New Directions in Cryptanalysis of Self-Synchronizing Stream Ciphers

Shahram Khazaei[1] and Willi Meier[2]

[1] EPFL, Lausanne, Switzerland
[2] FHNW, Windisch, Switzerland

Abstract. In cryptology we commonly face the problem of finding an unknown key K from the output of an easily computable keyed function $F(C, \mathsf{K})$ where the attacker has the power to choose the public variable C. In this work we focus on self-synchronizing stream ciphers. First we show how to model these primitives in the above-mentioned general problem by relating appropriate functions F to the underlying ciphers. Then we apply the recently proposed framework presented at AfricaCrypt'08 by Fischer *et. al.* for dealing with this kind of problems to the proposed T-function based self-synchronizing stream cipher by Klimov and Shamir at FSE'05 and show how to deduce some non-trivial information about the key. We also open a new window for answering a crucial question raised by Fischer *et. al.* regarding the problem of finding weak IV bits which is essential for their attack.

Key words: Self-synchronizing Stream Ciphers, T-functions, Key Recovery.

1 Introduction

The area of stream cipher design and analysis has made a lot of progress recently, mostly spurred by the eStream [6] project. It is a common belief that designing elegant strong synchronizing stream ciphers is possible, however, it is harder to come up with suitable designs for self-synchronizing ones. Despite numerous works on self-synchronizing stream ciphers in the literature, there is not yet a good understanding of their design and cryptanalytic methods. Many self-synchronizing stream ciphers have shown not to withstand cryptanalytic attacks and have been broken shortly after they have been proposed. In this work we show how to model a self-synchronizing stream cipher by a family of keyed functions $F(C, \mathsf{K})$. The input parameter C, called the *public variable*, can be controlled by the attacker while the input K is an unknown parameter to her called the extended key; it is a combination of the actual key used in the cipher and the unknown internal state of the cipher. The goal of the attacker would be to recover K or to get some information about it. The problem of finding the unknown key K, when access is given to the output of the function $F(C, \mathsf{K})$ for every C of the attacker's choice, is a very common problem encountered in cryptography. In general, when the keyed function F looks like a random function, the best way to solve the problem is to exhaust the key space. However, if F is far from being a random function there might be more efficient methods. Recently, Fischer *et. al.* [7] developed a method to recover the key faster than by exhaustive search in case F does not properly mix its input bits. The idea is to first identify some bits from C referred to as *weak public variable bits* and then to consider the coefficient of a monomial involving these weak bits in the algebraic normal form of F. If this coefficient does not depend on all the unknown bits of K, or it weakly depends on some of them, it can be exploited in an attack. Having modeled the self-synchronizing stream ciphers as the above-mentioned general problem, we consider the T-function based self-synchronizing stream cipher proposed by Klimov and Shamir [8] and use the framework from [7] to deduce some information about the key bits through some striking relations. Finding the weak public variable bits was raised as a crucial open question in [7] which was done mostly by random search there. In the second part of our work we try to shed some light in this direction in a more systematic way. The recently proposed cube attack by Dinur and Shamir [4], which has a strong connection to [7] and the present work, also includes some systematic procedure to find weak public variable bits.

The rest of the paper is organized as follows. In section 2 we review the method from [7] and try to make the connection between [4] and [7] clearer. In section 3 we describe the self-synchronizing stream ciphers and explain how to derive keyed functions $F(C, \mathsf{K})$ which suit the framework from [7]. Section 4 covers the description of the Klimov-Shamir T-function based self-synchronizing stream cipher along with its reduced word-size versions which will later be attacked in section 5 and 6. Section 6 also includes our new direction of finding weak bits in a systematic way.

2 An Approach for Key Recovery on a Keyed Function

Notations. We use $\mathbb{B} = \{0, 1\}$ for the binary field with two elements. A general m-bit vector in $\mathbb{B}^m$ is denoted by $C = (c_1, c_2, \ldots, c_m)$. By making a partition of C into $U \in \mathbb{B}^l$ and $W \in \mathbb{B}^{m-l}$, we mean dividing the variables set $\{c_1, c_2, \ldots, c_m\}$ into two disjoint subsets $\{u_1, ..., u_l\}$ and $\{w_1, ..., w_{m-l}\}$ and setting $U = (u_1, \ldots, u_l)$ and $W = (w_1, \ldots, w_{m-l})$. However, whenever we write $(U; W)$ we mean the original vector C. For example $U = (c_2, c_4)$ and $W = (c_1, c_3, c_5)$ is a partition for the vector $C = (c_1, c_3, c_4, c_5)$ and $(U; W)$ is equal to C and not to $(c_2, c_4, c_1, c_3, c_5)$. We also use the notation $U = C \setminus W$ and $W = C \setminus U$. A vector of size zero is denoted by $\emptyset$.

