Improving the Performance of the SYND Stream Cipher

Mohammed Meziani[1] and Gerhard Hoffmann[2], and Pierre-Louis Cayrel[3]

[1] CASED – Center for Advanced Security Research Darmstadt,
Mornewegstrasse 32, 64293 Darmstadt, Germany
mohammed.meziani@cased.de
[2] Technische Universität Darmstadt
Fachbereich Informatik
Kryptographie und Computeralgebra,
Hochschulstraße 10
64289 Darmstadt, Germany
hoffmann@mathematik.tu-darmstadt.de
[3] Laboratoire Hubert Curien, UMR CNRS 5516,
Bâtiment F 18 rue du professeur Benoît Lauras,
42000 Saint-Etienne, France
pierre.louis.cayrel@univ-st-etienne.fr

Abstract. In 2007, Gaborit et al. proposed the stream cipher SYND as an improvement of the pseudo random number generator due to Fischer and Stern. This work shows how to improve considerably the efficiency the SYND cipher without using the so-called regular encoding and without compromising the security of the modified SYND stream cipher. Our proposal, called XSYND, uses a generic state transformation which is reducible to the Regular Syndrome Decoding problem (RSD), but has better computational characteristics than the regular encoding. A first implementation shows that XSYND runs much faster than SYND for a comparative security level (being more than three times faster for a security level of 128 bits, and more than 6 times faster for 400-bit security), though it is still only half as fast as AES in counter mode. Parallel computation may yet improve the speed of our proposal, and we leave it as future research to improve the efficiency of our implementation.

Keywords: Stream ciphers, Provable security, Syndrome Decoding

1 Introduction

A stream cipher is a secret key cryptosystem that employs a symmetric secret key for producing an arbitrary long pseudo random sequence, called keystream. This keystream is then combined with the plaintext, typically by means of the bitwise XOR, to produce the ciphertext. Stream ciphers are necessary in many real-life applications, especially the wireless communication standards such as IEEE 802.11b [2] and Bluetooth [3]. Therefore, stream ciphers are usually required to be fast and implementable on constrained hardware.

It is easy to design a stream cipher. The challenge here is to make it theoretically secure and at the same time very efficient. A variety of efficient stream ciphers have been proposed, but most of them were proven to be insecure as reported during the eSTREAM project [1]. It is thus desirable to have provably secure stream ciphers, whose security is grounded on hard problems. The first constructions in this direction are [14, 13] whose security is based on the hardness of factoring problem. Another proposal was developed by Kaliski [24], its security relies on the intractability of the discrete logarithm problem. Assuming the hardness of solving RSA problem, Alexi et al. [4] proposed a pseudo-random number generator (PRNG). The one-way function hard-core bit construction by Goldreich et al. [19] has also led to the construction of the efficient PRNG, called BMGL [32], which was developed by Håstad and Näslund using Rijndael.

Although proving the hardness of all mentioned problems is an important open problem, they are all known to be easy on quantum attacks as shown in [31]. It is therefore advantageous to design stream ciphers whose security relies on other assumptions, and which are more promising even in the age of quantum computers. The first construction addressing this challenge is due to Impagliazzo et al. [23], based on the subset sum problem. Later, Fisher and Stern [16] proposed a PRNG whose security relies on the syndrome decoding (SD) problem [10]. Recently, further provably secure constructions have been proposed. The first one, called QUAD, due to Berbain et al. [9] under assumption that solving a multivariate quadratic system is hard (MQ-problem). The second one, named SYND, proposed by Gaborit et al. [18], is an improved variant of [16]. The security of SYND is also reducible to the SD problem. Recently, Meziani et al. [27] proposed the 2SC stream cipher based on the same problem, following the sponge construction. This cipher is much more efficient than SYND [18] in terms of performance and has small key/IV size, but it suffers from the drawback of having big matrices.

Our contribution. In this paper we propose an efficient variant of the SYND stream cipher [18], called XSYND, this new construction is reducible to the SD problem. This cipher is faster than all existing code-based stream ciphers [18, 27] and requires comparatively little storage capacity, making it attractive for practical implementations. We also propose parameters for fast keystream generation for different security levels.

Outline of the paper. Section 2 provides a background of coding theory. Section 3 describes the SYND stream cipher. A detailed description of the XSYND stream cipher is presented in Section 4, its security is discussed in Section 5. In Section 6 secure parameters and experimental results for XSYND are presented. Section 7 concludes this paper.

2 Coding Theory Background

This section provides a short introduction to error-correcting codes and recall some hard problems in this area.
In general, a linear code $\mathcal{C}$ is a k-dimensional subspace of an n-dimensional vector space over a finite field $\mathbb{F}_q$, where k and n are positive integers with $k < n$ and q a prime power. Elements of $\mathbb{F}_q^n$ are called *words* and elements of $\mathcal{C}$ are called *codewords*. The integer $r = n - k$ is called the co-dimension of $\mathcal{C}$. The *weight* of a word x, denoted by $w = wt(x)$, is the number of non-zero entries in x, and the *Hamming distance* between two words x and y is $wt(x - y)$. The minimum distance d of a code is the smallest distance between any two distinct codewords. A generator matrix G of $\mathcal{C}$ is a matrix whose rows form a basis of $\mathcal{C}$, .i.e., $\mathcal{C} = \{x \cdot G : x \in \mathbb{F}_q^k\}$. A parity check matrix H of $\mathcal{C}$ is defined by $\mathcal{C} = \{x \in \mathbb{F}_q^n : H \cdot x^T = 0\}$ and generates the dual space of the code $\mathcal{C}$.
A linear code $\mathcal{C}$ is called a cyclic code if any cyclic shift of a codeword is another codeword. That is, $x_0, \cdots, x_n \in \mathcal{C}$ implies $x_n, x_0, \cdots, x_{n-1} \in \mathcal{C}$. In this case, the parity check matrix of $\mathcal{C}$ can be only described by its first row. Furthermore, $\mathcal{C}$ is called a quasi cyclic code if its parity check matrix is composed of a number of cyclic submatrices. In practice, such codes are very good from the decoding capacity point of view and behave like random codes with small requirement on the length as shown in [17].Throughout this paper we consider $q = 2$.

Definition 1 (Regular word). *A regular word of length n and weight w is a word consisting of w blocks of length n/w, each with a single non-zero entry.*

In code-based cryptography, the security of most of the cryptographic primitives is related to the hardness of the following problems.

