Linear Cryptanalysis of TSC Stream Ciphers
Applications to the ECRYPT Proposal TSC-3

Frédéric Muller and Thomas Peyrin

DCSSI Crypto Lab
51, boulevard de La Tour-Maubourg
75700 PARIS-07 SP
Frederic.Muller@sgdn.pm.gouv.fr, Thomas.Peyrin@gmail.com

Abstract. In this paper, we introduce a new cryptanalysis method for stream ciphers based on T-functions and apply it to the TSC family which was proposed by Hong *et al.*. Our attack are based on linear approximations of the algorithms (in particular of the T-function). Hence, it is related to correlation attack, a popular technique to break stream ciphers with a linear update, like those using LFSR's.
We show a key-recovery attack for the two algorithms proposed at FSE 2005 : TSC-1 in $2^{25.4}$ computation steps, and TSC-2 in $2^{48.1}$ steps. The first attack has been implemented and takes about 4 minutes to recover the whole key on an average PC. Another algorithm in the family, called TSC-3, was proposed at the ECRYPT call for stream ciphers. Despite some differences with its predecessors, it can be broken by similar techniques. Our attack has complexity of 2^{42} known keystream bits to distinguish it from random, and about 2^{66} steps of computation to recover the full secret key.

1 Introduction

1.1 Background

Together with block ciphers, stream ciphers are the second important family of symmetric encryption primitive. They work by generating a long pseudo-random sequence (generally called the keystream) from a short key. Then, a message is encrypted by a simple XOR with the keystream and the decryption works the same way. The keystream should not be distinguishable from a random sequence to make the cipher secure. Even if the cryptographic security is the main issue, the efficiency of the algorithm has also to be taken in account. Indeed, speed is the main advantage of stream ciphers over block ciphers.

Nowadays, designing a stream cipher is risky and the existence of good block ciphers has brought some issues about the future of stream ciphers [2,23]. However, some particular domains continue to be active. For example, fast software-oriented stream ciphers may still be needed, as well as hardware-oriented designs with a small footprint for resource constrained devices. A call for primitive has recently been launched by the european ECRYPT project and many new algorithms have been proposed for this occasion [6,7].

A classical approach for stream cipher design is the use of Linear Feedback Shift Registers (LFSR). Such primitives have to be combined with nonlinear Boolean functions to break the linearity. Due to the apparition of new attacks (like algebraic attacks [1,4]), new primitives have been introduced to replace LFSR's. A nice example are the Triangular-functions (T-functions) by Klimov and Shamir [13,14]. They are a new class of mappings, with the property to be computable from Least Significant Bits (LSB) to Most Significant Bits (MSB). This is well suited for implementations, because many operations available on processors (like $+$,*,XOR,OR,AND) are T-functions. T-functions are not (necessarily) linear and, for appropriate choices, they can be permutations with one single cycle, which is useful for stream ciphers design. Klimov and Shamir also extended their theory to multi-word T-functions and provided some results in other domains such as block ciphers and hash functions [12,15,16,17].

The first T-function Based Stream Ciphers (TFBSC) were proposed in the original papers by Klimov and Shamir. More recently, Hong *et al.* proposed a new class of single cycle T-functions, which have the property to use S-boxes [10]. They described two new algorithms. The first one, TSC-1, is designed for hardware environment and the second, TSC-2, can be implemented very efficiently in software. Several attacks have also been published. At Asiacrypt 2004, Mitra and Sarkar [22] described a time-memory trade-off attack which breaks some of the algorithms proposed by Klimov and Shamir. Künzli, Junod and Meier recently found distinguishing attacks applicable to many TFBSC's [19]. Taking into account these results, Hong *et al.* proposed a new algorithm, called TSC-3 at the ECRYPT competition for stream cipher [7]. This algorithm is an improvement over its two predecessors, in order to thwart the published attack [11]. However, the basic construction remains roughly the same.

1.2 Contribution of the paper

Our contribution in this paper is to present a **new cryptanalysis method for TFBSC's**. Our idea is to mount a statistical attack using **linear approximations** of the cipher. First, we linearize the behavior of the T-function by considering several consecutive steps. Next, we linearize other components, like the output function. Then we describe how to recover the secret key by combining all these linear approximations. This framework is closely related to correlation attacks against LFSR-based stream ciphers [21,24] and also to linear cryptanalysis against block ciphers [20].

It applies very efficiently to the TSC family. Indeed, we can break TSC-1 with time complexity of $2^{25.4}$ steps and data complexity of $2^{21.4}$ keystream words. Similarly, TSC-2 can be broken with $2^{44.1}$ data and $2^{48.1}$ time. We implemented the first attack against TSC-1. It needs about 4 minutes to recover the whole initial secret key (Pentium-III 700 MHz).

This cryptanalysis method also applies against the ECRYPT proposal TSC-3, although some adaptations are needed. In particular, the linear approximations we use are a little bit more complicated than in the case of TSC-1 and TSC-2. We describe how to distinguish the output of TSC-3 from random data by processing about 2^{42} keystream words. This observation can be extended to a key-recovery attack with time complexity of 2^{66} and data complexity of about 2^{34} keystream bits.

Table 1. Summary of attacks against the TSC family

Algorithm	Type of Attack	Time	Data
TSC-1	Distinguishing [18,19]	2^{22}	2^{22}
TSC-1	Distinguishing	2^{19}	2^{15}
TSC-1	Key-recovery attack	$2^{25.4}$	$2^{21.4}$
TSC-2	Distinguishing[19]	2^{34}	2^{34}
TSC-2	Key-recovery attack	$2^{48.1}$	$2^{44.1}$
TSC-3	Distinguishing	2^{42}	2^{42}
TSC-3	Key-recovery attack	2^{66}	2^{34}

These attacks are the first key recovery attacks against the TSC family (distinguishing attacks have already been pointed out in [18,19]). Table 1 summarizes all these results. We also point out some important requirements for the design of T-function based stream ciphers. In particular, the existence of good linear approximations of the T-function over several consecutive steps should be avoided.

To begin, we review the basic properties of T-functions. Secondly, we overview the existing TFBSC and the existing attacks. In Section 4, we give a general framework to attack TFBSC. Next, we describe how this framework applies to break TSC-1, TSC-2 and TSC-3.

2 Introduction to T-functions

We give a short review of T-functions results; readers can see [12] for further details.

2.1 Single-word T-functions

Basically, a single-word T-function is a mapping on a n-bit word where the bit i of the output can depend only on bits $0, 1, \cdots, i$ of the input. For example, most arithmetic operations, like addition, subtraction and multiplication are T-functions. It is also the case of most logical operations (OR,AND,XOR). These operations are called primitive operations. They are useful because they are available on most processors and can generally be executed in one clock cycle.

Moreover, the composition of two T-functions is a T-function, which allows to design a large number of such functions. Klimov and Shamir developed tools in order to study their invertibility and their cycle structure. In particular, some families provide a great feature: a single cycle of maximal length. However, single-word T-functions are not useful by themselves as n is usually limited on modern processors (to 32 or 64 bits). To increase the state size, it is better to use, for instance 4 words of 64 bits instead of one word of 256 bits.

2.2 Multi-word T-functions

The definition of T-functions can be extended to multi-word T-functions: the bit i of any output word depends only on bits 0 to i of each input word.

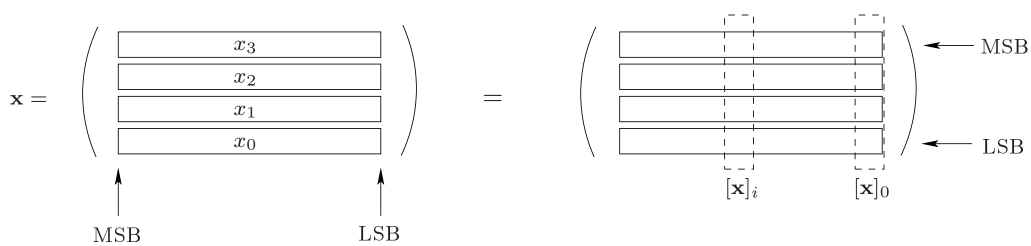
More formally, let $\mathbf{x}$ represent m words of n bits each denoted by x_i with $0 \leq i < m$. We get $\mathbf{x} = (x_j)_{j=0}^{m-1}$. Also, $[x_j]_i$ will refer to the i-th bit of a word x_j, seen as an integer:

$$[x_j] = \sum_{i=0}^{n-1} [x_j]_i 2^i.$$

Then $[\mathbf{x}]_i$ denotes the **layer** of i-th bits of the m words x_i composing $\mathbf{x}$. Thus we also get:

$$[\mathbf{x}]_i = \sum_{k=0}^{m-1} [x_k]_i 2^k.$$

Here is a clear depiction:



Definition 1. *A (multi-word) $T-function$ is a map*

$$\mathbf{T} : \begin{cases} (\{0,1\}^n)^m \longrightarrow (\{0,1\}^n)^m \\ \\ \quad\quad \mathbf{x} \quad\quad \longmapsto \mathbf{T}(\mathbf{x}) = (T_k(\mathbf{x}))_{k=0}^{m-1} \end{cases}$$

sending an m-tuple of n-bit words to another m-tuple of n-bit words, where each resulting n-bit word is denoted as $T_k(\mathbf{x})$, such that for each $0 \leq i < n$, the i-th bits of the resulting words $[\mathbf{T}(\mathbf{x})]_i$ are functions of just the lower input bits $[\mathbf{x}]_0, [\mathbf{x}]_1, \ldots, [\mathbf{x}]_i$.

