IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2023, No. 4, pp. 83–111. DOI:10.46586/tosc.v2023.i4.83-111

Improved Fast Correlation Attacks on the Sosemanuk Stream Cipher

Bin Zhang[1,2,3,4,5], Ruitao Liu[1,2,5], Xinxin Gong[3] and Lin Jiao[3]

[1] University of Chinese Academy of Sciences, Beijing, 100049, China
martin_zhangbin@hotmail.com, ruitao2021@iscas.ac.cn
[2] Trusted Computing and Information Assurance Laboratory, State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China
[3] State Key Laboratory of Cryptology, P.O. Box 5159, Beijing, 100878, China
xinxgong@126.com, jiaolin_jl@126.com
[4] Guizhou shujubao Network Technology Co., Ltd, Guizhou, China
[5] Zhongguancun Laboratory, Beijing, China

Abstract. In this paper, we present a new algorithm for fast correlation attacks on stream ciphers with improved cryptanalysis results on the Sosemanuk stream cipher, one of the 7 finalists in the eSTREAM project in 2008. The new algorithm exploits the direct sum construction of covering codes in decoding phase which approximates the random vectors to a nearest codeword in a linear code. The new strategy provides large flexibility for the adversary and could reduce the time/memory/data complexities significantly. As a case study, we carefully revisit Sosemanuk and demonstrate a state recovery attack with a time complexity of $2^{134.8}$, which is 2^{20} times faster than achievable before by the same kind of attack and is the fastest one among all known attacks so far. Our result indicates an inefficiency in longer keys than 135 bits and depicts that the security margin of Sosemanuk is around 2^8 for the 128-bit security for the first time.

Keywords: Fast correlation attack · Stream ciphers · Covering codes · Sosemanuk · Linear feedback shift register (LFSR)

1 Introduction

The European eSTREAM project [EST08] has a sustaining effect on the design and analysis of modern stream ciphers, which provides some typical design paradigms such as NFSR-based (nonlinear feedback shift register), ARX-based and LFSR + FSM (Finite State Machine) model. As one of the 7 eSTREAM finalists, Sosemanuk [BBC+08] is a word-oriented synchronous stream cipher which supports a variable key length ranging from 128 to 256 bits and an initial value (IV) of 128 bits with the 128-bit claimed security level. It follows the LFSR + FSM design strategy of the SNOW family of stream ciphers [EJ03, SNO06, EJMY19] in the keystream generation phase with the 4-bit S-box from the Serpent block cipher [BAK98] to enhance SNOW 2.0 from both security and efficiency points of view. The conclusion from the eSTREAM final report [BCC+08] on Sosemanuk is that it offers a very considerable margin for security as well as very reasonable performance trade-offs.

Correlation attack is a classical cryptanalysis method for LFSR-based stream ciphers, which exploits the statistical relation between the keystream and some subset of the involved LFSR sequences. The earliest studies can be traced back to the work in [Sie84] in last century with the correlation immunity proposed as a security criterion. Then fast correlation attacks (FCA), proposed by Meier and Staffelbach in [MS89], speeds up the

Received: 2023-09-01 Accepted: 2023-11-01 Published: 2023-12-08

exhaustive search of the involved LFSR state by some decoding algorithms from coding theory when the weight of the underlying LFSR is relatively low and the noise is of modest level. There are two kinds of algorithms involved here, i.e., the one-pass/information set decoding and probabilistic iterative decoding algorithms. Fast correlation attacks have been evolved over the years into some modern form, i.e., without any restriction on the involved LFSR, of finding the linear correlation by some theoretical and/or automatic searching strategy and of dealing with a highly noisy channel with dedicated decoding methods [CT00, JJ99b, JJ99a, CJM02, ZXM15].

Since 2005, there are lots of attention from the community to analyze the security of Sosemanuk. Various cryptanalysis methods have been tried to evaluate the security of Sosemanuk, to name but a few, Guess-and-Determine attacks [TSS+06, FLZ+10], time/memory/data tradeoff [BS00, DJGQ14] and linear/correlation attacks [LLP08, CH10]. The best attack on Sosemanuk available so far is the correlation attacks in [LLP08, CH10] with a time complexity of 2^{154} by using some bitwise linear approximation of a bias $2^{-21.41}$ between any two consecutive keystream words and the corresponding LFSR sequence. It is worth noting that given the correlation of $p = \frac{1}{2} + 2^{-21.41}$ and the number of binary variables $l = 320$, we can derive the information-theoretical bound [CS91] of the complexity for any attack aiming at the $l = 320$-bit entropy contained in the LFSR initial state, which is

$$\frac{4 \cdot l \cdot \ln 2}{1 - H(p)} \doteq 2^{51.08}$$

with $l = 320$ and $H(x) = -x \cdot \log(x) - (1-x) \cdot \log(1-x)$ being the binary entropy function; while the currently best known attacks all have a time complexity of around 2^{154}. The gap between the theory and the cryptanalysis practice on Sosemanuk is huge and a natural problem arising is whether we could narrow the gap by some improved decoding algorithm given the same correlation?

Contributions. In this paper, we present an improved fast correlation attack on stream ciphers which can make better use of the found linear correlation with respect to the corresponding information-theoretical bound.

We try to bridge this huge gap to some extent in theory by introducing new algorithmic procedures into fast correlation attacks, which are derived from the famous BKW algorithm [BKW00], a Gauss elimination procedure to transform the distribution of secret information and a direct sum construction of some well chosen covering codes. As analogues to similar BKW collision, distribution-transformation and code-reduction steps in the solving algorithms of the Learning Parity with Noise (LPN) problem [GJL20], a single-list BKW collision procedure and a random vector substitution procedure are introduced into FCA to reduce the dimension of the secret information in the LFSR initial state with another Gauss elimination step over finite fields converting the distribution of the initial LFSR state from uniform to some biased distribution. The single-list BKW collision step overcomes the lost-solution problem of the previous method, which splits the whole column-list of the derived generator matrix to several separate sub-lists, based on the generalized birthday problem [Wag02] to a large extent. Further recall that the parity-checks in the pre-processing phase of a FCA are usually constructed to reduce the dimension of the LFSR initial state that goes through the online decoding phase at the expense of the folded noise from the piling-up lemma [Mat94], thus it is naturally expected that we can somehow weaken the noise attenuation while still efficiently reduce the secret dimension. This inspires us to investigate the possibility of an integrated combination of some methods from LPN solvers with the existing FCAs on the algorithmic aspect.

Orderly, we first make the distribution transform to convert the distribution of the LFSR initial state from uniform to the same biased distribution as that of the found linear approximation by Gauss elimination. This step provides the possibility of reducing the complexity for recovering the LFSR initial state, as the entropy loss from a biased

distribution is usually larger than the uniform case. Then, the single-list BKW collision algorithm is introduced and optimized to reduce the secret dimension by some pre-computed tables of reasonably small size which can be accessed with a constant complexity under the same assumption adopted by default by almost all the cryptanalysis literatures. Then we study in details the random vector substitution procedure arising from the direct sum construction of covering codes in solving a noisy system of linear equations in the binary domain, a generic problem inherent not only in FCAs, but also in linear cryptanalysis (LC) of many symmetric key primitives.

We theoretically derive the averaged bias in the considered case when the covering codes are chosen from perfect codes with the description of relevant parameter-chosen method of linear programming. Then we make an integrated combination of these improvements with the Fast Walsh-Hadamard Transform (FWHT) technique to solve the resultant noisy linear system with the newly derived biased LFSR initial state as variables, which actually is a new framework for FCAs or a new solver of noisy linear system in the binary field.

Attacks	Attack Type	Complexities	
		Data	Time
[BS00]	Time/Memory/Data Tradeoff	$\mathcal{O}(2^{128})$	$\mathcal{O}(2^{256}+2^{256})$
[FLZ+10]	Guess-and-Determine	$2^{4.32}$	$\mathcal{O}(2^{176})$
[LLP08]	Fast correlation attack	$2^{145.50}$	$2^{155.7}$
This paper	Fast correlation attack	2^{135}	$2^{134.8}$

As a case study, we investigate the security of Sosemanuk by the new algorithm and compare it with the previously known best results, as shown above. By carefully tuning the parameters involved in the attack later depicted, we can have a state recovery attack in $2^{134.8}$ time with the 2^{10} times reduced data complexity of 2^{135}, which is around 2^{20} times faster than the best previously known results in [LLP08] and is the fastest attack on Sosemanuk among all the known attacks so far. Since the time complexity is the most important measurement when comparing different kinds of attacks, we focus on this aspect here. Note that the time complexity unit of the new FCA is the finite field operations such as multiplication with fixed values, while in [FLZ+10] the time complexity unit of the 2^{176} figure is not the same one as used in this paper. The guess-and-determine adversary has to guess a total of 175 bits of the internal state and works under one bit assumption, thus the inner deduction complexity for restoring the non-guessed part of the internal state has been absorbed and simplified into the time complexity unit. Thus, the actual time complexity of the guess-and-determine attack in [FLZ+10] is definitely much higher than 2^{176} when we take into account the detailed steps in the deduction phase. For the well-known time/memory/data tradeoff attack by Biryukov and Shamir in [BS00], if we take the typical point on the $TM^2D^2 = N^2$ tradeoff curve, i.e., $P = N^{2/3}$ preprocessing time, $T = N^{2/3}$ online attack time, $M = N^{1/3}$ memory space, and $D = N^{1/3}$ data for $N = 2^{320+64} = 2^{384}$ roughly, we have $T = 2^{256}$ which is $2^{121.2}$ times slower than the new FCA. Hence, our attack is the best known attack on Sosemanuk so far. More importantly, it indicates an inefficiency in longer keys than 135 bits and shows that the security margin of Sosemanuk is only 2^8 for the 128-bit security for the first time, which is somewhat contrary to the conclusion of the eSTREAM final report [BCC+08].

Since our work indicates and exploits some link between FCA, LC and LPN solvers, it is possible that more recent work on LPN solving can be integrated into a FCA framework and LC. Further, the new FCA framework has its own value in linear cryptanalysis of many symmetric key primitives as well. It is natural to investigate the immunity of many relevant symmetric key primitives against the newly evolved LC which includes a code-reduction procedure in appropriate parameters domain and to provide new provable security bounds

against the new form of LC.

Organization. This paper is organized as follows. We present the basic conceptions, notations and a brief review of fast correlation attacks, linear cryptanalysis and the related solving LPN issues in Section 2. In Section 3, we describe the general idea of the new algorithm, which will be more formally treated in Section 4 with complexity analysis. Then, we analyze Sosemanuk in Section 5 before Section 6 concludes the paper with some future work pointed out.

2 Preliminaries

2.1 Basic Notations

Denote the binary field by GF(2) and its m-dimensional extension field by GF(2^m). Let the usual xor operation be $\oplus$ and the inner product of two n-dimensional vectors $a = (a_{n-1}, \cdots, a_0)$ and $b = (b_{n-1}, \cdots, b_0)$ over GF(2^m) is $\langle a, b\rangle := \bigoplus_{i=0}^{n-1} a_i b_i$. We need the description of linear codes in general.

Definition 1. A $[n, k, d]$ linear code $\mathcal{C}$ over GF(2) is a subspace with dimension k of the vector space GF$(2)^n$, where n is the codeword length, k is the dimension of information bits and d is the minimum distance between any two codewords or the minimum weight of a non-zero codeword in $\mathcal{C}$, sometimes we refer to a $[n, k]$ linear code for simplicity.

A $[n, k]$ linear code $\mathcal{C}$ is usually described by its generator matrix $\mathbf{G}_{k\times n} = \big(\mathbf{g}_0, \ldots, \mathbf{g}_{n-1}\big)$ with $\mathbf{g}_i$ $(0 \le i \le n-1)$ being the $k \times 1$ column vector of $\mathbf{G}$ as

$$\mathbf{x} = (x_0, x_1, \ldots, x_{n-1}) = m \cdot \mathbf{G}_{k\times n} = \Big(\langle m, \mathbf{g}_0\rangle, \ldots, \langle m, \mathbf{g}_{n-1}\rangle\Big),$$

for all codeword $\mathbf{x} \in \mathcal{C}$, where $m = (m_0, m_1, \ldots, m_{k-1})_{1\times k}$ is the information vector and $\mathbf{G}$ is a $k \times n$ matrix whose rows form a vector space basis for $\mathcal{C}$, i.e., $\mathbf{G}$ is of full rank k. The choice of a basis in the vector space GF$(2)^n$ is not unique, thus code $\mathcal{C}$ has many different generator matrices which could be transformed into each other by elementary row operations in algebra. The systematic form of $\mathbf{G}$ is

$$\mathbf{G}_{sys} = \Big(I_{k\times k},\ A_{k\times(n-k)}\Big),$$

where $I_{k\times k}$ is the $k \times k$ identity matrix. Note that when $\mathbf{G}_{sys}$ is used to describe code $\mathcal{C}$, we have $x_i = m_i$ $(0 \le i \le k-1)$ for all codeword $\mathbf{x}$ and $\mathbf{G}_{sys}$ yields a permuted equivalent code. Dual to the generator matrix $\mathbf{G}$, a parity-check matrix $\mathbf{H}$ of code $\mathcal{C}$ is an $(n-k) \times n$ matrix satisfying $\forall \mathbf{x} \in \text{GF}(2)^n, \mathbf{x} \in \mathcal{C} \Leftrightarrow H\mathbf{x}^T = \mathbf{0}_{(n-k)\times 1}$, where $\mathbf{x}^T$ is the transpose of $\mathbf{x}$. If $\mathbf{G}$ has the systematic form as $\mathbf{G}_{sys}$, we have $H = \big(A^T, I_{(n-k)\times(n-k)}\big)$. Besides, the covering radius of code $\mathcal{C}$ is defined as

$$R(\mathcal{C}) = \max_{v\in GF(2)^n} \min_{\mathbf{x}\in\mathcal{C}} W_H(\mathbf{x} \oplus v),$$

where $W_H(\cdot)$ is the hamming weight of the argument. Thus, R is the smallest integer such that the union of Hamming spheres $B_R(\mathbf{x})$ around the codewords of $\mathcal{C}$ cover the entire vector space GF$(2)^n$. We have $R(\mathcal{C}) = \lfloor \frac{d-1}{2} \rfloor$ for a type of linear $[n, k, d]$ code $\mathcal{C}$, called perfect codes as discussed later in section 4.1, where $\lfloor\cdot\rfloor$ is the floor function. There are various ways to construct linear codes from known ones, among which the direct sum construction of linear codes is introduced as follows.

