14

New Classification of Existing Stream Ciphers

Khaled Suwais and Azman Samsudin
Universiti Sains Malaysia (USM)
Malaysia

1. Introduction

The demand on information security has extensively increased due to the sensitivity of the exchanged information over public communication channels. One of the primary goals of cryptographic systems (cryptosystems) is to help communicators exchanging their information securely. This goal is achieved by cryptographic applications and protocols. Transforming a message (plaintext) to an incomprehensive form (ciphertext) is accomplished by a process known as encryption. In contrast, transforming an encrypted message to its original form is accomplished by a process known as decryption.

In this chapter we focus on one type of symmetric key cryptosystems known as stream ciphers. Stream ciphers are important in securing static and streaming information. Therefore, this chapter intends to present a new classification of stream ciphers based on the keystream generators mechanism. The new classification is based on an extensive review of existing stream ciphers. This review showed that the constructional designs (underlying techniques) of some stream ciphers are similar, resulting in forming new categories for stream ciphers based on those similarities.

The main objectives of this chapter are summarized as follows: to provide comprehensive survey for existing stream ciphers based on their keystream generators, to analyze the security properties of each new category of stream ciphers and to explicate stream ciphers designs through a consistent classification that can assist the development process of new stream ciphers.

The new classification shows that stream ciphers are generally divided into three main categories: software-oriented, hardware-oriented and hybrid-design. This chapter will study the three categories extensively in order to understand the weak and strong points of each category.

2. Stream Cipher: Concept and Definition

Cryptographic systems are divided into two types of systems: Secret-key (Symmetric) and Public-key (Asymmetric) cryptosystems. In the later systems, the sender uses public information of the receiver to send a message securely to the receiver. The receiver then uses private information to recover the original message. In Secret-key cryptosystems, both the sender and receiver have previously set up secret information in which they use this information for encryption and decryption. Symmetric cryptosystems are further divided into block ciphers and stream ciphers.

The idea of stream ciphers was inspired from the famous cipher called the One-time Pad (Mollin, 2007; Delfs, 2002). This cipher is based on XOR'ing ($\oplus$) the message bits and the key bits. The One-time pad is defined by Delfs (2002) as shown in Equation 1:

$$\boldsymbol{E}: \{0,1\} \times \{0,1\} \to \{0,1\},\ (\boldsymbol{m}, \boldsymbol{k}) \to \boldsymbol{m} \oplus \boldsymbol{k} \tag{1}$$

where $\boldsymbol{m}$ and $\boldsymbol{k}$ denote plaintext and keystream bits respectively. The general formulas of encryption and decryption processes are described by Equation 2 and 3 respectively

$$\boldsymbol{E}_{k_i}(\boldsymbol{m}_i) = \boldsymbol{m}_i \oplus \boldsymbol{k}_i = \boldsymbol{c}_i \tag{2}$$

$$\boldsymbol{D}_{k_i}(\boldsymbol{c}_i) = \boldsymbol{c}_i \oplus \boldsymbol{k}_i = \boldsymbol{m}_i \tag{3}$$

Generally, stream cipher uses $\boldsymbol{n}$-iterations to generate $\boldsymbol{n}$-successive keystream based on the stream cipher internal state. The review conducted in this study shows that the processing techniques of the internal states of current stream ciphers are vary, where stream ciphers tend to be, in most cases, either hardware-oriented or software-oriented.

3. Stream Ciphers Categories

In contrast to block ciphers, stream ciphers have no standard model for their construction design, which leads cryptographers to construct various models for stream ciphers. This study classifies stream ciphers into categories whereby each category includes stream ciphers that share specific properties. The new classification divides stream ciphers into three main categories: hardware-based stream ciphers, software-based stream ciphers and hybrid designs of stream ciphers. The classification aims to look at stream ciphers from the implementation perspectives. The in-depth classification of hardware-based stream ciphers includes: FCSR/NLFSR-based, clock control-based and LFSR-based stream ciphers. On the other hand, software-based stream ciphers includes: T-function-based, block cipher-based, S-box-based and simple logical and arithmetic operations. The last category, the hybrid designs, includes those stream ciphers which depend on the combination of both hardware and software techniques in their constructional designs. The comprehensive classification of stream ciphers is illustrated by Fig. 1.

3.1 Hardware-Oriented Stream Cipher

The use of hardware implementations was significant in providing the security needed for various cryptographic applications. The widely used hardware implementation, as appears in the literature, relies on the use of LFSRs registers (Bojanic, et al., 2004; Ekdahl, 2003; Canteaut, et al., 2000). However, in this section we briefly introduce LFSRs and analyze the properties of each category and provide some examples on stream ciphers belonging to each category.

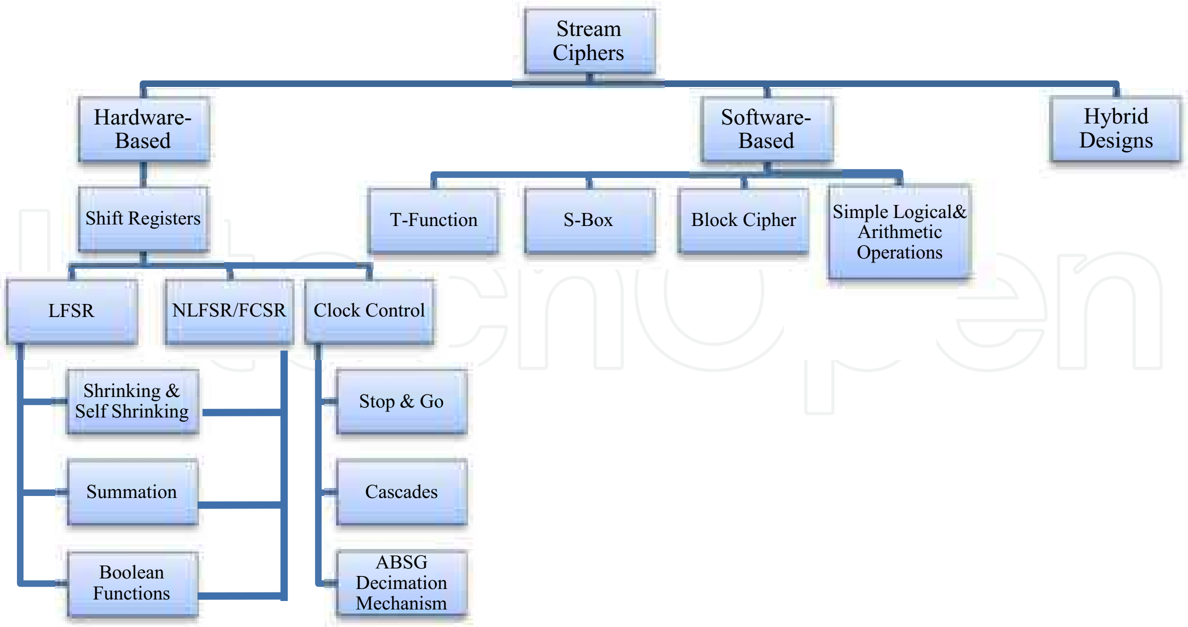


Fig. 1. Stream ciphers classifications

An LFSR is a shift register which is able to hold one symbol at a time and its input is a linear combination of the previous states. The symbols are normally elements from a field $\boldsymbol{F}_q$, where $\boldsymbol{q}$ = 2 refers to the binary fields and 2^w refers to some extension fields of the binary field for a given symbol's size w (Ekdahl, 2003).

Shift register of length $\boldsymbol{\ell}$ consists of $\boldsymbol{\ell}$ registers $0, \dots, \boldsymbol{\ell}-1$ as shown in Fig. 2. Each of these registers is able to hold one symbol, one input and one output. LFSRs rely on system clocks for their operations in which the system clock is responsible for timing all events. With every clocking of the LFSR, the registers read a new symbol from their input, and the symbols move forward from register $\boldsymbol{\ell}-1$ to register 0. However, the first register receives the new symbol as a linear combination of the symbols obtained from the previous clocking. Calculating the new symbol is basically determined by the feedback coefficients $\boldsymbol{C}_0, \boldsymbol{C}_1, \dots, \boldsymbol{C}_\ell$ as referred to in Fig. 2.

