aligned}$$

Next, the initial addition with $K_i'^{(1)}$ is consumed by round $i-1$ by defining $K_{i-1}'^{(1)} = K_{i-1}^{(1)} + C_{i-1}^{(1)} + K_i'^{(1)}$ (while $K_{i-1}'^{(0)} = K_{i-1}^{(0)} + C_{i-1}^{(0)}$ remains unchanged). Overall, the simplification eliminates the round key (and constant) addition into part 1 of the state in round i. This process can be applied recursively starting from round r down to the first round. As a result, round keys and round constants for $i \in \{1, \ldots, r\}$ are chopped down to a size of s bits and added directly after the non-linear layer. This requires changing the initial round key K_0 accordingly. We make a final modification and combine the (chopped down) round key and constant addition of each round $i \in \{1, \ldots, r\}$ with the non-linear layer S_i by defining $S'_i(x_i^{(0)}) = S_i(x_i^{(0)}) + K_i'^{(0)}$. The non-linear layers are now key-dependent, but this does not make any difference in the context of this paper and simplifies the presentation.

Algorithm 2 describes the simplified encryption procedure. Obviously, the transformation of Algorithm 1 into Algorithm 2, requires recalculating K_0 and the non-linear layers S_i, yet we use the same notation for them for simplicity.

2.3 Breaking Down the Linear Layers

Given L_i (which is an $n \times n$ matrix), we partition its n-bit input into the first s LSBs (part 0 of the state that is output by S_{i-1}) and the remaining $n-s$ bits (part 1 of the state). Similarly, we partition its n-bit output into the first s LSBs (that are inputs of S_{i+1}) and the remaining $n-s$ bits. We define 4 sub-matrices of L_i that map between the 4 possible pairs of state parts:

$$L_i^{00} \in GF(2)^{s \times s}, L_i^{01} \in GF(2)^{s \times (n-s)}, L_i^{10} \in GF(2)^{(n-s) \times s}, L_i^{11} \in GF(2)^{(n-s) \times (n-s)}.$$

Thus, in our notation L_i^{ab} for $a, b \in \{0, 1\}$ maps the part of the state denoted by b to the part of the state denoted by a.

$$L_i = \begin{array}{c} \left[\begin{array}{c|c} L_i^{00} & L_i^{01} \\ \hline \underbrace{L_i^{10}}_{s} & \underbrace{L_i^{11}}_{n-s} \end{array}\right] \end{array} \begin{array}{l} \} \, s \\ \} \, n-s \end{array}$$

We extend our notation L_i^{ab} by allowing $a, b \in \{0, 1, *\}$, where the symbol '$*$' denotes the full state. Therefore,

$$L_i^{0*} \in GF(2)^{s \times n}, L_i^{1*} \in GF(2)^{(n-s) \times n}, L_i^{*0} \in GF(2)^{n \times s}, L_i^{*1} \in GF(2)^{n \times (n-s)},$$

are linear transformations which are sub-matrices of L_i, as shown below.

$$L_i = \left[\begin{array}{c} L_i^{0*} \\ \hline \underbrace{L_i^{1*}}_{n} \end{array}\right] \begin{array}{l} \} \, s \\ \} \, n-s \end{array} \quad , L_i = \left[\begin{array}{c|c} \underbrace{L_i^{*0}}_{s} & \underbrace{L_i^{*1}}_{n-s} \end{array}\right] \} \, n$$

These linear transformation satisfy several basic equalities. For example, for each $y \in \{0, 1\}^n$:

$$L_i^{0*}(y) = L_i(y)^{(0)} = L_i^{00}(y^{(0)}) + L_i^{01}(y^{(1)}),$$
$$L_i^{1*}(y) = L_i(y)^{(1)} = L_i^{10}(y^{(0)}) + L_i^{11}(y^{(1)}).$$

2.4 General Matrix Notation

The superscript of L_i^{ab} has a double interpretation, as specifying both the dimensions of the matrix and its location in L_i. We will use this notation more generally to denote sub-matrices of some $n \times n$ matrix A, or simply to define a matrix with appropriate dimensions (e.g., $A^{01} \in GF(2)^{s \times (n-s)}$ may be defined without defining A and this should be clear from the context). Therefore, dimensions of the matrices in the rest of the paper will be explicitly specified in superscript as A^{ab}, where $a, b \in \{0, 1, *\}$ (we do not deal with matrices of other dimensions). In case the matrix A^{ab} is a sub-matrix of a larger matrix A, the superscript has a double interpretation as specifying both the dimensions of A^{ab} and its location in A. When no superscript is given, the relevant matrix is of dimensions $n \times n$. There will be two exceptions to this rule which will be specified separately.

2.5 Complexity Evaluation

In this paper we analyze the complexity of the linear layers of generalized LowMC schemes. We will be interested in the two natural measures of time complexity (measured by the number of bit operations) and memory complexities (measured by the number of stored bits) of a single encryption (or decryption) of an arbitrary plaintext (or ciphertext). The linear layers are naturally represented by matrices,[9] and thus evaluating a linear layer on a state is a simply a matrix-vector product. Since the time and memory complexities of evaluating and storing the linear layers are proportional in this paper, we will typically refer to both a the linear algebra complexity of the linear layers. For algorithms that generate GLMC instances, we will be interested in time complexity and in the number of random bits (or pseudo-random bits) that they use.

3 Linear Algebra Properties

In this section we describe the linear algebra properties that are relevant for the rest of this paper. We continue to use the notation of Section 2.3 and indicate the dimensions of some matrices using superscript notation.

3.1 Invertible Binary Matrices

Denote by α_n the probability that an $n \times n$ uniformly chosen binary matrix is invertible. We will use the following well-known fact:

Fact 1 *The probability that an $n \times n$ uniform binary matrix is invertible is $\alpha_n = \prod_{i=1}^{n}(1 - 1/2^i) > 0.2887$. More generally, for positive integers $d \leq n$, the probability that a $d \times n$ binary matrix, chosen uniformly at random, has full row rank of d is $\prod_{i=n-d+1}^{n}(1 - 1/2^i) = \big(\prod_{i=1}^{n}(1 - 1/2^i)\big)/\big(\prod_{i=1}^{n-d}(1 - 1/2^i)\big) = \alpha_n/\alpha_{n-d}$.*

We will be interested in invertibility of matrices of a special form, described in the following fact.

Fact 2 *An $n \times n$ binary matrix of the form*

$$\left[\begin{array}{c|c} A^{00} & A^{01} \\ \hline A^{10} & I_{n-s} \end{array}\right]$$

is invertible if and only if the $s \times s$ matrix $B^{00} = A^{00} + A^{01}A^{10}$ is invertible and its inverse is given by

$$\left[\begin{array}{c|c} (B^{00})^{-1} & -(B^{00})^{-1} \cdot A^{01} \\ \hline -A^{10} \cdot (B^{00})^{-1} & I_{n-s} - A^{10} \cdot (B^{00})^{-1} \cdot A^{01} \end{array}\right].$$

[9] This is indeed natural, as noted in Footnote 3.

Finally, we prove a simple proposition regarding random matrices.

Proposition 1. *Let $A \in GF(2)^{n\times n}$ be an invertible matrix chosen uniformly at random and let $B^{11} \in GF(2)^{(n-s)\times(n-s)}$ be an arbitrary invertible matrix (for $s \leq n$) that is independent from A. Then the matrix*

$$C = \left[\begin{array}{c|c} A^{00} & A^{01}\cdot B^{11} \\ \hline A^{10} & A^{11}\cdot B^{11} \end{array}\right]$$

is a uniform invertible matrix.

Proof. The proof follows from the bijection

$$C = \left[\begin{array}{c|c} A^{00} & A^{01}\cdot B^{11} \\ \hline A^{10} & A^{11}\cdot B^{11} \end{array}\right] = \left[\begin{array}{c|c} A^{00} & A^{01} \\ \hline A^{10} & A^{11} \end{array}\right] \cdot \left[\begin{array}{c|c} I_s & \mathbf{0}^{01} \\ \hline \mathbf{0}^{10} & B^{11} \end{array}\right]$$

between the uniform matrix A and C. This is indeed a bijection since B^{11} is invertible as the inverse is given by

$$\left[\begin{array}{c|c} I_s & \mathbf{0}^{01} \\ \hline \mathbf{0}^{10} & (B^{11})^{-1} \end{array}\right].$$

■

3.2 Normalized Matrices

Definition 1. *Let A^{1*} be a Boolean matrix with full row rank of $n-s$ (and therefore it has $n-s$ linearly independent columns). Let $COL(A)$ denote the first set of $n-s$ linearly independent columns of A^{1*} in a fixed lexicographic ordering of columns sets. Then, these columns form an $(n-s)\times(n-s)$ invertible matrix which is denoted by $\dot{A}$, while the remaining columns form an $(n-s)\times s$ matrix which is denoted by $\ddot{A}$. Moreover, denote $\hat{A} = \dot{A}^{-1}\cdot A^{1*} \in GF(2)^{(n-s)\times(n-s)}$ (in this matrix, the columns of $COL(A)$ form the identity matrix).*

Remark 1. The only exception to the rule of Section 2.4 has to do with Definition 1 (and later with the related Definition 2). In this paper, the decomposition of definition 1 is always applied to matrices $A^{1*} \in GF(2)^{(n-s)\times n}$ (in case A^{1*} is a sub-matrix of A, it contains the bottom $n-s$ rows of A). Hence the resultant matrices $\dot{A} \in GF(2)^{(n-s)\times(n-s)}$, $\ddot{A} \in GF(2)^{(n-s)\times s}$ and $\hat{A} = GF(2)^{(n-s)\times n}$ have fixed dimensions and do not need any superscript. On the other hand, we will use superscript notation to denote sub-matrices of these. For example $\hat{A}^{10} \in GF(2)^{(n-s)\times s}$ is a sub-matrix of $\hat{A}$, consisting of its first s columns.

It will be convenient to consider a lexicographic ordering in which the columns indices of A^{1*} are reversed, i.e., the first ordered set of $n-s$ columns is $\{n, n-1,\ldots,s+1\}$, the second is $\{n,n-1,\ldots,s+2,s\}$, etc. To demonstrate the above definition, assume that $COL(A) = \{n, n-1, \ldots, s+1\}$ is a consecutive set of linearly independent columns. Then, the matrix A^{1*} is shown below.

$$A^{1*} = \Big[\underbrace{\ddot{A}}_{s}\Big|\underbrace{\dot{A}}_{n-s}\Big] \quad \Big\} n-s$$

We can write $A = (\dot{A} \cdot \dot{A}^{-1}) \cdot A = \dot{A} \cdot (\dot{A}^{-1} \cdot A) = \dot{A} \cdot \hat{A}$, where

$$\hat{A} = \dot{A}^{-1} \cdot A^{1*} = \Big[\underbrace{\dot{A}^{-1} \cdot \ddot{A}}_{s}\Big|\underbrace{I_{n-s}}_{n-s}\Big] \quad \Big\} n-s. \tag{1}$$

Normalized Equivalence Classes Given an invertible matrix $A \in GF(2)^{n\times n}$, define

$$N(A) = \left[\begin{array}{c} A^{0*} \\ \hline \hat{A} \end{array}\right] = \left[\begin{array}{c} A^{0*} \\ \hline \dot{A}^{-1} \cdot A^{1*} \end{array}\right] = \left[\begin{array}{c|c} I_s & \mathbf{0}^{01} \\ \hline \mathbf{0}^{10} & \dot{A}^{-1} \end{array}\right] \cdot A.$$

The transformation $N(\cdot)$ partitions the set of invertible $n \times n$ boolean matrices into *normalized equivalence classes*, where A, B are in the same normalized equivalence class if $N(A) = N(B)$. We denote $A \leftrightarrow_N B$ the relation $N(A) = N(B)$.