Definition 2 (Binary Syndrome Decoding (SD) problem). *Given a binary $r \times n$ matrix H, a binary vector $y \in \mathbb{F}_2^r$, and an integer $w > 0$, find a word $x \in \mathbb{F}_2^n$ of weight $wt(x) = w$, such that $H \cdot x^T = y$.*

This problem is proven NP-complete in [10]. A particular case of this problem is the Regular Syndrome Decoding (RSD) problem, which has been proved to be NP-complete in [5]. It can be stated as follows.

Definition 3 (Regular Syndrome Decoding (RSD) Problem). *Given a binary $r \times n$ matrix H, a binary vector $y \in \mathbb{F}_2^r$,and an integer $w > 0$, find a regular word $x \in \mathbb{F}_2^n$ of weight $wt(x) = w$, such that $H \cdot x^T = y$.*

Through this paper, we will denote $\mathrm{RSD}(n, r, w)$ to indicate an instance of RSD problem with parameters (n, r, w). Before ending this section, we recall the definition of a hardcore bit (or hardcore predicate).

Definition 4 (Hardcore bit). *Let f be a one-way function. Let $h : \{0,1\}^* \to \{0,1\}$ be a polynomial-time computable function. We say that h is a hardcore bit for f if for all PPT adversary $\mathcal{A}$ there exists one negligible function ϵ, such that*

$$\Pr[A(f(x)) = h(x)] \leq \frac{1}{2} + \epsilon(n), \quad \forall n$$

where the probability is over x chosen randomly and the coin tosses of $\mathcal{A}$.

3 The SYND stream cipher

This section gives a short description of the original SYND design. SYND is a synchronous stream cipher with security reduction proposed in 2007 by Gaborit et. al [18]. SYND is a improved variant of Fisher-Stern's PRNG [16] with two improvements: the use of quasi-cyclic codes, which reduces the storage capacity and the introduction of regular words used in [5], which speeds up the keystream generation of the system. This PRNG can be seen as a finite automaton, S, determined by a set of inner states with lengths ranging from 256 to 1024 bits. SYND accepts keys of length 128 to 512 bits and produces a keystream twice as large as the key size in each round.
More precisely, let n, w, and r be three positive integers such that the ratio n/w is a power of two and $r = w \log_2(n/w)$. The key stream generation of SYND works in three steps using three different one-to-one functions called `Ini`, `Upd`, and `Out`, respectively (See Figure 1). The `Ini` function takes a secret key `K` concatenated with an initial vector `IV`, both of length $r/2$ bits, and returns an initial state $e_0 = \texttt{Ini}(\texttt{K}|\texttt{IV})$, which starts the key stream generation process, where $(a|b)$ denotes the concatenation of bit strings a and b. The `Ini` function is a three-Feistel transformation based on `Upd` and `Out`, and given by:

$$\texttt{Ini}(x) = y \oplus \texttt{Out}(x \oplus \texttt{Upd}(y)); \quad y = x \oplus \texttt{Upd}(y), \quad \forall x = (\texttt{K}, \texttt{IV}) \in \mathbb{F}_2^{r/2} \times \mathbb{F}_2^{r/2},$$

where `Upd` and `Out` functions are defined by

$$\texttt{Upd}(x) = A \cdot \theta(x); \quad \texttt{Out}(x) = B \cdot \theta(x), \quad \forall x \in \mathbb{F}_2^r.$$

Here, A and B are random binary matrices which describe the same binary quasi-cyclic (QC) code of length n, correcting up to w errors. The mapping $x \mapsto \theta(x)$ is an encoding algorithm which transforms an r-bit string into a regular word of length n and weight w. Starting from e_0, in each time unit i, S outputs a key bit $z_i = \texttt{Out}(e_i)$ and changes the inner state as follows: $e_{i+1} = \texttt{Upd}(e_i)$.

After generating the key bit stream $z_0, z_1, \cdots$, a cleartext bit stream $m_0, m_1, \cdots$ is encrypted into a cyphertext stream $c_0, c_1, \cdots$ by the bitwise XOR operator as $c_i = z_i \oplus m_i$. Knowing the secret state e_0 the receiver can generate the keystream $z_0, z_1, \cdots$ and therefore recover the cleartext bitstream by $m_i = z_i \oplus c_i$.

Thus, the evaluation of `Upd` and `Out` for state x is done by first encoding x into a regular word $\theta(x)$ of length n and weight w, and then multiplying the resulting word by a random $r \times n$ binary matrix. This process can be regarded as XORing w columns from the underlying random matrix with one another (these r-bit long columns correspond to the non-zero positions of the regular word $\theta(x)$). This idea was first introduced in the FSB hash family [5] in order to speed up the hashing process. In the next section, we show how to speed up SYND by eliminating the encoding $x \mapsto \theta(x)$, while at the same time preserving the security properties of the underlying scheme.

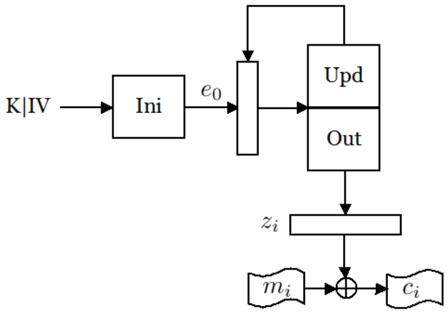

Fig. 1. A graphical illustration of the SYND stream cipher

4 Our proposal: XSYND

This section describes an eXtended SYND algorithm (XSYND), which adds two main features to the original SYND structure. In what follows, we use the notations of the previous section.

Firstly, we modify the `Ini` function such that it requires only two, rather than three function evaluations, without loss of security. We denote the new function by `XIni` and depict it in Fig. 2. Note that this modification does not affect the recovery of the secret `K` or the initial vector `IV`. In fact, it is straightforward to prove that, given an initial state e_0 output by `XIni`, if an adversary can extract `K` and `IV` from e_0, it can also easily solve an instance RSD(n, r, w). The new function `XIni` function is defined by:

$$\mathtt{XIni}(x) = y \oplus \mathtt{Out}(y); \quad y = x \oplus \mathtt{Upd}(x); \quad \forall x = (\mathtt{K}, \mathtt{IV}) \in \mathbb{F}_2^{r/2} \times \mathbb{F}_2^{r/2}.$$