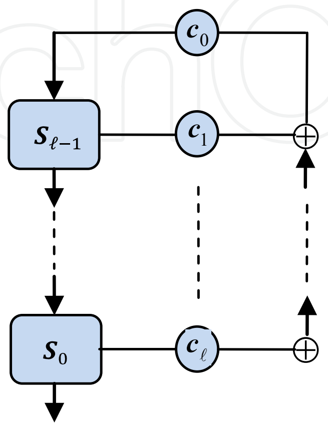


Fig. 2. LFSR of length $\boldsymbol{\ell}$

The concept of time clocking is important in LFSR functionality. When the device clocks at time $\boldsymbol{t} > 0$ we obtain a new symbol $\boldsymbol{S}_t \in \boldsymbol{F}_q$, where $\boldsymbol{S}_t$ is always satisfying the linear recurrence equation found in (Weisstein, 2008; Bolabattin, 2005) as shown in Equation 4:

$$\boldsymbol{c}_0^{-1}\boldsymbol{S}_{t+\ell} - \boldsymbol{c}_1\boldsymbol{S}_{t+\ell-1} - \boldsymbol{c}_1\boldsymbol{S}_{t+\ell-2} - \cdots - \boldsymbol{c}_\ell\boldsymbol{S}_t = 0,\ t > 0 \tag{4}$$

One important feature of using LFSR is its ability to produce an extremely long sequence of linear equation equal to 2$\boldsymbol{n}$-1, where $\boldsymbol{n}$ is the number of register elements in the LFSR. Moreover, LFSR is believed to deliver a stream cipher with uniformed distribution of the values generated by the keystream generator. However, the immediate output of LFSR is not acceptable to be used as a keystream since the production of the output value is done in linear fashion. In order to use LFSRs in generating keystream with minimum level of security, non-linear functions have to be added to LFSRs to make the bit production process after each clocking work in non-linear fashion. To achieve this purpose, different techniques have been introduced such as adding some non-linear filters, non-linear updates and irregular clocking to destroy the linearity found in LFSRs.

3.1.1 Shrinking and Self-Shrinking Generator

Coppersmith, et al., (1993) proposed a new generator which consists of two LFSRs and called it as a Shrinking Generator. The shrinking generator is designed as a pseudorandom keystream generator and it is preferred due to the simplicity of its design. The feedback coefficients are represented in polynomial representation. Each one of the LFSR produces a bit stream represented by $\boldsymbol{a}_i$ and $\boldsymbol{b}_i$ produced by LFSR-*A* and LFSR-*B* registers respectively to form the keystream $\boldsymbol{K}_s$. However, shrinking generators are subjected to known-plaintext distinguishing attack which is first introduced by (Ekdahl, et al., 2003). The attack detected some non-randomness in the distribution of the keystream bits. Note that the attacker is required to know the feedback polynomial of $\boldsymbol{A}$ to make this attack feasible.

Self-Shrinking generator is another variant of the shrinking generator concept. The generator rests on a single LFSR instead of using two different LFSRs as in shrinking generators. The procedure of clocking self-shrinking generators works by first clocking two bits from the LFSR, resulting in a pair of bits $(\boldsymbol{a}_1, \boldsymbol{a}_2)$. If $(\boldsymbol{a}_1, \boldsymbol{a}_2)$ equals to the value (1, 0) or (1, 1), it is taken to produce the pseudorandom bit 0 or 1 respectively. If the pair equals the value (0, 0) or (0, 1), the pair will be discarded because the output will always be a sequence of zeros as reported by Meier, et al., (1994).

Let $\boldsymbol{a} = (\boldsymbol{a}_0, \boldsymbol{a}_1, \boldsymbol{a}_2, \dots)$ be the output bits of a non-trivially initialized LFSRs of length $\boldsymbol{N}$. Therefore, $\boldsymbol{a}$ is a sequence with period 2^{N-1}. With respect to the period of $\boldsymbol{a}$, cryptanalysis attack in (Meier, et al., 1994) showed that if the period is at least $2^{N/2}$ and the linear complexity of the construction is $2^{\frac{N}{2}-1}$, attacker can attack the construction in $2^{0.7N}$ steps. Another attack based on a probabilistic approach was introduced by Mihaljevic (1996) and shows that self-shrinking generators can be attacked with complexity 2^{N-1} for any output sequence under certain limitation.

3.1.2 Summation Generator

Rainer Rueppel (1986) introduced a new generator based on the use of LFSRs called the Summation Generator. The idea behind this generator rests on the non-linearity provided by the carry-in integer addition. Rueppel uses this idea to use the output of several LFSRs through an adder with carry which in turn can provide a combination function with good non-linearity and high-order correlation properties (Robshaw, 1995). Rueppel's summation generator is described as in Equations 5 and 6 (Park, et al., 2005):

$$\boldsymbol{x}_i = \boldsymbol{a}_i \oplus \boldsymbol{b}_i \oplus \boldsymbol{c}_{i-1} \tag{5}$$

$$\boldsymbol{y}_i = \boldsymbol{a}_i \boldsymbol{b}_i \oplus (\boldsymbol{a}_i \oplus \boldsymbol{b}_i)\boldsymbol{y}_{i-1} \tag{6}$$

where $\boldsymbol{a}_i$ is the sequence generated by the first LFSR, $\boldsymbol{b}_i$ is the sequence generated by the second LFSR with the carry initialization value $\boldsymbol{y}_{i-1} = 0$.

In terms of the security of Rueppl's Summation Generator, the correlation probability of this generator showed that the generator is subjected to correlation attacks (Golic, 1996) since the probability of input-output correlation is ½ (Park, et al., 2005). However, several researchers have tried to improve the security of the summation generator to be used in stream ciphers. One example of stream ciphers using the summation generator is the E0 stream cipher which is used in the Bluetooth protocol (Kitsos, et al., 2003; Galanis, et al., 2005). E0 stream cipher consists of three components: payload key generator (initialize), keystream generator and summation combiner (encoder). However, various cryptanalysis and statistical attacks on E0 were presented in (Lu, et al., 2004), making E0 stream cipher insecure for cryptographic applications. Another example that appears in the literature is a parallelized stream cipher presented in 2002 by Lee and Moon (Lee, et al., 2002). The stream cipher rests on the improvement made on summation generators in (Lee, et al., 2000). Few years later, an algebraic attack against the improved generator was presented in (Han, et al., 2005), making the parallelized stream cipher subject to security vulnerability.

3.1.3 Boolean Function

In mathematics, a Boolean function is defined as a mapping of one or more binary input variables $\boldsymbol{L}^k$ to one binary output variable $\boldsymbol{L}$. Formally, we write the mapping function as in Equation 7:

$$\boldsymbol{\beta}: \boldsymbol{L}^k \rightarrow \boldsymbol{L} \tag{7}$$

where $\boldsymbol{L} = \{0,1\}$ is the Boolean domain of the Boolean function $\boldsymbol{\beta}$, and $\boldsymbol{k}$ is the non-negative integer called the rank of the function. One way of representing Boolean functions with a small number of input variables is by a truth table as illustrated in Table 1.

$\boldsymbol{a}_1$	$\boldsymbol{a}_2$	$F(a)$
0	0	0
0	1	0
1	0	1
1	1	0

Table 1. Truth table of the Boolean function $\boldsymbol{\beta}(\boldsymbol{a}_1, \boldsymbol{a}_2) = \boldsymbol{a}_1\boldsymbol{a}_2 + \boldsymbol{a}_{1i}$

For larger numbers of input variables, it is infeasible to list all the possible values of the truth table. Therefore, we have to use a compact description such as the Algebraic Normal Form (ANF) as shown Equation 8 (Ekdahl, 2003):

$$\boldsymbol{F} = \sum_{u \in F_2^n} \boldsymbol{\lambda}_u \left[\prod_{i=1}^{n} \boldsymbol{x}_i^{u_i}\right] \quad (8)$$

where $\boldsymbol{\lambda}_u \in \boldsymbol{F}_2$ and $\boldsymbol{u} = (\boldsymbol{u}_1, \boldsymbol{u}_2, \dots, \boldsymbol{u}_n)$. Another interesting property of Boolean function which attract several cryptographic applications is the balancing of the digits zero and one in the generated sequence. Generally, a Boolean function is said to be balanced if the probability of that function to produce bit 0 or 1, is ½ for all input variables chosen uniformly over $\boldsymbol{F}_2^n$.

Examples of stream ciphers based on the combination between LFSRs and Boolean functions are found in A5/1 (Biham, et al., 2000) and LILI-128 (Dawson, et al., 2000) stream ciphers. A5/1 was developed in 1987 and later became the most popular stream cipher in most European countries and United States to provide over-the-air communication privacy in GSM cellular telephone standard. The cipher is working in conjunction with three LFSRs (L-A, L-B, L-C) with irregular clocking. The three LFSRs vary in their length, in which the lengths are 19, 22 and 23 for L-A, L-B and L-C respectively. The main idea of A5/1 is to mix the cycled bits generated by the three LFSRs with respect to the irregularity in the clocking process. However, A5/1 seems to be vulnerable to cryptanalysis attacks as presented in (Biryukov, et al., 2000) and (Barkan, et al., 2003).