Proposition 2. *Two invertible $n \times n$ boolean matrices A, B satisfy $A \leftrightarrow_N B$ if and only if there exists an invertible matrix C^{11} such that*

$$A = \left[\begin{array}{c|c} I_s & \mathbf{0}^{01} \\ \hline \mathbf{0}^{10} & C^{11} \end{array}\right] \cdot B.$$

Proof. If $A \leftrightarrow_N B$ then

$$\left[\begin{array}{c|c} I_s & \mathbf{0}^{01} \\ \hline \mathbf{0}^{10} & \dot{A}^{-1} \end{array}\right] \cdot A = \left[\begin{array}{c|c} I_s & \mathbf{0}^{01} \\ \hline \mathbf{0}^{10} & \dot{B}^{-1} \end{array}\right] \cdot B$$

or

$$A = \left[\begin{array}{c|c} I_s & \mathbf{0}^{01} \\ \hline \mathbf{0}^{10} & \dot{A} \cdot \dot{B}^{-1} \end{array}\right] \cdot B$$

For the other direction, if

$$A = \left[\begin{array}{c|c} I_s & \mathbf{0}^{01} \\ \hline \mathbf{0}^{10} & C^{11} \end{array}\right] \cdot B$$

then

$$\left[\begin{array}{c} A^{0*} \\ \hline A^{1*} \end{array}\right] = \left[\begin{array}{c} B^{0*} \\ \hline C^{11} \cdot B^{1*} \end{array}\right].$$

Observe that $COL(A^{1*}) = COL(B^{1*})$, as left multiplication with the invertible C^{11} does not affect the linear dependencies among the columns of B^{1*}. Furthermore,

$$\hat{A} = \dot{A}^{-1} \cdot A^{1*} = \dot{A}^{-1} \cdot C^{11} \cdot B^{1*} =$$
$$\dot{A}^{-1} \cdot C^{11} \cdot \dot{B} \cdot \hat{B} = (\dot{A}^{-1} \cdot C^{11} \cdot \dot{B}) \cdot \hat{B}.$$

Thus, the normalized forms $\hat{A}, \hat{B}$ are related by the invertible linear transformation $\dot{A}^{-1} \cdot C^{11} \cdot \dot{B}$. Since $COL(A^{1*}) = COL(B^{1*})$, this linear transformation maps the identity matrix in $\hat{B}$ to the identity matrix in $\hat{A}$, implying that $\dot{A}^{-1} \cdot C^{11} \cdot \dot{B} = I_{n-s}$ and $\hat{A} = \hat{B}$. Combined with the equality $A^{0*} = B^{0*}$ we obtain $A \leftrightarrow_N B$. ■

Let $\Phi = \{N(A) \mid A \in GF(2)^{n \times n}$ is invertible$\}$ contain a representative from each normalized equivalence class.

Using Fact 1 and Proposition 2, we deduce the following corollary.

Corollary 1. *The following properties hold for normalized equivalence classes:*

1. *Each member of Φ represents a normalized equivalence class whose size is equal to the number of invertible $(n-s) \times (n-s)$ matrices C^{11}, which is $\alpha_{n-s} \cdot 2^{(n-s)^2}$.*
2. *The size of Φ is*

$$|\Phi| = \frac{\alpha_n \cdot 2^{n^2}}{\alpha_{n-s} \cdot 2^{(n-s)^2}} = \alpha_n / \alpha_{n-s} \cdot 2^{n^2-(n-s)^2}.$$

3.3 Matrix-Vector Product

Definition 2. *Let A^{1*} and B^{*1} be two Boolean matrices such that A^{1*} has full row rank of $n-s$. Define $\check{B}_A = B \cdot \dot{A} \in GF(2)^{n \times (n-s)}$.*

When A is understood from the context, we simply write $\check{B}$.

Remark 2. The notational conventions that apply to Definition 1 also apply to Definition 1 (see Remark 1), as it is always applied to matrices $A^{1*} \in GF(2)^{(n-s) \times n}$ and $B^{*1} \in GF(2)^{n \times (n-s)}$, where $\check{B} \in GF(2)^{n \times (n-s)}$ (and its sub-matrices are denoted using superscript).

Proposition 3. *Let A^{1*} and B^{*1} be two Boolean matrices such that A^{1*} has full row rank of $n-s$. Let $C = B^{*1} \cdot A^{1*} \in GF(2)^{n \times n}$. Then, after preprocessing A^{1*} and B^{*1}, C can be represented using $b = n^2 - s^2 + n$ bits. Moreover, given $x \in GF(2)^n$, the matrix-vector product Cx can be computed using $O(b)$ bit operations.*

Note that the above representation of the $n \times n$ matrix C is more efficient than the trivial representation that uses n^2 bits (ignoring the additive lower order term n). It is also more efficient than a representation that uses the decomposition $C = B^{*1} \cdot A^{1*}$ which requires $2n(n-s) = (n^2 - s^2) + (n-s)^2 \geq n^2 - s^2$ bits.

Proof. Since A^{1*} has full row rank of $n-s$, we use definitions 1 and 2, and write $C = B^{*1} \cdot A^{1*} = B^{*1} \cdot (\dot{A} \cdot \dot{A}^{-1}) \cdot A^{1*} = (B^{*1} \cdot \dot{A}) \cdot (\dot{A}^{-1} \cdot A^{1*}) = \check{B} \cdot \hat{A}$, where $\check{B}$ and $\hat{A}$ can be computed during preprocessing. Let us assume that the last $n-s$ columns of A^{1*} are linearly independent (namely, $COL(A^{1*}) = \{n, n-1, \ldots, s+1\}$). Then due to (1), $\hat{A}$ can be represented using $s(n-s)$ bits and the matrix-vector product Cx can be computed using $O(s(n-s) + n(n-s)) = O(n^2 - s^2)$ bit operations by computing $\hat{A}x = (\dot{A}^{-1} \cdot \ddot{A}) \cdot x[s|] + x[|n-s]$.

We assumed that the last $n-s$ columns of A^{1*} are linearly independent. If this is not the case, then $COL(A^{1*})$ can be specified explicitly (to indicate the columns of $\hat{A}$ that form the identity) using at most n additional bits. The product $\hat{A}x$ is computed by decomposing x according to $COL(A^{1*})$ (rather than according to its s LSBs). ■

Remark 3. Consider the case that A^{1*} is selected uniformly at random among all matrices of full row rank. Then, using simple analysis based on Fact 1, $n-s$ linearly independent columns of A^{1*} are very likely to be found among its $n-s+3$ last columns. Consequently, the additive low-order term n in the representation size of C can be reduced to an expected size of about $3\log n$ (specifying the 3 indices among are final $n-s+3$ that do not below in $COL(A^{1*})$). Moreover, computing the product $\hat{A}x$ requires permuting only 3 pairs of bits of x on average (and then decomposing it as in the proof above).

Remark 4. Instead of simplifying A^{1*} to contain the identity matrix, we can alternatively simplify B^{*1} assuming it has full column rank.[10] It is easy to verify that both simplifications give essentially the same result in terms of linear algebra complexity.

4 Optimized Encryption Algorithm

In this section we describe our encryption algorithm that optimizes the linear algebra of Algorithm 2. We begin by optimizing the implementation of a 2-round GLMC cipher and then consider a general r-round cipher.

4.1 Basic 2-Round Encryption Algorithm

We start with a basic algorithm that attempts to combine the linear algebra computation of two rounds. This computation can be written as

$$\left(\frac{x_3^{(0)}}{x_3^{(1)}}\right) = \left[\begin{array}{c|c} L_2^{00} & L_2^{01} \\ \hline L_2^{10} & L_2^{11} \end{array}\right] \left(\frac{y_2^{(0)}}{y_2^{(1)}}\right), \left(\frac{x_2^{(0)}}{x_2^{(1)}}\right) = \left[\begin{array}{c|c} L_1^{00} & L_1^{01} \\ \hline L_1^{10} & L_1^{11} \end{array}\right] \left(\frac{y_1^{(0)}}{y_1^{(1)}}\right).$$

Note that $x_2^{(0)}$ and $y_2^{(0)}$ are related non-linearly as $y_2^{(0)} = S_2(x_2^{(0)})$. On the other hand, since $x_2^{(1)} = y_2^{(1)}$ we can compute the contribution of $y_2^{(1)}$ to x_3 at once from y_1 by partially combining the linear operations of the two rounds as

$$\left(\frac{t_3^{(0)}}{t_3^{(1)}}\right) = \left[\begin{array}{c|c} L_2^{01}L_1^{10} & L_2^{01}L_1^{11} \\ \hline L_2^{11}L_1^{10} & L_2^{11}L_1^{11} \end{array}\right] \left(\frac{y_1^{(0)}}{y_1^{(1)}}\right). \quad (2)$$

The linear transformation of (2) is obtained from the product $L_2 \cdot L_1$ by ignoring the terms involving L_2^{00} and L_2^{10} (that operate on $y_2^{(0)}$). Note that (2) defines an $n \times n$ matrix that can be precomputed.

[10] One can also simplify both A^{1*} and B^{*1}, but this is never useful in our application.

We are left to compute the contribution of $y_2^{(0)}$ to x_3, which is done directly as in Algorithm 2 by

$$\begin{aligned} x_2^{(0)} &\leftarrow L_1^{0*}(y_1) \\ y_2^{(0)} &\leftarrow S_2(x_2^{(0)}) \\ t_3' &\leftarrow L_2^{*0}(y_2^{(0)}). \end{aligned} \tag{3}$$

This calculation involves $s \times n$ and $n \times s$ matrices. Finally, combining the contributions of (2) and (3), we obtain

$$x_3 \leftarrow t_3 + t_3'.$$

Overall, the complexity of linear algebra in the two rounds is $n^2 + 2sn$ instead of $2n^2$ of Algorithm 2. This is an improvement given that $s < n/2$, but is inefficient otherwise.