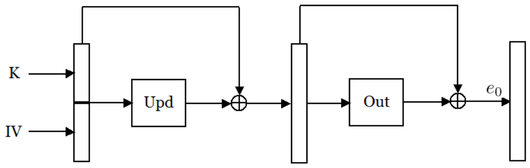

Fig. 2. The `XIni` function of XSYND

The second modification in XSYND is to avoid the regular encoding $x \mapsto \theta(x)$ in `Upd` and `Out` by using the randomize-then-combine paradigm due to Bellare et al. [6–8] as depicted in Figure 3. More precisely, given an input x consisting of w blocks $x_1, \ldots, x_w$, each block being b bits (where b is chosen at will), we first feed each block through a random function f, obtaining an output y_i. The values $y_1, y_2, \cdots, y_w$ are combined by bitwise XOR to generate the final output. In XSYND, we use the following function f: let H be a random binary matrix of size $wb \times w \cdot 2^b$, consisting of w submatrices $H_1 \ldots H_w$ of size $wb \times 2^b$ (we write $H = H_1 | \ldots | H_w$). If we write the submatrices as $H_i = (h_i^{(0)}, h_i^{(1)}, \ldots, h_i^{(2^b-1)})$, where $h_i^{(j)} \in \mathbb{F}^{wb}$ for $j \in \{0, 1, \ldots, 2^b - 1\}$, then we can define f by $y_i = h_i^{(j)}$ if and only if the decimal value of x_i is equal to j. We have 2^b possible value for each y_i, depending on the decimal value of the block x_i. In this way, we redefine the functions `Upd` and `Out` as follows (see also Fig. 4):

$$\texttt{Upd}(x) = a_1^{(x_1)} \oplus a_2^{(x_2)} \oplus \cdots \oplus a_w^{(x_w)}; \ \ \texttt{Out}(x) = b_1^{(x_1)} \oplus b_2^{(x_2)} \oplus \cdots \oplus b_w^{(x_w)}; \ \ \forall x \in \mathbb{F}_2^{wb}.$$

Here, $a_i^{(j)}$ (resp. $b_i^{(j)}$) is the j^{th} column of the i^{th} submatrix A_i (resp. B_i) of a random binary matrix A (resp. B), both of size $wb \times w2^b$.

Remark 1. It is worth noting that the same technique has been recently used by Berstein et al. [12] to improve the efficiency of the FSB hash family [5].

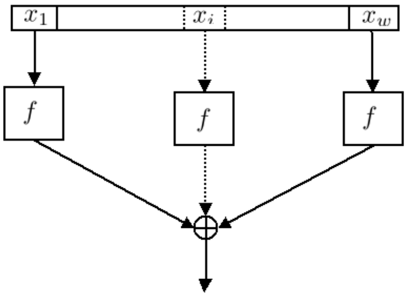

Fig. 3. Randomize-then-combine paradigm

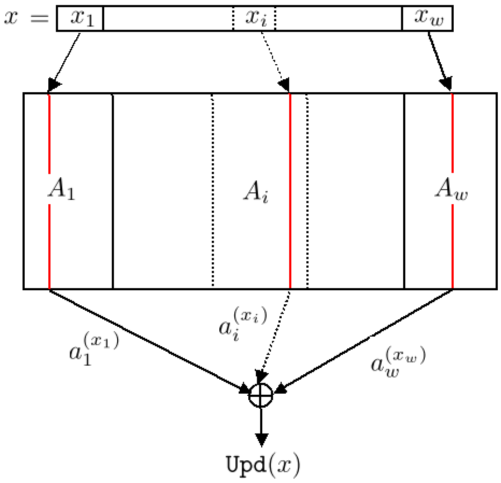

Fig. 4. The Update Function `Upd` of XSYND

5 Security of XSYND

5.1 Theoretical Security

In this section we present the theoretical security of our construction. The presentation is done in two steps. In the first step, we show that it is hard to find the secret state x given $\texttt{Upd}(x)$ and $\texttt{Out}(x)$ as described in section 4. More precisely, we show that inverting $\texttt{Upd}(x)$ and $\texttt{Out}(x)$ is reducible to the RSD problem. In the second step, we prove that XSYND is a pseudo-random generator, meaning that the key stream produced by XSYND is indistinguishable from truly random sequences.

Step 1: We consider general transformations g defined as:

$$g(x) = a_1^{(x_1)} \oplus a_2^{(x_2)} \oplus \cdots \oplus a_w^{(x_w)},\ \forall x = (x_1, \ldots, x_w) \in \mathbb{F}_2^{wb}.$$

In this transformation, $a_i^{(j)}$ for $j = 0, \ldots, 2^b$ is the $(j+1)^{th}$ column of the i^{th} submatrix A_i of a random binary matrix A of size $wb \times w2^b$. Note that both $\texttt{Upd}(x)$ and $\texttt{Out}(x)$ are particular instantiations of g, for random matrices A and B (see previous section). Our argument in this section is as follows: we first show that (1)for each x there exists a regular word z such that $g(x) = A \cdot z^T$, then prove that (2) learning x from $y = g(x)$ is equivalent to finding a regular word z such that $A \cdot z^T = y$ (this is an instantiation of RSD(n, r, w) for $r = wb$ and $n = w2^b$). Thus, under the RSD assumption, the modified XSYND protocol security can be reduced to the hardness of RSD.

First consider (1). We write $A = A_1|\ldots|A_w$ as in section 4, for $wb \times 2^b$ submatrices A_i. Each submatrix has columns $a_i^{(0)}, \ldots, a_i^{(2^b-1)}$. We note that any regular word z is in fact a word of length $n = w2^b$ and weight w, whose decimal entries $z_1, \ldots, z_w$ indicate the positions of its non-zero entries (and each z_i is a unique value between $(i-1)2^b + 1$ and $i2^b$ since the word is regular). Let $x = (x_1, \ldots, x_w)$ be a state in decimal notation. We associate each x with a value z whose decimal notation is $(z_1, \ldots, z_w)$ for $z_i = (x_i + 1) + (i-1)2^b$. The reverse transformation of z to x is obtained as follows:

$$\begin{cases} x_1 \equiv z_1 - 1 \pmod{2^b} \\ x_2 \equiv z_2 - 1 \pmod{2^b} \\ \cdots \quad \cdots \quad \cdots \quad \cdots \\ x_w \equiv z_w - 1 \pmod{2^b} \end{cases}$$

It is easy to check that:

$$A \cdot z^T = a_1^{(x_1)} \oplus a_2^{(x_2)} \oplus \cdots \oplus a_w^{(x_w)}.$$