We can also define a mapping from n-bit words to n-bit words in which the bit i of the output depends only on bits $0, 1, \ldots, i-1$ of the input. Such mappings are called **parameters** and are useful to construct interesting T-functions.

2.3 Properties of T-functions

We focus on **multi-word** T-functions, since they are the most interesting for stream cipher design. Basically, two properties can be expected :

- **invertibility :** This avoids a loss of entropy, if the T-function is used to update the state of a stream cipher.
- **single-cycle :** It is important for security that the sequence of internal states has a large period. A single cycle of maximal length 2^{nm} is even better, but is possible only if the T-function is invertible.

Klimov and Shamir proposed a method to construct T-functions which exhibits the single-cycle property. Their analysis is based on **odd** and **even parameters** (see [15] for more details).

Another approach was recently proposed by Hong *et al* [10] : Let $\mathbf{x} = (x_k)_{k=0}^{m-1}$ and $\mathbf{y} = (y_k)_{k=0}^{m-1}$ be two multi-words and let α be a single word. We note $\mathbf{x} \oplus \mathbf{y}$ and $\alpha \cdot \mathbf{x}$ defined as :

$$\mathbf{x} \oplus \mathbf{y} = (x_k \oplus y_k)_{k=0}^{m-1} \quad \text{and} \quad \alpha \cdot \mathbf{x} = (\alpha \wedge x_k)_{k=0}^{m-1}.$$

We also note $\sim \alpha$ the bitwise complement of α.

Theorem 1. *Let S be a single cycle S-box and let α be an odd parameter. If S^o is an odd power of S and S^e is an even power of S, the mapping*

$$T(\mathbf{x}) = (\alpha(\mathbf{x}) \cdot \mathbf{S^o}(\mathbf{x})) \oplus (\sim \alpha(\mathbf{x}) \cdot \mathbf{S^e}(\mathbf{x}))$$

defines a single cycle T-function.

3 Existing TFBSC's

3.1 Klimov and Shamir's ciphers

After introducing the concept of T-functions, Klimov and Shamir proposed several examples of TFBSC [15,16]. All are based on a similar construction.

Let C_0 be an odd number, $C_1 = 0\text{x}12481248$ and $C_3 = 0\text{x}48124812$. We set $a_0 = x_0$ and $a_{i+1} = a_i \wedge x_{i+1}$ for $i = 0, 1, 2$. We also have $\alpha = \alpha(\mathbf{x}) = (a_3 + C_0) \oplus a_3$. The following mapping is a single cycle T-function operating on 64-bit words:

$$\mathbf{T}\begin{pmatrix} x_3 \\ x_2 \\ x_1 \\ x_0 \end{pmatrix} \longmapsto \begin{pmatrix} x_3 \oplus (\alpha \wedge a_2) \oplus (2x_0(x_1 \vee C_1)) \\ x_2 \oplus (\alpha \wedge a_1) \oplus (2x_0(x_3 \vee C_3)) \\ x_1 \oplus (\alpha \wedge a_0) \oplus (2x_2(x_3 \vee C_3)) \\ x_0 \oplus \quad \alpha \quad \oplus (2x_2(x_1 \vee C_1)) \end{pmatrix} \tag{1}$$

Mitra and Sarkar described [22] a time-memory trade-off attack on a stream cipher based on (1) with a very simple output function. They analyzed the multiplicative part of the update function and managed to recover the initial secret key in 2^{40} time, 2^{24} space and less than five 128-bit blocks of keystream.

3.2 The TSC Family

Hong *et al.* provided two TFBSC's deduced from their new single-cycle T-functions family given in Theorem 1. For all algorithms, the number of words is $m = 4$. While Klimov-Shamir's proposal are software-oriented designs (with the use of integer multiplication), the TSC family is S-box oriented. In particular, the authors have suggested an implementation method for TSC-1 and TSC-3 which could make them suitable as hardware-oriented designs.

TSC-1

TSC-1 uses 4 words of $n = 32$ bits each, hence the internal state has size 128 bits. First a single-cycle S-box S_1 operating on 4 bits is defined :

$$S_1[16] = \{3, 5, 9, 13, 1, 6, 11, 15, 4, 0, 8, 14, 10, 7, 2, 12\};$$

The following function is an odd parameter :

$$\alpha(\mathbf{x}) = (p + C) \oplus p \oplus 2s,$$

where $C = 0x12488421$, $p = x_0 \wedge x_1 \wedge x_2 \wedge x_3$ and $s = x_0 + x_1 + x_2 + x_3$. According to Theorem 1 with $S^o = S_1$ and $S^e = S_1^2$, the following T-function is single-cycle :

$$T(\mathbf{x}) = (\alpha(\mathbf{x}) \cdot S_1(\mathbf{x})) \oplus (\sim \alpha(\mathbf{x}) \cdot S_1^2(\mathbf{x})). \tag{2}$$

Finally, 32 output bits are produced after application of $\mathbf{T}$ by:

$$f(\mathbf{x}) = (x_{0 \lll 9} + x_1)_{\lll 15} + (x_{2 \lll 7} + x_3), \tag{3}$$

where the symbol $\lll$ denotes left rotation. Every addition is done modulo 2^{32}. It is proven that the period of this T-function is 2^{128}.

TSC-2

TSC-2 is quite similar to TSC-1. It uses a different S-box :

$$S_2[16] = \{5, 2, 11, 12, 13, 4, 3, 14, 15, 8, 1, 6, 7, 10, 9, 0\};$$

and the following odd parameter:

$$\alpha_2(\mathbf{x}) = (p + 1) \oplus p \oplus 2s.$$

According to Theorem 1 with $S^o = Id$ and $S^e = S_2$:

$$\mathbf{x} \longmapsto \mathbf{x} \oplus (\alpha_2(\mathbf{x}) \cdot (\mathbf{x} \oplus S_2(\mathbf{x}))).$$

is single-cycle. Finally 32 keystream bits are obtained by:

$$f_2(\mathbf{x}) = (x_{0 \lll 11} + x_1)_{\lll 14} + (x_{0 \lll 13} + x_2)_{\lll 22} + (x_{0 \lll 12} + x_3).$$

TSC-3

At the ECRYPT competition for stream ciphers [11], Hong *et al.* proposed the stream cipher TSC-3. It differs from its two predecessors regarding several elements :

- First, it uses 4 words of size 40 bit each. This breaks the 32-bit oriented architecture, but it does not matter since the cipher is primarily designed for hardware implementations. In addition, this increases the state size to 160 bits. Therefore the expected level of security is 2^{80}, which can be reached by generic attacks, such as time-memory-data trade-offs [3].
- Secondly, each layer is still updated by S-boxes, but the branching function is more complex than for TSC-1 or TSC-2. Indeed, the parameter is made of 2 words p_0 and p_1. For the i-th layer, one first computes the value

  $$tmp = 2 * [p_1]_i + [p_0]_i \in \{0, 3\}$$

  According to the value of tmp, $[x]_i$ is update using either S, S^2, S^5 or S^6 where S is the same S-box as in TSC-1.
- The output function is also modified in TSC-3. One first starts by initializing 4 variables y_i of 32 bits each, by removing the 8 LSB's from each x_i. Then, the y_i's are permuted depending on the value of the least significant layer of the sate, $[x]_0$. Therefore there are $2^4 = 16$ possible permutations. Afterward, the output function looks very much like the ones used in TSC-1 and TSC-2 :

  $$f(\mathbf{y}) = (y_{0\lll 9} + y_{1\ggg 2})_{\lll 8} + (y_{2\lll 7} + y_3)_{\ggg 9}$$

- An initialization mechanism has also been added in order to set up the state from a key and an IV of variable length. This mechanism is based on the T-function itself, but is not described here.