In this section we review the framework from [7] which was inspired by results from [1, 5,11]. Let $F : \mathbb{B}^m \times \mathbb{B}^n \rightarrow \mathbb{B}$ be a keyed Boolean function which maps the m-bit public variable C and the n-bit secret variable K into the output bit $z = F(C, \mathsf{K})$. An oracle chooses a key K uniformly at random over $\mathbb{B}^n$ and returns $z = F(C, \mathsf{K})$ to an adversary for any chosen $C \in \mathbb{B}^m$ of adversary's choice. The oracle chooses the key K only once and keeps it fixed and unknown to the adversary. The goal of the adversary is to recover K by dealing with the oracle assuming that he has also the power to evaluate F for all inputs, *i.e.* all secret and public variables. To this end, the adversary can try all possble 2^n keys and filter the wrong ones by asking enough queries from the oracle. Intuitively each oracle query reveals one bit of information about the secret key if F mixes its input bits well enough to be treated as a random Boolean function with $n + m$ input bits. Therefore, assuming $\log_2 n \ll m$, then n key bits can be recovered by sending $\mathcal{O}(n)$ queries to the oracle. More precisely if the adversary asks the oracle

$n + \beta$ queries for some integer $\beta \gg 0$, then the probability that only the unknown chosen key by the oracle (*i.e.* the correct candidate) satisfies these queries while all the remaining $2^n - 1$ keys fail to satisfy all the queries is $(1 - 2^{-(n+\beta)})^{2^n-1} \approx 1 - e^{-2^{-\beta}}$ (for $\beta = 10$ it is about $1 - 10^{-3}$). The required time complexity is $\mathcal{O}(n2^n)$. However, if F extremely deviates from being treated as a random function, the secret key bits may not be determined uniquely. It is easy to argue that F divides $\mathbb{B}^n$ into J *equivalence classes* $\mathcal{K}_1,\ \mathcal{K}_2, \dots, \mathcal{K}_J$ for some $J \leq 2^n$, see Lemma 1 from [7]. Two keys K' and K'' belong to the same equivalence class iff $F(C, \mathsf{K}') = F(C, \mathsf{K}'')$ for all $C \in \mathbb{B}^m$. Let n_i denotes the number of keys which belong to the equivalence class $\mathcal{K}_i$. Note that we have $\sum_{i=1}^{J} n_i = 2^n$. A random key lies in the equivalence class $\mathcal{K}_i$ with probability $n_i/2^n$ in which case $(n - \log_2 n_i)$ bits of information can be achieved about the key. The adversary on average can get $\sum_{i=1}^{J} (n - \log_2 n_i)\frac{n_i}{2^n}$ bits of information about the n key bits by asking enough queries. It is difficult to estimate the minimum number of needed queries due to the statistical dependency between them. It highly depends on the structure of F but we guess that $O(n)$ queries suffice again. However, in case where F does not properly mix its input bits, there might be faster methods than exhaustive search for key recovery. We are interested in faster methods of recovering the unknown secret key in this case.

If one derives a weaker keyed function $\varGamma(W, \mathsf{K}) : \mathbb{B}^{m-l} \times \mathbb{B}^n \rightarrow \mathbb{B}$ from F which depends on the same key and a part of the public variables, the adversary-oracle interaction can still go on through $\varGamma$ this time. The idea of [7] is to derive such functions from the algebraic expansion of F by making a partition of the m-bit public variable C into $C = (U; W)$ with l-bit vector U and $(m - l)$-bit vector W. Let $F(C, \mathsf{K}) = \sum_{\alpha} \varGamma_{\alpha}(W, \mathsf{K}) U^{\alpha}$ where $U^{\alpha} = u_1^{\alpha_1} u_2^{\alpha_2} \dots u_l^{\alpha_l}$ for the multi-index $\alpha = (\alpha_1, \dots, \alpha_l)$. In other words, $\varGamma_{\alpha}(W, \mathsf{K})$ is the coefficient of U^{α} in the algebraic expansion of F. For every $\alpha \in \mathbb{B}^l$, the function $\varGamma_{\alpha}(W, \mathsf{K})$ can serve as a function $\varGamma$ derived from F. The function corresponding to $\alpha = (1, \dots, 1)$ is the coefficient of the maximum degree monomial. Previous works [5, 7] suggest that this function is usually more useful. We also only focus on the maximum degree monomial coefficient. Hence we drop the subscript α and write $\varGamma(W, \mathsf{K})$ instead of $\varGamma_{\alpha}(W, \mathsf{K})$ for $\alpha = (1, \dots, 1)$. Inspired by the terminology of [4] we refer to U as *cube vector* and to $\varGamma(W, \mathsf{K})$ as *superpoly* corresponding to cube vector U. Thanks to the relation $\varGamma(W, \mathsf{K}) = \bigoplus_{U \in \mathbb{B}^l} F((U; W), \mathsf{K})$, the adversary can still evaluate the superpoly for any W of his choice and for the same chosen key K by the oracle. This demands that the adversary sends 2^l queries to the oracle for each evaluation of $\varGamma$.