Definition 2. For linear codes $\mathcal{C}_1 = [n_1, k_1, d_1]$ and $\mathcal{C}_2 = [n_2, k_2, d_2]$, the direct sum of $\mathcal{C}_1$ and $\mathcal{C}_2$ is defined as $\mathcal{C}_1 \dot{+} \mathcal{C}_2 = \{(\mathbf{x}_1, \mathbf{x}_2) | \mathbf{x}_1 \in \mathcal{C}_1, \mathbf{x}_2 \in \mathcal{C}_2\}$. We have $\mathcal{C}_1 \dot{+} \mathcal{C}_2$ is a new $[n_1 + n_2, k_1 + k_2]$ linear code with radius $\lfloor \frac{d_1-1}{2} \rfloor + \lfloor \frac{d_2-1}{2} \rfloor$.

From Definition 2, the direct sum of two known codes is a new linear code constructed by concatenation, whose basic coding attribute can be determined accordingly.

Next, we will present some relevant facts of Boolean functions. A function $f : \mathrm{GF}(2)^n \to \mathrm{GF}(2)$ is called a Boolean function and a function $h = (h_1, \cdots, h_m) : \mathrm{GF}(2)^n \to \mathrm{GF}(2)^m$ when each h_i $(1 \le i \le m)$ is a Boolean function is defined as a m-dimensional vectorial Boolean function. Let X be a binary random variable, the correlation or bias of X is $c(X) = \Pr\{X = 0\} - \Pr\{X = 1\}$. Further, the correlation of a Boolean function f is defined as $c(f) = \Pr\{f(X) = 0\} - \Pr\{f(X) = 1\}$ with $X \in \mathrm{GF}(2)^n$ uniformly distributed. Given a vectorial Boolean function $h : \mathrm{GF}(2)^n \to \mathrm{GF}(2)^m$, define the distribution p_h of $h(X)$ with X uniformly distributed as $p_h(a) = \frac{\#\{X \in GF(2)^n | h(X)=a\}}{2^n}$ for all $a \in \mathrm{GF}(2)^m$. A natural measure of the bias of a distribution over a general alphabet is the Squared Euclidean Imbalance (SEI) [BJV04], which is also referred to as capacity in [HN10].

Definition 3. The SEI of a distribution p_h is $\Delta(p_h) = 2^m \sum_{a \in GF(2)^m} (p_h(a) - \frac{1}{2^m})^2$, which measures the distance between p_h and the uniform distribution over $GF(2)^m$.

SEI is used as a measure of the data complexity in attacks based on linear approximations over $\mathrm{GF}(2)^m$. In the bitwise case, we have $\Delta(p_h) = c^2(h)$ when $m = 1$. It is a well-known fact that the high-dimensional SEI $\Delta(p_h)$ will never be less than the low-dimensional counterpart, but we focus on the case of $m = 1$ in this paper. Further, the Walsh-Hadamard Transform plays the central role in linear cryptanalysis/correlation attacks to reduce the substitution complexity [CJM02].

Definition 4. For a function $f : \mathrm{GF}(2)^n \to \mathbf{R}$, where $\mathbf{R}$ is the set of real numbers, the Walsh-Hadamard Transform of f at point $\omega \in \mathrm{GF}(2^n)$ is defined as $\hat{f}(\omega) = \sum_{x \in GF(2^n)} f(x)(-1)^{\langle \omega, x \rangle}$.

The Walsh-Hadamard Transform of f can be computed efficiently with Fast Walsh-Hadamard Transform (FWHT), whose time complexity is $2^n + n2^n$ with 2^n memory, while the straight forward approach costs 2^{2n} time.

2.2 Correlation Attacks, Linear Cryptanalysis and Solving LPN

As stated, correlation attacks [Sie84, MS89] usually exploit the correlation between the keystream and the linear combinations of several LFSR sequences, which is directly applicable to memoryless combiners, but can be also generalized to combiners with memory [Gol96].

In most literature, the core problem is regarded as the decoding of a low-rate linear code transmitted through a binary channel, usually symmetric (BSC), with a low capacity, as depicted in Fig.1, where the nonlinearity introduced by the cipher can be typically represented as the noise from the channel. In Fig.1, the keystream z_t is regarded as the noisy version of the LFSR bit u_t and the aim is to reconstruct the original LFSR sequence from an given keystream segment. Since the seminal work of Meier and Staffelbach in [MS89] which exploits the low-weight linear relations, called parity-checks, among the codeword (LFSR) bits, huge efforts have been made to improve the performance and extend the application scope of fast correlation attacks [CT00, JJ99b, JJ99a, CJM02, ZXM15, CS91, TIM+18, MG90, MFI02, CJS01].

Linear cryptanalysis (LC) [Mat94] is a known-plaintext attack proposed by Matsui in 1993 to break DES, but can be seen as a more generic and closely related method to correlation attacks in symmetric key cryptanalysis. It has played an important role in block cipher domain, which first looks for bitwise linear approximations of the nonlinear components with a deviation from $\frac{1}{2}$ as much as possible and connect them together to build some probabilistic linear equations between several input/output bits and the key material, which can then be used to recover either 1 bit information of the key or part of

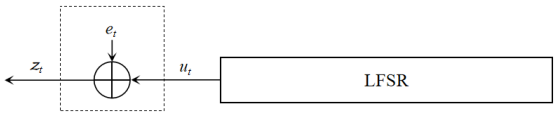

Figure 1: Model for a bitwise fast correlation attack

the last round key by statistical hypothesis test. It is easy to see that the key recovery routine is not necessarily be confined to the above cases. In general, we can loose the restrictions and try to establish a probabilistic linear system with the involved key bits as various variables, which is actually similar to fast correlation attacks [GBM02]. Thus, the final obstacle in LC is how to solve a noisy linear system over GF(2) as well.

The LPN problem is believed to be hard even given quantum computers, though no formal reduction from hard lattice problems exists (unlike the case of Learning with Errors (LWE) problem). Let $\mathtt{Ber}_\eta$ being the Bernoulli distribution with parameter η, i.e., if a random variable $e \leftarrow \mathtt{Ber}_\eta$ then $\Pr[e = 1] = \eta$ and $\Pr[e = 0] = 1 - \eta$ $(0 < \eta < \frac{1}{2})$, here instead of giving a more formal definition, we briefly recall the search LPN problem as follows.

Definition 5. For a secret vector $\mathbf{s} \in \mathrm{GF}(2)^k$, the adversary is given many pairs of the form $(\mathbf{g}, \langle s, \mathbf{g}\rangle \oplus e)$, where $\mathbf{g} \in \mathrm{GF}(2)^k$ is randomly generated and $e \leftarrow \mathtt{Ber}_\eta$, the task of the adversary is to recover $\mathbf{s}$ from the many given pairs.

Hence, if we take the secret vector $\mathbf{s}$ in LPN as the key material involved in LC of block ciphers, or the unknown initial state of the LFSR in correlation attacks on stream ciphers, then the above three problems actually result in the same mathematical model as solving a noisy system of linear equations over GF(2) whose bias is determined either by the linear approximations derived from the primitives or by the LPN oracle, given many observable noisy values of random linear combinations of the unknown variables, which is characterized as the data complexity in cryptanalysis.

The technique of covering codes plays an important role in the cryptographic hash function domain for memoryless near collision detecting [LR11, LMRS12, Leu14] and for solving LPN [GJL20, ZJW16], whose formal definition is as follows.

Definition 6. Some code $\mathcal{C}$ is a covering code if each vector in the vector space $\mathrm{GF}(2)^n$ is within a fixed distance $R_\mathcal{C}$ to some codeword $\mathbf{x}$ in $\mathcal{C}$, i.e., $\forall v \in \mathrm{GF}(2)^n$, $\exists\, \mathbf{x} \in \mathcal{C}$ such that $W_H(\mathbf{x} \oplus v) \leq R_\mathcal{C}$.

It is easy to see that a covering code provides a partition of the whole vector space $\mathrm{GF}(2)^n$, i.e., each random vector belongs to some Hamming sphere $B_R(\mathbf{x})$ around a codeword of the covering code $\mathcal{C}$. We will resort to this technique in some uniform case heavily hereafter, i.e., we will prefer to some uniform partition of the whole vector space when developing the new framework for FCAs.

3 A General Description of the New Algorithm

Fast correlation attacks consist of two phases: the pre-processing phase and the online decoding phase. For the attacks to be effective, the parity-check equations pre-computed have to match well with a series of incremental factors. First, they should be able to make a good use of the given linear correlation, which is crucial for the final complexity. This issue means that good parity-checks, if appropriately used, can exploit more information leakage represented by the correlation at a reasonable expense of time/memory/data

complexities. Second, the parity-checks should be specially tailored for the subsequent decoding procedure, i.e., the form of the parity-checks is usually determined by the solving method.

3.1 Sketch of Procedures in Fast Correlation Attacks

We first review the essential procedures used in non-iterative fast correlation attacks. The starting point is to look at the generator matrix $\mathbf{G}$ of the LFSR $[N, k]$ linear code where N is the length of the codeword and k is the dimension of the information, i.e., the LFSR length. Let $\mathbf{u} = (u_0, u_1, \cdots, u_{N-1})$, we have

$$\mathbf{u} = (u_0, u_1, \cdots, u_{k-1}) \cdot \mathbf{G} = (u_0, u_1, \cdots, u_{k-1}) \cdot \Big(\mathbf{g}_0, \mathbf{g}_1, \cdots, \mathbf{g}_{N-1}\Big), \tag{1}$$

where $\mathbf{g}_i = (\mathrm{g}_0^i, \mathrm{g}_1^i, \cdots, \mathrm{g}_{k-1}^i)^T$ is the i-th column vector. Given the model in Fig.1, we regard the column vectors $\mathbf{g}_i$ as random vectors over $\mathrm{GF}(2)^k$, though they are in fact algebraically related to each other. This is a reasonable assumption adopted in many literatures, which provides reliable theoretical results conforming to experiments. The parity-checks are usually constructed as

$$\bigoplus_{j=0}^{v-1} u_{i_j} = (u_0, u_1, \cdots, u_{k-1}) \cdot \bigoplus_{j=0}^{v-1} \mathbf{g}_{i_j} = \bigoplus_{j=0}^{k_1-1} c_j u_j,$$

where $\bigoplus_{j=0}^{v-1} \mathbf{g}_{i_j} = (c_0, c_1, \cdots, c_{k_1-1}, 0, \cdots, 0)^T$ with $k_1 < k$. Thus, the parity-checks are prepared as a linear combination of some LFSR bits, e.g., $\{u_{i_j}\}_{j=0}^{v-1}$ and some partial target bits, i.e., $\{u_j\}_{j=0}^{k_1-1}$ in the initial state of the LFSR. Once pre-computed, these linear equations are numerically evaluated on the noisy keystream bit z_t to gather the information leakage for the LFSR bits from this operation. Note that the above procedure is similar to the BKW collision in LPN solvers [GJL20], and the new noise variable is $e_1^1 = \bigoplus_{j=0}^{v-1} e_{i_j}$ with a bias ϵ^v given $\Pr\{e_{i_j} = 0\} = \frac{1}{2}(1+\epsilon)$. When the number of the parity-checks is statistically large enough to demonstrate the folded bias, some prediction can be made by a majority poll on the target bits in $(u_0, u_1, \cdots, u_{k_1-1})$. The evaluation of these parity-checks requires a complexity of $2^{k_1} N_1$ if this step is executed in a straight forward way, where N_1 is the cardinality of the parity-checks group on these target bits. Thanks to FWHT, this evaluation complexity can be reduced to $N_1 + 2^{k_1} k_1$ to fulfill the same task, which is detailed in Appendix A. This process can be seen as some decoding procedure of the newly derived $[N_1, k_1]$ linear code, which directly gives some partial information on the initial state of the LFSR in Fig.1. After restoring the first part of the LFSR initial state, the other parts of the LFSR initial state can be determined in a similar way with a much lower complexity. The required keystream length and the time complexity of the whole attack mainly depends on the absolute value of the finally derived correlation inherent in the decoding of the $[N_1, k_1]$ linear code.

3.2 New Algorithmic Procedures and their Integration

To facilitate the description of the new algorithmic procedures, we first go back to Figure 1. From Eq.(1), we have

$$\mathbf{z} = (z_0, z_1, \cdots, z_{N-1}) = (u_0, u_1, \cdots, u_{k-1}) \cdot \mathbf{G} \oplus \mathbf{e}, \tag{2}$$

where $\mathbf{e} = (e_0, e_1, \cdots, e_{N-1})$ is the noise vector with $\Pr\{e_i = 0\} = \frac{1}{2}(1+\epsilon)$ $(\epsilon > 0)$.

Gauss Elimination. As in the LPN solvers [GJL20, ZJW16], Gauss elimination can be adopted here to transform the distribution of the secret initial state $(u_0, u_1, \cdots, u_{k-1})$ of the

LFSR into the same distribution as that of the noise variables e_i. Precisely, the adversary first rewrites the matrix $\mathbf{G}$ into the systematic form through elementary row operations. Usually, in the LFSR case, this step is naturally fulfilled as can be seen from Eq.(1), i.e., $\mathbf{G} = \Big(I_{k\times k}, A_{k\times(n-k)}\Big)$. Then define $\hat{\mathbf{u}} = (u_0, u_1, \cdots, u_{k-1}) \oplus (z_0, z_1, \cdots, z_{k-1})$, now the adversary can derive that

$$\begin{aligned}\hat{\mathbf{z}} &= \mathbf{z} \oplus (z_0, z_1, \cdots, z_{k-1}) \cdot \mathbf{G} = (\mathbf{0}_{1\times k}, \hat{z}_{k+1}, \hat{z}_{k+2}, \cdots, \hat{z}_{N-1}), \\ &= (u_0, u_1, \cdots, u_{k-1}) \cdot \mathbf{G} \oplus \mathbf{e} \oplus (z_0, z_1, \cdots, z_{k-1}) \cdot \mathbf{G} \\ &= \hat{\mathbf{u}} \cdot \mathbf{G} \oplus \mathbf{e},\end{aligned} \tag{3}$$

where $\hat{\mathbf{u}} = (\hat{u}_0, \hat{u}_1, \ldots, \hat{u}_{k-1})$ is the new initial state whose distribution is $\Pr(\hat{u}_i = 0) = \Pr(e_i = 0) = \frac{1}{2}(1+\epsilon)$ from $\mathbf{0}_{1\times k}$ for $0 \le i \le k-1$. Since $(z_0, z_1, \cdots, z_{k-1})$ are concrete values (not random variables) when online and $\mathbf{G}$ has the systematic form, the first k bits of $\hat{\mathbf{z}}$ are all 0s, thus for each $\hat{u}_i$ $(0 \le i \le k-1)$, the event that $\{\hat{u}_i = 0\}$ implies the event that $\{e_i = 0\}$, which results in the distribution given above. As can be seen, Eq.(3) is obtained via a pure algebraic transformation, i.e., variable substitution, now the adversary can analyze the model in Eq.(3) whose secret vector has a biased distribution as the noise variable to derive information on the original initial state. We believe that the biased distribution will facilitate the partial exhaustive search of the new initial state with a lower complexity than the uniform case to a large extent and enlarge the usable parity-checks by the subsequent decoding.