LILI-128 is another stream cipher which was introduced in 2000 (Dawson, et al., 2000). It uses two binary LFSRs and two Boolean functions to generate a pseudorandom binary keystream. The two functions are evaluated on the current state data and the feedback bits are calculated. Basically, LILI-128 divides the overall work into two subsystems, in which the first subsystem generates some output values and controls the clocking irregularly to control the other subsystem. Nevertheless, several attacks presented in (Jönsson, et al., 2002) and (Tsunoo, et al., 2005) makes LILI-128 insecure.

Finally, there are many other examples on stream ciphers using different techniques (functions, filters, etc) in conjunction with LFSRs to achieve higher security. One example is the stream cipher SNOW (Ekdahl, et al., 2003). SNOW is based on the use of LFSR of the length 16 over an extension to a binary field of 32, feeding a finite state machine. However, SNOW was attacked as presented in (Coppersmith, et al., 2002), and therefore invalidate SNOW to be used for secure applications.

3.1.4 NLFSR and FCSR Registers

Non-Linear Feedback Shift Register (NLFSR) and Feedback with Carry Shift Register (FCSR) are two other types of shift registers used in stream ciphers. The main purpose of these registers is to eliminate and destroy the linearity found in LFSRs. The design of NLFSR applies a non-linear function in the shift register to ensure the non-linearity in the output values from the corresponding shift register. NLFSRs are used in several stream cipher designs such as the Grain stream cipher. Grain was developed in 2004 and submitted to

eSTREAM project for evaluation in 2005 (Hell, et al., 2005). However, Grain was attacked in 2006 by two different cryptanalysts as found in (Maximov, 2006) and (Kucuk, 2006).
FCSRs are similar to LFSR but different in the sense that the elementary addition in FCSR is with propagation of carrier instead of addition modulo 2 as in LFSR. An example of FCSR-based stream cipher is the new stream cipher F-FCSR which was developed recently and submitted for eSTREAM project evaluation (Arnault, et al., 2006). However, F-FCSR was attacked by (Jaulmes, et al., 2006) due to the weaknesses found in the initialization mechanisms as well as lack of entropy of the internal state.

3.1.5 Clock Control

One way of introducing the non-linearity in the generated keystream is by having a shift register clocked irregularly. In other words, the keystream generation is controlled by the varying rate of register clocking. One way of achieving that is by having two or more shift registers such that the clocking of one register is dependent on the other register in some ways. Fig. 3 shows an example of a clock controlled generator called the Altering Step generator where the output of one LFSR controls the other LFSRs.

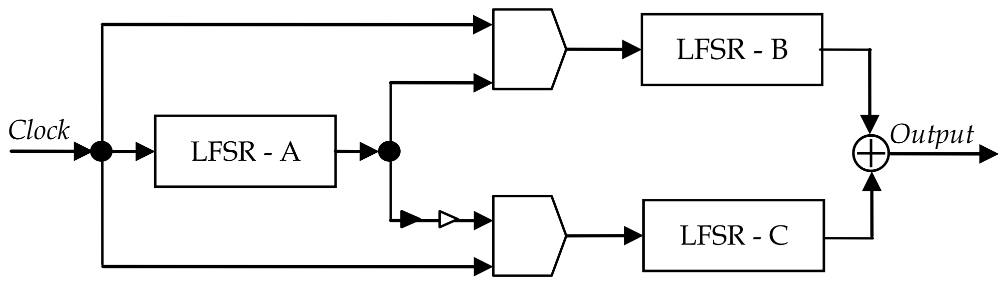


Fig. 3. Alternating step generator

There are various generators that are based on the idea of clock-controlling in shift registers for cryptographic purposes. Some of these generators are: Stop-and-Go, Cascades and ABSG Generators.

Stop-and-Go generator was first introduced in 1985 by Beth and Piper (Beth, et al., 1985). The idea of this generator is to let a control register R-A control the stepping of another register R-B. If the output of R-A is 1, then R-B is clocked. Otherwise R-B is not clocked. The output of R-B is then XORed with the output sequence of a third register R-C. The third register R-C has the same clocking ratio as in R-A. Beth and Piper believe that the stop-and-go generator is secure and immune against cryptanalysis attacks. However, the generator was subjected to efficient cryptanalysis attacks found in (Menezes, et al., 1997) and (Golic, et al., 2003).

Cascade generator is basically an extension of the stop-and-go generator, such that it is still relying on the idea that LFSRs are controlling each other. There are two types of cascades (Robshaw, 1995): The first type allows each register to generate ***l***-sequence and the second type restricts the length of each register to a prime length ***N*** with no feedback from any intermediate stage of the register. One example of the cascade stream ciphers is the Pomaranch stream cipher which is based on a Jump Controlled Sequence Generator

(cascade). Unfortunately, Pomaranch was vulnerable to several cryptanalysis attacks found in (Englund, et al., 2007) and (Cid, et al., 2006).
ABSG stream cipher is inspired by the shrinking and self-shrinking generator. Its main purpose was to provide irregularity for the generated keystream bits. Unlike shrinking generators, ABSG operates on a single input variable instead of two. ABSG also differs from the self-shrinking generator in that the production of $\boldsymbol{n}$-bits of output sequence requires approximate $3\boldsymbol{n}$-bit of input, while in self-shrinking, the production requires $4\boldsymbol{n}$-bit of input sequence (Afzal, et al., 2006). The stream cipher DECIM-128 presented in (Berbain, et al., 2005) is based on the use of LFSRs and ABSG decimation mechanism. It is a hardware-oriented stream cipher that handles a secret key of 80-bit length and public initialization vectors of 64-bit. The process of generating keystreams rests on the non-linearity filtered LFSR and the irregular decimation mechanism of ABSG. However, the attack presented in (Wu, et al., 2006) showed that DECIM is suffering from serious flaws in the initialization stage and the keystream generation algorithm stage.

3.2 Software-Oriented Stream Ciphers

In contrast to hardware-based stream ciphers, there are various designs of stream cipher based on bits manipulation (substitution, permutations, etc.), Boolean functions and other alternative designs. These designs of stream ciphers are classified under software-based stream ciphers in which they are not depending on hardware implementations for their security. This section will introduce a variety of stream cipher designs that are associated to different categories. The categorization is based on the mechanisms used in the process of generating keystream sequences used in these ciphers.

3.2.1 T-Function

In 2003, Klimov and Shamir introduced a new type of invertible round function (known as T-Function) by mixing some arithmetic and Boolean operations on full machine words (Klimov, et al., 2003). The name T-function refers to the triangular dependence between the columns of the operands. The function works as a mapping function formulated as in Equation 9:

$$\boldsymbol{f}\colon \boldsymbol{B}^{m\times n} \to \boldsymbol{B}^{k\times n} \tag{9}$$

where $\boldsymbol{B} = \{0,1\}$ is represented by a matrix and there is a dependency between the k-th column of the output with the first k set of columns of the input. It was designed to generate pseudorandom values of maximum length. The process of generating $\boldsymbol{X} = (\boldsymbol{x}_0, \boldsymbol{x}_1, \boldsymbol{x}_2, \dots)$ is described in (Klimov, et al., 2004) and shown in Equation 10:

$$\boldsymbol{x}_i = \boldsymbol{x}_{i-1}^2 \vee \boldsymbol{C} + \boldsymbol{x}_{i-1}\ mod\ 2^n \tag{10}$$

where $\vee$ referes to OR operation and $\boldsymbol{C}$ is used to determine a set of constants defined in the linear equation to hold all the sequences generated by the T-function. Since T-functions are so recent, only few stream ciphers appear in the literatures are based on them. One example is the stream cipher TSC-1 proposed by (Hong, et al., 2005). The proposed cipher is based on a single cycle T-function. TSC-1 works in conjunction with a filter function and 4×4 S-Box.

In general, T-function was subjected to several attacks such as the correlation attack based on the linear approximation of the T-function. The attack was successfully applied on TSC-128 with a complexity of 2^{42} known keystream bits to distinguish it from random (Muller, et al., 2005). The other attack presented in (Künzli, et al., 2005) describes a distinguishing attack on single-word and multi-word T-functions based on the deviation found in the integer differences of consecutive outputs with a complexity of 2^{32}. The importance of T-function comes from the efficiency of implementing it from both hardware and software perspectives. However, it seems that researchers need to put more efforts on developing and enhancing the security aspects of T-function.

3.2.2 S-Box

A substitution box or also known as S-box is an important component of different cryptographic primitives. S-box basically works as a mapping of $\boldsymbol{m}$ input bits into $\boldsymbol{n}$ output bits as visualized in Fig. 4, resulting in an $\boldsymbol{m} \times \boldsymbol{n}$ S-box.