In order to have an effective attack we need to have a weak superpoly function. In [7] several conditions were discussed under which the superpoly can be considered as a weak function and potentially lead to an attack. Refer to [4] for more scenarios and generalizations. In this paper we look for cube vectors U such that their superpoly does not depend on a large number of key bits. We refer to those key bits which $\varGamma(W, \mathsf{K})$ does not depend on as *neutral key bits*. This is a special case of the third scenario in [7] where *probabilistic neutral bits* were used instead. If the superpoly effectively depends on $t_k \leq n$ key bits and $t_w \leq m - l$ public key bits, assuming these $t_k + t_w$ bits are mixed reasonably well, the involved t_k secret bits can be recovered in time $t_k 2^{l+t_k}$ by sending $O(t_k 2^l)$ queries to the oracle. However, if the superpoly extremely deviates from being treated as a random function, as we already argued, it may even happen that the t_k key bits can not be determined uniquely. In this case one has to look at the corresponding equivalence classes to see how much information one can achieve about the involved t_k key bits. In sections 5 and 6 we will provide some examples by considering Klimov-Shamir's self-synchronizing stream cipher.

2.1 Connection With Previous Works

The attack is closely related to differential [2, 9] and integral [3, 10] kind of attacks, and the recent cube attack [4]. For $l = 0$ we have $U = \emptyset$ and $W = C$ and hence $\Gamma = F$, that is we are analyzing the original function. For $l = 1$ let's take $U = (c_i)$ and $W = C \setminus (c_i)$ for some $1 \leq i \leq m$. In this case we are considering a variant of (truncated) differential cryptanalysis, that is we have $\Gamma(W, \mathsf{K}) = F(C, \mathsf{K}) \oplus F(C \oplus \Delta C, \mathsf{K})$ where ΔC is an m-bit vector which is zero in all bit positions except the i-th one. For bigger l, this approach can be seen as an adaptive kind of higher order differential cryptanalysis. A more precise relation between the framework in [7] and (higher order) differential cryptanalysis seems to be as follows: The superpoly $\Gamma(W, \mathsf{K})$, which computes the coefficient of the maximum degree monomial, is computed as the sum of all outputs $F(C, \mathsf{K})$ where $C = (U; W)$ has a fixed part W and U varies over all possible values. This is what is also done in (higher order) differential cryptanalysis. However, in applications of the framework in [7], the values for W are often chosen adaptively. By adaptively we mean that a stronger deviation from randomness is observed for some specific choices for W (*e.g.* low weight W's) or even a specific value for W (*e.g.* W = 0). Whereas in most applications of (higher order) differential cryptanalysis, specific input values are of no favour. The recently proposed cube attack by Dinur and Shamir [4] still lies in the second scenario (condition) proposed in [7], having had been inspired by the earlier work by Vielhaber [12]. In [7] the public variable C was the Initial Vector of a stream cipher and the cube variables were called *weak IV bits* whenever the derived function Γ turned out to be weak enough to mount an attack. This concept can be adapted according to each context depending on the public variable (weak ciphertext bits, weak plaintext bits, weak message bits, *etc*). In general the terminology *weak public variables* can be used. On the whole, it is not easy to find week public variables. While [4] uses a more systematic procedure, [7] uses random search over cube vectors. In section 6, we will also take kind of systematic method. Another point which is worth mentioning is that cube attack [4] nicely works with complexity $O(n2^{d-1})$ if F is a random function of degree d in its $m + n$ input bits. In this case the superpoly corresponding to any cube vector of size $d - 1$ is weak, since it is a random linear function in key bits and remaining public variables.

3 Self-Synchronizing Stream Ciphers

A self-synchronizing stream cipher is built on an output filter $\mathcal{O} : \mathcal{K} \times \mathcal{S} \to \mathcal{M}$ and a self-synchronizing state update function (see Definition 1) $\mathcal{U} : \mathcal{M} \times \mathcal{K} \times \mathcal{S} \to \mathcal{M}$, where $\mathcal{S}$, $\mathcal{K}$ and $\mathcal{M}$ are the cipher state space, key space and plaintext space. We suppose that the ciphertext space is the same as that of the plaintext. Let $K \in \mathcal{K}$ be the secret key, and $\{S_i\}_{i=0}^{\infty}$, $\{p_i\}_{i=0}^{\infty}$ and $\{c_i\}_{i=0}^{\infty}$ denote the sequences of cipher state, plaintext and ciphertext respectively. The initial state is computed through the initialization procedure as $S_0 = \mathcal{I}(K, IV)$ from the secret key K and a public initial value IV. The ciphertext (in an additive stream cipher) is then computed according to the following relations:

$$c_i = p_i \oplus \mathcal{O}(K, S_i), \tag{1}$$

$$S_{i+1} = \mathcal{U}(c_i, K, S_i). \tag{2}$$

Definition 1. *[8] (SSF) Let $\{c_i\}_{i=0}^{\infty}$ and $\{\hat{c}_i\}_{i=0}^{\infty}$ be two input sequences, let S_0 and $\hat{S}_0$ be two initial states, and let K be a common key. Assume that the function $\mathcal{U}$ is used to update the state based on the current input and the key: $S_{i+1} = \mathcal{U}(c_i, K, S_i)$ and $\hat{S}_{i+1} = \mathcal{U}(\hat{c}_i, K, \hat{S}_i)$. The function $\mathcal{U}$ is called a self-synchronizing function (SSF) if equality of any r consecutive inputs implies the equality of the next state, where r is some integer, i.e.:*

$$c_i = \hat{c}_i, \ldots, c_{i+r-1} = \hat{c}_{i+r-1} \Rightarrow S_{i+r} = \hat{S}_{i+r}. \tag{3}$$

Definition 2. *The "resynchronization memory" of a function $\mathcal{U}$, assuming it is a SSF, is the least positive value of r such that Eq. 3 holds.*

3.1 Attack Models on Self-Synchronizing Stream Ciphers

There are two kinds of attack on synchronizing stream ciphers: *distinguishing attacks* and *key recovery attacks*[3]. The strongest scenario in which these attacks can be applied is a *known-keystream* attack model or a *chosen-IV-known-keystream* attack if the cipher uses an IV for initialization. It is not very clear how applying distinguishing attacks make sense for self-synchronizing stream ciphers. However, in the strongest scenario, one considers key recovery attacks in a *chosen-ciphertext* attack model or in a *chosen-IV-chosen-ciphertext* attack if the cipher uses an IV for initialization.

In this paper we only focus on chosen-ciphertext attacks. Our goal as an attacker is to efficiently recover the unknown key K by sending to the decryption oracle chosen ciphertexts of our choice. More precisely, we consider the family of functions $\{\mathcal{H}_i : \mathcal{M}^i \times \mathcal{K} \times \mathcal{S} \to \mathcal{M} | i = 1, 2, \ldots r\}$, where r is the resynchronization memory of the cipher and $\mathcal{H}_i(c_1, \ldots, c_i, K, S) = \mathcal{O}(K, \mathcal{G}_i(c_1, \ldots, c_i, K, S))$, where $\mathcal{G}_i : \mathcal{M}^i \times \mathcal{K} \times \mathcal{S} \to \mathcal{S}$ is recursively defined as $\mathcal{G}_{i+1}(c_1, \ldots, c_i, c_{i+1}, K, S) = \mathcal{U}(c_{i+1}, K, \mathcal{G}_i(c_1, \ldots, c_i, K, S))$ with initial condition $\mathcal{G}_1 = \mathcal{U}$.

Note that due to the self-synchronizing property of the cipher $\mathcal{H}_r(c_1, \ldots, c_r, K, S)$ is actually independent of the last argument S, however, all other $r-1$ functions depend on their last input. The internal state of the cipher is unknown at each step of operation of the cipher but because of the self-synchronizing property of the cipher it only depends on the last r ciphertext inputs and the key. We take advantage of this property and force the cipher to get stuck in a fixed but unknown state $\mathsf{S}^\star$ by sending the decryption oracle ciphertexts with some fixed prefix $(\mathsf{c}^\star_{-r+1}, \ldots, \mathsf{c}^\star_{-1})$ of our choice. Having forced the

[3] One could also think of *state recovery attack* in cases in which the synchronizing stream cipher is built based on a finite state machine and the internal state does not easily reveal the key.

cipher to fall in the unknown fixed state $\mathsf{S}^\star$, we can evaluate any of the functions $\mathcal{H}_i$, $i = 1, 2, \ldots, r$, at any point $(c_1, \ldots, c_i, K, \mathsf{S}^\star)$ for any input $(c_1, \ldots, c_i)$ of our choice by dealing with the decryption oracle. To be clearer let $z = \mathcal{H}_i(c_1, \ldots, c_i, K, \mathsf{S}^\star)$. In order to compute z for an arbitrary $(c_1, \ldots, c_i)$, we choose an arbitrary $\mathsf{c}^\star_{i+1} \in \mathcal{M}$ and ask the decryption oracle for $(p_{-r+1}, \ldots, p_{-1}, p_0, \ldots, p_{i+1})$– the decrypted plaintext corresponding to the ciphertext $(\mathsf{c}^\star_{-r+1}, \ldots, \mathsf{c}^\star_0,\ c_1, \ldots, c_i, \mathsf{c}^\star_{i+1})$. We then set $z = p_{i+1} \oplus \mathsf{c}^\star_{i+1}$.

To make notations simpler, we merge the unknown values K and $\mathsf{S}^\star$ in one unknown variable $\mathsf{K} = (K, \mathsf{S}^\star) \in \mathcal{K} \times \mathcal{S}$, called *extended unknown key*. We then use the simplified notation $\mathcal{F}_i(C, \mathsf{K}) = \mathcal{H}_i(c_1, \ldots, c_i, K, \mathsf{S}^\star) : \mathcal{M}^i \times (\mathcal{K} \times \mathcal{S}) \to \mathcal{M}$ where $C = (c_1, \ldots, c_i)$.