Toy Example. Let us consider $w = 3$ and $b = 2$. Then the matrix A should be $(3 \cdot 2) \times (3 \cdot 2^2) = 6 \times 12$ and binary. Consider in this example the following matrix A:

$$A = \left[\begin{array}{cccc|cccc|cccc}
a_1^{(0)} & a_1^{(1)} & a_1^{(2)} & a_1^{(3)} & a_2^{(0)} & a_2^{(1)} & a_2^{(2)} & a_2^{(3)} & a_3^{(0)} & a_3^{(1)} & a_3^{(2)} & a_3^{(3)} \\
\hline
1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\
0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0
\end{array}\right]$$

Let us consider a state x in decimal form, with $x = (2, 1, 0)$. Compute z in decimal form according to the formula $z_i = (x_i + 1) + (i - 1)2^b$. Thus $z_1 = 3$, $z_2 = 6$, and $z_3 = 9$. In binary notation, z_i denotes the positions of z's non-zero entries, i.e. $z = [0010|0100|1000]$. We can now verify that for this z we have

$$g(x) = a_1^{(2)} \oplus a_2^{(1)} \oplus a_3^{(0)} = [001111] = A \cdot z^T.$$

Now let us consider the security reduction of general transformations g to the RSD problem, i.e. step (2) outlined above. We have shown that for each input value x we can find a regular word z of weight w such that $A \cdot z^T = g(x)$. Assume that there exists an adversary that can invert $g(x)$, i.e. given $y = g(x)$, the adversary outputs x. Then the same adversary computes z as above and can thus, given a matrix A, and a value $y = g(x) = A \cdot z^T$, this adversary can output the regular word z. This is exactly an instantiation of $\mathrm{RSD}(n, r, w)$ for $r = wb$ and $n = w2^b$. In conclusion, we can reduce the security of XSYND to the hardness of the RSD problem.

Step 2: In this step, we prove that XSYND is a pseudo-random generator. Our proof is an adaption of that given for the Fischer-Stern's PRNG [16]. We will show that if there exists an algorithm that is able of distinguishing a random bit string from the output of the mapping $x \to (\mathtt{Out}(x), \mathtt{Upd}(x))$, then this algorithm can be converted into a predicator that can predicts the inner product of an input x and a random bit string chosen at random. Before doing so, we state the following assumptions.

1. *Indistinguishability:* The binary matrices A and B (both of size $r \times n$) are computationally indistinguishable from uniform matrices of the same dimensions.
2. *Regular syndrome decoding (RSD):* The family of mappings defined as $g_M(z) = M \cdot z^T$ for an uniform $2r \times n$ binary matrix M is one-way on the set of all regular words of length n and weight w.

As shown in the last subsection, the mapping $x \to \mathtt{Upd}(x)$ (resp. $x \to \mathtt{Out}(x)$) can be regarded as $f_u(z) = A \cdot z^T$ (resp. $f_o(z) = B \cdot z^T$), where A and B are binary matrices, both of size $r \times n$, and z is taken from the set of regular words. Therefore, from now on, we will use f_u (resp. f_o) instead of $\mathtt{Upd}$ (resp. $\mathtt{Out}$).

From A and B we build a $2r \times n$ block matrix M by stacking them vertically, i.e.

$$M = \left(\frac{A}{B}\right)$$

In this case, we can write the mapping $x \rightarrow (\texttt{Out}(x), \texttt{Upd}(x))$ as $g_M(z) = M \cdot z^T = (f_u(z), f_o(z))$. Consequently, in order to prove that XSYND is a pseudo-random generator, it is sufficient to prove that the output of $z \rightarrow g_M(z)$ is pseudo random as proved in [16]. Our proof is based on the Goldreich-Levin Theorem [19], which says that, for any one-way function, the inner product of its argument and a randomly chosen bit string is a hardcore bit (or hardcore predicate). Recall that the inner product of two bit strings a and b (of the same size) is defined by

$$\langle a, b \rangle = \sum_i a_i b_i \mod 2.$$

Theorem 1. *(Goldreich-Levin theorem) Let $f : \mathbb{F}_2^{\lambda(n)} \rightarrow \mathbb{F}_2^{\mu(n)}$ be a one-way function. For every PPT algorithm $\mathcal{A}$, for all polynomials p and all but finitely many n's,*

$$\Pr[A(f(x), \nu) = \langle x, \nu \rangle] \leq \frac{1}{2} + \frac{1}{p(n)}$$

where the probability is taken over x uniformly chosen x and $\nu \in \mathbb{F}_2^{\lambda(n)}$.

The theorem proving that XSYND is a pseudo-random generator is stated as follows.

Theorem 2. *Suppose n, r, and w are chosen such that the indistinguishability and the regular syndrome decoding assumptions hold. Then the output distribution of XSYND is computationally indistinguishable from a truly random distribution. That is, XSYND is a pseudo-random generator.*

Proof. (by contradiction). Let us assume that an $2r$-bit output of the mapping $g_M(z) = M \cdot z^T$ is not pseudo-random, and there exists a distinguisher $\mathcal{D}$, which is capable to differentiate this output of from a $2r$-bit random string υ. More precisely, $\mathcal{D}$ takes as input $2r \times n$ binary random matrix M and a random $\upsilon \in \{0,1\}^{2r}$ as a candidate being equal to $M \cdot z^T$ for some unknown regular word z. In the event that $M \cdot z^T = \upsilon$, $\mathcal{D}$ outputs 1 with probability above $\frac{1}{2} + \frac{1}{p(n)}$, for every polynomial $p(n)$. Otherwise, when υ is chosen uniformly from $\{0,1\}^{2r}$, $\mathcal{D}$ outputs 1 with probability at most $\frac{1}{2}$. Formally, the distinguisher $\mathcal{D}$ behaves as follows:

$$\begin{cases} \Pr[\mathcal{D}(M, \upsilon) = 1] \geq \frac{1}{2} + \frac{1}{p(n)}, & \text{if } \upsilon = M \cdot z^T, \text{ for some regular word } z \\ \Pr[\mathcal{D}(M, \upsilon) = 1] < \frac{1}{2}, & \text{if } \upsilon \text{ is taken uniformly from } \{0,1\}^{2r} \end{cases}$$

As next step, we will build an algorithm $\mathcal{P}$, which uses the distinguisher $\mathcal{D}$ as subroutine. This algorithm will predicts the inner product $\langle z, \nu \rangle$ with probability at least $\frac{1}{2} + \frac{1}{2p(n)}$, where z is an unknown regular word (an input of g_M) and ν a randomly chosen n-bit string. To this end, let write $\nu = (\nu_1, \cdots, \nu_n)$. In addition, let σ be the number of the positions j such that where $z_i = \nu_j = 1$, i.e. the size of the intersection $z \cap \nu$ and ρ its parity, i.e. the inner product $\langle z, \nu \rangle$. Then the algorithm $\mathcal{P}$ takes as input $g_M(z)$ and ν and executes the following steps:

- Select a random $\rho' \in \{0, 1\}$ as candidate to ρ
- Choose randomly $\xi \in \{0, 1\}^{2r}$
- Build a new $2r \times n$ binary matrix $\widehat{M} = (\widehat{m}_1, \cdots, \widehat{m}_n)$ such that for every $j \in \{1, \cdots, n\}$ it holds
$$\widehat{m}_j = \begin{cases} m_j + \xi & \text{if } \nu_j = 1, \\ m_j & \text{if } \nu_j = 0 \end{cases}$$
- Feed the distinguisher with $\widehat{M}$ and $g_M(z) + \rho' \cdot \xi$
- If the distinguisher outputs 1, then output $\rho' = \rho$. Otherwise, output the opposite of ρ'.

Now, we show next that $\mathcal{P}$ predicts the inner product $\langle z, \nu \rangle$ with probability above $\frac{1}{2} + \frac{1}{2p(n)}$. We have to consider two events:

(1) $\mathtt{E}_1$:"ρ is guessed correctly". Then the prognosticated value for the inner product $\langle z, \nu \rangle$ is correct if the distinguisher outputs 1. The distribution seen by the distinguisher on $(\widehat{M}, g_M(z) + \rho' \cdot \xi)$ is identical to the distribution on input $(M, g_M(z))$. By construction, this is the case with probability at least $\frac{1}{2} + \frac{1}{p(n)}$.
(2) $\mathtt{E}_2$:"ρ is not guessed correctly". The distinguisher receives uniformly distributed inputs because of the randomness of ξ. It then returns 1 with probability $\frac{1}{2}$.

Since $\Pr[\mathtt{E}_1] = \Pr[\mathtt{E}_2] = \frac{1}{2}$, we conclude that the overall probability of correctly predicting the inner product $\langle z, \nu \rangle$ is at least $\frac{1}{2} + \frac{1}{2p(n)}$. This contradicts the Theorem 1 because of the RSD assumption. ■

5.2 Practical Security

This section presents what are provably the most generic attacks against XSYND. We will only address the hardness of inverting the mapping g defined in the previous section, since this is the main building block of XSYND design. If an attacker can invert g, then she can recover the secret key and recover inner states.
In what follows, we denote by $\mathtt{WF}_Y(n, w, r)$ the work factor (i.e. number of binary operations) required to solve the instance $\mathrm{RSD}(n, w, r)$ by using an algorithm Y. Furthermore, in estimating the complexity of each attack against XSYND we use $r = wb$ with $b = \log_2\left(\frac{n}{w}\right)$.
There are essentially three types of known attacks that are applicable to XSYND:

1. **Linearization Attacks.** There are two types of linearization attacks that are relevant for XSYND, namely the Bellare-Micciancio (BM) attack [8] against the XHASH function [8], and the attack due to Saarinen [30]. We discuss these attacks below.

(a) The Bellare-Micciancio's attack. This is a preimage attack proposed by Bellare and Micciancio [8] against the so-called XHASH mapping. This attack relies on finding a linear dependency among w r-bit vectors, where w is the number of vectors XORred together and r, the length (in bits) of the target value. This is likely to succeed if the value w is close to r. More precisely, let l and k be two positive integers. Let f be a random function with $f : \mathbb{F}_2^l \mapsto \mathbb{F}_2^r$. Let $[i]$ denote the binary representation of an integer i. Based on f, the XHASH is defined as

$$\texttt{XHASH}(x) = f([1]|x_1) \oplus \cdots \oplus f([w]|x_w), \ \text{ with } \ x = (x_1, x_2, \ldots, x_w).$$

The BM attack finds a preimage x of a given $z = \texttt{XHASH}(x) \in F_2^r$ as follows. First, one finds w-bit string $y = (y_1, \ldots, y_w)$, with $y_i \in \mathbb{F}_2$, such that $\texttt{XHASH}(x^y) = z$, where $x^y = x_1^{y_1} \ldots x_w^{y_w}$. To achieve this, one first computes $2w$ values $\beta_i^k = f([i]|x_i^j)$ for $k \in \{0, 1\}$ and $i \in \{1, \ldots, w\}$; the next step is to try to solve the following system of equations over $\mathbb{F}_2$ using linear algebra:

$$\begin{cases} y_i \oplus \bar{y_i} = 1, & i \in \{1, \ldots, w\}, \\ \oplus_{i=1}^{w} \beta_i^0(j) y_i \oplus \beta_i^1(j) \bar{y_i} = z(i), & j \in \{1, \ldots, r\}. \end{cases}$$

Here, $\beta_i^0(j)$ (resp. $\beta_i^1(j)$) denotes the j^{-th} bit of β_i^0 (resp. β_i^1) and $\bar{y_i} = 1 - y_i$ are the unknowns. This system has $r + w$ equations in $2w$ unknowns and is easy to solve when $w = r + 1$. More generally, it was shown in [8] (Appendix A, Lemma A.1) that for all $y \in \mathbb{F}_2^w$ the probability to have $\texttt{XHASH}(x^y) \neq z$ is at most 2^{r-w}. That is, the complexity of inverting XHASH is at least 2^{r-w}; in our notation,

$$\texttt{WF}_{\texttt{BM}}(n, w, r) \geq 2^{r-w} = 2^{(b-1)w}.$$

(b) The Saarinen's attack. This attack is due to Saarinen [30] and it was proposed against the FSB [5] hash function. The main idea behind this attack is reducing the problem of finding collisions or preimages to that of solving systems of equations. This attack is very efficient when $r < 2w$. We briefly show how this attack works in our setting, where we must invert the map g.