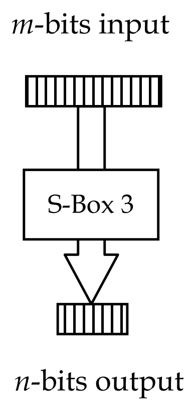


Fig. 4. Input/output mapping using S-Box

The design of S-box comes in two types: fixed and dynamic S-box. Fixed S-boxes rest on pre-computed values calculated in several ways based on the cryptographic component being used. Dynamic S-box are more interesting since the values in the S-box change during the execution. One way of representing S-boxes is by implementing them as table lookups of 2^n entries (Ekdahl, 2003). Another possibility of representing S-box is by calculating the S-box's entries by using a Boolean function as shown in Equation 11:

$$\boldsymbol{F}(\boldsymbol{x}) = \left(\boldsymbol{F}(\boldsymbol{x}_1), \boldsymbol{F}(\boldsymbol{x}_2), \ldots, \boldsymbol{F}(\boldsymbol{x}_m)\right) \tag{11}$$

In this category of stream ciphers, we found few ciphers whose designs are based on S-box. Two examples are discussed here: MUGI and WAKE stream ciphers. MUGI stream cipher was introduced in 2002 as an efficient stream cipher in hardware and software implementations (Watanabe, et al., 2002). MUGI uses a secret key and internal vector of 128-bit length to generate a random string of 64-bit length for each round. The internal state of MUGI consists of two internal states (state $\boldsymbol{a}$ and buffer $\boldsymbol{b}$) updates by two identical functions (called F-function). The F-function uses three main techniques: key addition, non-linear S-box and MDS (Maximum Distance Separable) matrix for linear transformation as described in Fig. 5.

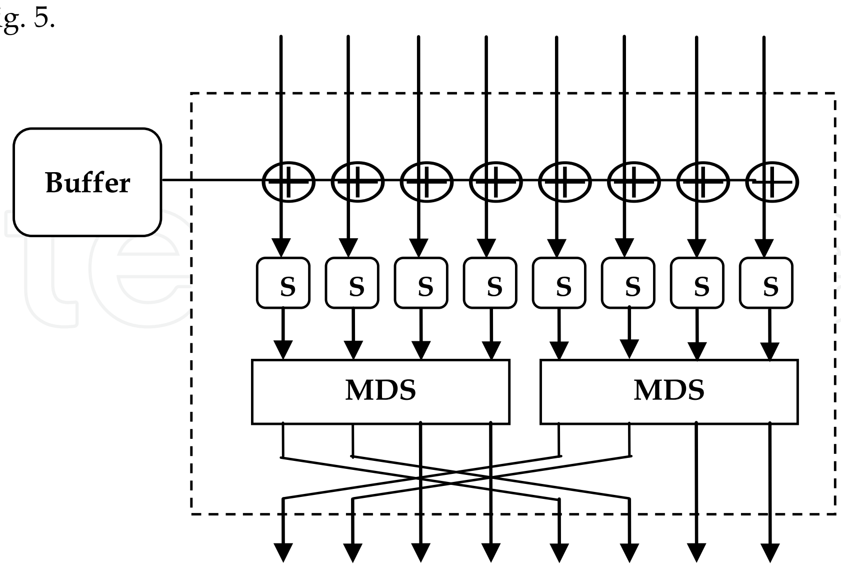


Fig. 5. F-function of MUGI

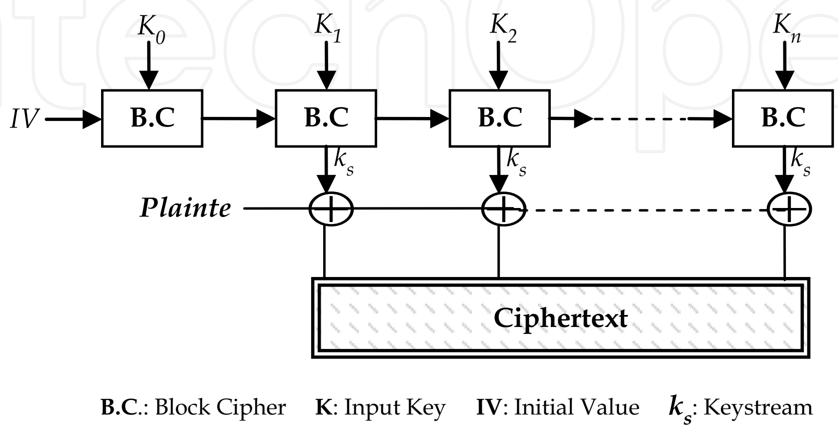
MUGI is not broken yet. However a weakness found in the linear part of MUGI was presented in (Golic, 2004), proved that the real response of the buffer without the feedback from the S-box consists of binary linear recurring sequences with linear complexity and with a very small period of 48 cycles. This theoretical analysis showed that by using the weakness mentioned above, it is possible to use linear cryptanalysis to attack MUGI.

Another example of stream cipher belonging to this category is the WAKE (Word Auto Key Encryption) stream cipher invented by David J. Wheeler (1993). WAKE has a simple structure and performs fast. It produces 4***n***-bit words to be XORed with plaintext to generate ciphertext, or with ciphertext to generate plaintext. The generation of new key depends on the ciphertext produced in the previous round. WAKE uses an S-box of 256 32-bit values with special property where some bytes are obtained from a permutation of all possible bytes, and some other bytes are generated randomly. The S-box of WAKE is not working independently from the overall process of keystream generation; instead it is working as part of function ***G*** which uses S-box in conjunction with other shifting operations. However, WAKE was subjected to a chosen plaintext or chosen ciphertext attack, which was fully analyzed in (Pudovkina, 2001). The analysis includes implementing two chosen plaintext attacks on WAKE with a complexity of $10^{19.2}$ and $10^{14.4}$ for the first and second attacks respectively.

It seems that S-box is efficient in providing non-linearity with efficient performance in the internal states of the keystream generators. Designing a cryptographically strong S-box is not easy. Therefore, any misuse of S-box in stream cipher leads to serious security vulnerabilities.

3.2.3 Block Cipher

This is another approach used in the design of stream ciphers. The block cipher is used as a core of the keystream generator of the corresponding stream cipher. The construction of the stream ciphers that belong to this category uses known block ciphers in their keystream generator such as using AES in the stream cipher (Biryukov, 2005). The general structure of stream ciphers based on block cipher is shown in Fig. 6.



K_0
K_1
K_2
K_n
IV
B.C
B.C
B.C
B.C
k_s
k_s
k_s
Plainte
Ciphertext
B.C.: Block Cipher **K**: Input Key **IV**: Initial Value k_s: Keystream

Fig. 6. Stream cipher based on block cipher scheme

Another design philosophy of stream ciphers in this category is based on the Substitution-Permutation Network (SPN) of block ciphers instead of using the components of block ciphers as appeared in Hermes8 stream cipher (Kaiser, 2005). The security of such a design depends on the underlying block cipher (component or technique) that resides at the core of the stream cipher. Up to this day, among the submitted stream ciphers based on block ciphers, LEX and Sosemanuk are the only two ciphers which have moved to the third phase of evaluation of eSTREAM project.

3.2.5 Simple Logical and Mathematical Operations

There are stream ciphers which do not fit into the mentioned categories above. Some of these ciphers are based on bitwise addition and bits rotation operations as in Phelix, SEAL and RC4, while others based on mixing various functions in conjunction with some addition and rotation operations as in Rabbit. In this category we will briefly describe Phelix, SEAL and Rabbit stream ciphers.

- **Phelix Stream Cipher**

Phelix stream cipher (Whiting, et al., 2005) is a high speed stream cipher selected for the software and hardware profiles of eSTREAM project for performance evaluation. Phelix supports an 8-bit to 256-bit length key and 128-bit nonce to generate the keystream bits with embedded MAC code for authentication. The main operations of Phelix are: addition modulo 2^{32}, bitwise XOR and rotation operations. The state of Phelix is broken into two groups: five state words called *active* states which are always participating in updating the internal function and four states called *old* state which is only used in the process of keystream generation.

Since Phelix provides authentication service during transmission, extra processing is done to produce a 128-bit MAC tag to be embedded to the message. Phelix requires 20 rounds in order to produce a single block. The main operations in one block of Phelix is only low-cost operation, in which they are fast in software and hardware implementations. However, Phelix has not moved to the third phase of the eSTREAM project evaluation due to some security vulnerability. Differential-linear attacks presented by Wu and Preneel (2007) showed that with the assumption of reusing the nonce, the key of Phelix can be recovered with complexity 2^{37} chosen plaintext words and $2^{41.5}$ operations. In this attack, the authors showed that Phelix is an insecure stream cipher since recovering the key by reusing the nonce (incorrect use of the nonce) is possible: "In practice an attacker may gain access to a Phelix encryption device for a while, reuse a nonce and recover the key. We thus consider Phelix as insecure" (Wu, et al., 2007).