BKW Collision-reduction. As stated just now, the preparation of parity-checks is very similar to the BKW collision procedure [BKW00] in LPN solvers, which solves LPN in sub-exponential time. In fact, the BKW procedure adopts an iterative sort-and-match procedure on the columns of the involved matrix to iteratively and effectively reduce the information dimension. We are interested in the reduction phase of the BKW algorithm in which the adversary iteratively searches for all the combinations of two columns in the current matrix that xor to zero in the last b bits. Given the $k \times n$ matrix $\mathbf{G}$, assume that we find two columns $\mathbf{g}_{i_0}$ and $\mathbf{g}_{i_1}$ satisfying

$$\mathbf{g}_{i_0} \oplus \mathbf{g}_{i_1} = (*, *, \cdots, *, \underbrace{0, 0, \cdots, 0}_{b}),$$

where $*$ denotes any value of GF(2), then it is said that $\mathbf{g}_{i_0}$ and $\mathbf{g}_{i_1}$ belong to the same partition or equivalence class. The concrete process of BKW reduction is shown in Algorithm 1 below, where two methods, LF1 and LF2, are described which only differ in how to merge and choose the columns and t is the number of reduction rounds.

Algorithm 1 BKW Reduction

Input: The matrix $\mathbf{G}_0 = \mathbf{G} = [\mathbf{g}_0, \mathbf{g}_1, \ldots, \mathbf{g}_{n-1}]$, the parameters t and b
Online: Reduction phase in BKW algorithm
1: **for** $i = 1$ **to** t **do**
2: Partition the columns of $\mathbf{G}_{i-1}$ according to the last $b \cdot i$ bits
Form pairs of the columns in each partition to obtain $\mathbf{G}_i$
2a: **LF1.** Partition $\mathbf{G}_{i-1} = V_0 \cup V_1 \cup \cdots \cup V_{2^b-1}$.
Randomly choose $\mathbf{v}^* \in V_j$ as the representative of V_j.
For each $\mathbf{v} \in V_j, \mathbf{v} \neq \mathbf{v}^*$, $\mathbf{G}_i = \mathbf{G}_i \cup (\mathbf{v} \oplus \mathbf{v}^*)$
2b: **LF2.** Partition $\mathbf{G}_{i-1} = V_0 \cup V_1 \cup \cdots \cup V_{2^b-1}$.
For each pair $(\mathbf{v}, \mathbf{v}') \in V_j, \mathbf{v} \neq \mathbf{v}'$, $\mathbf{G}_i = \mathbf{G}_i \cup (\mathbf{v} \oplus \mathbf{v}')$
Output: The matrix $\mathbf{G}_t$

Note that when we merge the two columns $\mathbf{g}_{i_0}$ and $\mathbf{g}_{i_1}$ in each iteration, the last b bits of 0 could be ignored in Algorithm 1. There are two routines, LF1 and LF2, involved in BKW reduction: the spirit of which are the same except the way how we choose and merge the

selected columns in each partition. LF1 randomly selects one column in each partition as the representative and then xor it with the remaining columns in V_j; while LF2 merges all pairs within the same partition V_j. Thus, LF2 could produce more data samples at the cost of some increased dependency which can be safely ignored according to experiments in [GJL20, ZJW16, LF06]. Further, the dependency does not affect the performance of the subsequent decoding. In some cases, this little dependency will facilitate the statistical decoding, as here the dependency is statistically linear dependency not the usual algebraic dependency [GBM02, LZFV13]. To fully exploit the given data samples, we usually take LF2 in the BKW collision-reduction based on the following fact.

Proposition 1 (Completeness)**.** *Given the matrix* $\mathbf{G}$*, one round of LF2 in BKW collision reduction can find all pairs of columns in* $\mathbf{G}$ *that xor to* 0 *in the last* b *bits.*

Depending on the concrete scenario, $t \geq 1$ rounds of LF2 may be applied to the matrix $\mathbf{G}$, as illustrated in Algorithm 1. It is worth noting that the BKW reduction works on a single list of the columns, i.e., there is no division operation on the matrix columns; while the method based on the generalized birthday problem in [Wag02] and adopted in [ZXM15] splits the whole set of columns in $\mathbf{G}$ into 2 or 4 sub-lists, which will lose quite some solutions to the real single-list κ-sum problem when the arguments are from the same sub-list.

Code-reduction. Another algorithmic procedure is the code-reduction that reduces the dimension of the secret information without having the piling-up lemma [Mat94] penalty of the folded noise. Essentially, this procedure rewrites each column of the resultant matrix $\mathbf{G}_t$ from some BKW reduction consisting of several rounds or with different values of v, as a sum of some codeword in a linear covering code and some sparse vectors easy to use in distinguishing process. The adversary rearrange these columns according to their nearest codewords and derive a new smaller linear code $[N_c, k_c]$ with $k_c < k_1$ in the $[N_1, k_1]$ linear code. Assume this procedure starts with some matrix $\mathbf{G}_t$ which can be regarded as certain random linear code, where we simply follow the matrix $\mathbf{G}$ as defined in Eq.(3). Precisely,

$$\begin{aligned}\mathbf{G}_t &= \left(I^{(t)}_{k_1\times k_1}, A^{(t)}_{k_1\times(N_1-k_1)}\right) = \left(I^{(t)}_{k_1\times k_1}, \mathbf{g}^{(t)}_{k_1+1}, \ldots, \mathbf{g}^{(t)}_{N_1-1}\right) \\ &= \left(I^{(t)}_{k_1\times k_1}, c_{k_1+1} \oplus \bar{e}_{k_1+1}, \ldots, c_{N_1-1} \oplus \bar{e}_{N_1-1}\right),\end{aligned} \tag{4}$$

where $\mathbf{g}_i = \mathbf{c}_i \oplus \bar{e}_i$ $(k_1 + 1 \leq i \leq N_1 - 1)$ with $\mathbf{c}_i$ being the nearest codeword from a $[k_1, k_c, d_c]$ covering code. Thus, it is natural that $W_H(\bar{e}_i) \leq R_c$, where $R_c = \lfloor \frac{d_c-1}{2} \rfloor$. Substituting Eq.(4) into Eq.(3), we have

$$\hat{\mathbf{z}} = \hat{\mathbf{u}} \cdot \mathbf{G} \oplus \mathbf{e} = \hat{\mathbf{u}} \cdot \left(I_{k_1\times k_1}, \mathbf{c}_{k_1+1} \oplus \bar{e}_{k_1+1}, \ldots, \mathbf{c}_{N_1-1} \oplus \bar{e}_{N_1-1}\right) \oplus \mathbf{e}, \tag{5}$$

where for each $i \geq k_1 + 1$, we have $\hat{\mathbf{z}}_i = \langle \hat{\mathbf{u}}, (\mathbf{c}_i \oplus \bar{e}_i)\rangle \oplus \mathbf{e}_i = \langle \hat{\mathbf{u}}, \mathbf{c}_i\rangle \oplus \langle \hat{\mathbf{u}}, \bar{e}_i\rangle \oplus \mathbf{e}_i$. Note that the term $\langle \hat{\mathbf{u}}, \bar{e}_i\rangle \oplus \mathbf{e}_i$ is the new noise variable introduced from this code-reduction procedure. Since k_c is usually much less than k_1 and is controllable by the adversary to a large extent, the dimension has been reduced a lot at the cost of the above newly introduced noise variable which may provide some trade-off beyond the piling-up lemma.

Based on the above three algorithmic procedures, by a flexible combination of the code-reduction with the BKW collision-reduction which is called dimension reduction hereafter, we can have a new high-level framework for fast correlation attacks as depicted below.

Algorithm 2 Improved FCA

Input: A keystream $\mathbf{z} = (z_0, z_1, \ldots, z_{k-1}, z_k, \ldots, z_{N-1})$
Online: Recover (partial) LFSR initial state which is consistent with $\mathbf{z}$
1: Apply Gauss elimination to derive the equivalent model in Eq.(3)
2: Select a dimension reduction strategy from Section 4 and build a new linear code accordingly
3: Decode the new code via FWHT
Output: The partial LFSR initial state involved in the new code

Compared with the previous framework for FCA, e.g., in [ZXM15], which only exploit the collision reduction to derive the required parity-checks, Algorithm 2 offers new space and freedom for the adversary to construct the targeted linear code whose decoding will provide the partial information of the original LFSR initial state. We will enrich the framework in Algorithm 2 with the corresponding technical details in the next section.

4 New Algorithms Based on Dimension Reduction

4.0.1 Overview.

We first show the overview before we detail the new algorithm. From Eq.(5), we have the equivalent model that is under consideration. There are some strategies in front of us to fulfil the dimension reduction task in Step 2 of Algorithm 2, from which we describe two candidate routines as follows.

- We first construct parity-check equations in the BKW collision reduction or in the generalized birthday problem [Wag02] manner as shown in Section 3.1, which are associated with $\bigoplus_{j=0}^{v-1} \mathbf{g}_{i_j}$ and $\bigoplus_{j=0}^{v-1} \mathbf{z}_{i_j}$. Then, the code-reduction procedure is conducted on the resultant smaller-size random linear code from the previous dimension reduction, to further lower the dimension of the remaining information as shown in Section 3.2 and below in details. We do not impose any restrictions on the number of times that a special reduction method is performed, i.e., it may be possible that there are not only one time of BKW or code-reduction in this routine before deriving the final linear code for the retrieval of the partial initial state of LFSR in Fig.1, which is mainly dependent on the involved parameter domain.

- We first perform the code-reduction procedure on the equivalent model to reduce the dimension, which may be favorable in some noise/secret information dimension parameter domain. Similarly, the number of times that the code-reduction conducted is unnecessarily restricted to one. Then we arrange the BKW reduction on the linear code by random collisions on the columns of its generator matrix. Again, we can allow several rounds of BKW reduction after the covering code procedure. The number of parity-checks constructed is the final code length and the remaining secret bits constitute the partial state information to be restored from decoding.

Actually, there are also some other combination approaches for the dimension reduction. For example, we can conceive the alternating routine, i.e., BKW reduction and covering code reduction are alternatingly executed for a very limited, but deliberately determined number of times until we reach some complexity trade-off that is better for the adversary to decode. In the typical parameter domain considered, we will not go into that direction and the above second direction further. After the dimension reduction, we use FWHT to get solutions with high correlation. In other words, we evaluate all the parity-check equations for all the possible values of the remaining information bits and maintain a list of candidates that have a high ranking according to the FWHT spectrum.

4.1 Detailed Algorithm

We will describe Algorithm 2 with the first routine, for we usually face the high noise and high secret dimension parameter domain in cryptanalysis. From the equivalent model after Gauss elimination, we have $\hat{\mathbf{z}} = \hat{\mathbf{u}} \cdot \mathbf{G} \oplus \mathbf{e}$ with $\Pr(\hat{u}_i = 0) = \frac{1}{2}(1+\epsilon)$ for $0 \le i \le k-1$. In order to have an efficient attack, the method illustrated in Section 3.1 is combined with a small-scale exhaustive search on the LFSR initial state $(\hat{u}_0, \hat{u}_1, \ldots, \hat{u}_{k-1})$, which is called the parity-checks with memory hereafter. Precisely, for the LFSR of length k, a small integer number of w bits $(\hat{u}_{k-w}, \ldots, \hat{u}_{k-1})$ of the initial state are guessed and the remaining $k-w$ bits $(\hat{u}_0, \hat{u}_1, \ldots, \hat{u}_{k-w-1})$ are to be determined later. We first look for some v-tuple column vectors $(\mathbf{g}_{i_0}, \mathbf{g}_{i_1}, \ldots, \mathbf{g}_{i_{v-1}})$ such that

$$\bigoplus_{j=0}^{v-1} \hat{u}_{i_j} = (\hat{u}_0, \hat{u}_1, \cdots, \hat{u}_{k-1}) \cdot \bigoplus_{j=0}^{v-1} \mathbf{g}_{i_j} = \bigoplus_{j=0}^{k_1-1} c_j \hat{u}_j \oplus \bigoplus_{j=w}^{1} c_{k-j} \hat{u}_{k-j},$$

where $\bigoplus_{j=0}^{v-1} \mathbf{g}_{i_j} = (c_0, c_1, \cdots, c_{k_1-1}, 0, \cdots, 0, \underbrace{*, \ldots, *}_{w})^T$ with $*$ being an arbitrary value in GF(2) and $k_1 < k-w$. Accordingly, we have

$$\bigoplus_{j=0}^{v-1} \hat{z}_{i_j} = (\hat{u}_0, \hat{u}_1, \cdots, \hat{u}_{k-1}) \cdot \bigoplus_{j=0}^{v-1} \mathbf{g}_{i_j} \oplus \bigoplus_{j=0}^{v-1} \mathbf{e}_{i_j} = \bigoplus_{j=0}^{k_1-1} c_j \hat{u}_j \oplus \bigoplus_{j=w}^{1} c_{k-j} \hat{u}_{k-j} \oplus \bigoplus_{j=0}^{v-1} \mathbf{e}_{i_j},$$

where $\Pr\{\bigoplus_{j=0}^{v-1} \mathbf{e}_{i_j} = 0\} = \frac{1}{2}(1+\epsilon^t)$. To simplify the notations, let $\mathbf{z}'_i = \bigoplus_{j=0}^{v-1} \hat{z}_{i_j} \oplus \bigoplus_{j=w}^{1} c_{k-j} \hat{u}_{k-j}$, $\mathbf{g}'_i = \bigoplus_{j=0}^{v-1} \mathbf{g}_{i_j}$ the truncated $k_1 \times 1$ column vector and $\mathbf{e}'_i = \bigoplus_{j=0}^{v-1} \mathbf{e}_{i_j}$ and taking the form of $\mathbf{g}'_i$ into account, we have

$$\mathbf{z}'_i = (\hat{u}_0, \hat{u}_1, \cdots, \hat{u}_{k_1-1}) \cdot \mathbf{g}'_i \oplus \mathbf{e}'_i, \tag{6}$$

for $0 \le i \le m_v - 1$, if we have found m_v similar equations like Eq.(6). This means that we have derived a $[m_v, k_1]$ linear code with its noisy transmission version $\{\mathbf{z}'_i\}_{i=0}^{m_v-1}$. The time/memory complexities are $2^w \cdot m_v^{\frac{1}{1+\lfloor \log v \rfloor}} v \cdot 2^{\frac{k-k_1}{1+\lfloor \log v \rfloor}}$ and $m_v^{\frac{1}{1+\lfloor \log v \rfloor}} v \cdot 2^{\frac{k-k_1}{1+\lfloor \log v \rfloor}}$, respectively from Wagner's k-tree algorithm [Wag02] with the relevant list of size $m_v^{\frac{1}{1+\lfloor \log v \rfloor}} \cdot 2^{\frac{k-k_1}{1+\lfloor \log v \rfloor}}$ if the condition $m_v \le 2^{\frac{k-k_1}{1+\lfloor \log v \rfloor}}$ holds. Unlike the original result in [Wag02], there is a 2^w factor in the above time complexity due to the partial exhaustive search of the initial state of the involved LFSR. Note that we do not need such a factor in the memory complexity, as the adversary could re-use the same memory stack for all the possible values of $(\hat{u}_{k-w}, \ldots, \hat{u}_{k-1})$ by an eraser. On the other hand, we will formally adopt Algorithm 1 of the BKW collision in this setting when v is a small integer like $v = 2$ or $v = 4$, for BKW reduction can make full use of the data samples to generate the parity-checks.