As shown in section 5.1, $g(x) = A \cdot z^T$, where A is the random binary matrix of size $r \times n$, whose entries define g, and z is a regular word of length n and weight w. We can in turn write $A \cdot z^T$ out as follows:

$$y = \oplus_{i=1}^{w} a_{(i-1)\frac{n}{w} + x_i + 1}, \ 0 \leq x_i \leq \frac{n}{w}, \tag{1}$$

where $x = (x_1, \ldots, x_w)$ and a_j denotes the j^{-th} column of A. For simplicity, assume that $x_i \in \{0, 1\}$. In this case, we define a constant r-bit vector c and an additional $r \times w$ binary matrix B as follows.

$$c = \oplus_{i=1}^{w} a_{(i-1)\frac{n}{w}+1}, \;\; B = [b_1 \cdots b_w] \text{ with } \;\; b_i = a_{(i-1)\frac{n}{w}+1} \oplus a_{(i-1)\frac{n}{w}+2}. \quad (2)$$

It is easy to check that $y = B \cdot x + c$. As a consequence if $r = w$, then B is square and we can find the preimage x from y as:

$$x = B^{-1} \cdot (y \oplus c), \quad (3)$$

where B^{-1} denotes the inverse of B. Note that this inverse exists with probability without proof of $\prod_{i=1}^{r}(1 - 1/2^i) \approx 0.29$ for r moderately large. The expected complexity of this attack is the the workload of inverting B, which is al most $0.29 \cdot r^3$. It has been proved in [30] that the same complexity is obtained even if $r \leq 2w$.
In the opposite direction, Saarinen also extended his attack for the case when $w \leq r/\alpha$ for $\alpha > 1$ and $x_i \notin \{0, 1\}$. In this case, the complexity is about $2^r/(\alpha+1)^w$. Moreover, the recent result[12] shows that if $\alpha = 2\beta$, for $\beta > 1$, this complexity becomes $2^r/(\beta + 1)^{2w}$. As consequence we obtain:

$$\mathtt{WF}_{\mathtt{Saarinen}}(n, w, r) \geq \begin{cases} 2^r/(\alpha+1)^w & \text{if } w \leq r/\alpha \\ 2^r/(\alpha+1)^{2w} & \text{if } w \leq r/2\alpha \end{cases}$$

which can be rewritten in our setting as:

$$\mathtt{WF}_{\mathtt{Saarinen}}(n, w, r) \geq \begin{cases} (\frac{2^b}{\alpha+1})^w & \text{if } \alpha \leq b \\ (\frac{2^b}{(\alpha+1)^2})^w & \text{if } \alpha \leq b/2 \end{cases}$$

2. **Generalized Birthday Attacks (GBA).** This class of attacks attempt to solve the following, so-called k-sum problem: given k random lists $L_1, L_2, \ldots, L_k$ of r-bit strings selected uniformly and independently at random, find $x_1 \in L_1, x_2 \in L_2, \ldots, x_k \in L_k$ such that $\oplus_{i=1}^{k} x_i = 0$. For $k = 2$, a solution can be found in time $2^{r/2}$ using the standard birthday paradox. For $k > 2$ Wagner's algorithm [33] and its extended variants $[5, 11, 28, 15]$ can be applied. When $k = 2^{j-1}$ and $|L_i| > 2^{r/j}$, Wagner's algorithm can find at least one solution in time $2^{r/j}$.

   The main idea behind a GBA algorithm is depicted Fig. **??**. We consider the case $k = 4$. Let $L_1, \ldots, L_4$ be four lists, each of length $2^{r/3}$. The algorithm proceeds in two iterations. In the first iteration, we build two new lists $L_{1,2}$ and $L_{3,4}$. The list $L_{1,2}$ contains all sums $x_1 \oplus x_2$ with $x_1 \in L_1$ and $x_2 \in L_2$ such that the first $r/3$ bits of the sum are zero. Similarly, $L_{3,4}$ contains all sums $x_3 \oplus x_4$ with $x_3 \in L_3$ and $x_4 \in L_4$ such that the first $r/3$ bits of the sum are zero. So the expected length of $L_{1,2}$ is equal to $2^{-r/3} \cdot |L_1| \cdot |L_2| = 2^{r/3}$. Similarly, the expected length of $L_{3,4}$ is also $2^{r/3}$. In the second iteration of the algorithm, we construct a new list L'_1 containing all pairs $(x'_1, x'_2) \in$

$L_{1,2} \times L_{3,4}$ such that the first $r/3$ bits of the sum $x'_1 \oplus x'_2$ are zero. Then the probability that $x'_1 \oplus x'_2$ equals zero is $2^{-2r/3}$. Therefore, the expected number of matching sums is $2^{-2r/3} \cdot |L_{1,2}| \cdot |L_{3,4}| = 1$. So we expected to find a solution. This idea can be generalized for $k = 2^{j-1}$ by repeating the same procedure $j-2$ times. In each iteration a, we construct lists, each containing $2^{r/j}$ elements that are zero on their first ar/j bits, until obtaining, on average, one r-bit element with all entries equal to 0.

We estimate the security of XSYND against GBA attacks by using the GBA algorithm from [15]. This algorithm attempts to find a set of indices $I = \{1, 2, \cdots, 2^{\gamma}\}$ satisfying $\oplus_{i \in I} H_i = 0$, where H_i are columns of the matrix H. As shown in [15], the algorithm is applicable when $\binom{2^b w}{2^{(1-\gamma)w}} \geq 2^{bw+\gamma(\gamma-1)}$. Under this condition, the cost of solving an instance RSD problem with parameters (n, r, w) is given by:

$$\mathtt{WF}_{\mathtt{GBA}}(n, w, r) \geq \left(\frac{wb}{\gamma} - 1\right) 2^{\frac{wb}{\gamma} - 1}.$$

Note that the recent result in [29] shows that the time and memory efficiency of GBA attacks can be improved, but only by a small factor. In Section 6 we take this improvement into account when proposing parameters for XSYND.

3. **Information Set Decoding (ISD).** ISD is one of the most important generic algorithm for decoding errors in an arbitrary linear code. An ISD algorithm consists (in its simplest form) in finding a valid, so-called information set, which is a subset of k error-free positions amongst the n positions of each codeword. Here, k is the dimension and n the length of the code. The validity of this set is checked by using Gaussian elimination on the $r \times n$ parity check matrix H. If we denote by $p(n, r, w)$ the probability of finding a valid information set and by $c(r)$ the cost of Gaussian elimination, then the overall cost of ISD algorithms equals the ratio $c(r)/p(n, r, w)$.