- **Rabbit Stream Cipher**

Rabbit is another design of stream ciphers based on iterating a set of coupled non-linear functions – or as the authors called them discretized chaotic maps (Boesgaard, et al., 2003). It uses a 128-bit key and 64-bit initial vector (IV) as input parameters to generate a stream of 128-bit blocks. The encryption is performed by XOR'ing this block with the plaintext block. The inner state of Rabbit consists of 513 bits. The first 512 bits represent 8-state variables ($\boldsymbol{X}_0$, …, $\boldsymbol{X}_7$) of 32-bit length each and 8-counter variables ($\boldsymbol{C}_0$, …, $\boldsymbol{C}_7$). The remainder bit is used as a counter carry bit, $\boldsymbol{b}$. The important part of any stream cipher is the next state function since it is the part that often needs to generate a new keystream. In Rabbit, the next state function is based on function $\boldsymbol{g}$ for mapping two 32-bit inputs to one 32-bit output. Rabbit uses function $\boldsymbol{g}$ to update the inner variables states as shown in Fig. 7.

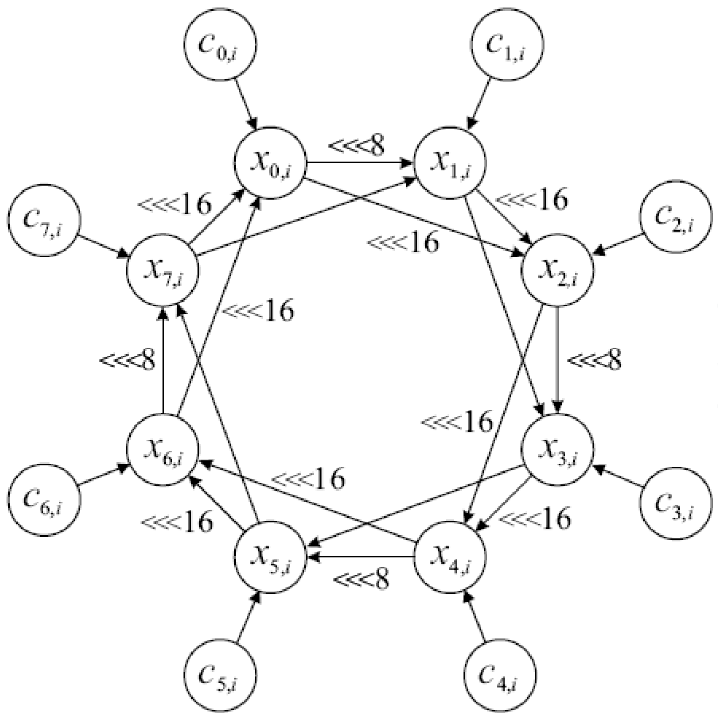


Fig. 7. Updating the inner states of Rabbit (Boesgaard, et al., 2003)

It seems that Rabbit stream cipher is strong against cryptanalysis attacks. It is selected among few other ciphers for further evaluation by eSTREAM project. However, a small bias in the output of Rabbit exists (Aumasson, 2007). Even so, Rabbit is still considered a secure stream cipher since the complexity of the distinguisher is significantly higher than the brute-force attack on the key space, 2^{128}.

- **SEAL Stream Cipher**

SEAL (Software-optimized Encryption Algorithm) is a stream cipher that was designed to work efficiently on software implementation (Rogaway, et al., 1994). SEAL is a length-increasing pseudorandom string mapping function that uses 160-bit encryption key to map (stretch) a 32-bit input length to an $\boldsymbol{n}$-bit output length. In the pre-processing stage, SEAL uses the hash algorithm SHA-1 (National, 2002) as a part of the table-generation function to stretch the key into a large table. Therefore, part of SEAL's security depends on the security of the used hash algorithm (SHA-1). In terms of the required computation, SEAL requires intensive pre-computation for initializing several large look-up tables with total size approximately 3 KB in size.

In terms of security, SEAL is designed to generate up to 2^{48} bytes of output per input seed. An attack in 1997 showed that this output can be distinguished from random after 2^{34} bytes of output (Coppersmith, et al., 2002). The attack was the reason behind the modification on SEAL, resulting in the modified algorithm SEAL 3.0. In 2001, Fluhrer introduced an attack on SEAL 3.0 that can distinguish the output from random after 2^{44} output bytes (Fluhrer, 2001). It is obvious that SEAL needs to avoid using the same seed after 2^{44} outputs to avoid these types of attacks.

- **RC4 Stream Cipher**

This is yet another important example of stream cipher design. The well known stream cipher is widely used in many security protocols and software applications such as SSL and WEP protocols integrated into Microsoft Windows, Lotus Notes, Apple AOCE, Oracle Secure SQL and many other applications. RC4 (Rivest, 1992) was developed by Ron Rivest in 1987 and the design was kept secret until 1994, until someone anonymously posted it to the Cypherpunks mailing list. The cipher uses a variable key-size with compact code size and it is suitable for byte-oriented processors. The encryption process in RC4 is done by generating a keystream to be XORed with a stream of plaintext to produce a stream of ciphertext.

Generating keystream in RC4 comprises two algorithms: The Key-Scheduling Algorithm (KSA) and the Pseudo-Random Generation Algorithm (PRGA). The KSA algorithm uses a permutation array S of all 256 possible bytes. The two algorithms cooperate with each other as follows: the KSA derives the internal secret state from a variable key size between 40 and 256 bits. PRGA in turn modifies the internal state and produces an output. The initialization process in PRGA sets $\boldsymbol{i}$ and $\boldsymbol{j}$ to 0, and then $\boldsymbol{i}$ is incremented as a counter and $\boldsymbol{j}$ is incremented by adding the value of the permutation array S pointed to by $\boldsymbol{i}$. The two values of S pointed to by $\boldsymbol{i}$ and $\boldsymbol{j}$ are swapped and the output is resulted by adding $S[\boldsymbol{i}] + S[\boldsymbol{j}]$ modulo 256 as shown in Fig. 8.

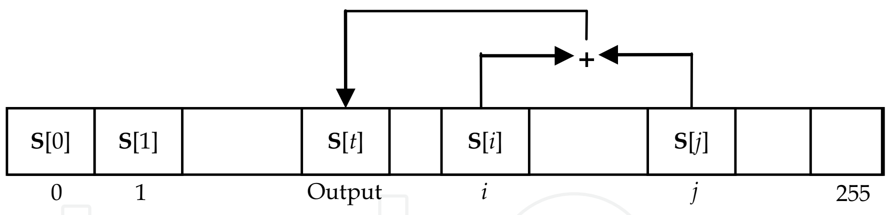


Fig. 8. PRGA round operation

Similar to PRGA, KSA initializes S to the identity permutation and initializes i and j to 0. Sequentially, KSA applies 256 rounds in which i stepped across S and j is updated by adding $S[i]$ to it and the next word of the key. At the present time, RC4 is not recommended for use in new applications. Several weaknesses of the KSA algorithm of RC4 (Fluhrer, et al., 2001) can be summarized in two points. First weakness is the existence of massive classes of weak keys. These classes enable the attackers to determine a large number of bits of KSA output by using a small part of the secret key. Thus, the initial outputs of the weak keys are disproportionately affected by a small portion of key bits. The second weakness rests on a related key vulnerability.

Brute Force attack on RC4 is possible by implementing exhaustive key-searches on Field Programmable Gate Array (FPGAs) using a Network on Chip (NoC) architecture (Couture, et al., 2004). The idea of this attack depends on two components: Key-Checker Unit and the Controller. The latter is responsible for distributing the key space. Key-Checker Unit is used to check each key independently. Therefore, using more than one Checker in a network will provide an adjustable level of parallelism. The research's results shows that RC4 is quite vulnerable to brute-force attack and it is possible to crack RC4 in minutes with a very large FPGA of 500 Checker units in a network.

Other kinds of attacks on RC4 have been presented recently. Results in (Mantin, 2007) showed a statistical bias of the digraphs distribution of the generated stream of RC4. Furthermore, a distinguishing attack was developed based on the statistical bias found in the output sequences (Tsunoo, et al., 2007). This bias is used along with the first two words of a keystream associated with approxiemtly 2^{30} secret keys.