4 Description of the Klimov-Shamir T-function Based Self-Synchronizing Stream Cipher

Shamir and Klimov [8] used the so-called multiword T-functions for a general methodology to construct a variety of cryptographic primitives. No fully specified schemes were given, but in the case of self-synchronizing stream ciphers, a concrete example construction was outlined. This section recalls its design. Let $\lll$, $+$, $\times$, $\oplus$ and $\vee$ respectively denote left rotation, addition modulo 2^{64}, multiplication modulo 2^{64}, bit-wise XOR and bit-wise OR operations on 64-bit integers. The proposed design works with 64-bit words and has a 3-word internal state $S = (s_0, s_1, s_2)^T$. A 5-word key $K = (k_0, k_1, k_2, k_3, k_4)$ is used to define the output filter and the state update function as follows:

$$\mathcal{O}(K, S) = \big((s_0 \oplus s_2 \oplus k_3) \lll 32\big) \times \big(((s_1 \oplus k_4) \lll 32) \vee 1\big), \tag{4}$$

and

$$\mathcal{U}(c,\ K,\ S) = \begin{pmatrix} \big(((s'_1 \oplus s'_2) \vee 1) \oplus k_0\big)^2 \\ \big(((s'_2 \oplus s'_0) \vee 1) \oplus k_1\big)^2 \\ \big(((s'_0 \oplus s'_1) \vee 1) \oplus k_2\big)^2 \end{pmatrix}, \tag{5}$$

where

$$\begin{aligned} s'_0 &= s_0 \oplus c \\ s'_1 &= s_1 - (c \lll 21) \\ s'_2 &= s_2 \oplus (c \lll 43). \end{aligned} \tag{6}$$

Generalized Versions: We also consider generalized versions of this cipher which use ω-bit words (ω even and typically $\omega = 8, 16, 32$ or 64). For ω-bit version the number of rotations in the output filter, Eq. 4, is $\frac{\omega}{2}$ and those of the state update function, Eq. 6, are $\lfloor \frac{\omega}{3} \rceil$ and $\lfloor \frac{2\omega}{3} \rceil$, $\lfloor x \rceil$ being the closest integer to x.

It can be shown [8] that the update function $\mathcal{U}$ is actually a SSF whose resynchronization memory is limited to ω steps and hence the resulting stream cipher is self-synchronizing indeed. Our analysis of the cipher for $\omega = 8,\ 16,\ 32$ and 64 shows that it resynchronizes after $r = \omega - 1$ steps (using $\omega(\omega - 1)$ input bits). It is an open question if this holds in general.

Remark 1. In [8] the notation $(k_0, k_1, k_2, k_{\mathcal{O}}, k'_{\mathcal{O}})$ is used for the key instead of the more standard notation $(k_0, k_1, k_2, k_3, k_4)$. The authors possibly meant to use a 3-word key (k_0, k_1, k_2) by deriving the other two key words ($k_{\mathcal{O}}$ and $k'_{\mathcal{O}}$ in their notations corresponding to k_3 and k_4 in ours) from first three key words. However, they do not specify how this must be done if they meant so. Also they did not introduce an initialization procedure for their cipher. In any case, we attack a more general situation where the cipher uses a 5-word secret key $K = (k_0, k_1, k_2, k_3, k_4)$ in chosen-ciphertext attack scenario. Moreover, for the 64-bit version the authors mentioned "the best attack we are aware of this particular example [64-bit version] requires $\mathcal{O}(2^{96})$ time", without mentioning the attack.

5 Analysis of the Klimov-Shamir T-Function Based Self-Synchronizing Stream Cipher

Let ω ($\omega = 8, 16, 32$ or 64) denote the word size and $r = \omega - 1$ be the resynchronization memory of the ω-bit version of the Klimov-Shamir self-synchronizing stream cipher. Let $\mathbb{B} = \{0, 1\}$ and $\mathbb{B}_\omega$ denote the binary field and the set of ω-bit words respectively. Following the general model of analysis of self-synchronizing stream ciphers in section 3.1, we focus on the family of functions $\mathcal{F}_i(C, \mathsf{K}) : \mathbb{B}_\omega^i \times \mathbb{B}_\omega^8 \to \mathbb{B}_\omega$, $i = 1, 2, \ldots, r$ where $C = (c_1, \ldots, c_i)$ and $\mathsf{K} = (K, \mathsf{S}^\star) = (k_0, k_1, k_2, k_3, k_4, \mathsf{s}_0^\star, \mathsf{s}_1^\star, \mathsf{s}_2^\star)$. We also look at a word b as an ω-bit vector $b = (b_0, \ldots, b_{\omega-1})$, b_0 being its LSB and $b_{\omega-1}$ its MSB. Therefore any vector $A = (a_0, a_1, ..., a_{p-1}) \in \mathbb{B}_\omega^p$ could be also treated as a vector in $\mathbb{B}^{p\times\omega}$ where the $(i\omega + j)$-th bit of A is $a_{i,j}$, the j-th LSB of the word a_i, for $i = 0, 1, \ldots, p-1$ and $j = 0, 1, \ldots, \omega - 1$ (we start numbering the bits of vectors from zero).