4.2 Optimized 2-Round Encryption Algorithm

The optimized algorithm requires a closer look at the linear transformation of (2). Note that this matrix can be rewritten as the product

$$\left(\frac{t_3^{(0)}}{t_3^{(1)}}\right) = \left[\frac{L_2^{01}}{L_2^{11}}\right] \left[L_1^{10} \middle| L_1^{11} \right] \left(\frac{y_1^{(0)}}{y_1^{(1)}}\right). \tag{4}$$

More compactly, this $n \times n$ linear transformation is decomposed as $L_2^{*1} \cdot L_1^{1*}$, namely, it is a product of matrices with dimensions $(n-s) \times n$ and $n \times (n-s)$. In order to take advantage of this decomposition, we use Proposition 3 which can be applied since L_1^{1*} has full row rank of $n - s$. This reduces linear algebra complexity of $L_2^{*1} \cdot L_1^{1*}$ from n^2 to $n(n-s) + n(n-s) - (n-s)^2 = n^2 - s^2$, ignoring an additive low order term of $3 \log n$, as computed in Remark 3.

```
Input: $x_0$
Output: $x_3$
begin
    $x_1 \leftarrow x_0 + K_0$
    $y_1 \leftarrow S_1(x_1^{(0)}) \| x_1^{(1)}$
    $x_2^{(0)} \leftarrow L_1^{0*}(y_1)$
    $y_2^{(0)} \leftarrow S_2(x_2^{(0)})$
    $x_3 \leftarrow L_2^{*0}(y_2^{(0)})$
    $x_3 \leftarrow x_3 + \check{L}_2(\hat{L}_1(y_1))$
    Output $x_3$
end
```

Algorithm 3: Optimized 2-Round Encryption

```
Input: $x_0$
Output: $x_3$
begin
    $x_1 \leftarrow x_0 + K_0$
    $y_1 \leftarrow S_1(x_1^{(0)}) \| x_1^{(1)}$
    $x_2^{(0)} \leftarrow L_1^{0*}(y_1)$
    $z_2^{(1)} \leftarrow \hat{L}_1(y_1)$
    $y_2^{(0)} \leftarrow S_2(x_2^{(0)})$
    $x_3 \leftarrow L_2^{*0}(y_2^{(0)})$
    $x_3 \leftarrow x_3 + \check{L}_2(z_2^{(1)})$
    Output $x_3$
end
```

Algorithm 4: Refactored 2-Round Encryption

Algorithm 3 exploits the decomposition $L_2^{*1} \cdot L_1^{1*} = \check{L}_2 \cdot \hat{L}_1$. Altogether, the linear algebra complexity of 2 rounds is reduced to

$$n^2 + 2sn - s^2 = 2n^2 - (n-s)^2$$

(or $2n^2 - (n-s)^2 + 3\log n$ after taking Remark 3 into account). This is an improvement by an additive factor of about s^2 compared to the basic 2-round algorithm above and is an improvement over the standard complexity of $2n^2$ for essentially all $s < n$.

4.3 Towards an Optimized r-Round Encryption Algorithm

The optimization applied in the 2-round algorithm does not seem to generalize to an arbitrary number of rounds in a straightforward manner. In fact, there is more than one way to generalize this algorithm (and obtain improvements over the standard one in some cases) using variants of the basic algorithm of Section 4.1 which directly combines more that two rounds. These variants are sub-optimal since they do not exploit the full potential of Proposition 3.

The optimal algorithm is still not evident since the structure of the rounds of Algorithm 3 does not resemble their structure in Algorithm 2 that we started with. Consequently, we rewrite it in Algorithm 4 such that $z_2^{(1)} = \hat{L}_1(y_1)$ is computed already in round 1 instead of round 2. The linear algebra in round 2 of Algorithm 4 can now be described using the $n \times n$ transformation

$$\left(\begin{array}{c} x_3^{(0)} \\ \hline x_3^{(1)} \end{array}\right) = \left[\begin{array}{c|c} L_2^{00} & \check{L}_2^{01} \\ \hline L_2^{10} & \check{L}_2^{11} \end{array}\right] \left(\begin{array}{c} y_2^{(0)} \\ \hline z_2^{(1)} \end{array}\right).$$

Note that $z_2^{(1)}$ is a value that is never computed by the original Algorithm 2.

When we add additional encryption rounds, we can apply Proposition 3 again and "push" some of the linear algebra of round 2 into round 3, then "push" some of the linear algebra of round 3 into round 4, etc. The full algorithm is described in detail next.

4.4 Optimized r-Round Encryption Algorithm

In this section, we describe our optimized algorithm for evaluating r rounds of a GLMC cipher. We begin by defining the following sequence of matrices.
For $i = 1$:

$$\begin{aligned} R_1^{1*} &= L_1^{1*} \\ \hat{R}_1 &= (\dot{R}_1)^{-1} \cdot R_1^{1*}. \end{aligned}$$

For $2 \leq i \leq r-1$:

$$\begin{aligned} \check{T}_i &= L_i^{*1} \cdot \dot{R}_{i-1} \\ R_i^{1*} &= L_i^{10} \| \check{T}_i^{11} \\ \hat{R}_i &= (\dot{R}_i)^{-1} \cdot R_i^{1*}. \end{aligned}$$

For $i = r$:

$$\check{T}_r = L_r^{*1} \cdot \dot{R}_{r-1}.$$

Basically, the matrix $\check{T}_i$ combines the linear algebra of round i with the linear algebra that is pushed from the previous round (represented by $\dot{R}_{i-1}$). The matrix $\hat{R}_i$ is the source of optimization, computed be applying Proposition 3 to the updated round matrix after computing $\check{T}_i$.

Before we continue, we need to prove the follow claim.

Proposition 4. *The matrix R_i^{1*} has full row rank of $n-s$ for all $i \in \{1, \ldots, r-1\}$, hence $(\dot{R}_i)^{-1}$ exists.*

Proof. The proof is by induction on i. For $i = 1$, $R_1^{1*} = L_1^{1*}$ has full row rank by the invertibility of L_1. For $i \in \{2, \ldots, r-1\}$, $R_i^{1*} = L_i^{10} \| \check{T}_i^{11}$. Observe that the matrix $L_i^{1*} = L_i^{10} \| L_i^{11}$ has full row rank and thus column rank of $n-s$. The matrix $\check{T}_i^{11}$ is obtained from L_i^{11} by right multiplication with the matrix $\dot{R}_{i-1}$ which has full rank by the induction hypothesis. Hence $\check{T}_i^{11}$ and L_i^{11} have the same column span. Therefore, the column spans of $L_i^{10} \| L_i^{11}$ and $R_i^{1*} = L_i^{10} \| \check{T}_i^{11}$ are identical, implying that their column and row ranks are $n-s$. ■

The general optimized encryption algorithm is given in Algorithm 5. At a high level, the first round can be viewed as mapping the "real state" $(y_1^{(0)}, y_1^{(1)})$ into the "shadow state" $(x_2^{(0)}, z_2^{(1)})$ using the linear transformation

$$\left(\begin{array}{c} x_2^{(0)} \\ \hline z_2^{(1)} \end{array} \right) = \left[\begin{array}{c|c} L_1^{00} & L_1^{01} \\ \hline \hat{R}_1^{10} & \hat{R}_1^{11} \end{array} \right] \left(\begin{array}{c} y_1^{(0)} \\ \hline y_1^{(1)} \end{array} \right).$$

In rounds $i \in \{2, \ldots, r-1\}$, the shadow state $(y_i^{(0)}, z_i^{(1)})$ (obtained after applying $S_i(x_i^{(0)})$) is mapped to the next shadow state $(x_{i+1}^{(0)}, z_{i+1}^{(1)})$ using the linear transformation

$$\left(\begin{array}{c} x_{i+1}^{(0)} \\ \hline z_{i+1}^{(1)} \end{array} \right) = \left[\begin{array}{c|c} L_i^{00} & \check{T}_i^{01} \\ \hline \hat{R}_i^{10} & \hat{R}_i^{11} \end{array} \right] \left(\begin{array}{c} y_i^{(0)} \\ \hline z_i^{(1)} \end{array} \right).$$

```
Input: $x_0$
Output: $x_{r+1}$
begin
    $x_1 \leftarrow x_0 + K_0$
    $y_1 \leftarrow S_1(x_1^{(0)}) \| x_1^{(1)}$                          ▷ Round 1
    $x_2^{(0)} \leftarrow L_1^{0*}(y_1)$
    $z_2^{(1)} \leftarrow \hat{R}_1(y_1)$
    for $i \in \{2, \dots, r-1\}$ do
        $y_i^{(0)} \leftarrow S_i(x_i^{(0)})$                             ▷ Round $i$
        $x_{i+1}^{(0)} \leftarrow L_i^{00}(y_i^{(0)})$
        $x_{i+1}^{(0)} \leftarrow x_{i+1}^{(0)} + \check{T}_i^{01}(z_i^{(1)})$
        $z_{i+1}^{(1)} \leftarrow \hat{R}_i(y_i^{(0)} \| z_i^{(1)})$
    end
    $y_r^{(0)} \leftarrow S_r(x_r^{(0)})$                                 ▷ Round $r$
    $x_{r+1} \leftarrow L_r^{*0}(y_r^{(0)})$
    $x_{r+1} \leftarrow x_{r+1} + \check{T}_r(z_r^{(1)})$
    Output $x_{r+1}$
end
```

Algorithm 5: Optimized r-Round Encryption

Finally, in round r, the shadow state $(y_r^{(0)}, z_r^{(1)})$ is mapped to the final real state $(x_{r+1}^{(0)}, x_{r+1}^{(1)})$ using the linear transformation

$$\begin{pmatrix} x_{r+1}^{(0)} \\ \hline x_{r+1}^{(1)} \end{pmatrix} = \left[\begin{array}{c|c} L_r^{00} & \check{T}_r^{01} \\ \hline L_r^{10} & \check{T}_r^{11} \end{array}\right] \begin{pmatrix} y_r^{(0)} \\ \hline z_r^{(1)} \end{pmatrix}.$$

While all the linear transformations are of dimensions $n \times n$, the gain from the standard algorithm is due to the special structure of $\hat{R}_i$ for $i \in \{1, 2, \dots, r-1\}$ as it contains the identity matrix.

Complexity Evaluation Algorithm 5 uses the following sequences of matrices:

$$\begin{aligned} &L_1^{0*}, \hat{R}_1, \\ L_i^{00}, \check{T}_i^{01}, \hat{R}_i \text{ for } i \in \{2, \dots, r-1\}&, \\ &L_r^{*0}, \check{T}_r. \end{aligned}$$

Hence, ignoring the linear algebra optimizations for each $\hat{R}_i$, the linear algebra complexity of each round is n^2, leading to a total complexity of $r \cdot n^2$. Taking the optimizations into account, for each $i \in \{1, \dots, r-1\}$, the actual linear algebra complexity of $\hat{R}_i$ is reduced by $(n-s)^2$ to $n^2 - (n-s)^2$. Therefore, the total linear algebra complexity is

$$r \cdot n^2 - (r-1)(n-s)^2.$$

Taking Remark 3 into account, we need to add another factor of $3(r-1)\log n$.

Remark 5. Note that Algorithm 5 is obtained from Algorithm 2 independently of how the instances of the cipher are generated. Hence, Algorithm 5 is applicable to all SP-networks with partial non-linear layers. However, the question of whether this algorithm is more efficient than the standard implementation (which may use very compact linear layers) depends on the actual cipher. Furthermore, in terms of the analysis, Remark 3 does not apply and the additive low-order term added to the complexity is different (but never larger than n bits per round).

Correctness We now prove correctness of Algorithm 5 by showing that its output value is identical to a standard implementation of the scheme in Algorithm 2. For each $i \in \{0, 1, \ldots, r+1\}$, denote by $\bar{x}_i$ the state value at the beginning of round i in a standard implementation and by $\bar{y}_i$ the state after the application of S_i.