3.3 Hybrid Designs

In this category we discuss other designs of stream ciphers based on a combination of hardware devices and software techniques to achieve the required security. Most of stream ciphers in this category depend on LFSRs as the main component in the core of the stream cipher. The software techniques vary from using T-function as in ABC stream cipher, dynamic permutations as in Polar Bear stream cipher, and look-up tables as in ORYX. In this section we will describe each stream cipher mentioned above and analyze the ciphers' structures and discuss their security strength.

3.3.1 ABC Stream Cipher

ABC is a stream cipher algorithm developed in 2005 (Anashin, et al., 2005) and submitted for eSTREAM project for further evaluation. It deals with a 128-bit key and 128-bit IV. ABC consists of 38, 32-bit registers. The registers are divided into two groups: 3 registers ($\boldsymbol{z}^0$, $\boldsymbol{z}^1$, **x**) are representing the state of ABC, and 35 registers ($\boldsymbol{d}_0$, $\boldsymbol{d}_1$, **e**, $\boldsymbol{e}_0$,…, $\boldsymbol{e}_{31}$) represent the constant parameters fed to the cipher. In conjunction with the LFSRs, ABC uses three main functions denoted by $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$ as shown in Fig. 9.

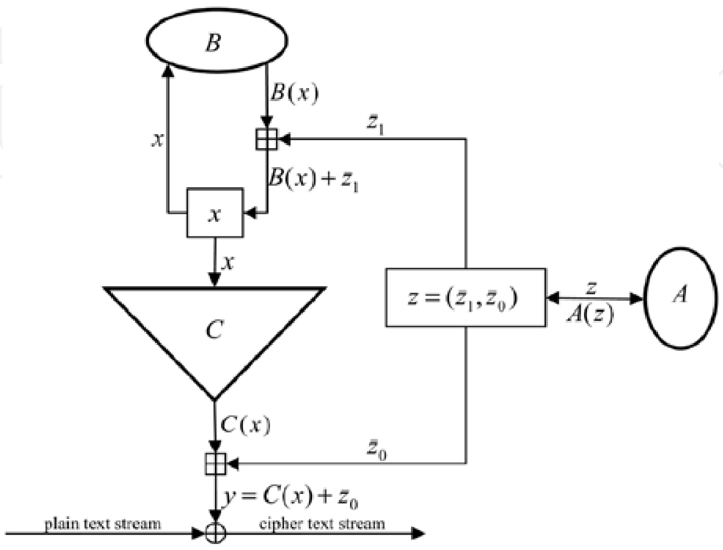


Fig. 9. Functions $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$ in the keystream generator ABC (Anashin, et al., 2005)

Function $\boldsymbol{A}$ is a linear transformation over the space $\boldsymbol{GF}(2^{64})$, and it is defined by a polynomial characteristic LFSR of length 64. Function $\boldsymbol{B}$ is a T-function with the restrictions that, for the two parameters $\boldsymbol{d}_0$ and $\boldsymbol{d}_1$, one must choose these two parameters such that $\boldsymbol{d}_0 \equiv 1 \ (mod\ 2)$ and $\boldsymbol{d}_1 \equiv 0 \ (mod\ 4)$ to guarantee that function $\boldsymbol{B}$ is a single cycle map. Lastly, function $\boldsymbol{C}$ is a highly non-linear mapping function (as the authors claimed).

In terms of the security, several attacks on ABC make it fails moving to the third phase of eSTREAM project. Based on the weakness of function $\boldsymbol{C}$ as illustrated in (Khazaei, 2005), a correlation based divide-and-conquer attack was able to find 63-bit of the state by searching 2^{63} possible choices. More specifically, the attack on ABC has a time complexity of 2^{108} to find the whole initial state bits, which is faster than brute-force attack.

A fast correlation attack on ABC was presented in (Zhang, et al., 2006). The attack depends on some weak keys to recover the internal state. Identifying one weak key and recovering the internal state of that key has low computation complexity. The attack is mainly looking for weak keys which were detected in function $\boldsymbol{C}$ for 2^{32} keystream generated from 2^{32} random keys (where each keystream is with 1615 output), the keystream can be distinguished from random. Finding one weak key based on this attack requires $2^{32} \times 1615 \times 4 = 2^{44.7}$ bytes, 2^{45} XOR and 2^{44} addition.

It is obvious that the ABC stream cipher was not strong enough against cryptanalysis attacks. The cipher is considered fast compared to other software-oriented stream ciphers. Nevertheless, choosing cryptographic primitives for secure applications requires more attention on the security side of those primitives. Hence, ABC failed to be considered as a member of the third phase of eSTREAM project due to the existence of some security vulnerability in its design.

3.3.2 Polar Bear Stream Cipher

Polar Bear was presented in 2005 and submitted to eSTREAM project for evaluation by Johan Hastad and Mats Naslund as reported in (Nada, et al., 2005). The cipher uses one 7-word LFSR ($\boldsymbol{R}^0$) and one 9-word LFSR ($\boldsymbol{R}^1$) of length 112-bit and 144-bit respectively. Updating the internal state depends on these two LFSRs along with dynamic permutation of bytes, $\boldsymbol{D}_8$. The cipher deals with 128-bit key and up to 32 byte initial vector. Polar Bear uses the Rijndael key schedule for its key schedule algorithm. The initialization process is achieved by taking the expanded key and initial vector, and applies 5 rounds of Rijndael encryption with block length 256. $\boldsymbol{R}^0$ and $\boldsymbol{R}^1$ are then loaded by the resulted ciphertext, and $\boldsymbol{D}_8$ as well as Rijndael S-box are initialized.

The authors of Polar Bear claimed that the cipher is efficient and secure due to the combination of LFSRs with the dynamical permutation. However, a Guess-and-Determine attack presented by Mattsson (Mattsson, 2006) and improved in (Hasanzadeh, et al., 2006), was able to recover the initial states of the registers with time complexity of 2^{79} (by Mattsson attack) and with time complexity of $2^{57.4}$ (by the improved attack). These two attacks showed that the Polar Bear stream cipher is not secure due to the inappropriate usage of the LFSR combined with the dynamical permutations. To counter this attack, it was suggested in (Hasanzadeh, et al., 2006) to initialize the dynamic permutation with an 8×8 key initial vector dependent S-box, provided that the permutation is random to attackers.

3.3.3 ORYX Stream Cipher

ORYX is a stream cipher algorithm that has been proposed for use in North American digital cellular systems. The structure of ORYX is very simple, based on binary LFSRs, S-box (look-up table) and permutation. More specifically, ORYX has four components, three LFSRs of 32-bit length ($\boldsymbol{LFSR}_A$, $\boldsymbol{LFSR}_B$, $\boldsymbol{LFSR}_K$), and an S-box containing a known permutation of the values ranging from 0 to 255, denoted by $\boldsymbol{L}$. The feedback function for $\boldsymbol{LFSR}_K$ (polynomial characteristic) is defined as in Equation 12:

$$\boldsymbol{x}^{32} + \boldsymbol{x}^{28} + \boldsymbol{x}^{19} + \boldsymbol{x}^{18} + \boldsymbol{x}^{16} + \boldsymbol{x}^{14} + \boldsymbol{x}^{11} + \boldsymbol{x}^{10} + \boldsymbol{x}^{9} + \boldsymbol{x}^{6} + \boldsymbol{x}^{5} + \boldsymbol{x} + 1 \tag{12}$$

while the feedback functions for $\boldsymbol{LFSR}_A$ is defined as in Equation 13:

$$\boldsymbol{x}^{32} + \boldsymbol{x}^{26} + \boldsymbol{x}^{23} + \boldsymbol{x}^{22} + \boldsymbol{x}^{16} + \boldsymbol{x}^{12} + \boldsymbol{x}^{11} + \boldsymbol{x}^{10} + \boldsymbol{x}^{8} + \boldsymbol{x}^{7} + \boldsymbol{x}^{5} + \boldsymbol{x}^{4} + \boldsymbol{x}^{2} + \boldsymbol{x} + 1 \tag{13}$$

and finally, $\boldsymbol{LFSR}_B$ is defined as in Equation 14 as follows:

$$\boldsymbol{x}^{32} + \boldsymbol{x}^{31} + \boldsymbol{x}^{21} + \boldsymbol{x}^{20} + \boldsymbol{x}^{16} + \boldsymbol{x}^{15} + \boldsymbol{x}^{6} + \boldsymbol{x}^{3} + \boldsymbol{x} + 1 \tag{14}$$

The keystream generation is performed by clocking the three LFSRs along with some fixed permutations in order to obtain the high bytes of the current state of each LFSR using a combining function as stated in Equation 15:

$$\boldsymbol{keystream} = (High8_k + \boldsymbol{L}[High8_A] + \boldsymbol{L}[High8_B]) \bmod 256 \quad (15)$$

ORYX is not a secure stream cipher due to the efficient attack presented in (Wagner, et al., 1998). The attack can directly recover the full 96 bits internal state using only 25-27 bytes of known plaintext with time complexity of 2^{16}. Therefore, these results showed that ORYX provides a very low level of security and not suitable for cryptographic applications.