For more information about these elements of TSC-3, the reader should refer directly to the specifications [10] or to the ECRYPT website [6].

4 Linear Cryptanalysis against TFBSC's

4.1 Context

Attacks based on linear approximations have many applications in cryptanalysis. For instance, Matsui's attack is the best cryptanalysis of DES [20] and more generally linear cryptanalysis has many applications for block ciphers. In the field of stream ciphers, popular attacks based on linear approximations have been developed for LFSR oriented designs and are generally referred to as **correlation attacks** [21,24]. Also linear cryptanalysis for stream ciphers has already been suggested [5,8] and has already been applied, for instance by Golic against RC4 [9].

In the case of TFBSC, the idea of using linear approximations was first introduced by Künzli, Junod and Meier. At the rump session of FSE 2005, they presented a distinguishing attack against the TSC-1 requiring about 2^{22} known keystream bits [18]. This idea is further developed in [19].

4.2 A Framework for Linear Cryptanalysis of TFBSC's

The attack we propose is composed by three steps :

1. find a **linear approximation of the T-function**. This provides a probabilistic relation between bits from the internal state of the stream cipher at different instants.
2. find a **linear approximation of the output function**. This provides a probabilistic relation between keystream bits and internal state bits.
3. **combine both approximations**. One goal may be to find relations involving keystream bits only, in order to obtain a distinguisher. But a more interesting idea is to guess some key bits in order to eliminate some terms in the approximations and therefore to **increase the bias**.

The general idea of this framework is to remove the non-linearity provided by the T-function. While steps 1 and 2 are almost always possible, it can be hard to combine the approximations in step 3.

More formally, let $[x_j]_i^t$ represent the value of the bit i from register j at time t. In the first step, we look for equations of the form :

$$\Pr\left(\bigoplus_{i,j}[x_j]_i^t = \bigoplus_{i,j}[x_j]_i^{t+\delta}\right) = \frac{1}{2}(1+\epsilon)$$

for some δ and with $|\epsilon|$ as big as possible. For the purpose of the attacks against TSC-1 and TSC-2, it turns out that we are only interested in the particular linear relations of the form :

$$[x_j]_i^t = [x_j]_i^{t+\delta}$$

This corresponds to the probability for a given bit in the internal state to flip between time t and time $t+\delta$, also called the **bit-flip probability**. While the design criteria of the TSC family [10,11] and the first known attacks [18,19] focused on these bit-flip properties, there is no reason to restrict the analysis to such particular cases. The cryptanalysis of TSC-3 (see Section 7) is a good example of attack where other types of linear approximations are needed.

The second step depends on how complex is the output function, but it is generally possible to find linear approximations for the algorithms of the TSC family. For instance, suppose we find a probabilistic linear relation between several state bits $[x_j]_i^t$ and several keystream bits $[s]_k^t$, at time t. We combine this relation with the first step, to obtain a linearized relation of the form :

$$\bigoplus_{i,j}([x_j]_i^t \oplus [x_j]_i^{t+\delta}) = \bigoplus_{k}([s]_k^t \oplus [s]_k^{t+\delta}) \tag{4}$$

which is equal to 0 with probability 0.5 $(1+\epsilon)$ and hopefully $|\epsilon| \gg 0$.

In the third step, we try to propose distinguishing attacks and key recovery attacks based on relation (4). A useful trick for T-functions, is that when we guess the i LSB's of each register in the initial state, we can predict these i LSB's at every instant because of the triangular structure.

5 The TSC-1 Case

In this section, we apply our framework to the TSC-1 case and show an efficient key-recovery attack. We explain the attack by following the three steps of our framework.

5.1 First Step

We want to approximate the behavior of the state-update function between time t and time $t+\delta$. By looking at the update function (2), we observe that the i-th layer's update depends on one parameter bit only, $[\alpha(\mathbf{x})]_i$. Depending on this bit, the 4 bits of the layer are updated using either S_1 or S_1^2 :

$$[\mathbf{T}(\mathbf{x})]_i = \begin{cases} S_1^2([\mathbf{x}]_i) & \text{if } [\alpha(\mathbf{x})]_i = 0 \\ \\ S_1([\mathbf{x}]_i) & \text{if } [\alpha(\mathbf{x})]_i = 1 \end{cases}$$

We assume that the parameter is uniformly distributed. Then

$$\Pr\left([\alpha(\mathbf{x})]_i = 0\right) = \Pr\left([\alpha(\mathbf{x})]_i = 1\right) = \frac{1}{2}$$

for $i \neq 0$. This property has been verified experimentally. We construct a binary tree describing the update of the i-th layer (see Figure 1). We start from an

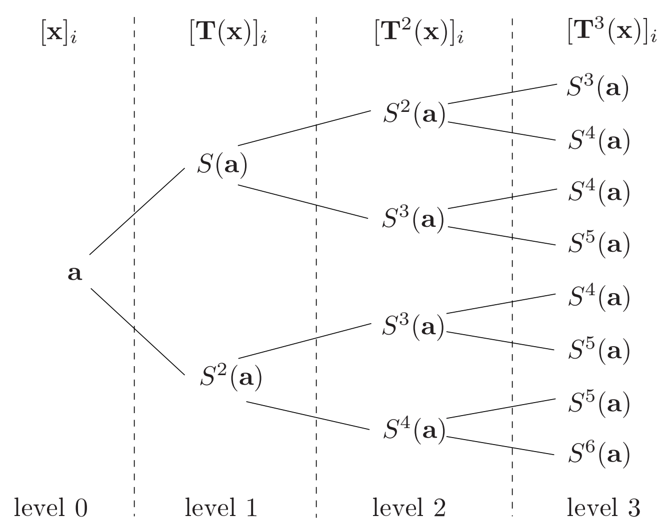


Fig. 1. Possible Evolutions of the i-th layer for TSC-1

unknown 4-bit value a and each branch corresponds to a value of $[\alpha(\mathbf{x})]_i$. After

j advances, there are 2^j leaves in the tree, each corresponding to a power of S_1. Let K_i^j be the number of occurrence of S_1^i at the level j of the tree. The coefficients K_i^j can be computed by the formula:

$$K_i^j = \binom{j}{i-j} \text{ with } i \geq j.$$

Using these coefficients, we can compute the probabilities of each output value after j advances, for each value of a. Then, we search for linear approximations between bits of the i-th layer at time t and at time $t+j$. In the case of TSC-1, we restrict our analysis to particular linear approximations where the same bit is considered twice (known as bit-flip probabilities). The authors of TSC-1 took them into account for the design, so the S-box has probability 1/2 to flip each input bit. The same holds for all powers S_1^i of the S-box, except for $i = 4, 8, 12$ and 16. So nothing will be observed at the level 1 in the tree, but at further levels, the "weak" powers may appear with high coefficients. We explored the

Table 2. TSC-1 : Bit Flip Probabilities for Different Depth j of the Tree

j	P	$\lvert\epsilon\rvert$	j	P	$\lvert\epsilon\rvert$
1	0.5000	0.0000	9	0.5264	0.0528
2	0.5937	0.1874	10	0.4143	0.1714
3	0.6406	0.2812	11	0.3993	0.2014
4	0.5078	0.0156	12	0.4849	0.0302
5	0.4219	0.1562	13	0.5587	0.1174
6	0.4473	0.1054	14	0.5507	0.1014
7	0.5479	0.0958	15	0.4972	0.0056
8	0.5996	0.1992	16	0.4717	0.0566

tree at depth j and computed the bit-flip probabilities for several values of j. The results are given in Table 2 (due to some symmetry properties, the probability is the same for the 4 input bits). We observe that the strongest bias are obtained with $j = 3, 5, 8, 11$. An example of good linear approximation is :

$$\Pr\left([x_i]_1^t \oplus [x_i]_1^{t+3} = 1\right) \simeq 0.64 = \frac{1}{2}(1 + 0.28).$$

for all $i = 0, \ldots, 3$.