   In the following, we estimate the cost of finding a solution to the regular syndrome decoding (RSD) problem, i.e. we wish to invert the map g. Let $n_s(n, r, w)$ be the expected number of solutions of RSD instance. This quantity is:

$$n_s(n, r, w) = \frac{\left(\frac{n}{w}\right)^w}{2^r} = 1,$$

   because $r = w \log_2(\frac{n}{w})$. In addition, let $p_v(n, r, w)$ be the probability that a given information set is valid for one given solution of RSD. As shown in [5], $p(n, r, w)$ can be approximated by: $p(n, r, w) \approx p_v(n, r, w) \cdot n_s(n, r, w)$. Furthermore, as shown in [5], $p_v(n, r, w)$ is given by:

$$p_v(n, r, w) = \left(\frac{r}{n}\right)^w = \left(\frac{\log_2(n/w)}{n/w}\right)^w$$

   We thus conclude that the probability of selecting a valid set to invert RSD is equal to: $p(n, r, w) = \left(\frac{b}{2^b}\right)^w$.

Hence, the cost $\mathtt{WF}_{\mathtt{ISD}}(n, w, r)$ of solving an instance of RSD with parameters (n, r, w) is approximately:

$$\mathtt{WF}_{\mathtt{ISD}}(n, w, r) \approx c(r) \cdot \left(\frac{2^b}{b}\right)^w . \tag{4}$$

If we assume that the complexity of Gaussian elimination is r^3, then $\mathtt{WF}_{\mathtt{ISD}}(n, w, r)$ becomes:

$$\mathtt{WF}_{\mathtt{ISD}}(n, w, r) \approx (wb)^3 \cdot \left(\frac{2^b}{b}\right)^w . \tag{5}$$

In practice, we use the lower bound for ISD algorithms presented in [26] to estimate the security of XSYND against ISD attacks and show our results in Table 1 .

Remark 2. One could also use Time Memory trade-off attacks against stream ciphers. This attack was first introduced in [21] as a generic method of attacking block ciphers. To make this attack unfeasible, one must adjust the cipher parameters as shown in [20, 22], i.e., the initial vector should be at least as large as the key, and the state should be at least twice the key.

Table 1 briefly summarizes the expected complexity of the previous attacks against XSYND.

Table 1. The estimated complexities of possible attacks against XSYND.

Attack	The binary logarithm of the complexity: $\log_2(\mathtt{WF}_{(.)}(n, w, r))$
BM	$w(b-1)$
Sarinnen	$\begin{cases} w(b - \log_2(\alpha+1)), & \text{if } \alpha \le b \\ w(b - 2\log_2(\alpha+1)), & \text{if } \alpha \le b/2 \end{cases}$
GBA	$wb/\gamma + \log_2(wb/\gamma - 1) - 1$ for $\gamma \in \mathbb{N}$
ISD	$w(b - \log_2(b)) + 3\log_2(wb)$

6 Parameters and Experimental Results

Suitable parameters (n, r, w) for XSYND must provide both efficiency and high security against all known attacks. Firstly, we account for Time Memory Trade-Off attacks (see section 5.2) and choose (n, r, w) such that:

$$r = w \log_2(n/w) \geq 2|\mathtt{IV}| \quad \text{and} \quad |\mathtt{IV}| \geq |\mathtt{K}|.$$

For XSYND we choose $r = w \log_2(n/w) = 2|\mathtt{IV}| = 2|\mathtt{K}|$. We then fix $b = \log_2(n/w) = 8$ and for each security level λ we vary w to obtain both high performance and a complexity of solving the RSD problem of at least 2^{λ}.

We have tested a large set of potential parameters for a number of security levels. Table 2 presents the optimal parameter sets (n, w, r) resulted from running our implementation for several security levels. Note that in our implementation, we only use random binary codes without any particular structure. But it is possible to find parameters providing the same security levels when the parity check matrix is quasi-cyclic as in [18]. In this case, r has to be a prime and 2 is primitive root of the finite field $\mathbb{F}_r^*$ in order to guarantee the randomness property of QC-codes as demonstrated in [17].

Table 2. Proposed parameters for XSYND.

Security Level	n	r	w	Key/IV size [bits]	Speed of XSYND [cpb]
80	8192	256	32	128	14.92
120	12288	384	48	192	16.98
160	16384	512	64	256	35.40
200	20480	640	80	320	43.68
240	24576	768	96	384	55.42
280	28672	896	112	448	77.09

The results shown in Table 2 are for a pure C/C++ implementation with additional use of C/C++-Intrinsics). The operating system was Debian 6.0.3, the source has been compiled with gcc (Debian 4.4.5-8) 4.4.5. All results have been gained on an AMD Phenom(tm) 9950 Quad-Core Processor, running at a clock rate of 1300 MHz. Due to the row-major convention of C/C++, the two matrices H_1 resp. H_2 have been used and stored in transposed form. In order to compare the speed of XSYND with the claimed speed of SYND [18] and 2SC [27] (Table 4), we have tested our implementation using the parameter sets suggested in [18]. Our results presented in Table 3 show that, for comparable security levels, XSYND runs faster than SYND [18] and 2SC cipher [27]. It is worth to stress that the authors of 2SC [27] compared the performance of 2SC [27] to that of SYND [18] based on their own implementations (of both schemes), because no freely-available implementation of SYND exists.

Compared to the (bitsliced and parallel) fastest software implementation of AES in CTR mode proposed by Käsper et al. [25], XSYND runs about two times slower. Indeed, this implementation is written in assembly using 128-bit XMM registers and runs at 7.59 cycles/byte on a Intel Core 2 Q9550 and 6.92 cycles/byte on Core i7 920. Note that our implementation could be sped up by using parallel computations achieving much better results than what Tables 2 and 3 show. It is therefore interesting to implement this to see how much further XSYND can be improved.

Table 3. Performance of XSYND vs. SYND using parameter sets proposed in [18].

Security Level	n	r	w	key/IV size [bits]	speed of SYND [cpb]	speed of XSYND [cpb]
80	8192	256	32	128	27	14.92
128	8192	384	48	192	47	16.86
180	8192	512	64	256	53	35.18
400	8192	1024	128	512	83	55.69

Table 4. Parameters and performance of 2SC cipher given in [27].

Security Level	n	r	w	key/IV size [bits]	speed of 2SC [cpb]
100	1572864	384	24	144	37
160	2228224	544	34	208	47
250	3801088	928	58	352	72

7 Conclusion

In this paper we presented XSYND, an improved variant of SYND stream cipher, without compromising its security. Our proposal uses a generic state transformation which is directly reducible to the regular syndrome decoding problem (RSD), but has better computational characteristics than the regular encoding introduced in the SYND system. A software implementation shows that our proposal runs much faster than all code-based stream ciphers for different security levels, but it is only half as fast as AES in counter mode without making any parallel computation. Moreover, unlike to SYND, we show how the security reduction of our proposal works.