If the code length is N_1, then we will have about $\frac{\binom{N_1}{2}}{2^b}$ parity-checks for one collision-reduction round. For t collision-reduction rounds, the BKW collision-reduction procedure can be sped up with a time complexity of $\sum_{i=1}^{t} \lceil \frac{N_1 - ib}{f} \rceil N_1$ after a pre-computation of $f2^{f-1}(2^f - 2)$, given $\frac{3}{2} f \cdot 2^{f-1}(2^f - 2)$ memory by pre-computing some tables of size around 2^{2f} to store the xor of all the possible combinations of two f-bit vectors [ZJW16]. Further, under some parameter settings, we need to optimize the complexity of reading these pre-computed tables with a constant complexity, which is achieved by dividing some relatively large value of f into some small factors such that the pre-computed table after factoring will be of practical size and could be read with a constant complexity. This is a common assumption in modern cryptanalysis literature that the adversary can read such small tables (not large tables) with a constant-time complexity. We will illustrate this issue in more details in Section 5.

Next, we make the code-reduction on this $[m_v, k_1]$ linear code. Let $\mathbf{z}' = (\mathbf{z}'_0, \mathbf{z}'_1, \cdots, \mathbf{z}'_{m_v-1})_{1\times m_v}$, $\hat{\mathbf{u}} = (\hat{u}_0, \cdots, \hat{u}_{k_1-1})_{1\times k_1}$, $\mathbf{G}' = \left(\mathbf{g}'_0, \mathbf{g}'_1, \cdots, \mathbf{g}'_{m_v-1}\right)_{k_1\times m_v}$ and $\mathbf{e}' = (\mathbf{e}'_0, \ldots, \mathbf{e}'_{m_v-1})_{1\times m_v}$, first rewrite this code in matrix form as

$$\mathbf{z}' = \hat{\mathbf{u}} \cdot \mathbf{G}' \oplus \mathbf{e}'. \tag{7}$$

Our aim is to construct a $[k_1, k_c]$ linear code $\mathcal{K}$ with covering radius d_c to regroup the columns in $\mathbf{G}'$, i.e., express $\mathbf{g}'_i = \mathbf{c}_i \oplus \bar{e}_i$ where $\mathbf{c}_i \in \mathcal{K}$ is the nearest codeword to the random column vector $\mathbf{g}'_i$, i.e., $W_H(\bar{e}_i) = W_H(\mathbf{g}'_i \oplus \mathbf{c}_i) \le d_c$. In the real-world cryptanalysis setting, the remaining dimension k_1 is still of a large size after BKW collision reduction, e.g., $k_1 > 100$, it is inconvenient to directly select a linear code of such a word length. Instead, we can build the new desirable linear code from some known good codes according to some well-understood approach such as the direct sum method, as in Definition 2 and the following Corollary.

Corollary 1. *For l linear codes $\mathcal{C}_i = [n_i, k'_i, d_i]$ $(1 \le i \le l)$, the direct sum of $\mathcal{C}_1$ to $\mathcal{C}_l$ is defined as $\mathcal{C}_1 \dot{+} \mathcal{C}_2 \dot{+} \cdots \dot{+} \mathcal{C}_l = \{(c_1, c_2, \cdots, c_l) | x_i \in \mathcal{C}_i, 1 \le i \le l\}$. Then $\mathcal{C}_1 \dot{+} \mathcal{C}_2 \dot{+} \cdots \dot{+} \mathcal{C}_l$ is a new linear code $[\sum_{i=1}^{l} n_i, \sum_{i=1}^{l} k'_i]$ with the radius $\sum_{i=1}^{l} \lfloor \frac{d_i-1}{2} \rfloor$.*

From Corollary 1, the new linear code constructed is also a linear code [CHLL05], which can be characterized by its generator matrix $\mathbf{G}'_{sys} = \left(I_{k_c\times k_c}, A_{k_c\times(k_1-k_c)}\right)$ and the corresponding parity-check matrix $\mathbf{H} = \left(A^T, I_{(k_1-k_c)\times(k_1-k_c)}\right)$, where $k_c = \sum_{i=1}^{l} k'_i$ and $k_1 = \sum_{i=1}^{l} n_i$. The operation of regrouping the $\mathbf{g}'_i$s according to the constructed linear code $\mathcal{K}$ can be fulfilled by the well-known syndrome-decoding in coding theory.

Syndrome Decoding. The table-based syndrome decoding is a generic method for decoding linear codes. There are two columns in the prepared table: one stores the syndrome $\mathbf{H} \cdot \mathbf{g}'^T_i$ and the other is the corresponding minimum-weight error vector. When decoding, one just computes the syndrome from the received vector, look up the pre-computed table to find a match and add the corresponding error vector with the received vector to get the nearest codeword. Coming back to our context, a constant-time accessible table $\mathtt{T}_c$ of $2^{k_1-k_c}$ items is pre-computed with each item storing the syndrome $\mathbf{H} \cdot \mathbf{g}'^T_i$ and its minimum-weight error vector. Then in the online phase, if a concrete syndrome is computed, the adversary can make a table look-up of the pre-computed table $\mathtt{T}_c$ with a constant complexity, find the corresponding error vector and xor them together to get the nearest codeword $\mathbf{c}_i$. Since we adopt the direct sum construction of covering codes and each employed code is of practical size, the syndrome table $\mathtt{T}_c$ can be assumed as being accessible with a constant time complexity. Further, we will resort to this table-based technique heavily in the attack on Sosemanuk in Section 5.2, for which we also build pre-computed tables of small size which can be read with a constant complexity.

Since we usually build the syndrome-decoding table $\mathtt{T}_c$ for linear codes of a practically small codeword size, e.g., $n_i \le 23$, the time complexity of pre-computing these tables are relatively low and can be negligible compared to those of the other procedures in the algorithm. Thus, we have

$$\begin{aligned}\mathbf{z}'_i &= \langle \hat{\mathbf{u}}, \mathbf{c}_i \rangle \oplus \langle \hat{\mathbf{u}}, \bar{e}_i \rangle \oplus \mathbf{e}_i = \langle \hat{\mathbf{u}}, x_i \mathbf{G}'_{sys} \rangle \oplus \langle \hat{\mathbf{u}}, \bar{e}_i \rangle \oplus \mathbf{e}_i \\ &= \langle \hat{\mathbf{u}} \mathbf{G}'^T_{sys}, x_i \rangle \oplus \langle \hat{\mathbf{u}}, \bar{e}_i \rangle \oplus \mathbf{e}_i.\end{aligned}$$

Let $\tilde{e}_i = \langle \hat{\mathbf{u}}, \bar{e}_i \rangle \oplus \mathbf{e}_i$ be the new noise variable whose average bias can be computed as described below. We rewrite the above equation as

$$\mathbf{z}'_i = \langle \mathbf{y}, x_i \rangle \oplus \langle \hat{\mathbf{u}}, \bar{e}_i \rangle \oplus \mathbf{e}_i,$$

where $\mathbf{y} = \hat{\mathbf{u}}\mathbf{G}'^{T}_{sys}$ is the derived information vector after code-reduction. Although the code-reduction is not one-to-one invertible, the adversary can still reveal quite some information of the initial state, please refer to section 5.5 in [GJL20] for this issue.

For the new noise, we first treat the error $\mathbf{e}_i$ and $\langle \hat{\mathbf{u}}, \bar{e}_i \rangle$ as independent noise variable which can be combined by the piling-up lemma. From the direct sum construction in Corollary 1, we also regard the error sub-vector $\langle \hat{\mathbf{u}}_j, \bar{e}_{i,j} \rangle$ oriented from different segments of $\mathbf{c}_i$, i.e., $c_{i,j}$, for $1 \le j \le l$ as independent noise variables and we have

$$\langle \hat{\mathbf{u}}, \bar{e}_i \rangle = \bigoplus_{j=1}^{l} \langle \hat{\mathbf{u}}_j, \bar{e}_{i,j} \rangle.$$

Hence, it suffices to compute the noise introduced from $\langle \hat{\mathbf{u}}_j, \bar{e}_{i,j} \rangle$, and combine them together as independent variables. In other words, we need to compute the bias ϵ_j corresponding to a linear covering code $[n_j, k'_j, d_j]$ for $1 \le j \le l$. In general, there are two ways to fulfill this task. One is to compute the averaged value of bias ϵ_j which has already been verified in many experiments including [GJL20]; the other is the bias analysis conditioned on the partial key information, as in [GJL20], which yields a bit conservative estimate compared to the experiments and the former. Thus, we select the averaged bias analysis to deal with ϵ_j under some reasonable independent assumption, i.e., we adopt the well-known random code assumption.

Precisely, we have the bias $\epsilon_j = 1 - 2\Pr\{\langle \hat{\mathbf{u}}_j, \bar{e}_{i,j} \rangle = 1\}$ and the event $\langle \hat{\mathbf{u}}_j, \bar{e}_{i,j} \rangle = 1$ means that for the support set of $\bar{e}_{i,j}$, there is an odd number of the coordinate positions in the corresponding $\hat{\mathbf{u}}_j$ taking the value 1, i.e.,

$$\begin{aligned}
&\Pr\{\langle \hat{\mathbf{u}}_j, \bar{e}_{i,j} \rangle = 1\} \\
&= \sum_{\substack{t=1 \\ t\&\texttt{0x1}=1}}^{d_c} \binom{n_j}{t} \Pr^t\{\hat{\mathbf{u}}_j^t \cdot \bar{e}_{i,j}^t = 1\}(1 - \Pr\{\hat{\mathbf{u}}_j^t \cdot \bar{e}_{i,j}^t = 1\})^{n_j - t} \\
&= \sum_{\substack{t=1 \\ t\&\texttt{0x1}=1}}^{d_c} \binom{n_j}{t} \Pr^t\{\hat{\mathbf{u}}_j^t = 1\}\Pr^i\{\bar{e}_{i,j}^t = 1\}(1 - \Pr\{\hat{\mathbf{u}}_j^t = 1\}\Pr\{\bar{e}_{i,j}^t = 1\})^{n_j - t},
\end{aligned} \tag{8}$$

where $\Pr\{\hat{\mathbf{u}}_j^t \cdot \bar{e}_{i,j}^t = 1\} = \Pr\{\hat{\mathbf{u}}_j^t = 1\}\Pr\{\bar{e}_{i,j}^t = 1\}$ is the probability that at each coordinate position t,$(1 \le t \le n_j)$, the event $\hat{\mathbf{u}}_j^t \cdot \bar{e}_{i,j}^t = 1$ holds for the chosen covering code. It is known from coding theory that the covering spheres centered at each codeword are disjoint for perfect codes and all the random vectors distribute among these covering spheres uniformly, i.e., if the minimum weight of the considered perfect code is d_p, then each random vector of the same bit-length has a distance less than or equal to $\lfloor \frac{d_p - 1}{2} \rfloor$ to exactly one codeword.

Algorithm 3 Computing $\Pr\{\bar{e}_{i,j}^t = 1\}$ for a fixed position t

Parameter: $[n_j, k'_j, d_j]$ code $\mathcal{C}_j$ with its parity-check matrix $\mathbf{H}$, t

Initialize the array $A[n_j] = \{0\}$
for each $0 \le x \le 2^{n_j} - 1$ **do**
    compute the syndrome $\mathbf{H} \cdot x^t$
    decode x into the nearest codeword $\mathbf{c}_x$ by syndrome-decoding
    $A[x] = x \oplus c_x$
Initialize a counter $c = 0$
for each $0 \le x \le 2^{n_j} - 1$ **do**
    if $A[x]_t = 1$ **then** $c \leftarrow c + 1$

Output: $\Pr\{\bar{e}_{i,j}^t = 1\} = \frac{c}{2^{n_j}}$

Table 1: All the binary perfect codes with the codeword/information length and the covering radius

Code	Codeword	Information	Covering radius
`Hamming`	2^i-1	2^i-i-1	1
`Repetition`	$2i+1$	1	i
`Golay`	23	12	3
$\{0\}^i$	i	0	i
$\{0,1\}^i$	i	i	0

Table 2: Coordinate distribution in the error vector of the Repetition Codes

`Repetition` Codes							
i	2	3	4	5	6	7	8
$\Pr\{\bar{e}_{i,j}^t=1\}$	$\frac{5}{16}$	$\frac{11}{32}$	$\frac{95}{256}$	$\frac{193}{512}$	$\frac{793}{2048}$	$\frac{1619}{4096}$	$\frac{26333}{65536}$

It is well-known that there are very limited types of perfect codes in binary domain, which is listed in Table 1 with the corresponding basic attributes. We need to compute $\Pr\{\bar{e}_{i,j}^t=1\}$ for each t and for each perfect code in Table 1. Given a code $\mathcal{C}_j$, we have devised Algorithm 3 to fulfill this task. We have run Algorithm 3 for each perfect code in Table 1 and found that for a $[2^i-1, 2^i-i-1, 3]$ Hamming code, we have $\Pr\{\bar{e}_{i,j}^t=1\}=\frac{1}{2^i}$ for $1\le t\le 2^i-1$; for the $[23,12,7]$ Golay code and each t-th coordinate in the error vector, we have $\Pr\{\bar{e}_{i,j}^t=1\}=\frac{127}{1024}$. Table 2 shows the relevant value for a repetition code of some relevant sizes. Accordingly, we can come back to Eq.(8) with the corresponding probability $\Pr\{\bar{e}_{i,j}^t=1\}$.