Proposition 5. *For each $i \in \{1, \ldots, r-1\}$ in Algorithm 5, $y_i^{(0)} = \bar{y}_i^{(0)}, x_{i+1}^{(0)} = \bar{x}_{i+1}^{(0)}$ and $z_{i+1}^{(1)} = (\dot{R}_i)^{-1}(\bar{x}_{i+1}^{(1)})$.*

Proof. The proof is by induction on i. For $i = 1$, clearly $y_1^{(0)} = \bar{y}_1^{(0)}, x_2^{(0)} = \bar{x}_2^{(0)}$ and

$$z_2^{(1)} = \hat{R}_1(y_1) = (\dot{R}_1)^{-1} \cdot R_1^{1*}(\bar{y}_1) = (\dot{R}_1)^{-1} \cdot L_1^{1*}(\bar{y}_1) = (\dot{R}_1)^{-1}(\bar{x}_2^{(1)}).$$

For $i \in \{2, \ldots, r-1\}$, using the induction hypothesis we obtain

$$y_i^{(0)} = S_i(x_i^{(0)}) = S_i(\bar{x}_i^{(0)}) = \bar{y}_i^{(0)},$$

and

$$\begin{aligned} x_{i+1}^{(0)} = L_i^{00}(y_i^{(0)}) + \check{T}_i^{01}(z_i^{(1)}) = \\ L_i^{00}(\bar{y}_i^{(0)}) + \check{T}_i^{01}\big((\dot{R}_{i-1})^{-1}(\bar{x}_i^{(1)})\big) = \\ L_i^{00}(\bar{y}_i^{(0)}) + L_i^{01} \cdot \dot{R}_{i-1} \cdot (\dot{R}_{i-1})^{-1}(\bar{x}_i^{(1)}) = \\ L_i^{00}(\bar{y}_i^{(0)}) + L_i^{01}(\bar{y}_i^{(1)}) = \\ L_i^{0*}(\bar{y}_i) = \\ \bar{x}_{i+1}^{(0)}. \end{aligned}$$

Finally,

$$\begin{aligned}
z_{i+1}^{(1)} = \hat{R}_i(y_i^{(0)} \| z_i^{(1)}) = \\
(\dot{R}_i)^{-1} \cdot R_i^{1*}\big(\bar{y}_i^{(0)} \| (\dot{R}_{i-1})^{-1}(\bar{x}_i^{(1)})\big) = \\
(\dot{R}_i)^{-1} \cdot \big(L_i^{10} \| \check{T}_i^{11}\big)\big(\bar{y}_i^{(0)} \| (\dot{R}_{i-1})^{-1}(\bar{x}_i^{(1)})\big) = \\
(\dot{R}_i)^{-1} \cdot \big(L_i^{10}(\bar{y}_i^{(0)}) + \check{T}_i^{11} \cdot (\dot{R}_{i-1})^{-1}(\bar{x}_i^{(1)})\big) = \\
(\dot{R}_i)^{-1} \cdot \big(L_i^{10}(\bar{y}_i^{(0)}) + L_i^{11} \cdot \dot{R}_{i-1} \cdot (\dot{R}_{i-1})^{-1}(\bar{x}_i^{(1)})\big) = \\
(\dot{R}_i)^{-1} \cdot \big(L_i^{10}(\bar{y}_i^{(0)}) + L_i^{11}(\bar{x}_i^{(1)})\big) = \\
(\dot{R}_i)^{-1} \cdot \big(L_i^{1*}(\bar{y}_i)\big) = \\
(\dot{R}_i)^{-1}(\bar{x}_{i+1}^{(1)}).
\end{aligned}$$

■

Proposition 6. *Algorithm 5 is correct, namely* $x_{r+1} = \bar{x}_{r+1}$.

Proof. By Algorithm 5 and using Proposition 5,

$$\begin{aligned}
x_{r+1} = L_r^{*0}(y_r^{(0)}) + \check{T}_r(z_r^{(1)}) = \\
L_r^{*0}(\bar{y}_r^{(0)}) + L_r^{*1} \cdot \dot{R}_{r-1}\big((\dot{R}_{r-1})^{-1}(\bar{x}_r^{(1)})\big) = \\
L_r^{*0}(\bar{y}_r^{(0)}) + L_r^{*1}(\bar{y}_r^{(1)}) = \\
L_r(\bar{y}_r) = \\
\bar{x}_{r+1}.
\end{aligned}$$

■

5 Optimized Sampling of Linear Layers

In this section we optimize the sampling of linear layers of generalized LowMC ciphers, assuming they are chosen uniformly at random from the set of all invertible matrices. Sampling the linear layers required by Algorithm 5 in a straightforward manner involves selecting r invertible matrices and applying additional linear algebra operations that transform them to normalized form. This increases the complexity compared to merely sampling these r matrices in complexity $O(r \cdot n^3)$ using a simple rejection sampling algorithm (or asymptotically faster using the algorithm of [17]) and encrypting with Algorithm 2.

We show how to reduce the complexity from $O(r \cdot n^3)$ to[11]

$$O(n^3 + (r-1)(s^2 \cdot n)),$$

which is a significant improvement for small s (and sufficiently large r). We also reduce the amount of (pseudo) random bits requires to sample the linear layers from about $r \cdot n^2$ to about $r \cdot n^2 - (r-1)\big((n-s)^2 - 2(n-s)\big)$.

[11] Further asymptotic improvements are possible using fast matrix multiplication.

The linear layer sampling complexity is reduced in three stages. The first stage breaks the dependency between matrices of different rounds. The second stage breaks the dependency in sampling the bottom part of each round matrix (containing $n-s$ rows) from its top part. Finally, the substantial improvement in complexity for small s is obtained in the third stage that optimizes the sampling of the bottom part of the round matrices. Although the first two stages do not significantly reduce the complexity, they are necessary for applying the third stage and are interesting in their own right.

We note that sampling the matrices that determine the (truncated) round keys of LowMC (following the simplification derived in [16]) can also be optimized using similar techniques to the ones we use in this section for sampling the linear layers. Overall, the complexity of sampling the linear layers remains the bottleneck.

5.1 Breaking Dependencies Among Different Round Matrices

Recall that for $i \in \{2, \ldots, r\}$, the linear transformation of round i is generated from the matrix

$$\left[\begin{array}{c|c} L_i^{00} & \check{T}_i^{01} \\ \hline L_i^{10} & \check{T}_i^{11} \end{array}\right] \tag{5}$$

where

$$\check{T}_i = L_i^{*1} \cdot \dot{R}_{i-1}.$$

For $i = r$, this gives the final linear transformation, while for $i < r$, the final transformation involves applying the decomposition of Definition 1 to $L_i^{10} \| \check{T}_i^{11}$. Since $\check{T}_i$ depends on the invertible $(n-s) \times (n-s)$ matrix $\dot{R}_{i-1}$ (computed in the previous round), a naive linear transformation sampling algorithm would involve computing the linear transformations in their natural order by computing $\dot{R}_{i-1}$ in round $i-1$ and using it in round i. However, this is not required, as the linear transformation of each round can be sampled independently. Indeed, by using Proposition 1 with the invertible matrix $B^{11} = \dot{R}_{i-1}$, we conclude that in round i we can simply sample the matrix given in (5) as a uniform invertible $n \times n$ matrix without ever computing $\dot{R}_{i-1}$. Therefore, the linear transformation sampling for round r simplifies to selecting a uniform invertible $n \times n$ matrix, L_r. For rounds $i \in \{1, \ldots, r-1\}$, we can select a uniform invertible $n \times n$ matrix, L_i, and then normalize it and discard $\dot{R}_i$ after the process. This simplifies Algorithm 5, and it can be rewritten as in Algorithm 6. Note that we have renamed the sequence $\{z_i^{(1)}\}$ to $\{x_i^{(1)}\}$ for convenience.

We stress that the dependency between the round matrices could be broken in Algorithm 6 only since the linear transformation in each round is a uniform invertible matrix. If this is not the case, one can still rename the matrices of Algorithm 5 and derive an algorithm of the form of Algorithm 6. However, computing these matrices would still require deriving $\check{T}_i$ and $\hat{R}_i$ as defined in Section 4.4.

```
Input: $x_0$
Output: $x_{r+1}$
begin
    $x_1 \leftarrow x_0 + K_0$
    for $i \in \{1, \ldots, r-1\}$ do
        $y_i \leftarrow S_i(x_i^{(0)}) \| x_i^{(1)}$                    ▷ Round $i$
        $x_{i+1}^{(0)} \leftarrow L_i^{0*}(y_i)$
        $x_{i+1}^{(1)} \leftarrow \hat{L}_i(y_i)$
    end
    $y_r \leftarrow S_r(x_r^{(0)}) \| x_r^{(1)}$                        ▷ Round $r$
    $x_{r+1} \leftarrow L_r(y_r)$
    Output $x_{r+1}$
end
```

Algorithm 6: Simplified Optimized r-Round Encryption

5.2 Reduced Sampling Space

We examine the sample space of the linear layers more carefully.

For each of the first $r-1$ rounds, the sampling procedure for Algorithm 6 involves selecting a uniform invertible matrix and then normalizing it according to Definition 1. However, by Corollary 1, since each normalized equivalence class contains the same number of $\alpha_{n-s} \cdot 2^{(n-s)^2}$ invertible matrices, this is equivalent to directly sampling a uniform member from Φ to represent its normalized equivalence class. If we order all the matrices in Φ, then sampling from it can be done using $\log|\Phi|$ uniform bits. However, encrypting with Algorithm 6 requires an explicit representation of the matrices and using an arbitrary ordering is not efficient in terms of complexity. In the rest of this section, our goal is to optimize the complexity of sampling from Φ, but first we introduce notation for the full sampling space.

Let the set Λ_r contain r-tuples of matrices defined as

$$\Lambda_r = \Phi^{r-1} \times \{A \in GF(2)^{n \times n} \text{ is invertible}\},$$

where $\Phi^{r-1} = \underbrace{\Phi \times \Phi \ldots \times \Phi}_{r-1 \text{ times}}$.

The following corollary is a direct continuation of Corollary 1.

Corollary 2. *The following properties hold:*

1. *Each r-tuple $(L_1, \ldots, L_{r-1}, L_r) \in \Lambda_r$ represents a set of size $(\alpha_{n-s})^{r-1} \cdot 2^{(r-1)(n-s)^2}$ containing r-tuples of matrices $(L'_1, \ldots, L'_{r-1}, L'_r)$ such that*

$$\big(N(L'_1), \ldots, N(L'_{r-1}), L'_r\big) = (L_1, \ldots, L_{r-1}, L_r).$$

2. *Λ_r contains*

$$|\Lambda_r| = \frac{(\alpha_n)^r \cdot 2^{n^2}}{(\alpha_{n-s})^{r-1} \cdot 2^{(r-1)(n-s)^2}} =$$

$$(\alpha_n)^r / (\alpha_{n-s})^{r-1} \cdot 2^{r \cdot n^2 - (r-1)(n-s)^2}$$

r-tuples or matrices.

As noted above, sampling from Λ_r reduces to sampling the first $r-1$ matrices uniformly from Φ and using a standard sampling algorithm for the r'th matrix.