4. Discussion and Conclusion

Increasing the security of the keystream generator is the primary goal for researchers who intend to develop new stream cipher algorithms. The classification presented in this chapter showed that stream ciphers are mainly either software oriented or hardware oriented. In some cases, there are stream ciphers which rely on a combination of hardware devices and software techniques in the design of their keystream generators.

From the security perspective, several stream ciphers (hardware-oriented and software-oriented) are found vulnerable to either cryptanalysis attacks, statistical biased or both. Cryptanalysis attacks on stream ciphers come in two types: hardware-based attacks and software-based attacks. In both types of attacks, attacker tries to extract useful information from the keystream generator in order to obtain the secret key or the plaintext message. Statistical biased such as correlation between keystreams, patterns and randomness are the main issues found in hardware-oriented stream ciphers. On the other hand, the underlying techniques used in software-oriented stream ciphers are found vulnerable to cryptanalysis attacks, due to the relative simplicity of their constructional designs.

Reviewing the constructional designs of stream ciphers leads us to the fact that the keystream generator must be constructed on solid bases. These solid bases can be represented by: linearity-elimination techniques, mathematical hard problems, chaotic behaviours or other secure techniques. The main goal of these new techniques is to provide cryptographic applications with secure stream ciphers against cryptanalysis and statistical attacks.

5. References

Afzal, M. K., & Masood, A. (2006). Comparative Analysis of the Structures of eSTREAM Submitted Stream Ciphers. *In Proc. The Second International Conference on Emerging Technologies* (pp. 245-250). Peshawar, Pakistan: IEEE-ICET.

Anashin, V. B., & Kumar, A. (2005, April 29). *ABC: A New Fast Flexible Stream Cipher.* Retrieved May 20, 2008, from The eSTREAM Project: http://www.ecrypt.eu.org/stream/ciphers/abc/abc.pdf

Arnault, F. B., & Lauradoux, C. (2006, January 2). *Update on F-FCSR Stream Cipher.* Retrieved May 26, 2008, from The eSTREAM Project: http://www.ecrypt.eu.org/stream/papersdir/2006/025.pdf

Aumasson, J. (2007, January 2). *On bias of Rabbit.* Retrieved May 30, 2008, from The eSTREAM Project: http://www.ecrypt.eu.org/stream/papersdir/2007/033.pdf

Barkan, E. B., & Keller, N. (2003). Instant Ciphertext-Only Cryptanalysis of GSM Encrypted Communication. In *Advances in Cryptology - CRYPTO 2003* (Vol. 2729 of LNCS, pp. 600-616). Berlin: Springer.

Berbain, C. B., & Sibert, H. (2005, April 29). *DECIM-128.* Retrieved May 26, 2008, from The eSTREAM Project: http://www.ecrypt.eu.org/stream/p3ciphers/decim/decim128_ p3.pdf

Beth, T., & Piper, F. (1985). The stop-and-go generator. *In Proc. of the EUROCRYPT 84 workshop on Advances in cryptology: theory and application of cryptographic techniques* (pp. 88 - 92). Paris, France: Springer-Verlag.

Biham, E., & Dunkelman, O. (2000). Cryptanalysis of the A5/1 GSM Stream Cipher. In *Progress in Cryptology – INDOCRYPT 2000* (Vol. 1977, pp. 43-51). Berlin: Springer.

Biryukov, A. (2005, April 29). *A new 128 bit key stream cipher : LEX.* Retrieved June 2, 2008, from The eSTREAM Project: http://www.ecrypt.eu.org/stream/ciphers/lex/lex.pdf

Biryukov, A. S., & Wagner, D. (2000). Real Time Cryptanalysis of A5/1 on a PC. In *Proc. Fast Software Encryption*, (pp. 1-18). New York.

Boesgaard, M. V., & Scavenius, O. (2003). Rabbit: A New High-Performance Stream Cipher. In *Fast Software Encryption* (Vol. 2887 of LNCS, pp. 307-329). Springer.

Bojanic, S. C., & Taladriz, O. (2004). Stream Cipher Cryptanalysis Based on Edit-Distance: A Hardware Approach. *Tatra Mt. Math. Pub* , 17-29.

Bolabattin, N. (2005, May 13). *Random Number Generator Using Leap-Forward Techniques*. Retrieved May 21, 2008, from http://www.pldesignline.com/showArticle.jhtml?articleID=192200271

Canteaut, A., & Filiol, E. (2000). *Ciphertext Only Reconstruction of LFSR-based Stream Ciphers.* LE CHESNAY: Unit´e de recherche INRIA Rocquencourt.

Cid, C. G., & Johansson, T. (2006). Cryptanalysis of Pomaranch. *In Proc. of IEE Information Security*, *153*, pp. 51-53.

Coppersmith, D. H., & Jutla, C. (2002). Cryptanalysis of stream ciphers with linear masking. In *Advances in Cryptology - CRYPTO'02* (Vol. 2442 of LNCS, pp. 515-532). Springer.

Coppersmith, D. H., & Jutla, C. (2002). Scream: A Software-Efficient Stream Cipher. In *Fast Software Encryption* (Vol. 2365 of LNCS). Springer.

Couture, N., & Kent, K. (2004). The Effectiveness of Brute Force on RC4. *In Proc. of the Second Annual Conference on Communication Networks and Services Research* (pp. 333-336). Washington, USA : IEEE Computer Society.

Dawson, E. C., & Simpson, L. (2000). The LILI-128 Keystream Generator. *In Proc. of First NESSIE Workshop.* Heverlee, Belgium.

Delfs, H. (2002). *Introduction to Cryptography: Principles and Applications.* Springer.

Ekdahl, P. M., & Johansson, T. (2003). Predicting the Shriking Generator with Fixed Connections. In E. Biham (Ed.), *Advances in Cryptology - EUROCRYPT2003* (Vol. 2656 of LNCS, pp. 330-344). Springer.

Ekdahl, P. (2003). *On LFSR Based Stream Ciphers: Analysis and Design.* Lund, Sweden : Lund University.

Ekdahl, P., & Johansson, T. (2003). A New Version of the Stream Cipher SNOW. In *Selected Areas in Cryptography* (Vol. 2595 of LNCS, pp. 47-61). Berlin: Springer.

Englund, H. H., & Johansson, T. (2007). Two General Attacks on Pomaranch-Like Keystream Generators. In *Fast Software Encryption* (Vol. 4593 of LNCS, pp. 274-289). Berlin: Springer.

Fluhrer, S. (2001). Cryptanalysis of the SEAL 3.0 pseudorandom function family. In *Fast Software Encryption* (Vol. 2355 of LNCS, pp. 135–143). Springer.

Fluhrer, S. M., & Shamir, A. (2001). Weaknesses in the Key Scheduling Algorithm of RC4. In *Selected Areas in Cryptography* (Vol. 2259 of LNCS, pp. 1-24). Berlin: Springer.

Freier, A. K., & Kocher, P. (1996). *The SSL Protocol Version 3.0.* Retrieved January 15, 2008, from http://wp.netscape.com/eng/ssl3/ssl-toc.html.

Galanis, M. K., & Goutis, C. (2005). Comparison of the Hardware Implementation of Stream Ciphers. *The International Arab Journal of Information Technology , 2* (4), 267-274.

Golic, D., & Menicocci, R. (2003). Edit probability correlation attacks on stop/go clocked keystream generators. *Journal of cryptology , 16* (1), 41-68.

Golic, J. (2004). A Weakness of the Linear Part of Stream Cipher MUGI. In *Fast Software Encryption* (Vol. 3017 of LNCS, pp. 178-192). Berlin: Springer.

Golic, J. (1996). Correlation properties of a general combiner with memory. *Journal of Cryptology* , 111-126.

Han, D., & Lee, M. (2005). An algebraic attack on the improved summation generator with 2-bit memory. *Information Processing Letters , 93* (1), 43 - 46.