In Table 7 of the Appendix, we give experimental results. They show that the observed bias match the theoretical analysis. Therefore the **initial assumption that the parameter bits are uniformly distributed is satisfied**. The only exception concerns the LSB of the registers. Indeed, the parameter bit is constant at position 0, so the previous assumption no longer holds. This analysis explains what Künzli *et al.* observed [18] with $j = 8$ and 11, although the best bias is obtained with $j = 3$.

5.2 Second Step

In this step, we want to "linearize" the behavior of the output function of TSC-1 defined by (3). This function uses addition and left rotation on 32-bit words. Left rotation is already linear, so we only have to linearize the additions. This can be naturally done by introducing a **carry bit**. For instance, when adding two integers a_0 and a_1, we can express the i-th bit of the result by the linear expression :

$$[a_0]_i \oplus [a_1]_i \oplus R_i$$

where R_i depends on layers $< i$.

Consider the addition of n integers of 32 bits called $a_0, \ldots, a_{n-1}$. We note $A = \sum_{k=0}^{n-1} a_k$ and $R(i)$ the i-th carry. For $n = 2$ terms, the carry is simply one bit, but more generally, it is an integer formally defined by :

$$R(i) = \frac{\sum_{k=0}^{n-1}(a_k \bmod 2^i) - (\sum_{k=0}^{n-1}(a_k \bmod 2^i)) \bmod 2^i}{2^i}.$$

with $R(0) = 0$. The linearized expression of the i-th bit of A is given by :

$$[A]_i = \left[\sum_{k=0}^{n-1} a_k\right]_i = [R(i)]_0 \oplus \bigoplus_{k=0}^{n-1} [a_k]_i. \tag{5}$$

In the case of TSC-1, the output function is composed by an addition with 2 terms ($E = x_{0 \lll 9} + x_1$) and an addition with 3 terms ($S = E_{\lll 15} + x_{2 \lll 7} + x_3$) where S represents the output. Hence, using linearized relations (5), for any bit i we have:

$$\begin{cases} [E]_i = [x_0]_{(i+23)} \oplus [x_1]_{(i)} \oplus [R_E(i)]_0 \\ \\ [S]_i = E_{(i+17)} \oplus [x_2]_{(i+25)} \oplus [x_3]_{(i)} \oplus [R_S(i)]_0 \end{cases}$$

where R_E and R_S represent the carry for the 2-term and 3-term addition respectively. All indexes are taken modulo 32. We can note that $R_E(i) \in \{0, 1\}$ and $R_S(i) \in \{0, 1, 2\}$. Finally, we obtain :

$$[S]_i = [x_0]_{(i+8)} \oplus [x_1]_{(i+17)} \oplus [R_E(i+17)]_0 \oplus [x_2]_{(i+25)} \oplus [x_3]_{(i)} \oplus [R_S(i)]_0.$$

which is a **linear approximation of the output function**.

We would like to XOR this relation at two instants t and $t + \delta$ for instance with $\delta = 3$, since this is the value identified in the first step. We already now the bit-flip probabilities of the register bits. Now the problem is to determine the bit-flip probabilities of the carry bits between t and $t + 3$.

5.3 Bit Flip Property of Carries

Basically, each input bit in the additions E and S is flipped with a known probability, different from 0.5. As a consequence, we may expect that the carries also flip with probabilities different from 0.5. The goal of this Section is to evaluate this probability.

We define the "general carry" as $[R_G(i)] = [R_E(i+17)] \oplus [R_S(i)]$. We also call $X_{R_G}(i) = [R_G(i)]_0^t \oplus [R_G(i)]_0^{t+3}$ and $X_j(i) = [x_j]_i^t \oplus [x_j]_i^{t+3}$. From the previous section, we get :

$$[S]_i^t \oplus [S]_i^{t+3} = X_{R_G}(i) \oplus X_0(i+8) \oplus X_1(i+17) \oplus X_2(i+25) \oplus X_3(i)$$

From the first step, we know that $\Pr(X_j(i) = 1) = \frac{1}{2}(1 + \epsilon_i^j)$. The biases ϵ_i^j are given in Table 7 in the Appendix. So the only remaining term is $X_{R_G}(i)$. Experimentally, we observed that

$$\Pr(X_{R_G}(i) = 1) = \frac{1}{2}(1 + \epsilon_i^G) \text{ with } |\epsilon_i^G| \gg 0$$

and that ϵ_i^G apparently depends on the position i considered. Unfortunately, we also observed that the two "internal" carries $R_S(i)$ and $R_E(i)$ are not independent, so it is not possible to handle them separately.

To explain this bias, we model the phenomenon as a **Markov chain**. Indeed, carries at layer $i+1$ are computed only from the carries at layer i and from the terms in the addition, so we do not need to remember what happened previously. We implemented a recursive algorithm to evaluate the following probability, starting from the least significant bit $i = 0$:

$$\begin{aligned} \Pr_i(a,b,c,d) = \Pr\ (\ & (R_S(i)^t = a) \wedge (R_S(i)^{t+3} = b) \\ & \wedge (R_E(i+17)^t = c) \wedge (R_E(i+17)^{t+3} = d)) \end{aligned}$$

for all possible $a, b, c, d \in \{0,1,2\}^2 \times \{0,1\}^2$. To compute $\Pr_{i+1}(a,b,c,d)$, we examine all cases at layer i : we try all values of the terms in the addition, we try all values of the carries at layer i, and we compute the new carries. Each event at layer i is associated with its corresponding probability, and we increment accordingly the probabilities of layer $i+1$. After examining all cases, we know $\Pr_{i+1}(a,b,c,d)$. Then, we increment i and jump to the next layer [1].

In the end, we obtain the bit-flip probability of the general carry by :

$$\Pr(X_{R_G}(i) = 1) = \sum_{a,b,c,d | \mathrm{LSB}(a) \oplus \mathrm{LSB}(b) \oplus c \oplus d = 1} \Pr_i(a,b,c,d).$$

The experiments on TSC-1 returned the same probabilities as our computation by a Markov chain. These results are listed in the rightmost column of Table 7 (see the Appendix). We now have biased linear approximations which involve only TSC-1's output bits and internal state bits, so we can continue to the third step.

[1] There is a slight technicality, since the layer 0 actually depends from the layer 31 due to the left rotation, so we do not know how to initialize the recursion. Actually, probabilities are quite independent from the initial value, so we can handle this difficulty.

5.4 Third Step

Distinguishing attacks

It is easy to use a bias on the output of a stream cipher for a distinguishing attack : one just produces enough keystream bits and checks if the bias is satisfied or not. For a bias ε, it it well known that about ε^{-2} samples are needed. As an example, Künzli *et al.* pointed out a distinguisher requiring 2^{22} output bits for TSC-1 [18]. Similarly, our previous analysis provides a distinguishing attack. For example, consider the layer $i = 1$ of the output. We have

$$[S]_1^t \oplus [S]_1^{t+3} = X_{R_G}(1) \oplus X_0(9) \oplus X_1(18) \oplus X_2(26) \oplus X_3(1)$$

Assuming the terms are independent, the bias are just multiplied, so

$$\Pr\left([[S]_1^t \oplus [S]_1^{t+3} = 1\right) = 0.5\,(1 + \varepsilon_D)$$

with :

$$\begin{aligned}\epsilon_D &= \epsilon(X_0(9)) \times \epsilon(X_1(18)) \times \epsilon(X_2(26)) \times \epsilon(X_3(1)) \times \epsilon(X_{R_G}(1)) \\ &= 0.2834 * 0.2824 * 0.2732 * 0.2812 * (-0.0874) \\ &= -2^{-10.86}\end{aligned}$$

using Table 7 in the Appendix. This gives a data complexity of $\epsilon_D^{-2} \simeq 2^{21.7}$ keystream bits, which is slightly better than [19].

Key recovery attacks

As pointed out in Section 4.2, if we guess the i LSB's of each register in the initial state, we can predict these bits at any moment. This idea can be used to **eliminate many terms in the linear approximations**.

First, let us guess the LSB of each register. There are $2^4 = 16$ possibilities. For any instant t, we can predict these LSB and thus eliminate all terms of the form $X_i(0)$ in the linear approximations. For instance, we can predict $[S]_0^t \oplus [S]_0^{t+3} \oplus X_3(0)$ which is biased with

$$\epsilon = -0.2826 * 0.2818 * 0.2826 * 0.1906 = -2^{-7.86}$$

according to Table 7 of the Appendix. This bias will be observed only for the correct guess. So, with a sufficient amount of data, we can find which of the 16 guesses is correct. The process can be repeated to successively guess all layers of the initial state, starting from the least significant ones.