5.3 Breaking Dependencies Between Round Sub-Matrices

We describe how to further simplify the algorithm for sampling the linear layers by breaking the dependency between sampling the bottom and top sub-matrices in each round. From this point, we will rename the round matrix L_i to a general matrix $A \in GF(2)^{n \times n}$ for convenience. In order to sample from Φ, the main idea is to sample the bottom $n-s$ linearly independent rows of A first, apply the decomposition of Definition 1 and then use this decomposition in order to efficiently sample the remaining s linearly independent rows of A. Therefore, we never directly sample the larger $n \times n$ matrix, but obtain the same distribution on output matrices as the original sampling algorithm.

Sampling the Bottom Sub-Matrix We begin by describing in Algorithm 7 how to sample and compute $\hat{B}$ (which will be placed in the bottom $n-s$ rows of A) and $COL(B^{1*})$ using simple rejection sampling. It uses the sub-procedure $GenRand(n_1, n_2)$ that samples an $n_1 \times n_2$ binary matrix uniformly at random.

Correctness of the algorithm follows by construction. In terms of complexity, we keep track of the span of $\dot{B}$ using simple Gaussian elimination. Based on Fact 1, the expected complexity of (a naive implementation of) the algorithm until it succeeds is $O((n-s)^3 + s^2(n-s))$ due to Gaussian elimination and matrix multiplication.

The Optimized Round Matrix Sampling Algorithm Let us first assume that after application of Algorithm 7, we obtain $\hat{B}, COL(B^{1*})$ such that $COL(B^{1*})$ includes the $n-s$ last columns (which form the identity matrix in $\hat{B}$). The matrix A is built by placing $\hat{B}$ in its bottom $n-s$ columns, and in this case it will be of the block form considered in Fact 2. There is a simple formula (stated in Fact 2) that determines if such matrices are invertible, and we can use this formula to efficiently sample the top s rows of A, while making sure that the full $n \times n$ matrix is invertible. In case $COL(B^{1*})$ does not include the $n-s$ last columns, then a similar idea still applies since A would be in the special form after applying a column permutation determined by $COL(B^{1*})$. Therefore, we assume that A is of the special form, sample the top s rows accordingly and then apply the inverse column permutation to these rows.

Algorithm 8 gives the details of this process. It uses a column permutation matrix, denoted by P (computed from $COL(B^{1*})$, such that $\hat{B} \cdot P = \big((\dot{B})^{-1} \cdot \ddot{B}\big) \| I_{n-s}$ is of the required form. The algorithm also uses two sub-procedures:

1. $GenRand(n_1, n_2)$ samples an $n_1 \times n_2$ binary matrix uniformly at random.
2. $GenInv(n_1)$ samples a uniform invertible $n_1 \times n_1$ matrix.

```
Output: $\hat{B}, COL(B^{1*})$
begin
    while true do
        $B^{1*} \leftarrow \mathbf{0}^{(n-s)\times n}, \dot{B} \leftarrow \mathbf{0}^{(n-s)\times(n-s)}$
        $COL(B^{1*}) \leftarrow \emptyset$
        $rank \leftarrow 0$
        for $i \in \{n, n-1, \ldots, 1\}$ do
            $B^{1*}[*, i] \leftarrow GenRand(n-s, 1)$                ▷ sample column uniformly
            if $rank = n-s$ then
                continue                                             ▷ already obtained full rank
            end
            if $B^{1*}[*, i] \notin span(\dot{B})$ then
                                                                     ▷ new column contributes to rank
                $rank \leftarrow rank + 1$
                $COL(B^{1*}) \leftarrow COL(B^{1*}) \cup \{i\}$
                $\dot{B}[*, rank] \leftarrow B^{1*}[*, i]$
            end
        end
        if $rank = n-s$ then
            $\hat{B} \leftarrow (\dot{B})^{-1} \cdot B^{1*}$
            Output $\hat{B}, COL(B^{1*})$
        end
    end
end
```

Algorithm 7: *SampleBottom*()

```
Output: Round matrix for Algorithm 6
begin
    $\hat{B}, COL(B^{1*}) \leftarrow SampleBottom()$
    $A^{1*} \leftarrow \hat{B}$                                      ▷ bottom $n-s$ rows of $A$
    $C^{00} \leftarrow GenInv(s)$
    $A'^{01} \leftarrow GenRand(s, n-s)$
    $D^{10} \leftarrow (\hat{B} \cdot P)^{10}$                        ▷ $D^{10} = (\dot{B})^{-1} \cdot \ddot{B}$
    $A'^{00} \leftarrow C^{00} + A'^{01} \cdot D^{10}$
    $A^{0*} \leftarrow (A'^{00} \| A'^{01}) \cdot P^{-1}$             ▷ top $s$ rows of $A$
    Output $A$
end
```

Algorithm 8: Optimized Round Matrix Sampling

The complexity of the algorithm is $O((n-s)^3+s^2(n-s)+s^3+s^2(n-s)+sn) = O((n-s)^3 + s^2(n-s) + s^3)$ (using naive matrix multiplication and invertible matrix sampling algorithms), where the dominant factor for small s is $(n-s)^3$. The algorithm requires about $sn + n(n-s) = n^2$ random bits.

Proposition 7. *Algorithm 8 selects a uniform matrix in Φ, namely, the distribution of the output A is identical to the distribution generated by sampling a* *uniform invertible $n \times n$ matrix and applying the transformation of Definition 1 to its bottom $n - s$ rows.*

Proof. First, note that the bottom $n - s$ rows of A are in normalized form after the transformation $A^{1*} = \hat{B} = (\dot{B})^{-1} \cdot B^{1*}$ in *SampleBottom*(). We show that A is sampled correctly by showing that reversing this transformation gives a uniform invertible matrix. Namely,

$$H = \left[\begin{array}{c} A^{0*} \\ \hline B^{1*} \end{array}\right] = \left[\begin{array}{c} A^{0*} \\ \hline \dot{B} \cdot \hat{B} \end{array}\right]$$

is a uniform invertible random matrix.

In order to show that H is invertible, note that

$$\left[\begin{array}{c|c} I_s & \mathbf{0}^{01} \\ \hline \mathbf{0}^{10} & (\dot{B})^{-1} \end{array}\right] \cdot H = \left[\begin{array}{c|c} I_s & \mathbf{0}^{01} \\ \hline \mathbf{0}^{10} & (\dot{B})^{-1} \end{array}\right] \cdot \left[\begin{array}{c} A^{0*} \\ \hline \dot{B} \cdot \hat{B} \end{array}\right] = \left[\begin{array}{c} A^{0*} \\ \hline \hat{B} \end{array}\right] = A.$$

Therefore, it suffices to prove that A is invertible, which is true if and only if $A \cdot P$ is invertible. We have

$$A^{1*} \cdot P = \hat{B} \cdot P = \big((\dot{B})^{-1} \cdot \ddot{B}\big) \| I_{n-s} = D^{10} \| I_{n-s}.$$

Hence

$$A \cdot P = \left[\begin{array}{c} A^{0*} \cdot P \\ \hline A^{1*} \cdot P \end{array}\right] = \left[\begin{array}{c|c} A'^{00} & A'^{01} \\ \hline D^{10} & I_{n-s} \end{array}\right] = \left[\begin{array}{c|c} C^{00} + A'^{01} \cdot D^{10} & A'^{01} \\ \hline D^{10} & I_{n-s} \end{array}\right].$$

By Fact 2 (and change of notation), in order to show that $A \cdot P$ is an invertible matrix, we need to verify that $(C^{00} + A'^{01} \cdot D^{10}) + A'^{01} \cdot D^{10} = C^{00}$ is invertible. This indeed holds by the way that C^{00} is sampled.

It remains to show that H is sampled uniformly among invertible matrices. For this purpose, we show that (a) the number of possible triplets of B^{1*}, C^{00}, A'^{01} sampled uniformly in Algorithm 8 is $\prod_{i=1}^{n}(1 - 1/2^i) \cdot 2^{n^2} = \alpha_n \cdot 2^{n^2}$, and (b) every such triplet gives a different (invertible) matrix H. Since the number of invertible matrices is $\alpha_n \cdot 2^{n^2}$ by Fact 1, (a) and (b) combined with the fact that the algorithm only samples invertible matrices H completes the proof.

We begin by counting the number of triplets B^{1*}, C^{00}, A'^{01}. By Fact 1, the number of possible values for B^{1*} is $\alpha_n / \alpha_s \cdot 2^{n(n-s)}$, as it is a uniform $(n-s) \times n$ matrix with full row rank. Again, by Fact 1, the number of invertible matrices C^{00} is $\alpha_s \cdot 2^{s^2}$, while the number of values of A'^{01} is $2^{s(n-s)}$. Altogether, the number of triplets is

$$\alpha_n / \alpha_s \cdot 2^{n(n-s)} \cdot \alpha_s \cdot 2^{s^2} \cdot 2^{s(n-s)} = \alpha_n \cdot 2^{n^2},$$

as claimed.

Finally, we show that every such triplet gives a different matrix H. Let $B_1^{1*}, C_1^{00}, A_1'^{01}$ and $B_2^{1*}, C_2^{00}, A_2'^{01}$ be two triplets sampled by executions of Algorithm 8 and assume that they both give rise to the same matrix H. We show that these triplets are equal. First, note that $H^{1*} = B_1^{1*} = B_2^{1*}$ which also implies that the value of D^{10} and P computed in Algorithm 8 (which only depends on B^{1*}) is identical for both executions. Next, the value of $H^{0*} = A^{0*}$ is computed in two different ways as

$$\begin{aligned} A^{0*} = (A_1'^{00} \| A_1'^{01}) \cdot P^{-1} = (A_2'^{00} \| A_2'^{01}) \cdot P^{-1} \rightarrow \\ A_1'^{00} \| A_1'^{01} = A_2'^{00} \| A_2'^{01} \rightarrow \\ A_1'^{01} = A_2'^{01} \text{ and } A_1'^{00} = A_2'^{00} \rightarrow \\ A_1'^{01} = A_2'^{01} \text{ and } C_1^{00} + A_1'^{01} \cdot D^{10} = C_2^{00} + A_2'^{01} \cdot D^{10} \rightarrow \\ A_1'^{01} = A_2'^{01} \text{ and } C_1^{00} = C_2^{00}, \end{aligned}$$

as claimed. ■

5.4 Optimized Sampling of the Bottom Sub-Matrix

For a small value of s, the complexity of Algorithm 8 is dominated by *SampleBottom*() (Algorithm 7), whose complexity is $O((n-s)^3 + s^2(n-s))$. We now show how to reduce this complexity to $O(s(n-s))$ on average. Thus, the total expected complexity of Algorithm 8 becomes

$$O(s^2(n-s) + s^3) = O(s^2 \cdot n)$$

(using naive matrix multiplication and invertible matrix sampling algorithms). Moreover, the randomness required by the algorithm is reduced from about $sn + n(n-s) = n^2$ to about

$$sn + (s+2)(n-s) = n^2 - (n-s)^2 + 2(n-s).$$

Recall that the output of *SampleBottom*() consists of $\hat{B}, COL(B^{1*})$, where $\hat{B}$ contains I_{n-s} and s additional columns of $n-s$ bits. The main idea is to directly sample $\hat{B}$ without ever sampling the full B^{1*} and normalizing it. In order to achieve this, we have to artificially determine the column set $COL(B^{1*})$ (which contains the identity matrix in $\hat{B}$), and the values of the remaining s columns.