Now given $[k_1,k_c]$, we need to explicitly configure the direct sum $\mathcal{C}_1\dot{+}\mathcal{C}_2\dot{+}\cdots\dot{+}\mathcal{C}_l$ which will yield the maximized bias after code-reduction. In other words, with a given $\Pr\{\hat{\mathbf{u}}_j^t=1\}$, let $\epsilon^i_{\texttt{Hamming}}$ be the bias of a $[2^i-1,2^i-i-1,3]$ Hamming code for such i that $2^i-1\le k_1$ and $2^i-i-1\le k_c$, $\epsilon^i_{\texttt{Repet}}$ be the bias of a $[2i+1,1,2i+1]$ Repetition code for such i that $2i+1\le k_1$, $\epsilon_{\texttt{Golay}}$ be the bias of the $[23,12,7]$ Golay code, $\epsilon_{\{0\}^i}$ be the bias when substituting the vector with a bit-string of 0 and $\epsilon_{\{0,1\}^i}$ be the bias when keeping the vector as it is. Similarly, let $\lambda^i_{\texttt{Hamming}}$ be the number of identical $[2^i-1,2^i-i-1,3]$ Hamming code in the direct sum $\mathcal{C}_1\dot{+}\mathcal{C}_2\dot{+}\cdots\dot{+}\mathcal{C}_l$, $\lambda^i_{\texttt{Repet}}$ be the number of $[2i+1,1,2i+1]$ Repetition code, $\lambda_{\texttt{Golay}}$ be the number of the $[23,12,7]$ Golay code and $\lambda_{\{0\}^i},\lambda_{\{0,1\}^i}$ be the relevant numbers of the two trivial codes in the direct sum. Then the optimal configuration of the direct sum is equivalent to the maximized solution of

$$\begin{aligned}\max\quad & \sum_{\substack{i:2^i-1\le k_1\\ 2^i-i-1\le k_c}}\lambda^i_{\texttt{Hamming}}\log\epsilon^i_{\texttt{Hamming}}+\sum_{i:2i+1\le k_1}\lambda^i_{\texttt{Repet}}\log\epsilon^i_{\texttt{Repet}}\\ & +\lambda_{\texttt{Golay}}\log\epsilon_{\texttt{Golay}}+\lambda_{\{0\}^i}\log\epsilon_{\{0\}^i}+\lambda_{\{0,1\}^i}\log\epsilon_{\{0,1\}^i}\end{aligned}$$

under the constraint of

$$\begin{cases}\sum\limits_{\substack{i:2^i-1\le k_1\\ 2^i-i-1\le k_c}}(2^i-1)\lambda^i_{\texttt{Hamming}}+\sum\limits_{i:2i+1\le k_1}(2i+1)\lambda^i_{\texttt{Repet}}+23\lambda_{\texttt{Golay}}+\lambda_{\{0\}^i}+\lambda_{\{0,1\}^i}=k_1\\ \sum\limits_{\substack{i:2^i-1\le k_1\\ 2^i-i-1\le k_c}}(2^i-i-1)\lambda^i_{\texttt{Hamming}}+\sum\limits_{i:2i+1\le k_1}\lambda^i_{\texttt{Repet}}+12\lambda_{\texttt{Golay}}+\lambda_{\{0,1\}^i}=k_c.\end{cases}$$

This is a typical integer linear programming which can be solved efficiently by Sagemath. Thus, the solution to this linear programming problem will provide us an optimal configuration of the direct sum construction for the deployed covering codes.

4.2 Complexity Analysis

Now we present a summary of the theoretical justification of the new framework for fast correlation attacks in Algorithm 2. Note that some analysis has already been involved above, here we just put them together in the following Theorem 1. We adopt the finite field operations such as multiplication with fixed values as the time complexity unit in this paper. As stated before, the pre-processing phase can be divided into two parts, i.e., the BKW collision-reduction and the code-reduction. The time complexity of the code-reduction depends on the involved parameters of the linear covering code, e.g., k_l and k_c according to the syndrome-decoding based on some small tables, which can be assumed as a constant access complexity as explained before. Thus, this part of complexity will not be dominant at all compared with the time complexities of the other parts of the attack. We will adopt the sort-and-match routine as described before in Section 3.2 when the BKW collision-reduction procedure is adopted in Algorithm 2, whose time complexity is $\mathcal{O}(N + (\frac{4 \cdot k_1 \cdot \ln 2}{\varepsilon^2}) \cdot (k + 1 - b))$ after one-round of BKW reduction, where N is the available keystream length, k_1 is the remaining information bits after the corresponding dimension reduction step.

Theorem 1. *Let ε_g, ε_c be the correlations introduced in the BKW collision-reduction and the code-reduction procedure, respectively. Denote $\varepsilon_f = \varepsilon_g \cdot \varepsilon_c$, then Algorithm 2 has the following complexities.*

- *The time complexity of one-round BKW collision-reduction is $\mathcal{O}(N + (\frac{4 \cdot k_1 \cdot \ln 2}{\varepsilon^2}) \cdot (k + 1 - b))$.*

- *The online decoding time complexity for recovering the k_c-bit part of the LFSR initial state is $\mathcal{O}\Big(\frac{4 \cdot k_c \cdot \ln 2}{\varepsilon_f^2} + k_c 2^{k_c}\Big)$. The other bits in the LFSR initial state can be recovered similarly with a much lower complexity.*

- *The required length N of the observed keystream segment for Algorithm 2 to have a non-negligible success probability higher than 0.5 has to satisfy the relation $m_v = \binom{N/2^b}{2} \cdot 2^b = \frac{4 \cdot k_c \cdot \ln 2}{\varepsilon_f^2}$.*

Proof. For the complexity of BKW collision-reduction, it suffices to note that the adversary puts the captured keystream before the first row of the generator matrix $\mathbf{G}$ and exploit LF2 to eliminate b bits for the columns. If there is k_1 bits left, then from the unique distance in correlation attacks and linear cryptanalysis [CJS01], we have the remaining number of columns as $\frac{4 \cdot k_1 \cdot \ln 2}{\varepsilon^2}$. Besides, N is the complexity of sorting in the BKW sort-and-match partition.

For the online decoding complexity, it suffices to note that now the length of the new code is $\frac{4 \cdot k_c \cdot \ln 2}{\varepsilon_f^2}$ and from FWHT and the arguments behind Definition 4, we will have the complexity expression $\frac{4 \cdot k_c \cdot \ln 2}{\varepsilon_f^2} + k_c 2^{k_c}$.

For the data complexity N, it suffices to note that $\binom{N/2^b}{2} \cdot 2^b$ is the expected number of samples after one round of LF2, which have to meet the requirement of the unique distance decoding. This completes the proof. □

Based on the framework established in this section, we will present an improved fast correlation attack on Sosemanuk in the next section, in which we will have some further optimization of the complexity results shown in Theorem 1 based on a delicate usage of some carefully pre-computed tables.

5 Application to Sosemanuk

In this section, we will demonstrate a new state recovery attack against Sosemanuk by the newly developed Algorithm in Section 4.

5.1 Specification of Sosemanuk

Sosemanuk inherits the design spirit of SNOW 2.0, which is depicted in Fig.2. It consists of three main components, i.e., a LFSR of length 10 with each cell of 32-bit, a Finite State Machine (FSM) consisting of 2 32-bit words, and a non-linear output function `Serpent1` that operates in a bitsliced manner.

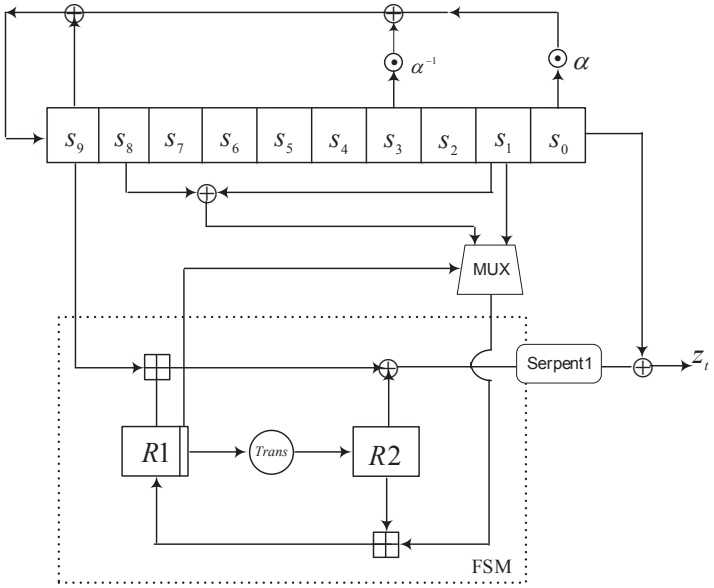

Figure 2: The keystream generation of Sosemanuk

The feedback polynomial of the LFSR is $x^{10} \oplus x^9 \oplus \alpha^{-1}x^3 \oplus \alpha \in \mathrm{GF}(2^{32})[x]$, where $\alpha \in \mathrm{GF}(2^{32})$ is a root of the primitive polynomial $y^4 \oplus \beta^{23}y^3 \oplus \beta^{245}y^2 \oplus \beta^{48}y \oplus \beta^{239} \in \mathrm{GF}(2^8)[y]$, and β is a root of the polynomial $x^8 \oplus x^7 \oplus x^5 \oplus x^3 \oplus 1 \in \mathrm{GF}(2)[x]$. Denote the LFSR state at time t by $(s_{t+9}, s_{t+8}, \cdots, s_t)$ for each $s_{t+i} \in \mathrm{GF}(2^{32})$ $(0 \le i \le 9)$. The FSM state at time t is denoted by $(R1_t, R2_t)$. Then the FSM is updated as

$$R1_t = R2_{t-1} \boxplus (s_{t+1} \oplus \mathtt{lsb}(R1_{t-1})s_{t+8}),$$
$$R2_t = \mathtt{Trans}(R1_{t-1}) \triangleq (M \times R1_{t-1} \bmod 2^{32})^{\lll 7},$$

where $M = \mathtt{0x54655307}$ is the constant value in the hexadecimal expression of the first ten decimals of π and $\lll$ is the bitwise rotation of a 32-bit word by 7 bits. The FSM has the output $f_t = (s_{t+9} \boxplus R1_t) \oplus R2_t$ for each time t. The keystream words are generated as follows,

$$(z_{t+3}, z_{t+2}, z_{t+1}, z_t) = \mathtt{Serpent1}(f_{t+3}, f_{t+2}, f_{t+1}, f_t) \oplus (s_{t+3}, s_{t+2}, s_{t+1}, s_t),$$

where `Serpent1` denotes the fastest Serpent [BAK98] S-box S_2 which is applied in a bitsliced manner and $t \equiv 1 (\bmod 4)$, i.e., four words are output vs 4 LFSR clockings.

5.2 An Improved Attack on Sosemanuk

According to the theoretical result on keystream generators with memory [Gol96], there should exist some linear approximation relation between the LFSR bits and the corresponding keystream bits when 2 consecutive steps of the FSM update are considered, e.g., t and $t+1$. From the description of keystream generation, we have

$$\begin{aligned} f_t \oplus R2_t &= s_{t+9} \boxplus \mathtt{Trans}^{-1}(\mathtt{R2_{t+1}}) \\ f_{t+1} \oplus R2_{t+1} &= s_{t+10} \boxplus (R2_t \boxplus (s_{t+2} \oplus a_t s_{t+9})). \end{aligned}$$

Applying the linear masks, we have

$$\begin{aligned} &\langle \Gamma, f_t \rangle \oplus \langle \Gamma, R2_t \rangle = \langle \Gamma, s_{t+9} \rangle \oplus \langle \Gamma, R2_{t+1} \rangle, \\ &\langle \Lambda, f_{t+1} \rangle \oplus \langle \Lambda, R2_{t+1} \rangle = \langle \Lambda, s_{t+10} \rangle \oplus \langle \Lambda, R2_t \rangle \oplus \langle \Lambda, s_{t+2} \rangle \oplus \langle \Lambda a_t, s_{t+9} \rangle, \end{aligned} \tag{9}$$

where $a_t = \mathtt{lsb}(R1_t)$ as shown in Fig.2 and $\Gamma_i, \Lambda_i \in \mathrm{GF}(2)^{32}$ are the corresponding linear masks. If we let $\Gamma = \Lambda$, then we can consider the following associated approximation relation

$$\langle \Gamma, (f_t \oplus f_{t+1}) \rangle \oplus \langle \Gamma, z_t \rangle \oplus \langle \Gamma, s_t \rangle \oplus \langle \Gamma, z_{t+3} \rangle \oplus \langle \Gamma, s_{t+3} \rangle = 0, \tag{10}$$

and $a_t \Gamma \cdot s_{t+9} = \Gamma \cdot s_{t+9}$ which holds with the correlation $\frac{1}{2}$. By the relations Eq.(9) and Eq.(10), we can derive

$$\langle \Gamma, z_t \rangle \oplus \langle \Gamma, z_{t+3} \rangle = \langle \Gamma, s_t \rangle \oplus \langle \Gamma, s_{t+2} \rangle \oplus \langle \Gamma, s_{t+3} \rangle \oplus \langle \Gamma, s_{t+10} \rangle, \tag{11}$$

where $\Gamma_i \in \mathrm{GF}(2)^{32}$ are the corresponding linear masks. We will not go into the search details of the linear masks making Eq.(11) hold with a large bias, which is actually similar to the previous work in [LLP08], as our aim is to make a better decoding by the newly developed framework given some fixed linear correlation, as stated before. When $\Gamma = \mathtt{0x03004001}$, we found this is the best linear mask found so far with a correlation of $2^{-21.41}$ for Eq.(11). Based on the framework presented in Algorithm 2, we can mount a bitwise fast correlation attack on Sosemanuk as follows.

First, we derive the bitwise state transition matrix of the LFSR in Sosemanuk, as shown in Appendix B and then exploit the LF2 routine in BKW collision-reduction to find the desirable parity checks, which sort the corresponding query matrix $\mathbf{G}$ into 2^b partitions according to the last b bits, thus there are on average $\frac{N}{2^b}$ samples in each partition. There is little dependency introduced by one round of LF2. Fortunately, this problem has been addressed by several previous papers on fast correlation attacks on non-linearly filtered keystream generators already. The final conclusion of the previous work is that the dependency does not affect the performance of the subsequent decoding. In some cases, this little dependency will facilitate the statistical decoding, as here the dependency is linear statistical dependency not the usual algebraic dependency. Then from the depiction of LF2 in Algorithm 1, we can get

$$m_v = 2^b \binom{N/2^b}{2} \text{ for } v = 2.$$

Thus, the correlation ϵ_g introduced in the BKW collision-reduction procedure is $\epsilon_g = (2^{-21.41})^2 = 2^{-42.82}$. By the tuned parameters of the code-length $k = 320$ and $k_1 = 185$, we have reduced 135 bits of the secret information. The process is shown in Fig.3 below.

By solving the integer program formulated in Sec.4, we construct a covering code with the parameters $k_1 = 185, k_c = 127$, which are the best parameters we have found through a exhaustive search over feasible parameter domain in terms of the total time/memory/data complexities via Sagemath, which can be seen from the accompanying Sagemath codes.