The complexity of guessing each layer depends on the best bias that can be found. For the first step of the attack, the bias is $\epsilon = -2^{-7.86}$ so we need

$$M = \epsilon^{-2} = 2^{15.72}$$

keystream bits to find the correct guess. The time complexity is about

$$T = 2^{15.72} \times 2^4 = 2^{19.72}$$

steps. If we stop the attack after this step, we obtain a distinguishing attack which is slightly better than [19]. At each step, we can choose between several linear approximations (one for each of the 32 keystream bits). We always pick the position which gives the best results (see Table 3 for more details). Note that after

Table 3. Possible Attack Schedule for TSC-1

round	bit position attacked	register attacked	number of known terms	bias obtained	time complexity
0	0	3	1	$2^{-7.86}$	$2^{19.72}$
1	1	3	1	$2^{-9.03}$	$2^{22.06}$
2	9	2	1	$2^{-9.41}$	$2^{22.82}$
3	10	2	1	$2^{-9.29}$	$2^{22.58}$
4	11	2	1	$2^{-9.33}$	$2^{22.66}$
5	12	2	1	$2^{-9.39}$	$2^{22.78}$
6	13	2	1	$2^{-9.31}$	$2^{22.62}$
7	7	3	2	$2^{-7.48}$	$2^{18.96}$
8	0	0	2	$2^{-6.04}$	$2^{16.08}$
9	1	0	2	$2^{-7.21}$	$2^{18.42}$
10	10	3	2	$2^{-7.47}$	$2^{18.94}$
11	11	3	2	$2^{-7.46}$	$2^{18.92}$
12	19	2	2	$2^{-7.49}$	$2^{18.98}$
13	20	2	2	$2^{-7.51}$	$2^{19.02}$
14	21	2	2	$2^{-7.50}$	$2^{19.00}$
15	15	3	3	$2^{-4.81}$	$2^{13.62}$
16	8	0	3	$2^{-6.07}$	$2^{16.14}$
17	9	0	3	$2^{-5.76}$	$2^{15.52}$
18	10	0	3	$2^{-5.64}$	$2^{15.28}$
19	11	0	3	$2^{-5.63}$	$2^{15.26}$
20	12	0	3	$2^{-5.69}$	$2^{15.38}$
21	13	0	3	$2^{-5.66}$	$2^{15.32}$
22	14	0	3	$2^{-5.64}$	$2^{15.28}$
23	15	0	4	$2^{-2.94}$	$2^{9.88}$
24	24	3	4	$2^{-3.84}$	$2^{11.68}$
25	0	2	4	$2^{-2.39}$	$2^{8.78}$
26	1	2	4	$2^{-3.52}$	$2^{11.04}$
27	10	1	4	$2^{-3.82}$	$2^{11.64}$
28	11	1	4	$2^{-3.81}$	$2^{11.62}$
29	12	1	4	$2^{-3.82}$	$2^{11.64}$
30	13	1	4	$2^{-3.83}$	$2^{11.66}$
31	14	1	4	$2^{-3.82}$	$2^{11.64}$

guessing the layer 7, we can eliminate two terms in the linear approximations, so the complexity drops. Similarly, the complexity drops after the layer 15 (3 terms are eliminated) and after the layer 23 (4 terms are eliminated). The full cost of the attack is dominated by the first layers (layer number 2 in particular). The total complexity is of $2^{21.4}$ data and $2^{25.4}$ time.

6 The TSC-2 Case

As in the previous Section, we explain the attack by following the three steps of our framework. The idea is essentially the same as for TSC-1.

6.1 First Step

In TSC-2, a different S-box and a different T-function are used. The new mapping of the update function can be better understood written as:

$$[\mathbf{T}(\mathbf{x})]_i = \begin{cases} [\mathbf{x}]_i & \text{if } [\alpha(\mathbf{x})]_i = 0 \\ \\ S_2([\mathbf{x}]_i) & \text{if } [\alpha(\mathbf{x})]_i = 1 \end{cases}$$

Like before, we could assume that the parameter is uniformly distributed and apply the "tree" approach used against TSC-1. The tree is different but the idea would essentially be the same. However, the designers added a special property to S_2 which apparently prevents such an analysis : the application of S_2 always flips the LSB of the input. Therefore, register 0 has a bit-flip probability of exactly $\frac{1}{2}$ at any depth in the tree. The idea here is to guarantee at least one "unbiased register" and to use it several times in the output function to "mask" the other "biased registers". Unfortunately, there is still a second order inbalance, because **the parameter itself is not perfectly balanced**. Indeed, we have :

$$\alpha(\mathrm{x}) = (p+1) \oplus p \oplus s$$

where $s = x_0+x_1+x_2+x_3$. The term $(p+1) \oplus p$ is generally equal to 0 (except for the least significant positions), so it does not really matter. Besides, we can hope that s has a bit-flip probability slightly different from $1/2$ due to the interaction with the T-function update. Indeed, our experiments showed that :

$$\Pr\left([\alpha(\mathbf{x})]_i^t \oplus [\alpha(\mathbf{x})]_i^{t+1} = 1\right)$$

slightly deviates from $\frac{1}{2}$.

Therefore the branches in the tree are not perfectly balanced, like we assumed before. This second-order correction is negligible in the case of TSC-1, but is important for TSC-2. We modified accordingly the probability of each leaf. After these corrections, biased probabilities appear like in the case of TSC-1, by considering $\delta = 2$. However, the biases are not the same for all registers. In particular, they are smaller for x_0 than for x_1, x_2 and x_3 (see Table 8 in the Appendix).

6.2 Second Step

In this part of the attack, we use exactly the same technique as for TSC-1 since the output functions of the two ciphers look alike. In TSC-2, the output function uses two 2-term additions ($E_1 = x_{0\lll 11} + x_1$, $E_2 = x_{0\lll 13} + x_2$) and then a 4-term addition ($S = E_{1\lll 14} + E_{2\lll 22} + x_{0\lll 12} + x_3$). We get:

$$\begin{aligned}[S]_i = [x_0]_{(i+7)} \oplus [x_1]_{(i+18)} \oplus [R_{E_1}(i+18)]_0 \oplus [x_0]_{(i+29)} \oplus [x_2]_{(i+10)} \\ \oplus [R_{E_2}(i+10)]_0 \oplus [x_0]_{(i+20)} \oplus [x_3]_{(i)} \oplus [R_S(i)]_0\end{aligned}$$

We redefine the general carry $[R_G(i)] = [R_{E_1}(i+18)] \oplus [R_{E_2}(i+10)] \oplus [R_S(i)]$ and $X_{R_G}(i) = [R_G(i)]_0^t \oplus [R_G(i)]_0^{t+2}$. By setting $X_j(i) = [x_j]_i^t \oplus [x_j]_i^{t+2}$ and combining the first and the second step, we get:

$$\begin{aligned}[S]_i^t \oplus [S]_i^{t+2} = X_0(i+7) \oplus X_0(i+29) \oplus X_0(i+20) \\ \oplus X_1(i+18) \oplus X_2(i+10) \oplus X_3(i) \oplus X_{R_G}(i)\end{aligned}$$

We use the method of Section 5.3 to predict the bit-flip probabilities of the carries, with a recursive algorithm. The results are given in Table 8 of the Appendix.

6.3 Third Step

In this step we can execute a distinguishing or a key recovery attack. This step is exactly the same as for TSC-1: using the biased linear approximations we attack one by one the 32 layers of 4 bits. The difficulty is to find which biased linear approximation is the best for each round. We give in Table 9 of the Appendix a possible choice for each layer and the corresponding complexities. We finally get an attack recovering the initial key with $2^{44.1}$ data and $2^{48.1}$ time.

7 The TSC-3 Case

Since TSC-3 has some particular features compared to the two previous algorithms, the application of the attack is not exactly the same. However, it roughly follows the same framework.