Hasanzadeh, M. S., & Khazaei, S. (2006). *Improved Cryptanalysis of Polar Bear.* Retrieved May 29, 2008, from The eSTREAM Project: http://www.ecrypt.eu.org/stream/papersdir/084.pdf

Hell, H. J., & Meier, W. (2005, April 29). *Grain - A Stream Cipher for Constrained Environments.* Retrieved May 26, 2008, from The eSTREAM Project: http://www.ecrypt.eu.org/stream/ciphers/grain/grain.pdf

Helleseth, T. J., & Kholosha, A. (2006, January 2). *Pomaranch - Design and Analysis of a Family of Stream Ciphers.* Retrieved May 27, 2008, from The eSTREAM Project: http://www.ecrypt.eu.org/stream/papersdir/2006/008.pdf

Hong, J. L., & Han, D. (2005). A New Class of Single Cycle T-Functions. In *Fast Software Encryption* (Vol. 3557 of LNCS, pp. 68-82). Berlin: Springer.

Jaulmes, É., & Muller, F. (2006). Cryptanalysis of the F-FCSR Stream Cipher Family. In *Selected Areas in Cryptography* (Vol. 3897 of LNCS, pp. 20-35). Berlin: Springer.

Jönsson, F., & Johansson, T. (2002). A fast correlation attack on LILI-128. *Information Processing Letters , 81* (3), 127 - 132.

Kaiser, U. (2005, April 29). *Hermes Stream Cipher.* Retrieved May 20, 2008, from eSTREAM PHASE 2: http://www.ecrypt.eu.org/stream/ciphers/hermes8/hermes8.pdf

Khazaei, S. (2005). *Divide and conquer attack on ABC stream cipher.* Retrieved May 15, 2008, from eSTREAM, ECRYPT Stream Cipher Project: http://www.ecrypt.eu.org/stream

Kitsos, P. N., & Koufopavlou, O. (2003). Hardware Implementation of Bluetooth Security. *IEEE Pervasive Computing , 2* (1), 21-29.

Klimov, A., & Shamir, A. (2003). A New Class of Invertible Mappings. In *Cryptographic Hardware and Embedded Systems - CHES 2002* (Vol. 2523 of LNCS, pp. 470-483). London, UK: Springer.

Klimov, A., & Shamir, A. (2004). New Cryptographic Primitives Based on Multiword T-Functions. In *Fast Software Encryption* (Vol. 3017 of LNCS, pp. 1-15). Berlin: Springer.

Kucuk, O. (2006, July 16). *Slide Resynchronization Attack on the Initialization of Grain 1.0.* Retrieved May 25, 2008, from The eSTREAM Project: http://www.ecrypt.eu.org/stream/papersdir/2006/044.ps

Künzli, S. J., & Meier, W. (2005). Distinguishing Attacks on T-Functions. In *Progress in Cryptology – Mycrypt 2005* (Vol. 3715 of LNCS, pp. 2-15). Berlin: Springer.

Lee, H., & Moon, S. (2000). On an improved summation generator with 2-bit memory. *Signal Processing , 80* (1), 211-217.

Lee, H., & Moon, S. (2002). Parallel stream cipher for secure high-speed communications. *Signal Processing* (82), 259-265.

Lu, Y., & Vaudenay, S. (2004). Cryptanalysis of Bluetooth Keystream Generator Two-Level E0. In *Advances in Cryptology - ASIACRYPT 2004* (Vol. 3329 of LNCS, pp. 483-499). Berlin: Springer.

Mantin, I. (2007). Predicting and Distinguishing Attacks on RC4 Keystream Generator. In *Advances in Cryptology* (Vol. 3494 of LNCS, pp. 491-506). Berlin: Springer.

Mattsson, J. (2006). *A Guess-and-Determine Attack on the Stream Cipher Polar Bear.* Retrieved May 10, 2008, from The eSTREAM Project: http://www.ecrypt.eu.org/stream/papersdir/2006/017.pdf

Maximov, A. (2006). Cryptanalysis of the "Grain" family of stream ciphers. *In Proc. of the 2006 ACM Symposium on Information, computer and communications security* (pp. 283 - 288). Taipei, Taiwan: ACM.

Meier, W., & Staffelbach, O. (1994). The Self-Shrinking Generator. In *Eurocrypt 94* (Vol. 950 of LNCS, pp. 205-214). Springer.

Menezes, A. O., & Vanstone, S. (1997). *Handbook of Applied Cryptography* . Boca Raton, FL: CRC Press.

Mister, S., & Adams, C. (1996). Practical S-Box Design. *Workshop on Selected Areas in Cryptography (SAC '96)* (pp. 61–76). Philadelphia, Pennsylvania: ACM.

Molland, H., & Helleseth, T. (2005). Alinear weakness in the Klimov-Shamir T-function. *In Proc. International Symposium on Information Theory, ISIT 2005.* (pp. 1106 - 1110). Adelaide, Australia: IEEE.

Mollin, R. A. (2007). *An Introduction to Cryptography* (2nd Edition ed.). (K. H. Rosen, Ed.) Boca Raton: Chapman & Hall/CRC.

Muller, F., & Peyrin, T. (2005). Linear Cryptanalysis of the TSC Family of Stream Ciphers. In *Advances in Cryptology - ASIACRYPT 2005* (Vol. 3788 of LNCS, pp. 373-394). Berlin: Springer.

Nada, J., & Naslund, M. (2005). *The Stream Cipher Polar Bear.* Retrieved May 10, 2008, from The eSTREAM Project: http://www.ecrypt.eu.org/stream/ciphers/polarbear/polarbear.pdf

National, S. A. (2002). *Announcing the Secure Hash Standard.* Federal Information Processing Standards Publications 180-2.

Park, M., & Park, D. (2005). A New Stream Cipher Using Two Nonlinear Functions. In *Computational Science and Its Applications – ICCSA 2005* (Vol. 3481 of LNCS, pp. 235-244). Berlin: Springer.

Pudovkina, M. (2001). *Analysis of chosen plaintext attacks on the WAKE Stream Cipher.* Retrieved May 29, 2008, from eprint: http://eprint.iacr.org/2001/065.pdf

Rivest, R. (1992). *The RC4 Encryption Algorithm.* RSA Data Security Inc.: Document No. 003-013005-100-000000.

Robshaw, M. (1995). *Stream Ciphers.* CA: RSA Laboratories.

Rogaway, P., & Coppersmith, D. (1994). A software-optimized encryption algorithm. In *Fast Software Encryption* (Vol. 809 of LNCS, pp. 56-63). Springer.

Stalling, W. (2003). *Cryptography and network security: principles and practice* (3rd ed.). New Jersey: Prentice Hall.

Tsunoo, Y. K., & Suzaki, T. (2007). A Distinguishing Attack on a Fast Software-Implemented RC4-Like Stream Cipher. *IEEE Trans. on Information Theory. 53*, pp. 3250-3255. IEEE Computer Society.

Tsunoo, Y. S., & Minematsu, K. (2005). Shorter Bit Sequence Is Enough to Break Stream Cipher LILI-128. *IEEE Trans. on Information Theory. 51 (12)*, pp. 4312-4319. IEEE Computer Society.

Wagner, D. S., & Schneier, B. (1998). Cryptanalysis of ORYX. In *Selected Areas in Cryptography* (Vol. 1556 of LNCS, pp. 296-305). Springer.

Watanabe, D. F., & Preneel, B. (2002). A New Keystream Generator MUGI. In *Fast Software Encryption* (Vol. 2365 of LNCS, pp. 179-194). Berlin: Springer.

Weisstein, E. W. (2008). *Gram-Schmidt Orthonormalization.* Retrieved July 20, 2008, from MathWorld--A Wolfram Web Resource: http://mathworld.wolfram.com/Gram-SchmidtOrthonormalization.html

Wheeler, D. (1993). A Bulk Data Encription Algorithm. In *Fast Software Encryption, Cambridge Security Workshop* (Vol. 809 of LNCS, pp. 127 - 134). London, UK: Springer.

Whiting, D. S., & Muller, F. (2005, April 29). *Phelix: Fast Encryption and Authentication in a Single Cryptographic Primitive.* Retrieved May 12, 2008, from The eSTREAM Project: http://www.ecrypt.eu.org/stream/ciphers/phelix/phelix.pdf

Wu, H., & Preneel, B. (2007). Differential-Linear Attacks Against the Stream Cipher Phelix. In *Fast Software Encryption* (Vol. 4593 of LNCS, pp. 87-100). Berlin: Springer.

Wu, H., & Preneel, P. (2006). Cryptanalysis of the Stream Cipher DECIM. In *Fast Software Encryption* (Vol. 4047 of LNCS, pp. 30-40). Berlin: Springer.

Zenner, E. (2004). *Cryptanalysis of LFSR-based Pseudorandom Generators - a Survey.* Reihe Informatik.

Zhang, H. L., & Wang, X. (2006). *Fast Correlation Attack on Stream Cipher ABC v3.* Retrieved May 18, 2008, from http://www.ecrypt.eu.org/stream/papersdir/2006/049.pdf