Remark 6. In general, the distribution of $\hat{B}$ in some alternative *SampleBottom*() implementation does not have to be identical to the one of Algorithm 7, as we can select $COL(B^{1*})$ in a different way (i.e., using a different method to enumerate the columns). The important requirement is that under any enumeration, $\dot{B} \cdot \hat{B} = B^{1*}$ should be a uniform matrix of full row rank. Consider the following trivial optimization attempt of *SampleBottom*(): sample $COL(B^{1*})$ uniformly at random among all column sets of $n-s$ indices (and then sample the remaining columns of $\hat{B}$ uniformly). This algorithm does not satisfy the requirement, as the distribution of $\dot{B} \cdot \hat{B}$ for $\hat{B}$ sampled with this algorithm gives more weight to any matrix with many sets of $n-s$ linearly independent columns over any matrix with fewer such sets.

The main idea is to simulate *SampleBottom*() (Algorithm 7), while sampling concrete vectors only when necessary. As a simple example, consider the case where the first $n-s$ columns sampled by *SampleBottom*() happen to be linearly independent and form $\dot{B}$ (which occurs with probability of more than $\alpha_n \approx 0.288$). Then, we simply replace $\dot{B}$ with the identity matrix (without sampling its columns) and sample the remaining s columns that supposedly form $(\dot{B})^{-1} \cdot \ddot{B}$ uniformly at random. This has complexity of about $s(n-s)$ and the output distribution is identical to that of *SampleBottom*(). Indeed, in this case, the s columns of $\ddot{B}$ are uniform and are independent of $(\dot{B})^{-1}$, hence they remain uniform after multiplication with $(\dot{B})^{-1}$ in *SampleBottom*(). When the first $n-s$ columns sampled by *SampleBottom*() are not linearly independent, then some of the columns of $\ddot{B}$ are no longer uniform conditioned on $COL(B^{1*})$ (these are the columns sampled before full rank is obtained and are not in $COL(B^{1*})$) and we have to take this into account. Algorithm 9 gives the full procedure.

Proposition 8. *The output distribution of OptSampleBottom() (Algorithm 9) is identical to the output distribution of a single iteration of SampleBottom() (Algorithm 7).*

Proof. First, we show that the distributions of $COL(B^{1*})$ in each stage $i \in \{n, n-1, \dots, 1\}$ of the algorithms are identical. Note that in both algorithms, the size of $COL(B^{1*})$ at each stage is *rank*, hence this will also show that the distributions of *rank* are identical (and imply that both algorithms have the same failure probability). At the beginning, $COL(B^{1*}) = \emptyset$ in both and it suffices to show that this variable is updated correctly in *OptSampleBottom*() for each $i \in \{n, n-1, \dots, 1\}$. Indeed, in *SampleBottom*() column i is added to $COL(B^{1*})$ if the currently sampled vector is not in the subspace spanned by the previously sampled vectors (whose size is 2^{rank}). This occurs with probability $1 - 2^{rank}/2^{n-s} = 1 - 2^{(n-s)-rank}$ and is simulated exactly by the $(n-s) - rank$ coin tosses of *OptSampleBottom*().

It remains to show that the output distributions of $\hat{B}$ in the two algorithms are identical. Since we showed that the distributions of $COL(B^{1*})$ are identical at each stage $i \in \{n, n-1, \dots, 1\}$, it is sufficient to show that the output distributions of $\hat{B}$ are identical, conditioned on $COL(B^{1*})$. Clearly, the columns of $COL(B^{1*})$, which contain the identity matrix in both algorithms, are identical. Moreover, the columns sampled after $rank = n-s$ are uniformly distributed in both algorithms (as multiplication with the independent invertible matrix $(\dot{B})^{-1}$ does not change their distribution in *SampleBottom*()). It remains to consider the columns of $i \in \{n, n-1, \dots, 1\}$ which are sampled when $rank < n-s$ and are not added to $COL(B^{1*})$. In *SampleBottom*(), such a column i is sampled uniformly from the subspace spanned by the previously sampled vectors whose size is 2^{rank}. The final multiplication with $(\dot{B})^{-1}$ is a change of basis which transforms the basis of the previously sampled columns to the last *rank* vectors in the standard basis $e_{(n-s)-rank+1}, e_{(n-s)-rank+2}, \dots, e_{n-s}$. Hence, after fixing $COL(B^{1*})$, in the output $\hat{B}$ of *SampleBottom*(), column i is a uniform vector in the subspace spanned by $e_{(n-s)-rank+1}, e_{(n-s)-rank+2}, \dots, e_{n-s}$, which is identical to its distribution in *OptSampleBottom*(). ∎

```
Output: $\hat{B}, COL(B^{1*})$
begin
    $COL(B^{1*}) \leftarrow \emptyset$
    $rank \leftarrow 0$
    for $i \in \{n, n-1, \ldots, 1\}$ do
        if $rank = n - s$ then
                                                          ▷ already obtained full rank
            $\hat{B}[*, i] \leftarrow GenRand(n - s, 1)$              ▷ sample column uniformly
            continue
        end
        for $j \in \{1, \ldots, (n - s) - rank\}$ do
            $b \underset{rand}{\leftarrow} \{0, 1\}$                             ▷ sample uniform bit
            if $b = 1$ then
                                                          ▷ column in $COL(B^{1*})$
                $rank \leftarrow rank + 1$
                $COL(B^{1*}) \leftarrow COL(B^{1*}) \cup \{i\}$
                continue                                  ▷ sample next column
            end
        end
                                                          ▷ column not in $COL(B^{1*})$
        if $rank = 0$ then
            $\hat{B}[*, i] \leftarrow \mathbf{0}$
            continue
        end
        $v \underset{rand}{\leftarrow} \{0, 1\}^{rank}$                         ▷ sample $rank$ uniform bits
        $\hat{B}[*, i] \leftarrow 0^{(n-s)-rank} \| v$            ▷ prepend $(n - s) - rank$ zeros to $v$
    end
    if $rank < n - s$ then
        Output FAIL
    end
    Output $\hat{B}, COL(B^{1*})$
end
```

Algorithm 9: *OptSampleBottom*() Iteration

Complexity The computational effort of the algorithm is proportional to its number of coin tosses (note that it does not involve any linear algebra). Hence, to analyze the complexity, we count the expected number of coin tosses. Simple probabilistic analysis shows that when the iteration succeeds, the expected number of coin tosses can be upper bounded by $2(n - s) + s(n - s) = (s + 2)(n - s)$, where the factor $2(n - s)$ accounts for coin tosses for the $n - s$ columns in $COL(B^{1*})$ and the $s(n - s)$ accounts for sampling the s columns outside of $COL(B^{1*})$. For a reasonable[12] value of $s \geq 3$, by Fact 1, the first iteration will succeed with probability of more than 0.87. Moreover, we can reduce the expected number of coin tosses (while maintaining the output distribution) to less

[12] For $s = 1$ or $s = 2$, the "non-linear" permutation layer is actually linear which makes the cipher insecure.

than $(s+2)(n-s)+0.3n \approx (s+2)(n-s)$ by sampling the columns that are not in $COL(B^{1*})$ only if an iteration succeeds. In fact, it is possible to sample $COL(B^{1*})$ directly without using rejection sampling, but this is more complicated and does not lead to a substantial improvement in complexity.

Decryption We conclude this section by considering efficient sampling of linear layers for decryption. The inverse of the round encryption matrix is of the form shown in Fact 2 after a row permutation (which is the inverse of a column permutation induced by $COL(B^{1*})$). This inverse is generated as a byproduct of Algorithm 8 above for sampling the encryption matrix (which uses the optimized *OptSampleBottom*()). Furthermore, matrix-vector product with the inverse matrix (during decryption) can be computed in about $n^2-(n-s)^2$ bit operations, hence decryption can be performed in about the same complexity as encryption.

6 Optimality of Linear Representation

In this section, we prove that the representation of the linear layers used by Algorithm 6 for a GLMC cipher is essentially optimal. Furthermore, we show that the number of uniform (pseudo) random bits used by the sampling algorithm derived in Section 5 is close to optimal. More specifically, we formulate two assumptions and prove the following theorem under these assumptions, recalling the value of $|\Lambda_r|$ from Corollary 2.

Theorem 1. *Sampling an instance of a GLMC cipher with uniform linear layers must use at least*

$$b = \log|\Lambda_r| = \log\left((\alpha_n)^r/(\alpha_{n-s})^{r-1}\cdot 2^{r\cdot n^2-(r-1)(n-s)^2}\right) \geq \\ r\cdot n^2-(r-1)(n-s)^2-3.5r.$$

uniform random bits and its encryption (or decryption) algorithm requires at least b bits of storage on average. Moreover, if a secure PRG is used to generate the randomness for sampling, then it must produce at least b pseudo-random bits and the encryption (and decryption) process requires at least b bits of storage on average, assuming that it does not have access to the PRG.

We mention that the theorem does not account for the storage required by the non-linear layers. The theorem implies that the code size of Algorithm 6 is optimal up to an additive factor of about $r\cdot(3.5+3\log n)$, which is negligible (less than $0.01\cdot b$ for reasonable choices of parameters).

6.1 Basic Assumptions

The proof relies on the following two assumptions regarding a GLMC cipher.

1. If a PRG is used for the sampling process, it is not used during the encryption process.

2. The linear layers are stored in a manner which is independent of the specification of the non-linear layers. Namely, changing the specification of the non-linear layers does not affect the way that the linear layers are stored.

We now motivate these assumptions. We do so by arguing that implementations that bypass these assumptions are either not useful in practice, or they do not decrease by much the total amount of memory required to encrypt with an instance of a GLMC cipher (even though they may reduce the storage size of the linear layers).

Regarding the first assumption, it is possible to encrypt (or decrypt) using very little memory by generating the linear layers from the seed on-the-fly during the encryption (or decryption) process. However, this is very inefficient in terms time complexity. Therefore, we assume that the encryption maintains an explicit representation of the linear layers (as indeed maintained by current LowMC implementations), and for this purpose, they can be assumed to be truly random based on the pseudo-randomness of the generator.

The second assumption is bypassed (for example) by standard AES software implementations that combine the linear layers with the AES Sbox and form look-up tables which clearly depend on the AES Sbox specification. However, this only increases the total size of the code (in exchange for improved performance on some platforms) and is not a useful way to bypass the second assumption with respect to implementation size. A more meaningful way to bypass the second assumption consists of optimizations that make use of equivalent representations of the cipher which are dependent on its non-linear layers. For example, if the "non-linear" layers are actually linear, then all the linear layers can be combined into a single matrix, equivalent to all combined matrices. Of course, this results in a linear and insecure cipher, but a more realistic approach would consider non-linear layers that are self-affine equivalent (cf. [5]), which implies that they have several equivalent representations. However, unless the non-linear layers are close to being truly linear, the number of such equivalent representations is small compared to the total possible number of linear layers in a GLMC cipher (and in the specific case of LowMC) and they do not allow a substantial saving in the implementation size.[13]

Finally, we note that implementations which try to combine the linear layers in various ways (that are independent of the non-linear layers), or manipulate them (such as the "method of the four Russians") do fall within our model and we prove that they cannot reduce the storage size.