References

1. http://www.ecrytp.eu.org/stream.
2. Overview of IEEE 802.11b Security, Intel Technology Journal Q2. 2000.
3. Specification of the Bluetooth system, vol. 1.1. Feb.2001. http://www.bluetooth.org/spec/.
4. W. Alexi, B. Chor, O. Goldreich, and C. P. Schnorr. RSA and Rabin functions: certain parts are as hard as the whole. *SIAM J. Comput.*, 17(2):194–209, 1988.
5. D. Augot, M. Finiasz, and N. Sendrier. A Family of Fast Syndrome Based Cryptographic Hash Functions. In E. Dawson and S. Vaudenay, editors, *Mycrypt 2005*, volume 3715, pages 64–83. Springer, 2005.
6. M. Bellare, O. Goldreich, and S. Goldwasser. Incremental cryptography: The case of hashing and signing. In *Proceedings of the 14th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '94, pages 216–233. Springer, 1994.
7. M. Bellare, O. Goldreich, and S. Goldwasser. Incremental cryptography and application to virus protection. In *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, STOC '95, pages 45–56. ACM, 1995.
8. M. Bellare and D. Micciancio. A new paradigm for collision-free hashing: incrementality at reduced cost. In *Proceedings of the 16th annual international conference on Theory and application of cryptographic techniques*, EUROCRYPT'97, pages 163–192. Springer, 1997.
9. C. Berbain, H. Gilbert, and J. Patarin. QUAD: A multivariate stream cipher with provable security. *J. Symb. Comput.*, 44(12):1703–1723, 2009.
10. E. Berlekamp, R. McEliece, and H. van Tilborg. On the inherent intractability of certain coding problems. *IEEE Transactions on Information Theory*, 24(2):384–386, May 1978.
11. D. J. Bernstein. Better price-performance ratios for generalized birthday attacks. In *Workshop Record of SHARCS07: Special-purpose Hardware for Attacking Cryptographic Systems (2007)*, 2007.
12. D. J. Bernstein, T. Lange, C. Peters, and P. Schwabe. Really Fast Syndrome-Based hashing. In A. Nitaj and D. Pointcheval, editors, *Progress in Cryptology–AFRICACRYPT 2011*, volume 6737 of *LNCS*, pages 134–152. Springer, 2011.
13. L Blum, M Blum, and M Shub. A simple unpredictable pseudo random number generator. *SIAM J. Comput.*, 15(2):364–383, 1986.
14. M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM J. Comput.*, 13(4):850–864, 1984.
15. M. Finiasz and N. Sendrier. Security Bounds for the Design of Code-based Cryptosystems. In Mitsuru Matsui, editor, *Advances in Cryptology - ASIACRYPT 2009*, number 5912 in LNCS, pages 88–105. Springer, 2009.
16. J.-B. Fischer and J. Stern. An efficient pseudo-random generator provably as secure as syndrome decoding. In *EUROCRYPT'96: Proc. of the 15th annual international conference on Theory and application of cryptographic techniques*, pages 245–255. Springer, 1996.
17. P. Gaborit and G. Zémor. Asymptotic improvement of the Gilbert-Varshamov bound for linear codes. volume abs/0708.4164, 2007.
18. Ph. Gaborit, C. Laudaroux, and N. Sendrier. SYND : a very fast code-based cipher stream with a security reduction. In *IEEE Conference, ISIT'07*, pages 186–190, Nice, France, July 2007.

19. O. Goldreich and L. A. Levin. A hard-core predicate for all one-way functions. In *STOC '89: Proc. of the twenty-first annual ACM symposium on Theory of computing*, pages 25–32. ACM, 1989.
20. J.Dj. Golic. Cryptanalysis of alleged A5 stream cipher. In *Proceedings of the 16th annual international conference on Theory and application of cryptographic techniques*, EUROCRYPT'97, pages 239–255. Springer, 1997.
21. M. Hellman. A cryptanalytic time-memory trade-off. *IEEE Transactions on Information Theory*, 26:401–406, 1980.
22. J. Hong and P. Sarkar. Rediscovery of time memory tradeoffs. Cryptology ePrint Archive, Report 2005/090, 2005. `http://eprint.iacr.org/`.
23. R. Impagliazzo and M. Naor. Efficient cryptographic schemes provably as secure as subset sum. *J. Cryptology*, 9(4):199–216, 1996.
24. B. S. Kaliski. *Elliptic Curves and Cryptography: A Pseudorandom Bit Generator and Other Tools*. Phd thesis, MIT, Cambridge, MA, USA, 1988.
25. E. Käsper and P. Schwabe. Faster and timing-attack resistant AES-GCM. In Christophe Clavier and Kris Gaj, editors, *Cryptographic Hardware and Embedded Systems – CHES 2009*, volume 5747 of *LNCS*, pages 1–17. Springer, 2009.
26. A. May, A. Meurer, and E. Thomae. Decoding Random Linear Codes in $\tilde{\mathcal{O}}(2^{0.054n})$. In *ASIACRYPT*, volume 7073 of *Lecture Notes in Computer Science*. Springer, 2011.
27. M. Meziani, P.-L. Cayrel, and S. M. Alaoui El Yousfi. 2SC: An Efficient Code-Based Stream Cipher. In *ISA*, volume 200 of *Communications in Computer and Information Science*, pages 111–122. Springer, 2011.
28. L. Minder and A. Sinclair. The extended k-tree algorithm. In *Proc. of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'09, pages 586–595, 2009.
29. R. Niebuhr, P.-L. Cayrel, and J. Buchmann. Improving the efficiency of Generalized Birthday Attacks against certain structured cryptosystems. In *WCC 2011*, LNCS, pages 163–172. Springer, Apr 2011.
30. M.-J. O. Saarinen. Linearization attacks against syndrome based hashes. In K. Srinathan, C. Pandu Rangan, and Moti Yung, editors, *INDOCRYPT*, volume 4859 of *LNCS*, pages 1–9. Springer, 2007.
31. P. W. Shor. Algorithms for Quantum Computation: Discrete Logarithms and Factoring. In *SFCS '94: Proc. of the 35th Annual Symposium on Foundations of Computer Science*, pages 124–134. IEEE Computer Society, 1994.
32. J. Håstad and M. Näslund. BMGL: Synchronous key-stream generator with provable security, 2001.
33. D. Wagner. A generalized birthday problem. In Moti Yung, editor, *CRYPTO*, volume 2442 of *LNCS*. Springer, 2002.