7.1 First Step

The updating of any layer $[x]_i$ of the state can still be represented in a **tree-oriented fashion**, although it is no longer a binary tree (each node has 4 branches). Let us first suppose that the parameter words are uniformly distributed. Then, after applying the T-function, $[T(x)]_i$ has probability 1/4 to be equal to any of the $S^j([x]_i)$, for $j = 1, 2, 5, 6$. Similarly, after t updates, one can easily compute the probability for $[T^t(x)]_i$ to be equal to each power of the S-box[2]. This is summarized in Table 4. Then we can apply essentially the same analysis than for TSC-1 and TSC-2. However, here we are not interested only in bit-flip properties. Linear relations involving one input bit and another output bit may be of interest, because the registers are permuted in the output function,

[2] Remember that $S^{16} = I$

Table 4. Exploration of the tree for TSC-3: Probability that $[T^t[x]]_i = S^j([x]_i)$

j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$t=0$	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$t=1$	0	1/4	1/4	0	0	1/4	1/4	0	0	0	0	0	0	0	0	0
$t=2$	0	0	1/16	1/8	1/16	0	1/8	1/4	1/8	0	1/16	1/8	1/16	0	0	0
$t=3$	0.047	0.047	0.016	0.016	0.047	0.047	0.016	0.047	0.141	0.141	0.047	0.047	0.141	0.141	0.047	0.016
$t=4$	0.039	0.063	0.094	0.063	0.023	0.031	0.047	0.031	0.023	0.063	0.094	0.063	0.039	0.094	0.141	0.094

so we may compare bits belonging to different registers in the next steps of the attack. So we focus on the linear relations of the form :

$$[x_j]_i^t = [x_{j'}]_i^{t+\delta} \tag{6}$$

for two different register indexes $j, j' \in \{0, 3\}$ and for some depth δ. While the S-box of TSC-3 (the same as the one used in TSC-1) has good bit-flip properties, such advanced linear approximations have not been taken into account by the designers.

Table 5. Probability that $[x_j]_i^t = [x_{j'}]_i^{t+\delta}$

Case $\delta = 1$

Input \ Output	$[x_0]_i^t$	$[x_1]_i^t$	$[x_2]_i^t$	$[x_3]_i^t$
$[x_0]_i^{t+1}$	0.5	0.53125	0.46875	0.5
$[x_1]_i^{t+1}$	0.46875	0.5	0.5	0.46875
$[x_2]_i^{t+1}$	0.5	0.53125	0.5	0.53125
$[x_3]_i^{t+1}$	0.53125	0.5	0.46875	0.5

Case $\delta = 2$

Input \ Output	$[x_0]_i^t$	$[x_1]_i^t$	$[x_2]_i^t$	$[x_3]_i^t$
$[x_0]_i^{t+2}$	0.515625	0.515625	0.5	0.5
$[x_1]_i^{t+2}$	0.5	0.515625	0.46875	0.5
$[x_2]_i^{t+2}$	0.5	0.5	0.515625	0.515625
$[x_3]_i^{t+2}$	0.5	0.5	0.5	0.515625

Case $\delta = 3$

Input \ Output	$[x_0]_i^t$	$[x_1]_i^t$	$[x_2]_i^t$	$[x_3]_i^t$
$[x_0]_i^{t+3}$	0.51172	0.49610	0.51172	0.50391
$[x_1]_i^{t+3}$	0.50391	0.51172	0.49610	0.51172
$[x_2]_i^{t+3}$	0.49610	0.48828	0.51172	0.49610
$[x_3]_i^{t+3}$	0.48828	0.49610	0.50391	0.51172

From Table 4, it is easy to derive the probability that relation (6) holds, for any pair of positions (j, j'). These results do not depend on which layer i we consider although some side effects are observed at the least significant positions[3]. The results for certain values of δ are given in Table 5. They have been verified experimentally, by running the cipher on a random initial state.

7.2 Second Step

In the case of TSC-3, the output function is not directly applied to the state registers, but to 4 registers y_0, y_1, y_2 and y_3 which are truncated and permuted copies of the state registers x_0, x_1, x_2 and x_3. First we linearize the output function as we did for TSC-1 and TSC-2 :

$$[S]_i = [y_0]_{(i+17)} \oplus [y_1]_{(i+26)} \oplus [y_2]_{(i+2)} \oplus [y_3]_{(i+9)} \oplus [R_G(i)]_0$$

where $R_G(i)$ is the "general carry", defined as before. Then, we replace the bits from the registers y_i by the appropriate bits from the state registers x_i. Because of the truncation and the permutation, we have $[y_j]_i = [x_{\pi(j)}]_{i+8}$ where π is a 4-bit permutation determined by the layer $[x]_0$ of the internal state. The linear approximations depend on this permutation. Suppose that we are in the particular case where :

$$[x]_0^t = 4$$

Then the next value of this layer is :

$$[x]_0^{t+1} = 1$$

Looking at the permutations π associated with these particular values, we get

$$[S]_i^t = [x_0]_{(i+25)}^t \oplus [x_1]_{(i+34)}^t \oplus [x_3]_{(i+10)}^t \oplus [x_2]_{(i+17)}^t \oplus [R_G(i)]_0^t$$

and

$$[S]_i^{t+1} = [x_1]_{(i+25)}^{t+1} \oplus [x_0]_{(i+34)}^{t+1} \oplus [x_2]_{(i+10)}^{t+1} \oplus [x_3]_{(i+17)}^{t+1} \oplus [R_G(i)]_0^{t+1}$$

Using the Table 5, we observe that :

$$\begin{aligned}
\Pr([x_0]_{(i+25)}^t = [x_1]_{(i+25)}^{t+1}) &= 0.46875 \\
\Pr([x_1]_{(i+34)}^t = [x_0]_{(i+34)}^{t+1}) &= 0.53125 \\
\Pr([x_3]_{(i+10)}^t = [x_2]_{(i+10)}^{t+1}) &= 0.53125 \\
\Pr([x_2]_{(i+17)}^t = [x_3]_{(i+17)}^{t+1}) &= 0.46875
\end{aligned}$$

These 4 probabilities are of the form $0.5\ (1 \pm 2^{-4})$. We tried to consider other values of $[x]_0$ than 4, but it seems to be the best choice, since the highest probabilities in Table 5 appear. Combining the two relations at instants t and $t+1$, we get :

$$[S]_i^t \oplus [S]_i^{t+1} = [R_G(i)]_0^t \oplus [R_G(i)]_0^{t+1}$$

[3] Contrarily to TSC-1 or TSC-2, these side effects are not bothering for TSC-3, since layers 0 to 7 are discarded by the output function

with probability $0.5\,(1+\varepsilon)$ and $|\varepsilon| = (2^{-4})^4 = 2^{-16}$.

Like for TSC-1 and TSC-2, the carries from the additions involved in the output function are not independent from each other. So it is not easy to express simply the probability that $[R_G]$ changes between t and $t+1$. Like before, modeling this phenomenon by a Markov chain could provide more precise results, but we choose to measure the probability experimentally for the sake of simplicity. Results for several values of i are given in Table 6. For some well-

Table 6. TSC-3 : Bias measured experimentally on the Carry

Position	$\Pr([R_G]_i^t \oplus [R_G]_i^{t+1})$	Position	$\Pr([R_G]_i^t \oplus [R_G]_i^{t+1})$
$i=0$	0.5001	$i=16$	0.4993
$i=1$	0.4921	$i=17$	0.4998
$i=2$	0.4968	$i=18$	0.4997
$i=3$	0.4989	$i=19$	0.4998
$i=4$	0.5001	$i=20$	0.5003
$i=5$	0.5003	$i=21$	0.5002
$i=6$	0.5004	$i=22$	0.4996
$i=7$	0.4999	$i=23$	0.4452
$i=8$	0.4442	$i=24$	0.4862
$i=9$	0.4871	$i=25$	0.4962
$i=10$	0.4967	$i=26$	0.4995
$i=11$	0.4999	$i=27$	0.5003
$i=12$	0.4996	$i=28$	0.5005
$i=13$	0.4997	$i=29$	0.4997
$i=14$	0.5002	$i=30$	0.4996
$i=15$	0.4997	$i=31$	0.5007

chosen positions (typically those were one of the carries is guaranteed to be 0), the probability deviates significantly from 0.5. We observed biases as high as $\epsilon \simeq 2^{-3}$ for "good" positions such as $i=8$ or $i=23$. As a consequence,

$$[S]_{23}^t \oplus [S]_{23}^{t+1}$$

is equal to 0 with probability of $0.5\,(1+\varepsilon)$ and $|\varepsilon| \simeq 2^{-16} \times 2^{-3} = 2^{-19}$.

This bias is only valid when $[x_0]^t = 4$, which is the case for exactly one position over 16 in the keystream sequence. It is straightforward to determine which positions should be analyzed if we guess the 4 LSB's of the initial state. In the next section, we show how to exploit this bias for distinguishing and key-recovery attacks.