6.2 Model Formalization

We now define our model which formalizes the assumptions above and allows to prove the optimality of our representation.

[13] It is possible to optimize the representation of Algorithm 6 to take advantage of self-affine equivalent non-linear layers. However, this is out of the scope of this paper and as noted above, has very limited effect for a reasonable choice of non-linear layers.

Definition 3. *Given a triplet of global parameters (n, s, r), a (simplified) standard representation of a GLMC cipher is a triplet $\mathcal{R} = (K_0, \mathcal{S}, \mathcal{L})$ such that $K_0 \in \{0,1\}^n$, $\mathcal{S} = (S_1, S_2, \ldots, S_r)$ is an r-tuple containing the specifications of r non-linear invertible layers $S_i : \{0,1\}^s \to \{0,1\}^s$ and $\mathcal{L} = (L_1, L_2, \ldots, L_r)$ is an r-tuple of invertible matrices $L_i \in GF(2)^{n \times n}$. The r-tuple $\mathcal{L}$ is called a standard linear representation.*

To simplify our notation, given a standard representation $\mathcal{R} = (K_0, \mathcal{S}, \mathcal{L})$, we denote the encryption algorithm defined by Algorithm 2 as $E_{\mathcal{R}} : \{0,1\}^n \to \{0,1\}^n$.

Definition 4. *Two standard cipher representations $\mathcal{R}, \mathcal{R}'$ are equivalent (denoted $\mathcal{R} \equiv \mathcal{R}'$) if for each $x \in \{0,1\}^n$, $E_{\mathcal{R}}(x) = E_{\mathcal{R}'}(x)$.*

Definition 5. *Two standard linear representations $\mathcal{L}, \mathcal{L}'$ are equivalent (denoted $\mathcal{L} \equiv \mathcal{L}'$) if for each tuple of non-linear layers $\mathcal{S}$, and key K_0, $(K_0, \mathcal{S}, \mathcal{L}) \equiv (K, \mathcal{S}, \mathcal{L}')$.*

The requirement that $(K, \mathcal{S}, \mathcal{L}) \equiv (K, \mathcal{S}, \mathcal{L}')$ for *any* $\mathcal{S}, K_0$ captures the second assumption of Section 6.1 that a standard representation of the linear layers is independent of the non-linear layers (and the key).

Clearly, the linear equivalence relation partitions the r-tuples of standard linear representations into linear equivalence classes. It is important to mention that Theorem 1 does not assume that the encryption algorithm uses Algorithm 2 or represents the linear layers as an r-tuple of matrices. These definitions are merely used in its proof, as shown next.

6.3 Proof of Theorem 1

We will prove the following lemma regarding linear equivalence classes, from which Theorem 1 is easily derived.

Lemma 1. *For any $\mathcal{L} \neq \mathcal{L}' \in \Lambda_r$, $\mathcal{L} \not\equiv \mathcal{L}'$.*

The lemma states that each r-tuple of Λ_r is a member of a distinct equivalence class, implying that we have precisely identified the equivalence classes.

Proof (of Theorem 1). Lemma 1 asserts that there are at least $|\Lambda_r|$ linear equivalence classes. Corollary 2 asserts that each r-tuple in Λ_r represents a set of linear layers of size $(\alpha_{n-s})^{r-1} \cdot 2^{(r-1)(n-s)^2}$, hence every r-tuple in Λ_r has the same probability weight when sampling the r linear layers uniformly at random. The theorem follows from the well-known information theoretic fact that sampling and representing a uniform string (an r-tuple in Λ_r) chosen out of a set of 2^k strings requires at least k bits on average (regardless of any specific sampling or representation methods). ■

The proof of Lemma 1 relies on two propositions which are implications of the definition of equivalence of standard linear representations (Definition 5).

Proposition 9. *Let $\mathcal{L} \equiv \mathcal{L}'$ be two equivalent standard linear representations. Given $K_0, \mathcal{S}$, let $\mathcal{R} = (K_0, \mathcal{S}, \mathcal{L})$ and $\mathcal{R}' = (K_0, \mathcal{S}, \mathcal{L}')$. Fix any $x \in \{0,1\}^n$ and $i \in \{0, 1, \ldots, r+1\}$, and denote by x_i (resp. x'_i) the value $E_{\mathcal{R}}(x)$ (resp. $E_{\mathcal{R}'}(x)$) at the beginning of round i. Then $x_i^{(0)} = x_i'^{(0)}$.*

Namely, non-linear layer inputs (and outputs) have to match at each round when encrypting the same plaintext with ciphers instantiated with equivalent standard linear representations (and use the same key and non-linear layers).
Proof. The proposition clearly holds for $i = 0$ and $i = r+1$ (as the outputs of equivalent ciphers have to match for any x). Assume towards contradiction that $x_i^{(0)} \neq x_i'^{(0)}$ for some $i \in \{1, \ldots, r\}$. Recall that linear equivalence implies that $(K_0, \mathcal{S}, \mathcal{L}) \equiv (K_0, \mathcal{S}, \mathcal{L}')$ for any $K_0, \mathcal{S}$. In particular, consider $\mathcal{S}^*$ for which the only change from $\mathcal{S}$ is that S_i is modified to S_i^* and exchanges the output values of $x_i'^{(0)}$ and some $u_i^{(0)}$ (for $u_i^{(0)} \neq x_i^{(0)}$ and $u_i^{(0)} \neq x_i'^{(0)}$), namely, $S_i^*(x_i'^{(0)}) = S_i(u_i^{(0)})$ and $S_i^*(u_i^{(0)}) = S_i(x_i'^{(0)})$.

Since $S_i(x_i'^{(0)}) \neq S_i^*(x_i'^{(0)})$, then $E_{(K_0,\mathcal{S},\mathcal{L}')}(x) \neq E_{(K_0,\mathcal{S}^*,\mathcal{L}')}(x)$. Indeed, the state values up to the i'th non-linear layer match in $E_{(K_0,\mathcal{S},\mathcal{L}')}(x)$ and $E_{(K_0,\mathcal{S}^*,\mathcal{L}')}(x)$, but then diverge as $S_i(x_i'^{(0)}) \neq S_i^*(x_i'^{(0)})$. Since the remaining partial encryption algorithm (consisting of L'_i and rounds $i+1, \ldots, r$) is a permutation which is identical in both ciphers, the state values cannot converge.

On the other hand $E_{(K_0,\mathcal{S}^*,\mathcal{L})}(x) = E_{(K_0,\mathcal{S},\mathcal{L})}(x)$ as $S_i(x_i^{(0)}) = S_i^*(x_i^{(0)})$. Therefore, either $E_{(K_0,\mathcal{S},\mathcal{L})}(x) \neq E_{(K_0,\mathcal{S},\mathcal{L}')}(x)$ or

$$E_{(K_0,\mathcal{S}^*,\mathcal{L})}(x) = E_{(K_0,\mathcal{S},\mathcal{L})}(x) = E_{(K_0,\mathcal{S},\mathcal{L}')}(x) \neq E_{(K_0,\mathcal{S}^*,\mathcal{L}')}(x),$$

and in any case $\mathcal{L} \not\equiv \mathcal{L}'$ in contradiction. ■

Proposition 10. *Let $\mathcal{L} \equiv \mathcal{L}'$ be two equivalent standard linear representations. Given $K_0, \mathcal{S}$, let $\mathcal{R} = (K_0, \mathcal{S}, \mathcal{L})$ and $\mathcal{R}' = (K_0, \mathcal{S}, \mathcal{L}')$. Fix any $x \in \{0,1\}^n$ and $i \in \{0, 1, \ldots, r+1\}$, and denote by x_i (resp. x'_i) the value $E_{\mathcal{R}}(x)$ (resp. $E_{\mathcal{R}'}(x)$) at the beginning of round i. Moreover, fix $\bar{x} \neq x$ such that $\bar{x}_i = \bar{x}_i^{(0)}, \bar{x}_i^{(1)}$, where $\bar{x}_i^{(0)} \neq x_i^{(0)}$, but $\bar{x}_i^{(1)} = x_i^{(1)}$. Then, $\bar{x}_i'^{(1)} = x_i'^{(1)}$.*

The proposition considers two plaintexts x and $\bar{x}$ whose encryptions under the first cipher in round i differ only in the 0 part of the state. We then look at the second cipher (formed using equivalent standard linear representations) and claim that the same property must hold for it as well. Namely, the encryptions of x and $\bar{x}$ under the second cipher in round i differ only on the 0 part of the state.
Proof. Consider $\mathcal{S}^*$ for which the only change from $\mathcal{S}$ is that S_i is modified to S_i^* and exchanges the output values of $x_i^{(0)}$ and $\bar{x}_i^{(0)}$ (in particular $S_i^*(\bar{x}_i^{(0)}) = S_i(x_i^{(0)})$). Consider $E_{(K_0,\mathcal{S}^*,\mathcal{L})}(\bar{x})$ and note that the state obtained after i rounds (that are unchanged from $(K_0, \mathcal{S}, \mathcal{L})$) is equal to

$$\bar{x}_i = \bar{x}_i^{(0)}, \bar{x}_i^{(1)} = \bar{x}_i^{(0)}, x_i^{(1)}.$$

After application of S_i^*, the 0 part of the state is modified to $S_i^*(\bar{x}_i^{(0)}) = S_i(x_i^{(0)}) = y_i^{(0)}$. Hence, the full state of $E_{(K_0,\mathcal{S}^*,\mathcal{L})}(\bar{x})$ after i encryption rounds and the non-linear layer application is $y_i^{(0)}, x_i^{(1)} = y_i^{(0)}, y_i^{(1)} = y_i$.

Since the mappings from y_i to the output in the remaining i'th linear layer and last $r-i+1$ round functions in $(K_0, \mathcal{S}^*, \mathcal{L})$ and $(K_0, \mathcal{S}, \mathcal{L})$ are identical, we have

$$E_{(K_0,\mathcal{S}^*,\mathcal{L})}(\bar{x}_i) = E_{(K_0,\mathcal{S},\mathcal{L})}(x_i).$$

In other words, exchanging the non-linear layer values canceled the change caused the exchanging the plaintexts.

Since $\mathcal{L} \equiv \mathcal{L}'$, then

$$E_{(K_0,\mathcal{S}^*,\mathcal{L}')}(\bar{x}_i) = E_{(K_0,\mathcal{S}^*,\mathcal{L})}(\bar{x}_i) = E_{(K_0,\mathcal{S},\mathcal{L})}(x_i) = E_{(K_0,\mathcal{S},\mathcal{L}')}(x_i).$$

Once again, the last $r - i + 1$ round functions after applications of S_i^* and S_i in $(K_0, \mathcal{S}^*, \mathcal{L}')$ and $(K_0, \mathcal{S}, \mathcal{L}')$ (respectively) are identical permutations and their outputs are identical by the above equality. Hence, the inputs to the final rounds (obtained after i encryption rounds and application of the i'th non-linear layer) are identical, and in particular the values of their part 1 of the state are identical. In $E_{(K_0,\mathcal{S}^*,\mathcal{L}')}(\bar{x}_i)$ it is $\bar{y}_i'^{(1)} = \bar{x}_i'^{(1)}$ as the first r rounds of $E_{(K_0,\mathcal{S}^*,\mathcal{L}')}$ and $E_{(K_0,\mathcal{S},\mathcal{L}')}$ are identical. In $E_{(K_0,\mathcal{S},\mathcal{L}')}(x_i)$ is it $y_i'^{(1)} = x_i'^{(1)}$ by definition. We conclude that $\bar{x}_i'^{(1)} = x_i'^{(1)}$ as claimed. ∎

Proof (of Lemma 1). The proof is by induction of r.