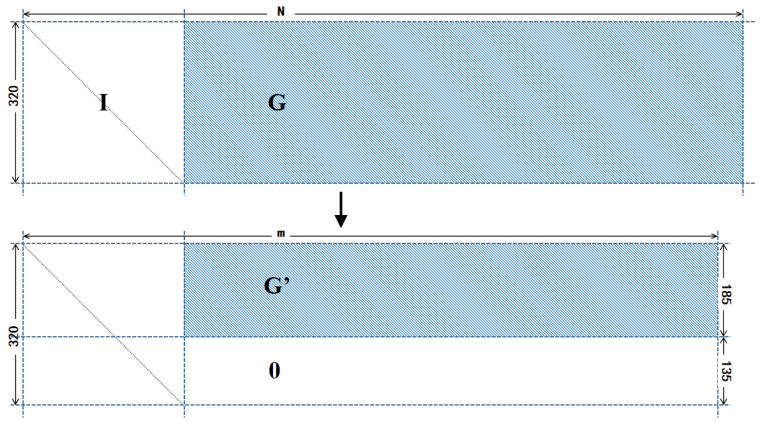

Figure 3: The description of the adopted BKW collision-reduction

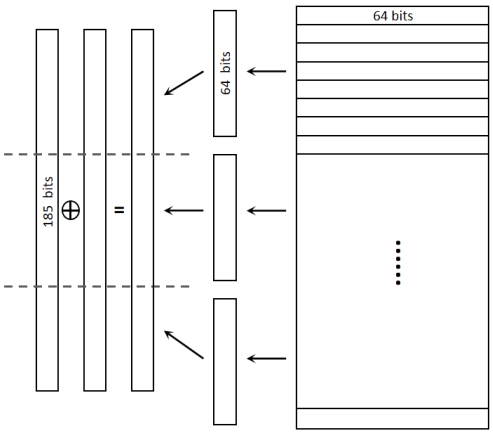

Figure 4: Schematic of the 64-dimensional xor in BKW collision-reduction

By applying the method in Section 4, we have found the following optimal problem for code-reduction and the corresponding constraint in this phase is

$$\begin{cases} \sum\limits_{\substack{i:2^i-1\leq 185\\ 2^i-i-1\leq 127}} (2^i-1)\lambda^i_{\texttt{Hamming}} + \sum\limits_{i:2i+1\leq 185} (2i+1)\lambda^i_{\texttt{Repet}} + 23\lambda_{\texttt{Golay}} + \lambda_{\{0\}^i} + \lambda_{\{0,1\}^i} = 185 \\ \sum\limits_{\substack{i:2^i-1\leq 185\\ 2^i-i-1\leq 127}} (2^i-i-1)\lambda^i_{\texttt{Hamming}} + \sum\limits_{i:2i+1\leq 185} \lambda^i_{\texttt{Repet}} + 12\lambda_{\texttt{Golay}} + \lambda_{\{0,1\}^i} = 127 \end{cases}$$

We have solved the above objective function by SageMath and get the optimal cascaded perfect codes for the direct sum construction. By concatenating two $\mathcal{H}_4 = [15, 11]$ Hamming codes, one $\mathcal{H}_6 = [63, 57]$ Hamming code, and four $\mathcal{G} = [23, 12]$ Golay codes, we can construct a $[185, 127]$ linear code, i.e., the direct sum construction of the covering code $\mathcal{C}$ exploited in Algorithm 2 for Sosemanuk is

$$\mathcal{C} = 2 \cdot \mathcal{H}_4 \ \dot{+} \ \mathcal{H}_6 \ \dot{+} \ 4 \cdot \mathcal{G}.$$

For this specified code construction, the bias introduced in the code-reduction phase can

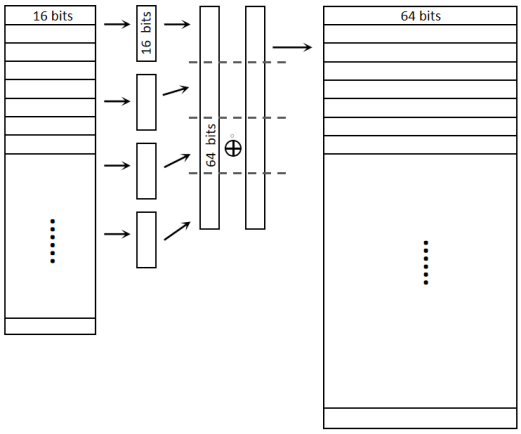

Figure 5: Detailed schematic of 16-dimensional xor in BKW collision-reduction

be computed as:

$$\begin{aligned}
\varepsilon_c &= \mathtt{bias}(15, 11, 1, \tfrac{1}{16})^2 \cdot \mathtt{bias}(63, 57, 1, \tfrac{1}{64}) \cdot \mathtt{bias}(23, 12, 3, \tfrac{127}{1024})^4 \\
&= \left[\sum_{\substack{0 \le i \le 1 \\ i \,\&\, 0x1 = 1}} \binom{15}{i} \left(\tfrac{1}{16}\right)^i \cdot \left(\tfrac{15}{16}\right)^{15-i} \right]^2 \cdot \left[\sum_{\substack{0 \le i \le 1 \\ i \,\&\, 0x1 = 1}} \binom{63}{i} \left(\tfrac{1}{64}\right)^i \cdot \left(\tfrac{63}{64}\right)^{63-i} \right] \\
&\quad \cdot \left[\sum_{\substack{0 \le i \le 3 \\ i \,\&\, 0x1 = 1}} \binom{23}{i} \left(\tfrac{127}{1024}\right)^i \cdot \left(\tfrac{897}{1024}\right)^{23-i} \right]^4 \\
&= 2^{-19.53215483015486}
\end{aligned}$$

Hence, the final correlation introduced in both BKW collision-reduction phase and code-reduction phase is $\varepsilon_f = \varepsilon_g \cdot \varepsilon_c = 2^{-42.82} \cdot 2^{-19.53} \doteq 2^{-62.35}$.

To optimize the collision-reduction procedure, we will adopt the table-based strategy to save the sum of 2 columns, each of which has f-dimension. The k_c-dimensional xor is divided into $\lceil \frac{k_c}{f} \rceil$ parts, and we store a table of all possible xors of two f-dimensional vectors and read it up to $\lceil \frac{k_c}{f} \rceil$ times. Then the computational complexity of this step is $\lceil \frac{k_c}{f} \rceil \cdot m_v$, and the cost for constructing the relevant table is estimated as $f \cdot 2^{f-1} \cdot (2^f - 2)$ with the memory complexity $\frac{3}{2} f 2^{f-1}(2^f - 2)$ as in [ZJW16]. As shown in Fig.4 and Fig.5, if we choose $f = 64$, then we need to store the table for all the possible xors of the 2 64-dimensional vectors which will not be accessible by a constant complexity in general. Instead, we proceed as follows to avoid this huge table and frequent reading problem. For computing the xor of two 64-dimensional vectors, we continue to divide the 64-dimensional vectors into 4 parts, each of which has a dimension of 16. Similar to accessing the pre-computed tables, say Table T_c for syndrome-decoding, since the size of such tables is usually small enough to be loaded into the memory that can be fast accessed by CPU, we adopt the common assumption in modern cryptanalysis literature that the adversary can read such small tables with a constant complexity. This is also a common assumption in symmetric key cryptanalysis. Note that this issue is quite different from the large table access case, which will have a complexity dependent on the concrete table size. That is why the small tables we constructed as above can be accessed with a constant complexity. Further, after BKW collision-reduction procedure, we get about $\frac{4 \cdot k_c \cdot ln2}{\varepsilon_f^2} = 2^{133.16}$ parity check equations and the data complexity is $N = 2^{135}$. For constructing the pre-computed tables, the time

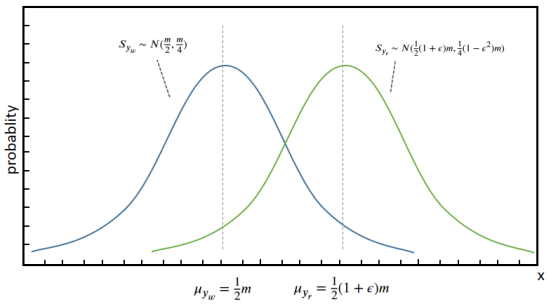

Figure 6: The distinguishing effect of the two distributions (correct vs. random) in the attack on Sosemanuk

complexities is about $\lceil \frac{k_1+1}{f} \rceil \cdot \frac{4k_c \ln 2}{\varepsilon_f^2} + f2^{f-1}(2^f - 2) = 2^{134.798}$ and the computational complexity of the online decoding phase is $\frac{4k_c \ln 2}{\varepsilon_f^2} + k_c 2^{k_c} = 2^{134.634}$, we can recover the $k_c = 127$ bits of the initial state of LFSR, the other bits can be recovered with a much lower complexity. Fig.6 depicts the distinguishing effect of the two involved distribution between the correct and random wrong candidates when the adversary launches the attack. Therefore, the final time/memory/data complexities are all below $2^{134.80}$, which is about 2^{20} times faster than the previous best result at Asiacrypt 2008 in [LLP08], and the fastest attack available so far among all known attacks on Sosemanuk. Though the 2^{20} times complexity reduction is the comprehensive result of several factors such as the 320-bit LFSR internal state size in Sosemanuk, the $2^{-21.41}$ bias of the found linear approximation, the essential improvement comes from the covering code reduction as can be seen from the computation of ϵ_c. By the covering code method, the dimension of the secret information has been decreased to $k_c = 127$ with a total bias of $\varepsilon_f = 2^{-62.35}$, which becomes easier to decode by FWHT. This obviously confirms the advantages of the newly developed Algorithm 2.

5.3 Experimental Results

We also applied the new framework to a reduced-version of Sosemanuk to check the validity of our attack. In general, the essential covering code procedure has been successfully verified not only on the reduced version but also on Sosemanuk itself. The theoretical prediction of the bias after covering code matches well with the simulation results, which clearly support the validity of the new framework.

Below is a description of the experimental flow on the reduced version of Sosemanuk, called s-Sosemanuk. The s-Sosemanuk stream cipher consists of a LFSR of length 5 with 16-bit cells and two 16-bit register words in the FSM part, whose detailed specification is given in Appendix C. We randomly choose some fixed values for 48 bits of the LFSR initial state and try to restore the remaining 32-bit state by the attack. First, we run s-Sosemanuk to generate the keystream $\{z_t\}$ for a randomly generated 80-bit LFSR initial state generated by AES-128. We generated 2^{28} keystream words, associated with the corresponding LFSR output sequence $\{s_t\}$, stored in two arrays respectively. The linear approximation with the largest bias found in experiments is $I_t = \Gamma \cdot z_t \oplus \Gamma \cdot z_{t+3} \oplus \Gamma \cdot s_t \oplus \Gamma \cdot s_{t+2} \oplus \Gamma \cdot s_{t+3} \oplus \Gamma \cdot s_{t+4}$ for $t \geq 0$,where $\Gamma = \texttt{0x0019} \in \mathrm{GF}(2)^{16}$ and the bias is around $\varepsilon = 2^{-3.607}$. Second, for each I_t, we express each s_t $(t > 4)$ as the linear combination of the LFSR initial state $(s_0, s_1, s_2, s_3, s_4)$ according to its feedback polynomial. Now we get the equivalent model in Eq.(3) with the generator matrix being in the systematic form. Third, we make the BKW collision-reduction by LF2 and the essential code-reduction to reduce the dimension of the secret information to $k_c = 22$ bits. We can get $k_1 = 28$, $k_c = 22$ and construct a

[28, 22] linear code as

$$\mathcal{C} = \mathcal{H}_2 \dotplus \mathcal{H}_4 \dotplus 10 \cdot \{0,1\}^{\mathtt{i}},$$

where k_1 denotes the number of information bits after BKW collision-reduction and k_c is the dimension of the employed linear code in code-reduction, i.e., the length of the remaining secret information. The final correlation is $\epsilon_f = (2^{-3.607})^2 \cdot 2^{-2.345} = 2^{-9.559}$. The bias introduced by covering code are shown in Table 3 for each component perfect code listed in Table 3.

Table 3: Comparison of the bias introduced by covering code and estimated in theory on the reduced version of Sosemanuk

	Experimental	Theoretical		Experimental	Theoretical
$\mathcal{H}_2$	0.425	0.425	$\mathcal{H}_3$	0.405	0.403
$\mathcal{H}_4$	0.398	0.398	$\mathcal{H}_5$	0.396	0.396
$\mathcal{H}_6$	0.3946	0.3948	$\mathcal{H}_7$	0.3940	0.3938
$\mathcal{G}$	0.0676	0.0665			

The results in Table 3 shows that the bias introduced by covering code are well consistent with the theoretical predictions. Note that the comparison between the experimental and theoretical results for the covering code has no restriction on the way how the checked pseudo-random sequence is generated, which means that this code-reduction procedure can be verified on Sosemanuk itself as well. Then, for each possible value of the $k_c = 22$-bit partial state, we use FWHT to compute the correlations and get solutions with the high correlation. We ran the experiments 100 times with different LFSR initial states generated randomly, and we found that the correct key always ranks in the top 10 in the candidates list, which clearly verified the theoretical analysis.

Therefore, the experimental results have proved the correctness of the newly developed method in Algorithm 2 and we can make reliable theoretical analysis when the simulation is infeasible to execute.

6 Conclusion

In this paper we have presented a new framework for fast correlation attack on stream ciphers with the integrated algorithmic procedures Gauss elimination, BKW collision-reduction and code-reduction to efficiently reduce the dimension of the secret information in the involved LFSR initial state. We further carefully revisit the Sosemanuk stream cipher, one of the 7 finalists in the eSTREAM project based on the new framework and have achieved improved and fine-grained cryptanalysis results on it with a time complexity of $2^{134.8}$, which is around 2^{20} times faster than the best previously known results at Asiacrypt 2008. Our result indicates an inefficiency in longer keys than 135 bits and shows that the security margin of Sosemanuk is around 2^8 for the 128-bit security for the first time, which is somewhat contrary to the official conclusion of the eSTREAM final report in 2008. A natural future work is to investigate the applicability of the new algorithm to other related symmetric key ciphers and hard problems such as solving LPN and LWE algorithms.

Acknowledgments

We would like to thank the anonymous reviewers and the shepherd for providing very valuable comments which improve the paper greatly. This work is supported by the program of the National Natural Science Foundation of China (Grant No. 62272446).

References

[BAK98] Eli Biham, Ross J. Anderson, and Lars R. Knudsen. Serpent: A new block cipher proposal. In Serge Vaudenay, editor, *FSE'98*, volume 1372 of *LNCS*, pages 222–238. Springer, Heidelberg, March 1998.

[BBC+08] Côme Berbain, Olivier Billet, Anne Canteaut, Nicolas T. Courtois, Henri Gilbert, Louis Goubin, Aline Gouget, Louis Granboulan, Cédric Lauradoux, Marine Minier, Thomas Pornin, and Hervé Sibert. Sosemanuk, a fast software-oriented stream cipher. In Matthew J. B. Robshaw and Olivier Billet, editors, *New Stream Cipher Designs - The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 98–118. Springer, 2008.