Now, for any $i = 1, \ldots, r$ and $j = 0, \ldots, \omega - 1$ we consider the family of Boolean functions $\mathcal{F}_{i,j} : \mathbb{B}^{i\omega} \times \mathbb{B}^{8\omega} \to \mathbb{B}$ which maps the $i\omega$-bit input C and the 8ω-bit extended key K into the j-th LSB of the word $\mathcal{F}_i(C, \mathsf{K})$. Any of these keyed functions can be put into the framework from [7] explained in section 2. The next step is to consider a partitioning $C = (U; W)$ with l-bit segment U and $(i\omega - l)$-bit segment W to derive the (hopefully weaker) functions $\mathit{\Gamma}_{i,j}^U : \mathbb{B}^{i\omega - l} \times \mathbb{B}^{8\omega} \to \mathbb{B}$ where $\mathit{\Gamma}_{i,j}^U$ is the superpoly in $\mathcal{F}_{i,j}$ corresponding to the cube vector U. Whenever there is no ambiguity we drop the superscript or the subscripts. We may also use $\mathit{\Gamma}_{i,j}^U[\omega]$ in some cases to emphasize the word-size. We are now ready to give our simulation results.

Note: Instead of giving giving the variables of cube vector U we give the bit numbers. For example for $\omega = 16$, the set $\{0, 18, 31, 32\}$ stands for the cube vector $U = (c_{1,0}, c_{2,2}, c_{2,15}, c_{3,0})$.

Example 1. For all possible common word sizes ($\omega = 8, 16, 32$ or 64) we have been able to find some i, j and U such that $\mathit{\Gamma}$ is independent of W and only depends on three key bits $k_{0,0}$, $k_{1,0}$ and $k_{2,0}$. Table 1 shows some of these quite striking relations. We also found relations $\mathit{\Gamma}_{1,0}^{\{3\}}[8] = 1 + k_{2,0}$ and $\mathit{\Gamma}_{1,0}^{\{6,7,8,9,10\}}[16] = 1 + k_{0,0}$ involving only one key bit. For $\omega = 64$, the three relations in Table 1 give 1.75 bits of information about $(k_{0,0}, k_{1,0}, k_{2,0})$.

ω	i	j	U	$\mathit{\Gamma}_{i,j}^{U}[\omega]$
8	2	0	$\{2\}$	$1 + k_{0,0}k_{1,0} + k_{0,0}k_{2,0} + k_{1,0}k_{2,0}$
16	3	0	$\{5\}$	$1 + k_{0,0}k_{1,0} + k_{2,0} + k_{0,0}k_{1,0}k_{2,0}$
16	3	0	$\{10\}$	$1 + k_{0,0} + k_{1,0}k_{2,0} + k_{0,0}k_{1,0}k_{2,0}$
32	5	0	$\{11\}$	$1 + k_{0,0}k_{1,0} + k_{2,0} + k_{0,0}k_{1,0}k_{2,0}$
32	16	0	$\{96, 97, 98\}$	$1 + k_{0,0} + k_{2,0} + k_{0,0}k_{2,0}$
64	11	0	$\{21\}$	$1 + k_{0,0}k_{1,0} + k_{2,0} + k_{0,0}k_{1,0}k_{2,0}$
64	11	0	$\{42\}$	$1 + k_{0,0} + k_{1,0}k_{2,0} + k_{0,0}k_{1,0}k_{2,0}$
64	12	0	$\{20\}$	$1 + k_{0,0}k_{1,0} + k_{0,0}k_{2,0} + k_{1,0}k_{2,0}$

Table 1. Simple relations on three key bits $(k_{0,0}, k_{1,0}, k_{2,0})$

A more detailed analysis of the functions $\mathit{\Gamma}_{i,j}^{U}[\omega](W, \mathsf{K})$ for different values of i, j and U reveals that many of these functions depend on only few bits of their $(i\omega - l)$-bit and 8ω-bit arguments. Let t_w and t_k respectively denote the number of bits of W and K which $\mathit{\Gamma}$ effectively depends on. In addition let t'_k out of t_k bits come from K and the remaining $t_s = t_k - t'_k$ bits from $\mathsf{S}^\star$ (remember $\mathsf{K} = (K, \mathsf{S}^\star)$). Table 2 shows these values for some of these functions.

Having in mind what we mentioned in section 2 and being too optimistic, we give the following proposition.