7.3 Third Step

If we exploit positions 8 or 23 of the output word, we showed in the previous section a bias of the order of $\varepsilon = 2^{-19}$. This can be used to distinguish TSC-3's output sequence from random data, provided $\varepsilon^{-2} = 2^{38}$ samples are given. Since only one position out of 16 in the output sequence is useful, it means that :

$$M = 16 \times 2^{38} = 2^{42}$$

output words are needed. In addition, we must try all values for the initial state's LSB, so the time complexity is about

$$T = 2^4 \times 2^{38} = 2^{42}$$

computation steps.

To mount a key-recovery attack, we start by guessing the 9 least significant layers of the initial state (36 bits in total), in order to predict $[x]_8^t$ for all t. This layer is also the least significant layer of the registers y_i, and it turns out that it is also used in one of the "best" linear approximations : $[S]_{23}^t \oplus [S]_{23}^{t+1}$.

Therefore, we can eliminate one term in this approximation which increases the bias from 2^{-19} to 2^{-15}. Once we found the correct guess for these 36 state bits, it is straightforward to continue the attack, like we did for TSC-1 and TSC-2. The first step is clearly the most expensive, because we must guess 36 bits at the same time. So, the time complexity is

$$T = 2^{36} \times (2^{15})^2 = 2^{66}$$

computation steps. The data complexity of this attack is only :

$$M = 16 \times 2^{30} = 2^{34}$$

output words.

These two attacks show that the stream cipher TSC-3 does not reach the expected security level.

8 Criteria for future design

First, we can notice that the 3 separate steps in our linear cryptanalysis framework are always possible, to some extent.

- The periodicity of the least significant layers in multi-word T-functions is always small, by construction. The periodicity of the i-th layer is always 2^{mi} at most for a state of m words. Therefore the following linear relation always holds with probability 1 :

  $$[x_j]_i^t \oplus [x_j]_i^{t+2^{mi}} = 0$$

  Other approximations can exist depending on the nature of the T-function, as illustrated in the case of the TSC family.
- For any choice of the output function, there exist linear approximations between input and output bits. Unless the function is very complex (but it is generally not the case, because the output function needs to be fast), it is likely that approximations with good probability can even be found.
- If the approximations of step 1 and step 2 can be combined, it is generally feasible to exploit these biased relations into a key-recovery attack.

Therefore the difficulty does not lie in finding linear approximations or exploiting them, but on combining all approximations to describe the complete cipher. This is something we did not manage to do for Klimov and Shamir's proposal for instance [16]. It is likely that T-function will receive a lot of attention in the future for stream cipher design. To prevent the application of linear cryptanalysis, we suggest to use several safeguards.

- Never use the least significant half of the registers in the output function, because of the small periodicity (this countermeasure was already applied by Klimov and Shamir in several proposals, and TSC-3 has also taken a step in this direction compared to its two predecessors).
- Use rotations in the output function in order to combine the bits from all registers. The output function of TSC-1 or TSC-3 is probably too simple, which makes the analysis easier.
- Try to avoid simple linear approximations for the T-functions over several consecutive steps. For the S-box based T-functions proposed by Hong *et al.*, it is an open problem to say if this is possible. It seems that the current proposals do not provide enough diffusion, but maybe for an appropriate instantiation, the existence of good linear approximations may be avoided. This is an interesting topic for future research.
- Try to take advantage of the "complex" operations which are available on processors. For instance, we believe it is a good idea to use the integer multiplication, when possible, even in the output function.

All these countermeasures may have a negative impact on the encryption speed, but this must be put into the balance with the increased level of security.

9 Conclusion and Comments

In this paper, we give a general framework of linear cryptanalysis for stream ciphers using a T-function. The idea consists in linearizing separately the T-function and the output function, and then connecting both approximations. We successfully applied it to the TSC family of stream ciphers but we believe it can have many applications against this emerging family.

We managed to find a key recovery attack requiring $2^{21.4}$ data with $2^{25.4}$ time for TSC-1, and $2^{44.1}$ data with $2^{48.1}$ time for TSC-2. The attack against TSC-1 has been implemented and requires about 4 minutes of analysis on an average PC. Thus, TSC-1 and TSC-2 are not secure enough for stand-alone use.

An advanced version of our attack also allows to break TSC-3, one of the stream ciphers recently proposed for the ECRYPT project. This attack is very interesting because the designers took into account distinguishing attacks by Künzli *et al.* and added countermeasures. However, our general framework still allows to break the cipher. TSC-3 can be distinguished from random by processing 2^{42} output words, and its secret key can be recovered with 2^{66} computation steps and 2^{34} known output words.

For future designs of stream ciphers, we suggest to benefit from complex operations that allow T-functions. For instance, integer multiplication has good diffusion properties and prevents good linear approximations. Moreover, we recommend never to use LSB's of the state registers in the output function.

References

1. F. Armknecht and M. Krause. Algebraic Attacks on Combiners with Memory. In D. Boneh, editor, *Advances in Cryptology – Crypto'03*, volume 2729 of *Lectures Notes in Computer Science*, pages 162–175. Springer, 2003.
2. S. Babbage. Stream Ciphers: What Does the Industry Want ? In *State of the Art of Stream Ciphers* workshop (SASC'04), 2004.
3. A. Biryukov and A. Shamir. Cryptanalytic time/memory/data tradeoffs for stream ciphers. In T. Okamoto, editor, *Advances in Cryptology – Asiacrypt'00*, volume 1976 of *Lectures Notes in Computer Science*, pages 1–13. Springer, 2000.
4. N. Courtois and W. Meier. Algebraic Attacks on Stream Ciphers with Linear Feedback. In E. Biham, editor, *Advances in Cryptology – Eurocrypt'03*, volume 2656 of *Lectures Notes in Computer Science*, pages 345–359. Springer, 2003.
5. C. Ding, G. Xiao, and W. Shan. *The Stability Theory of Stream Ciphers*, volume 561 of *Lectures Notes in Computer Science*. Springer, 1991. see Section 3.3.
6. ECRYPT Network of Excellence in Cryptology. `http://www.ecrypt.eu.org/index.html`.
7. ECRYPT Stream Cipher Project. See `http://www.ecrypt.eu.org/stream/`.
8. J. Golić. Linear Cryptanalysis of Stream Ciphers. In B. Preneel, editor, *Fast Software Encryption – 1994*, volume 1008 of *Lectures Notes in Computer Science*, pages 154–169. Springer, 1995.
9. J. Golić. Linear Statistical Weakness of Alleged RC4 Keystream Generator. In W. Fumy, editor, *Advances in Cryptology – Eurocrypt'97*, volume 1233 of *Lectures Notes in Computer Science*, pages 226–238. Springer, 1997.
10. J. Hong, D. Lee, Y. Yeom, and D. Han. A New Class of Single Cycle T-functions. In H. Gilbert and H. Handschuh, editors, *Fast Software Encryption – 2005*, volume 3557 of *Lectures Notes in Computer Science*, pages 68–82. Springer, 2005.
11. J. Hong, D. Lee, Y. Yeom, D. Han, and S. Chee. T-function Based Stream Cipher TSC-3. ECRYPT Stream Cipher Project Report 2005/031, 2005. http://www.ecrypt.eu.org/stream.
12. A. Klimov. *Applications of T-functions in Cryptography*. PhD thesis, Weizmann Institute of Science, 2004. http://www.wisdom.weizmann.ac.il/ ask/.
13. A. Klimov and A. Shamir. A New Class of Invertible Mappings. In B. Kaliski, Ç. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES) – 2002*, volume 2523 of *Lectures Notes in Computer Science*, pages 470–483. Springer, 2002.
14. A. Klimov and A. Shamir. Cryptographic Applications of T-functions. In M. Matsui and R. Zuccherato, editors, *Selected Areas in Cryptography – 2003*, volume 3006 of *Lectures Notes in Computer Science*, pages 248–261. Springer, 2004.
15. A. Klimov and A. Shamir. New Cryptographic Primitives Based on Multiword T-Functions. In B. Roy and W. Meier, editors, *Fast Software Encryption – 2004*, volume 3017 of *Lectures Notes in Computer Science*, pages 1–15. Springer, 2004.
16. A. Klimov and A. Shamir. The TFi Family of Stream Ciphers, 2004. Handout given at the SASC'04 workshop.