For $r = 1$, Λ_1 is the set of invertible matrices. We show that every different invertible matrix $\mathcal{L} = (L_1)$ forms a linear equivalence class. Indeed, assume that for some $K, \mathcal{S}$ and all x, $E_{(K_0,\mathcal{S},(L_1))}(x) = E_{(K_0,\mathcal{S},(L_1'))}(x)$.[14] Then after adding K_0 and applying S_1 (which are the same for both schemes), we get $L_1(y_1) = L_1'(y_1)$. In particular, this holds for the n vectors of the standard basis $y_1 \in \{e_1, e_2, \ldots, e_n\}$ which give (in matrix notation) $L_1 \cdot I_n = L_1' \cdot I_n$ or $L_1 = L_1'$.

Assume correctness for $i = r$ and add a round at the beginning, noticing that rounds $2, \ldots, r+1$ form an r-round scheme with zero initial round key.

Let $(L_1, \mathcal{L}) \neq (L_1', \mathcal{L}') \in \Lambda_{r+1}$ such that $\mathcal{L}, \mathcal{L}' \in \Lambda_r$. We need to prove $(L_1, \mathcal{L}) \not\equiv (L_1', \mathcal{L}')$. We divide the proof into three cases.

If $\mathcal{L} = \mathcal{L}'$ but $L_1 \neq L_1'$, then for any corresponding ciphers, we have $E_{(K_0,\mathcal{S},(L_1,\mathcal{L}))}(x) \neq E_{(K_0,\mathcal{S},(L_1',\mathcal{L}))}(x)$ on every x for which the outputs of the first round differs (since rounds $2, \ldots, r+1$ are identical).

If $\mathcal{L} \neq \mathcal{L}'$ and $L_1 = L_1'$, then by the induction hypothesis, $\mathcal{L} \not\equiv \mathcal{L}'$. If we start with non-equivalent r-round ciphers and add an identical round at the beginning, clearly the schemes remain non-equivalent.

Finally, it remains to prove the induction step for the case that $\mathcal{L} \neq \mathcal{L}'$ and $L_1 \neq L_1'$. Assume towards contradiction that $(L_1, \mathcal{L}) \equiv (L_1', \mathcal{L}')$. We will prove that in this case, $L_1 \leftrightarrow_N L_1'$. However, by the definition of Λ_{r+1}, we select

[14] The definition of linear equivalence requires that $E_{(K_0,\mathcal{S},(L_1))}(x) = E_{(K_0,\mathcal{S},(L_1'))}(x)$ holds for any $K_0, \mathcal{S}$, so it obviously must hold for an arbitrary choice of $K_0, \mathcal{S}$.

only one matrix from each normalized equivalence class and hence $L_1 = L'_1$ in contradiction.

In order to derive $L_1 \leftrightarrow_N L'_1$, fix a key K_0, an $(r+1)$-tuple of non-linear layers $\mathcal{S}$ and pick some plaintext $x \in \{0,1\}^n$. Denote the state values for round i of the first cipher $E_{(K_0,\mathcal{S},(L_1,\mathcal{L}))}(x)$ by x_i and y_i and similarly, denote by x'_i and y'_i these values for $E_{(K_0,\mathcal{S},(L'_1,\mathcal{L}'))}(x)$. Since the initial round keys and S_1 are identical, $y_1 = y'_1$ and therefore,

$$x_2 = L_1 \cdot \left(\frac{y_1^{(0)}}{y_1^{(1)}}\right) \ , \ x'_2 = L'_1 \cdot \left(\frac{y_1^{(0)}}{y_1^{(1)}}\right),$$

for the encryption of any $x \in \{0,1\}^n$. We consider the transformation

$$\big(L'_1 \cdot (L_1)^{-1}\big)(x_2) = x'_2.$$

Our goal is to show that

$$L'_1 \cdot (L_1)^{-1} = \left[\begin{array}{c|c} I_s & \mathbf{0}^{01} \\ \hline \mathbf{0}^{10} & C^{11} \end{array}\right] \tag{6}$$

for some invertible C^{11}, which proves $L_1 \leftrightarrow_N L'_1$ by Proposition 2.

Since we assume $(L_1, \mathcal{L}) \equiv (L'_1, \mathcal{L}')$, then by Proposition 9 we must have $x_2'^{(0)} = x_2^{(0)}$ at the input of the second linear layer. By setting $x_2 \in \{e_1, e_2, \ldots, e_n\}$ to be the n standard basis vectors and using the equality $x_2'^{(0)} = x_2^{(0)}$, we conclude that the top s rows of $L'_1 \cdot (L_1)^{-1}$ are of the form of (6).

It remains to prove that the bottom-left $(n-s)\times s$ block of $L'_1 \cdot (L_1)^{-1}$ is equal to $\mathbf{0}^{10}$. Equivalently, we need to show that changing $x_2^{(0)}$ (without changing $x_2^{(1)}$) at the input of $L'_1 \cdot (L_1)^{-1}$ does not change $x_2'^{(1)}$ at the output (but will obviously set $x_2'^{(0)} = x_2^{(0)}$). Indeed, this property is directly implied by Proposition 10. ■

References

1. M. R. Albrecht, G. V. Bard, and W. Hart. Algorithm 898: Efficient multiplication of dense matrices over GF(2). *ACM Trans. Math. Softw.*, 37(1):9:1–9:14, 2010.
2. M. R. Albrecht, C. Rechberger, T. Schneider, T. Tiessen, and M. Zohner. Ciphers for MPC and FHE. In Oswald and Fischlin [15], pages 430–454.
3. A. Bar-On, I. Dinur, O. Dunkelman, V. Lallemand, N. Keller, and B. Tsaban. Cryptanalysis of SP networks with partial non-linear layers. In Oswald and Fischlin [15], pages 315–342.
4. E. Barkan and E. Biham. In How Many Ways Can You Write Rijndael? In Y. Zheng, editor, *Advances in Cryptology - ASIACRYPT 2002, 8th International Conference on the Theory and Application of Cryptology and Information Security, Queenstown, New Zealand, December 1-5, 2002, Proceedings*, volume 2501 of *Lecture Notes in Computer Science*, pages 160–175. Springer, 2002.

5. A. Biryukov, C. D. Cannière, A. Braeken, and B. Preneel. A Toolbox for Cryptanalysis: Linear and Affine Equivalence Algorithms. In E. Biham, editor, *Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings*, volume 2656 of *Lecture Notes in Computer Science*, pages 33–50. Springer, 2003.
6. D. Boneh, S. Eskandarian, and B. Fisch. Post-Quantum Group Signatures from Symmetric Primitives. *IACR Cryptology ePrint Archive*, 2018:261, 2018.
7. M. Chase, D. Derler, S. Goldfeder, C. Orlandi, S. Ramacher, C. Rechberger, D. Slamanig, and G. Zaverucha. Picnic: A Family of Post-Quantum Secure Digital Signature Algorithms. https://microsoft.github.io/Picnic/.
8. M. Chase, D. Derler, S. Goldfeder, C. Orlandi, S. Ramacher, C. Rechberger, D. Slamanig, and G. Zaverucha. Post-Quantum Zero-Knowledge and Signatures from Symmetric-Key Primitives. In B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1825–1842. ACM, 2017.
9. D. Derler, S. Ramacher, and D. Slamanig. Post-Quantum Zero-Knowledge Proofs for Accumulators with Applications to Ring Signatures from Symmetric-Key Primitives. In T. Lange and R. Steinwandt, editors, *Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018, Fort Lauderdale, FL, USA, April 9-11, 2018, Proceedings*, volume 10786 of *Lecture Notes in Computer Science*, pages 419–440. Springer, 2018.
10. B. Gérard, V. Grosso, M. Naya-Plasencia, and F. Standaert. Block Ciphers That Are Easier to Mask: How Far Can We Go? In G. Bertoni and J. Coron, editors, *Cryptographic Hardware and Embedded Systems - CHES 2013 - 15th International Workshop, Santa Barbara, CA, USA, August 20-23, 2013. Proceedings*, volume 8086 of *Lecture Notes in Computer Science*, pages 383–399. Springer, 2013.
11. O. Goldreich, S. Micali, and A. Wigderson. How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In A. V. Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 218–229. ACM, 1987.
12. J. Katz, V. Kolesnikov, and X. Wang. Improved Non-Interactive Zero Knowledge with Applications to Post-Quantum Signatures. *IACR Cryptology ePrint Archive*, 2018:475, 2018. Accepted to ACM CCS 2018.
13. V. Kolesnikov and T. Schneider. Improved Garbled Circuit: Free XOR Gates and Applications. In L. Aceto, I. Damgård, L. A. Goldberg, M. M. Halldórsson, A. Ingólfsdóttir, and I. Walukiewicz, editors, *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II - Track B: Logic, Semantics, and Theory of Programming & Track C: Security and Cryptography Foundations*, volume 5126 of *Lecture Notes in Computer Science*, pages 486–498. Springer, 2008.
14. NIST's Post-Quantum Cryptography Project. Round 1 Submissions. https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-1-Submissions.
15. E. Oswald and M. Fischlin, editors. *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*. Springer, 2015.

16. L. Perrin, A. Promitzer, S. Ramacher, and C. Rechberger. Improvements to the Linear Layer of LowMC: A Faster Picnic. *IACR Cryptology ePrint Archive*, 2017:1148, 2017.
17. D. Randall. Efficient Generation of Random Nonsingular Matrices. *Random Struct. Algorithms*, 4(1):111–118, 1993.
18. S. Rasoolzadeh, Z. Ahmadian, M. Salmasizadeh, and M. R. Aref. Total Break of Zorro using Linear and Differential Attacks. *ISeCure, The ISC International Journal of Information Security*, 6(1):23–34, 2014.
19. C. Rechberger, H. Soleimany, and T. Tiessen. The LowMC Cipher Breaking Challenge. https://www.nuee.nagoya-u.ac.jp/labs/tiwata/fse2017/slides/Rump-11.pdf.
20. Y. Wang, W. Wu, Z. Guo, and X. Yu. Differential Cryptanalysis and Linear Distinguisher of Full-Round Zorro. In I. Boureanu, P. Owesarski, and S. Vaudenay, editors, *Applied Cryptography and Network Security - 12th International Conference, ACNS 2014, Lausanne, Switzerland, June 10-13, 2014. Proceedings*, volume 8479 of *Lecture Notes in Computer Science*, pages 308–323. Springer, 2014.
21. A. C. Yao. How to Generate and Exchange Secrets (Extended Abstract). In *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*, pages 162–167. IEEE Computer Society, 1986.