Proposition 1. *If a function $\mathit{\Gamma}_{i,j}^{U}$ is random-looking enough, recovering the t_k unknown bits of the extended key takes expected time $i \times t_k \times 2^{l+t_k}$.*

The unity of time is processing one ciphertext word of the underlined self-synchronizing stream cipher. The factors 2^l and i come from the following facts: computing $\mathit{\Gamma}$ from $\mathcal{F}_i$ needs 2^l evaluations of $\mathcal{F}_i$ (remember $\mathit{\Gamma}_{i,j}^{U}(W, \mathsf{K}) = \bigoplus_{U \in \mathbb{B}^l} \mathcal{F}_{i,j}((U; W), \mathsf{K})$) and computing $\mathcal{F}_i$ needs i iterations of the cipher.

Even if the ideal condition of Proposition 1 is not satisfied, the only thing which is not guaranteed is that the t_k involved unknown bits are uniquely determined. Yet some information about them can be achieved. Refer to the note in section 2 regarding the equivalence classes.

Example 2. Take the relation $\mathit{\Gamma}_{3,0}^{\{18\}}[8](W, \mathsf{K})$ from Table 2. This particular function depends on $t_k = 5$ bits $(k_{0,0}, k_{0,1}, k_{1,0}, k_{2,0}, k_{2,1})$ of the key and on $t_w = 7$ bits

ω	i	j	U	t_k	t_w	t'_k	t_s	comment
8	1	0	$\emptyset$	20	8	9	11	
16	1	0	$\emptyset$	40	16	17	23	
32	1	0	$\emptyset$	80	32	33	47	
64	1	0	$\emptyset$	160	64	65	95	
8	1	0	$\{1\}$	9	5	3	6	
16	1	0	$\{u\}$	18	11	6	12	$8 \leq u \leq 10$
32	1	0	$\{u\}$	42	23	14	28	$16 \leq u \leq 20$
64	1	0	$\{u\}$	90	51	30	60	$32 \leq u \leq 42$
8	3	0	$\{8\}$	4	6	4	0	
8	3	0	$\{18\}$	5	7	5	0	
16	7	0	$\{16\}$	17	58	17	0	
16	7	0	$\{34\}$	16	52	16	0	
16	7	0	$\{33, 34\}$	12	33	12	0	
16	7	0	$\{38, 39\}$	12	30	12	0	
32	15	0	$\{32\}$	41	293	41	0	
32	15	0	$\{66\}$	40	279	40	0	
32	15	0	$\{76, 77\}$	36	231	36	0	
64	31	0	$\{64\}$	89	1274	89	0	
64	31	0	$\{130\}$	88	1243	88	0	
64	31	0	$\{129, 130\}$	84	1158	84	0	
64	31	0	$\{150, 151\}$	84	1155	84	0	

Table 2. Effective number of bits of each argument which Γ depends on. Note that the functions having the same number of effective bits do not necessarily have the same involved variables.

$(c_{1,4}, c_{1,5}, c_{1,6}, c_{2,0}, c_{2,1}, c_{2,5}, c_{2,6})$ of the ciphertext. The ANF of this function is:

$$\begin{aligned}\Gamma_{3,0}^{\{18\}}[8] = 1 &+ k_{0,0}k_{0,1} + k_{0,0}k_{0,1}k_{2,0} + k_{2,0}k_{2,1} + k_{0,0}c_{1,4} + \\ &k_{0,0}k_{2,0}c_{1,4} + k_{0,0}k_{1,0}c_{1,5} + k_{0,0}k_{1,0}k_{2,0}c_{1,5} + \\ &k_{0,0}c_{1,6} + k_{0,0}k_{2,0}c_{1,6} + k_{2,0}c_{2,0} + k_{0,0}k_{2,0}c_{2,0} + \\ &k_{2,0}c_{2,1} + c_{2,0}c_{2,1} + k_{0,0}c_{2,0}c_{2,1} + k_{2,0}c_{2,0}c_{2,1} + \\ &k_{0,0}k_{2,0}c_{2,0}c_{2,1} + k_{1,0}k_{2,0}c_{2,5} + k_{2,0}c_{2,6}.\end{aligned} \tag{7}$$

This equation can be seen as a system of $2^{t_w} = 128$ equations versus $t_k = 5$ unknowns. Our analysis of this function shows that only 48 of the equations are independent which on average can give 3.5 bits of information about the five unknown bits (2 bits of information for 25% of the keys and 4 bits for the remaining 75% of the keys).

Example 3. Take the relation $\Gamma_{7,0}^{\{33,34\}}[16](W, \mathsf{K})$ from Table 2. This particular function depends on $t_k = 12$ key bits and on $t_w = 33$ ciphertext bits. Our analysis of this function shows that on average about 2.41 bits of information about the 12 key bits can be achieved (10 bits of information for 12.5% of the keys, 3 bits for 25% of the keys and 0.67 bits about the remaining 62.5% of the keys).