17. A. Klimov and A. Shamir. New Applications of T-functions in Block Ciphers and Hash Functions. In H. Gilbert and H. Handschuh, editors, *Fast Software Encryption – 2005*, volume 3557 of *Lectures Notes in Computer Science*, pages 18–31. Springer, 2005.
18. S. Künzli, P. Junod, and W. Meier. Attacks Against TSC. Rump Session at *Fast Software Encryption* (FSE'05), 2005.
19. S. Künzli, P. Junod, and W. Meier. Distinguishing Attacks on T-Functions. In *International Conference on Cryptology in Malaysia (MyCrypt 2005)*, 2005. To appear.
20. M. Matsui. Linear Cryptanalysis Method for DES Cipher. In T. Helleseth, editor, *Advances in Cryptology – Eurocrypt'93*, volume 765 of *Lectures Notes in Computer Science*, pages 386–397. Springer, 1993.
21. W. Meier and O. Staffelbach. Fast Correlations Attacks on Certain Stream Ciphers. In *Journal of Cryptology*, pages 159–176. Springer, 1989.
22. J. Mitra and P. Sarkar. Time-Memory Trade-Off Attacks on Multiplications and T-functions. In P. Lee, editor, *Advances in Cryptology - Asiacrypt'04*, volume 3329 of *Lectures Notes in Computer Science*, pages 468–482. Springer, 2004.
23. A. Shamir. Stream Ciphers: Dead or Alive ? Invited talk presented at Asiacrypt'04, 2004.
24. T. Siegenthaler. Correlation-immunity of Nonlinear Combining Functions for Cryptographic Applications. In *IEEE Transactions on Information Theory*, volume 30, pages 776–780, 1984.

Appendix

Table 7. TSC-1 for $t/t+3$: Bit Flip Probabilities of the Registers and of the LSB of the General Carry

bit position	register 0	register 1	register 2	register 3	LSB(R_G)
0	0.5000	0.5000	0.5000	0.5000	0.5953
1	0.6406	0.6406	0.6406	0.6406	0.4563
2	0.6479	0.6479	0.6479	0.6479	0.4948
3	0.6446	0.6446	0.6446	0.6446	0.5247
4	0.6442	0.6442	0.6442	0.6442	0.5328
5	0.6427	0.6427	0.6427	0.6427	0.5343
6	0.6356	0.6356	0.6356	0.6356	0.5356
7	0.6412	0.6412	0.6412	0.6412	0.5352
8	0.6413	0.6413	0.6413	0.6413	0.5263
9	0.6417	0.6416	0.6416	0.6417	0.5337
10	0.6417	0.6417	0.6417	0.6417	0.5355
11	0.6364	0.6364	0.6364	0.6364	0.5357
12	0.6408	0.6408	0.6409	0.6409	0.5354
13	0.6412	0.6411	0.6412	0.6412	0.5352
14	0.6416	0.6416	0.6415	0.6416	0.5354
15	0.6417	0.6416	0.6416	0.6417	0.4348
16	0.6364	0.6364	0.6364	0.6363	0.4952
17	0.6408	0.6409	0.6409	0.6408	0.5253
18	0.6412	0.6412	0.6412	0.6412	0.5332
19	0.6416	0.6416	0.6417	0.6416	0.5349
20	0.6364	0.6364	0.6364	0.6364	0.5355
21	0.6409	0.6408	0.6408	0.6409	0.5359
22	0.6412	0.6412	0.6412	0.6413	0.5355
23	0.6365	0.6365	0.6365	0.6365	0.5360
24	0.6408	0.6408	0.6408	0.6409	0.5349
25	0.6413	0.6413	0.6413	0.6413	0.5266
26	0.6366	0.6365	0.6366	0.6366	0.5333
27	0.6409	0.6408	0.6408	0.6409	0.5351
28	0.6414	0.6413	0.6413	0.6412	0.5357
29	0.6365	0.6366	0.6365	0.6365	0.5357
30	0.6408	0.6409	0.6408	0.6408	0.5355
31	0.6412	0.6412	0.6412	0.6412	0.5351

Table 8. TSC-2 for $t/t+2$: Bit Flip Probabilities of the Registers and of the LSB of the General Carry

bit position	register 0	register 1	register 2	register 3	LSB(R_G)
0	0.0000	0.5000	0.5000	0.6250	0.4929
1	0.5000	0.4062	0.4062	0.4453	0.4960
2	0.5312	0.3848	0.3848	0.4146	0.4942
3	0.4980	0.3744	0.3744	0.4057	0.4943
4	0.4957	0.3739	0.3739	0.4054	0.4894
5	0.4858	0.3715	0.3715	0.4036	0.4940
6	0.4811	0.3703	0.3703	0.4027	0.4955
7	0.4792	0.3698	0.3698	0.4023	0.4949
8	0.4785	0.3697	0.3697	0.4022	0.4952
9	0.4782	0.3695	0.3695	0.4021	0.4948
10	0.4780	0.3695	0.3695	0.4022	0.4947
11	0.4781	0.3694	0.3695	0.4022	0.4949
12	0.4781	0.3695	0.3695	0.4021	0.4944
13	0.4781	0.3696	0.3696	0.4021	0.4892
14	0.4781	0.3695	0.3696	0.4021	0.4877
15	0.4781	0.3695	0.3694	0.4021	0.4963
16	0.4781	0.3695	0.3696	0.4021	0.4952
17	0.4781	0.3695	0.3696	0.4021	0.4949
18	0.4781	0.3696	0.3696	0.4022	0.4948
19	0.4781	0.3695	0.3695	0.4021	0.4945
20	0.4782	0.3695	0.3696	0.4021	0.4947
21	0.4781	0.3695	0.3695	0.4020	0.4944
22	0.4781	0.3696	0.3695	0.4022	0.4875
23	0.4782	0.3695	0.3695	0.4021	0.4945
24	0.4781	0.3696	0.3696	0.4022	0.4953
25	0.4781	0.3696	0.3696	0.4021	0.4946
26	0.4781	0.3695	0.3695	0.4022	0.4896
27	0.4781	0.3695	0.3696	0.4022	0.4942
28	0.4781	0.3695	0.3695	0.4022	0.4955
29	0.4781	0.3695	0.3695	0.4022	0.4951
30	0.4782	0.3696	0.3696	0.4022	0.4949
31	0.4781	0.3695	0.3695	0.4022	0.4951

Table 9. Possible Attack Schedule for TSC-2

round	bit position attacked	register attacked	number of known terms	bias obtained	time complexity
0	3	0	1	$2^{-21.76}$	$2^{47.52}$
1	13	0	1	$2^{-20.79}$	$2^{45.58}$
2	14	0	2	$2^{-18.66}$	$2^{41.32}$
3	3	3	2	$2^{-19.36}$	$2^{42.72}$
4	4	3	2	$2^{-18.46}$	$2^{40.92}$
5	5	3	2	$2^{-19.28}$	$2^{42.56}$
6	6	3	2	$2^{-19.70}$	$2^{43.40}$
7	7	3	2	$2^{-19.52}$	$2^{43.04}$
8	8	3	2	$2^{-19.60}$	$2^{43.20}$
9	12	0	2	$2^{-17.22}$	$2^{38.44}$
10	3	0	3	$2^{-14.84}$	$2^{33.68}$
11	4	0	3	$2^{-13.95}$	$2^{31.90}$
12	5	0	3	$2^{-14.77}$	$2^{33.54}$
13	3	2	4	$2^{-12.91}$	$2^{29.82}$
14	14	3	4	$2^{-11.80}$	$2^{27.60}$
15	5	2	4	$2^{-12.83}$	$2^{29.66}$
16	6	2	4	$2^{-13.25}$	$2^{30.50}$
17	7	2	4	$2^{-13.07}$	$2^{30.14}$
18	8	2	4	$2^{-13.15}$	$2^{30.30}$
19	9	2	4	$2^{-13.04}$	$2^{30.08}$
20	0	0	5	$2^{-10.65}$	$2^{25.30}$
21	14	0	5	$2^{-7.28}$	$2^{18.56}$
22	15	0	5	$2^{-9.01}$	$2^{22.02}$
23	3	0	6	$2^{-6.45}$	$2^{16.90}$
24	4	0	6	$2^{-5.56}$	$2^{15.12}$
25	5	0	6	$2^{-6.38}$	$2^{16.76}$
26	6	0	6	$2^{-6.80}$	$2^{17.60}$
27	7	0	6	$2^{-6.62}$	$2^{17.24}$
28	8	0	6	$2^{-6.70}$	$2^{17.40}$
29	9	0	6	$2^{-6.59}$	$2^{17.18}$
30	10	0	6	$2^{-6.56}$	$2^{17.12}$
31	11	0	6	$2^{-6.62}$	$2^{17.24}$