[BCC+08] Steve Babbage, Christophe De Cannière, Anne Canteaut, Carlos Cid, Henri Gilbert, Thomas Johansson, Matthew Parker, Bart Preneel, Vincent Rijmen, and Matthew Robshaw. The eSTREAM Portfolio, 2008. https://www.ecrypt.eu.org/stream/portfolio.pdf.

[BGM06] Côme Berbain, Henri Gilbert, and Alexander Maximov. Cryptanalysis of Grain. In Matthew J. B. Robshaw, editor, *FSE 2006*, volume 4047 of *LNCS*, pages 15–29. Springer, Heidelberg, March 2006.

[BJV04] Thomas Baignères, Pascal Junod, and Serge Vaudenay. How far can we go beyond linear cryptanalysis? In Pil Joong Lee, editor, *ASIACRYPT 2004*, volume 3329 of *LNCS*, pages 432–450. Springer, Heidelberg, December 2004.

[BKW00] Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. In *32nd ACM STOC*, pages 435–440. ACM Press, May 2000.

[BS00] Alex Biryukov and Adi Shamir. Cryptanalytic time/memory/data tradeoffs for stream ciphers. In Tatsuaki Okamoto, editor, *ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 1–13. Springer, Heidelberg, December 2000.

[CH10] Joo Yeon Cho and Miia Hermelin. Improved linear cryptanalysis of SOSEMANUK. In Donghoon Lee and Seokhie Hong, editors, *ICISC 09*, volume 5984 of *LNCS*, pages 101–117. Springer, Heidelberg, December 2010.

[CHLL05] Gérard D. Cohen, Iiro S. Honkala, Simon Litsyn, and Antoine Lobstein. Covering codes. In *North-Holland Mathematical Library*, 2005.

[CJM02] Philippe Chose, Antoine Joux, and Michel Mitton. Fast correlation attacks: An algorithmic point of view. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 209–221. Springer, Heidelberg, April / May 2002.

[CJS01] Vladimor V. Chepyzhov, Thomas Johansson, and Ben J. M. Smeets. A simple algorithm for fast correlation attacks on stream ciphers. In Bruce Schneier, editor, *FSE 2000*, volume 1978 of *LNCS*, pages 181–195. Springer, Heidelberg, April 2001.

[CS91] Vladimor V. Chepyzhov and Ben Smeets. On a fast correlation attack on certain stream ciphers. In Donald W. Davies, editor, *EUROCRYPT'91*, volume 547 of *LNCS*, pages 176–185. Springer, Heidelberg, April 1991.

[CT00] Anne Canteaut and Michaël Trabbia. Improved fast correlation attacks using parity-check equations of weight 4 and 5. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 573–588. Springer, Heidelberg, May 2000.

[DJGQ14] Lin Ding, Chenhui Jin, Jie Guan, and Chuanda Qi. New treatment of the BSW sampling and its applications to stream ciphers. In David Pointcheval and Damien Vergnaud, editors, *AFRICACRYPT 14*, volume 8469 of *LNCS*, pages 136–146. Springer, Heidelberg, May 2014.

[EJ03] Patrik Ekdahl and Thomas Johansson. A new version of the stream cipher SNOW. In Kaisa Nyberg and Howard M. Heys, editors, *SAC 2002*, volume 2595 of *LNCS*, pages 47–61. Springer, Heidelberg, August 2003.

[EJMY19] Patrik Ekdahl, Thomas Johansson, Alexander Maximov, and Jing Yang. A new SNOW stream cipher called SNOW-V. *IACR Trans. Symm. Cryptol.*, 2019(3):1–42, 2019.

[EST08] The ECRYPT Stream Cipher Project: eSTREAM Portfolio of Stream Ciphers., 2008. http://www.ecrypt.eu.org/stream/.

[FLZ+10] Xiutao Feng, Jun Liu, Zhaocun Zhou, Chuankun Wu, and Dengguo Feng. A byte-based guess and determine attack on SOSEMANUK. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 146–157. Springer, Heidelberg, December 2010.

[GBM02] Jovan Dj. Golic, Vittorio Bagini, and Guglielmo Morgari. Linear cryptanalysis of Bluetooth stream cipher. In Lars R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 238–255. Springer, Heidelberg, April / May 2002.

[GJL20] Qian Guo, Thomas Johansson, and Carl Löndahl. Solving LPN using covering codes. *Journal of Cryptology*, 33(1):1–33, January 2020.

[Gol96] Jovan Dj. Golic. Correlation properties of a general binary combiner with memory. *Journal of Cryptology*, 9(2):111–126, March 1996.

[HN10] Miia Hermelin and Kaisa Nyberg. Dependent linear approximations: The algorithm of Biryukov and others revisited. In Josef Pieprzyk, editor, *CT-RSA 2010*, volume 5985 of *LNCS*, pages 318–333. Springer, Heidelberg, March 2010.

[JJ99a] Thomas Johansson and Fredrik Jönsson. Fast correlation attacks based on turbo code techniques. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 181–197. Springer, Heidelberg, August 1999.

[JJ99b] Thomas Johansson and Fredrik Jönsson. Improved fast correlation attacks on stream ciphers via convolutional codes. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 347–362. Springer, Heidelberg, May 1999.

[Leu14] Gaëtan Leurent. Time-memory trade-offs for near-collisions. In Shiho Moriai, editor, *FSE 2013*, volume 8424 of *LNCS*, pages 205–218. Springer, Heidelberg, March 2014.

[LF06] Éric Levieil and Pierre-Alain Fouque. An improved LPN algorithm. In Roberto De Prisco and Moti Yung, editors, *SCN 06*, volume 4116 of *LNCS*, pages 348–359. Springer, Heidelberg, September 2006.

[LLP08] Jung-Keun Lee, Dong Hoon Lee, and Sangwoo Park. Cryptanalysis of Sosemanuk and SNOW 2.0 using linear masks. In Josef Pieprzyk, editor, *ASIACRYPT 2008*, volume 5350 of *LNCS*, pages 524–538. Springer, Heidelberg, December 2008.

[LMRS12] Mario Lamberger, Florian Mendel, Vincent Rijmen, and Koen Simoens. Memoryless near-collisions via coding theory. *Des. Codes Cryptogr.*, 62(1):1–18, 2012.

[LR11] Mario Lamberger and Vincent Rijmen. Optimal covering codes for finding near-collisions. In Alex Biryukov, Guang Gong, and Douglas R. Stinson, editors, *SAC 2010*, volume 6544 of *LNCS*, pages 187–197. Springer, Heidelberg, August 2011.

[LZFV13] Zhenqi Li, Bin Zhang, Junfeng Fan, and Ingrid Verbauwhede. A new model for error-tolerant side-channel cube attacks. In Guido Bertoni and Jean-Sébastien Coron, editors, *CHES 2013*, volume 8086 of *LNCS*, pages 453–470. Springer, Heidelberg, August 2013.

[Mat94] Mitsuru Matsui. Linear cryptanalysis method for DES cipher. In Tor Helleseth, editor, *EUROCRYPT'93*, volume 765 of *LNCS*, pages 386–397. Springer, Heidelberg, May 1994.

[MFI02] Miodrag J. Mihaljevic, Marc P. C. Fossorier, and Hideki Imai. Fast correlation attack algorithm with list decoding and an application. In Mitsuru Matsui, editor, *FSE 2001*, volume 2355 of *LNCS*, pages 196–210. Springer, Heidelberg, April 2002.

[MG90] Miodrag J. Mihaljevic and Jovan Dj. Golic. A fast iterative algorithm for a shift register initial state reconstruction given the nosiy output sequence. In Jennifer Seberry and Josef Pieprzyk, editors, *AUSCRYPT'90*, volume 453 of *LNCS*, pages 165–175. Springer, Heidelberg, January 1990.

[MS89] Willi Meier and Othmar Staffelbach. Fast correlation attacks on certain stream ciphers. *Journal of Cryptology*, 1(3):159–176, October 1989.

[Sie84] Thomas Siegenthaler. Correlation-immunity of nonlinear combining functions for cryptographic applications. *IEEE Trans. Inf. Theory*, 30(5):776–780, 1984.

[SNO06] ETSI/SAGE. Specification of the 3gpp confidentiality and integrity algorithms UEA2 & UIA2. In *Document 2: SNOW 3G Specification, version 1.1*, 2006. http://www.3gpp.org/ftp/.

[TIM+18] Yosuke Todo, Takanori Isobe, Willi Meier, Kazumaro Aoki, and Bin Zhang. Fast correlation attack revisited - cryptanalysis on full Grain-128a, Grain-128, and Grain-v1. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 129–159. Springer, Heidelberg, August 2018.

[TSS+06] Yukiyasu Tsunoo, Teruo Saito, Maki Shigeri, Tomoyasu Suzaki, Hadi Ahmadi, Taraneh Eghlidos, and Shahram Khazaei. Evaluation of sosemanuk with regard to guess-and-determine attacks. 2006.

[Wag02] David Wagner. A generalized birthday problem. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 288–303. Springer, Heidelberg, August 2002.

[ZJW16] Bin Zhang, Lin Jiao, and Mingsheng Wang. Faster algorithms for solving LPN. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part I*, volume 9665 of *LNCS*, pages 168–195. Springer, Heidelberg, May 2016.

[ZXM15] Bin Zhang, Chao Xu, and Willi Meier. Fast correlation attacks over extension fields, large-unit linear approximation and cryptanalysis of SNOW 2.0. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 643–662. Springer, Heidelberg, August 2015.

A Details of the FWHT Procedure when Evaluating the Parity-checks

When the adversary comes to the step of evaluating the resultant parity-checks on the target bits $\mathbf{r} = (u_0, u_1, \cdots, u_{k_1-1})$ with N_1 such parity-checks, we can regard this problem as the decoding of a $[N_1, k_1]$ linear code of length N_1 and dimension k_1 with the generator matrix $\hat{\mathbf{G}} = \left(\hat{\mathbf{g}}_0, \hat{\mathbf{g}}_1, \cdots, \hat{\mathbf{g}}_{N_1-1}\right)$ with $\hat{\mathbf{g}}_i$ being the i-th $k_1 \times 1$ column vector. Assume the resultant keystream is $\hat{\mathbf{z}} = \mathbf{r} \cdot \hat{\mathbf{G}} \oplus \tilde{\mathbf{e}}'$ with $\tilde{\mathbf{e}}'$ being the corresponding $1 \times N_1$ noise vector.

Let $\hat{\mathbf{u}} = \mathbf{r} \cdot \hat{\mathbf{G}} = (u_0, u_1, \cdots, u_{N_1-1})$, we introduce an integer-valued function,

$$h(\hat{\mathbf{u}}') = \sum_{0 \le i \le N_1-1 : \hat{\mathbf{u}}' = \hat{\mathbf{g}}_i^T} (-1)^{\hat{\mathbf{z}}_i},$$

for all $\hat{\mathbf{u}}' \in \mathrm{GF}(2)^{k_1}$. We compute the Walsh transform $\hat{H}$ of h and then we can get a 2^{k_1}-dimensional array storing the correlation indexed by $\hat{\mathbf{u}}'$, which is

$$\begin{aligned}\hat{H}(\mathbf{r}) &= \sum_{\hat{\mathbf{u}}' \in GF(2)^{k_1}} h(\hat{\mathbf{u}}')(-1)^{\langle \mathbf{r}, \hat{\mathbf{u}}' \rangle} = \sum_{i=0}^{N_1-1} (-1)^{\hat{\mathbf{z}}_i \oplus \langle \mathbf{r}, \hat{\mathbf{g}}_i^T \rangle} = \sum_{i=0}^{N_1-1} (-1)^{\hat{\mathbf{z}}_i \oplus u_i} \\ &= \sum_{i=0}^{N_1-1} (-1)^{\tilde{e}'_i} = N_{st} - N_{us},\end{aligned}$$

where N_{st} is the number of parity-checks that evaluates to 0 and N_{us} is the number of parity-checks that evaluates to 1. Note that we can make a ranking of the candidates according to the Walsh spectrum $\hat{H}(\mathbf{r})$ and take $\mathbf{r} = \mathbf{r}_{max}$ with the highest $\max_{\mathbf{r} \in GF(2)^{k_1}} \hat{H}(\mathbf{r})$. In cryptanalysis, the adversary could take a list of candidates that rank in the top to assure a good success probability. Since the preparation of the function h takes time N_1 and the time complexity for computing $\hat{H}$ is $k_1 \cdot 2^{k_1}$ with a memory complexity of 2^{k_1}, the final time complexity of FWHT is $N_1 + k_1 2^{k_1}$. Please refer to [BGM06] for more information.

B Binary State Transition Matrix of the LFSR over GF(2^{32}) in Sosemanuk

Sosemanuk uses multiplications and divisions of elements in $\mathrm{GF}(2^{32})$ by α, which is the root of the primitive polynomial over $\mathrm{GF}(2^8)[x]$. Multiplication of $z \in \mathrm{GF}(2^{32})$ by α corresponds to a left shift by 8 bits of z, followed by an xor with a 32-bit value which depends only on the most significant byte of z. Thus we can change the LFSR recurrence from $\mathrm{GF}(2^{32})$ to $\mathrm{GF}(2)$, and derive the binary generator matrix $\mathbf{G}$ accordingly.

Precisely, from the feedback polynomial of the LFSR in Sosemanuk $x^{10} \oplus x^9 \oplus \alpha^{-1}x^3 \oplus \alpha \in \mathrm{GF}(2^{32})[x]$, where $\alpha \in \mathrm{GF}(2^{32})$ is a root of the primitive polynomial $y^4 \oplus \beta^{23}y^3 \oplus \beta^{245}y^2 \oplus \beta^{48}y \oplus \beta^{239} \in \mathrm{GF}(2^8)[y]$ and β is a root of the polynomial $x^8 \oplus x^7 \oplus x^5 \oplus x^3 \oplus 1 \in \mathrm{GF}(2)[x]$, we have

$$s_{t+10} = s_{t+9} \oplus \alpha^{-1} s_{t+3} \oplus \alpha s_t.$$

Thus, it suffices to represent the multiplication $\alpha^{-1}s_{t+3}$ and αs_t in GF(2) form. We will first look at $\alpha \cdot s_t$. Without loss of generality, let us represent the relation as $\mathbf{x} = \alpha \cdot \mathbf{y}$, where $\mathbf{x} = (x_{31}, x_{30}, \cdots, x_0)$ and $\mathbf{y} = (y_{31}, y_{30}, \cdots, y_0)$ with x_i and y_i belong to GF(2). First we have

$$\alpha^4 = \beta^{23}\alpha^3 \oplus \beta^{245}\alpha^2 \oplus \beta^{48}\alpha \oplus \beta^{239}$$

and rewrite $\mathbf{x}$ as $\mathbf{x} = \mathbf{x}_3\alpha^3 \oplus \mathbf{x}_2\alpha^2 \oplus \mathbf{x}_1\alpha \oplus \mathbf{x}_0$ based on the polynomial basis $\{\alpha^3, \alpha^2, \alpha^1, 1\}$, where $\mathbf{x}_3 = (x_{31}, \cdots, x_{24})$, $\mathbf{x}_2 = (x_{23}, \cdots, x_{16})$, $\mathbf{x}_1 = (x_{15}, \cdots, x_8)$ and $\mathbf{x}_0 = (x_7, \cdots, x_0)$. We can further represent each byte $\mathbf{x}_i = (x_{8i+7}, \cdots, x_{8i})$ for $0 \le i \le 3$ based on the polynomial basis of $\{\beta^7, \beta^6, \beta^5, \beta^4, \beta^3, \beta^2, \beta^1, 1\}$. Finally, we get

$$\begin{aligned}
x_{31} &= y_{26} \oplus y_{24} \oplus y_{23},\\
x_{30} &= y_{31} \oplus y_{26} \oplus y_{25} \oplus y_{24} \oplus y_{22},\\
x_{29} &= y_{30} \oplus y_{25} \oplus y_{24} \oplus y_{21},\\
x_{28} &= y_{31} \oplus y_{29} \oplus y_{26} \oplus y_{20},\\
x_{27} &= y_{30} \oplus y_{28} \oplus y_{25} \oplus y_{19},\\
x_{26} &= y_{29} \oplus y_{27} \oplus y_{26} \oplus y_{18},\\
x_{25} &= y_{28} \oplus y_{26} \oplus y_{25} \oplus y_{17},\\
x_{24} &= y_{27} \oplus y_{25} \oplus y_{24} \oplus y_{16},\\
x_{23} &= y_{30} \oplus y_{28} \oplus y_{27} \oplus y_{26} \oplus y_{25} \oplus y_{24} \oplus y_{15},\\
x_{22} &= y_{30} \oplus y_{29} \oplus y_{28} \oplus y_{14},\\
x_{21} &= y_{31} \oplus y_{29} \oplus y_{28} \oplus y_{27} \oplus y_{13},\\
x_{20} &= y_{31} \oplus y_{25} \oplus y_{24} \oplus y_{12},\\
x_{19} &= y_{30} \oplus y_{24} \oplus y_{11},\\
x_{18} &= y_{31} \oplus y_{30} \oplus y_{29} \oplus y_{28} \oplus y_{27} \oplus y_{26} \oplus y_{25} \oplus y_{24} \oplus y_{10},\\
x_{17} &= y_{30} \oplus y_{29} \oplus y_{28} \oplus y_{27} \oplus y_{26} \oplus y_{25} \oplus y_{24} \oplus y_9,\\
x_{16} &= y_{31} \oplus y_{29} \oplus y_{28} \oplus y_{27} \oplus y_{26} \oplus y_{25} \oplus y_{24} \oplus y_8,\\
x_{15} &= y_{31} \oplus y_{27} \oplus y_{24} \oplus y_7,\\
x_{14} &= y_{30} \oplus y_{27} \oplus y_{26} \oplus y_{24} \oplus y_6,\\
x_{13} &= y_{29} \oplus y_{26} \oplus y_{25} \oplus y_5,\\
x_{12} &= y_{28} \oplus y_{27} \oplus y_{25} \oplus y_4,\\
x_{11} &= y_{31} \oplus y_{27} \oplus y_{26} \oplus y_{24} \oplus y_3,\\
x_{10} &= y_{30} \oplus y_{27} \oplus y_{26} \oplus y_{25} \oplus y_{24} \oplus y_2,\\
x_9 &= y_{29} \oplus y_{26} \oplus y_{25} \oplus y_{24} \oplus y_1,\\
x_8 &= y_{28} \oplus y_{25} \oplus y_{24} \oplus y_0,\\
x_7 &= y_{31} \oplus y_{30} \oplus y_{29} \oplus y_{28} \oplus y_{27},\\
x_6 &= y_{26},\\
x_5 &= y_{25},\\
x_4 &= y_{31} \oplus y_{30} \oplus y_{29} \oplus y_{28} \oplus y_{27} \oplus y_{24},\\
x_3 &= y_{30} \oplus y_{29} \oplus y_{28} \oplus y_{27} \oplus y_{26},\\
x_2 &= y_{31} \oplus y_{30} \oplus y_{26} \oplus y_{25},\\
x_1 &= y_{31} \oplus y_{30} \oplus y_{29} \oplus y_{25} \oplus y_{24},\\
x_0 &= y_{31} \oplus y_{30} \oplus y_{29} \oplus y_{28} \oplus y_{24},
\end{aligned}$$

which is the bitwise representation of $\mathbf{x} = \alpha \cdot \mathbf{y}$. Similarly for $\mathbf{x} = \alpha^{-1} \cdot \mathbf{y}$, from

$$\alpha^{-1} = \beta^{16}\alpha^3 \oplus \beta^{39}\alpha^2 \oplus \beta^6\alpha \oplus \beta^{64},$$

we can get

$$\begin{aligned}
x_{31} &= y_7 \oplus y_6 \oplus y_3,\\
x_{30} &= y_7 \oplus y_5 \oplus y_3 \oplus y_2,\\
x_{29} &= y_7 \oplus y_6 \oplus y_4 \oplus y_2 \oplus y_1,\\
x_{28} &= y_5 \oplus y_1 \oplus y_0,
\end{aligned}$$

$$
\begin{aligned}
x_{27} &= y_4 \oplus y_0,\\
x_{26} &= y_6,\\
x_{25} &= y_5,\\
x_{24} &= y_7 \oplus y_4 \oplus y_1,\\
x_{23} &= y_{31} \oplus y_7 \oplus y_6 \oplus y_4,\\
x_{22} &= y_{30} \oplus y_5 \oplus y_4 \oplus y_3,\\
x_{21} &= y_{29} \oplus y_4 \oplus y_3 \oplus y_2,\\
x_{20} &= y_{28} \oplus y_6 \oplus y_4 \oplus y_{31} \oplus y_2 \oplus y_1,\\
x_{19} &= y_{27} \oplus y_7 \oplus y_5 \oplus y_3 \oplus y_2 \oplus y_1 \oplus y_0,\\
x_{18} &= y_{26} \oplus y_7 \oplus y_2 \oplus y_1 \oplus y_0,\\
x_{17} &= y_{25} \oplus y_6 \oplus y_1 \oplus y_0,\\
x_{16} &= y_{24} \oplus y_7 \oplus y_5 \oplus y_0,\\
x_{15} &= y_{23} \oplus y_6 \oplus y_5 \oplus y_3 \oplus y_2 \oplus y_1,\\
x_{14} &= y_{22} \oplus y_6 \oplus y_4 \oplus y_3 \oplus y_0,\\
x_{13} &= y_{21} \oplus y_5 \oplus y_3 \oplus y_0,\\
x_{12} &= y_{20} \oplus y_6 \oplus y_5 \oplus y_4 \oplus y_3,\\
x_{11} &= y_{19} \oplus y_5 \oplus y_4 \oplus y_3 \oplus y_2,\\
x_{10} &= y_{18} \oplus y_6 \oplus y_5 \oplus y_4,\\
x_9 &= y_{17} \oplus y_7 \oplus y_5 \oplus y_4 \oplus y_3,\\
x_8 &= y_{16} \oplus y_7 \oplus y_6 \oplus y_4 \oplus y_2,\\
x_7 &= y_{15} \oplus y_6 \oplus y_3 \oplus y_0,\\
x_6 &= y_{14} \oplus y_6 \oplus y_5 \oplus y_3 \oplus y_2 \oplus y_0,\\
x_5 &= y_{13} \oplus y_7 \oplus y_5 \oplus y_4 \oplus y_2 \oplus y_1,\\
x_4 &= y_{12} \oplus y_4 \oplus y_1,\\
x_3 &= y_{11} \oplus y_3 \oplus y_0,\\
x_2 &= y_{10} \oplus y_6 \oplus y_3 \oplus y_2 \oplus y_0,\\
x_1 &= y_9 \oplus y_5 \oplus y_2 \oplus y_1,\\
x_0 &= y_8 \oplus y_7 \oplus y_4 \oplus y_1 \oplus y_0,
\end{aligned}
$$

which is the bitwise representation of $\mathbf{x} = \alpha^{-1} \cdot \mathbf{y}$. We have made experiments to check the correctness of the above relations and have got the confirmed results. With these bitwise representations, we can rewrite the state transition matrix of the LFSR in Sosemanuk as shown in Eq.(12):

$$
\left(\begin{array}{c|c}
\mathrm{I}_{32\times 32} & \\
\hline
0_{160\times 32} & \mathrm{I}_{288\times 288}\\
\hline
\mathrm{A}^{-1} & \\
\hline
0_{64\times 32} & \\
\hline
\mathrm{A} & 0_{32\times 288}
\end{array}\right), \tag{12}
$$

where $\mathrm{I}_{i\times i}$ for $i = 32, 128$ is the identity matrix of size $i\times i$, $0_{i\times j}$ for $(i, j) = (160, 32), (64, 32), (32, 288)$ are the sub-matrices of size $i \times j$ and the sub-matrices denoted by A associated

with α and by A^{-1} associated with α^{-1} are depicted below.

$$A = \begin{pmatrix}
0,1,0,1,0,0,0,0,0,0,1,1,0,1,0,1,1,0,0,0,1,0,0,0,1,0,0,1,0,1,1,1\\
0,0,1,0,1,0,0,0,1,1,0,0,1,1,1,0,0,1,0,0,0,1,0,0,1,0,0,1,1,1,1,1\\
0,0,0,1,0,1,0,0,0,1,1,0,0,1,1,1,0,0,1,0,0,0,1,0,1,0,0,1,1,0,1,1\\
0,0,0,0,1,0,1,0,1,1,1,0,0,1,1,1,0,0,0,1,0,0,0,1,1,0,0,1,1,0,0,1\\
0,0,0,0,0,1,0,1,1,0,1,0,0,1,1,1,1,1,0,1,1,1,0,0,1,0,0,1,1,0,0,0\\
1,1,0,1,0,1,1,0,1,0,0,0,0,1,1,1,0,1,1,0,1,1,1,0,0,1,0,0,1,1,0,0\\
0,1,1,0,1,0,1,1,1,0,0,1,0,1,1,1,0,0,1,1,0,1,1,1,0,0,1,0,0,1,1,0\\
1,1,1,0,0,0,0,1,1,0,0,1,1,1,1,1,1,1,0,0,1,1,1,1,0,0,0,1,0,0,1,1\\
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0\\
0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0\\
0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0\\
0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0\\
0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0\\
0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0\\
0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0\\
0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0\\
0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0\\
0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0\\
0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0\\
0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0\\
0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0\\
0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0\\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0\\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0\\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0\\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0\\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0\\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0\\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0\\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0\\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0\\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0\\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0\\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0\\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0\\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0\\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0\\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0\\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0\\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1\\
1,1,1,0,0,0,0,1,1,0,0,0,1,1,0,1,0,0,0,0,0,0,1,1,0,0,1,0,0,0,0,1\\
1,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,1,1,0,1,0,1,0,1,1,1,0,0,0,1,0,0\\
0,1,0,1,0,0,1,0,0,1,0,0,1,0,0,1,1,0,1,0,1,1,1,1,1,0,0,1,1,0,0,1,0\\
0,0,1,0,1,0,0,1,1,1,1,1,0,0,0,0,0,1,0,1,1,1,1,1,1,0,0,1,1,0,0,0,1\\
1,1,0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,1,1,1,1,1,1,0,1,1,1,1,0,0,1,1,0\\
0,1,1,0,0,0,0,0,0,0,0,1,1,1,1,0,0,1,0,1,0,1,0,0,1,0,1,1,0,0,1,1,0\\
0,0,1,1,0,0,0,0,0,0,0,0,1,1,1,1,0,1,0,0,0,0,0,0,0,0,0,1,1,0,0,1,1\\
0,0,0,1,1,0,0,0,0,0,0,0,0,1,1,1,1,0,1,0,0,0,0,0,0,1,1,0,0,1,1,0,1
\end{pmatrix}$$

$$A^{-1} = \begin{pmatrix}
0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0\\
0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0\\
0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0\\
0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0\\
0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0\\
0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0\\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0\\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0\\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0\\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0\\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0\\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0\\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0\\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0\\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0\\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0\\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0\\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0\\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0\\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0\\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0\\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0\\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0\\
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1\\
1,1,1,0,0,0,0,1,1,0,0,0,1,1,0,1,0,0,0,0,0,0,1,1,0,0,1,0,0,0,0,1\\
1,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,1,1,0,1,0,1,0,1,1,1,0,0,0,1,0,0\\
0,1,0,1,0,0,1,0,0,1,0,0,1,0,0,1,1,0,1,0,1,1,1,1,1,0,0,1,1,0,0,1,0\\
0,0,1,0,1,0,0,1,1,1,1,1,0,0,0,0,0,1,0,1,1,1,1,1,1,0,0,1,1,0,0,0,1\\
1,1,0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,1,1,1,1,1,1,0,1,1,1,1,0,0,1,1,0\\
0,1,1,0,0,0,0,0,0,0,0,1,1,1,1,0,0,1,0,1,0,1,0,0,1,0,1,1,0,0,1,1,0\\
0,0,1,1,0,0,0,0,0,0,0,0,1,1,1,1,0,1,0,0,0,0,0,0,0,0,0,1,1,0,0,1,1\\
0,0,0,1,1,0,0,0,0,0,0,0,0,1,1,1,1,0,1,0,0,0,0,0,0,1,1,0,0,1,1,0,1
\end{pmatrix}.$$

C A Reduced Version of Sosemanuk

The LFSR consists of 5 cells and each cell is a 16-bit word in $\mathrm{GF}(2^{16})$. The feedback polynomial of the LFSR is $\pi(x) = \alpha x^5 \oplus \alpha^{-1}x^3 \oplus x^2 \oplus 1 \in \mathrm{GF}(2^{16})[x]$, where α is a root of the primitive polynomial $x^4 \oplus \beta^{10}x^3 \oplus \beta^6 x^2 \oplus x \oplus \beta^{11}$ and β is a root of $x^4 \oplus x \oplus 1 \in \mathrm{GF}(2)[x]$. The FSM has two 16-bit memory registers $R1$ and $R2$ updated as

$$R1_t = R2_{t-1} \boxplus (s_{t+1} \oplus \mathtt{lsb}(R1_{t-1})s_{t+3}),$$
$$R2_t = \mathtt{Trans}(R1_{t-1}) \triangleq (M \times R1_{t-1} \bmod 2^{16})^{\lll 7},$$

where $M = \mathtt{0x0021}$ is the constant value in the hexadecimal expression and $\lll$ is the bitwise rotation of a 16-bit word by 7 bits. The FSM has the output $f_t = (s_{t+4} \boxplus R1_t) \oplus R2_t$ for each time t. The generated keystream is $z_t = f_t \oplus s_t$.