Example 4. Take the relation $\Gamma_{7,0}^{\{38,39\}}[16](W, \mathsf{K})$ from Table 2. This particular function depends on $t_k = 12$ key bits and on $t_w = 30$ ciphertext bits. Our analysis of this function shows that on average about 1.94 bits of information about the 12 key bits can be achieved (10 bits of information for 12.5% of the keys, 3 bits for another 12.5% of the keys and 0.42 bits for the remaining 75% of the keys).

Example 5. Take the relation $\Gamma_{7,0}^{\{34\}}[16](W, \mathsf{K})$ from Table 2. This particular function depends on $t_k = 16$ key bits and on $t_w = 52$ ciphertext bits. Our analysis of this function shows that on average about 5.625 bits of information about the 16 key bits can be achieved (13 bits of information for 25% of the keys, 11 bits for 12.5% of the keys, 4 bits for another 12.5% of the keys, and 1 bit for the remaining 50% of the keys).

For larger values of i we expect Γ to fit better the ideal situation of Proposition 1. Therefore, we give the following claim about the security of the 64-bit version of Klimov-Shamir's proposal.

Proposition 2. *We expect each of the functions* $\Gamma_{31,0}^{\{129,130\}}[64]$ *and* $\Gamma_{31,0}^{\{150,151\}}[64]$ *to reveal a large amount of information about the corresponding* $t_k = 84$ *involved key bits for a non-negligible fraction of the keys. The required computational time is* $31 \times 84 \times 2^{2+84} \approx 2^{92.8}$.

In [7] the bits of the set U were called *weak IV bits*. With the same terminology, here we call them *weak ciphertext bits*. How to find these weak bits was raised as an open question in [7]. In the next section we present a systematic procedure to find weak ciphertext bits, with the consequence of improving Proposition 2.

6 Towards a Systematic Approach to Find Weak Ciphertext Bits

The idea is to start with a set U and extend it gradually. At each step we examine all the ciphertext bits which Γ^U depends on, to choose an extended U for the next step which results in a Γ which depends on the least number of key bits. Table 3 shows our simulation results by starting from function $\Gamma_{1,0}^{\{41\}}[64]$ from Table 2 which effectively depends on $t_k = 90$ extended key bits and $t_w = 51$ ciphertext bits. Similar to Proposition 2, one expects each of the functions $\Gamma_{1,0}^{U}[64]$ in Table 3 to reveal a large amount of information about the corresponding t_k involved extended key bits (including t'_k effective key bits) for a non-negligible fraction of the keys in time $t_k 2^{l+t_k}$, as indicated in the last column. In particular by starting from the function in the bottom of Table 3, (the promised large amount of information about) the involved $t'_k = 12$ key bits and $t_s = 33$ internal state bits can be gained in time $2^{69.5}$ (for a non-negligible fraction of the keys). Notice, that once we have the correct value for the unknown extended key for some function in Table 3, those of the previous function can be recovered by little effort. Therefore we present the following proposition.

Proposition 3. *We expect that by starting from* $\Gamma_{1,0}^{\{1-9,32-41\}}[64]$ *and going backwards to* $\Gamma_{1,0}^{\{41\}}[64]$ *as indicated in Table 3, a large amount of information about the involved* $t_k = 90$ *unknown bits (including* $t'_k = 30$ *effective key bits) is revealed for a non-negligible fraction of the keys in time* $2^{69.5}$.

Remark 2. By combining the results of different functions Γ one can get better results. Finding an optimal combination demands patience and detailed examination of different Γ's. We make this statement clearer by an example as follows. Detailed analysis of $\Gamma_{31,0}^{\{129,130\}}[64]$ and $\Gamma_{31,0}^{\{150,151\}}[64]$ shows that the key bits which they depend on are $\{0-27, 64-90, 128-156\}$ and $\{0-28, 64-90, 128-155\}$, respectively. These two functions have respectively 27 and 28 bits in common with the 30 key bits $\{0-19, 21-30\}$ involved in $\Gamma_{1,0}^{\{41\}}[64]$. They also include the key bits $\{0, 32, 64\}$ for which 1.75 information can be easily gained according to Ex. 1. Taking it altogether it can be said that a large amount of information about the 88 key bits $\{0-30, 32, 64-90, 128-156\}$ can be achieved in time $2^{69.5}$ with a non-negligible probability.

7 Conclusion

In this work we proposed a new analysis method for self-synchronizing stream ciphers which was applied to Klimov-Shamir's example of a construction of a T-function based self-synchronizing stream cipher. We did not fully break this proposal but the strong key leakage demonstrated by our results makes us believe a total break is not out of reach. In future design of self-synchronizing stream ciphers one has to take into account and counter potential key leakage.

Acknowledgement: We would like to thank Martijn Stam for his helpful editorial comments.

References

1. J.-Ph. Aumasson, S. Fischer, S. Khazaei, W. Meier and C. Rechberger. New Features of Latin Dances: Analysis of Salsa, ChaCha, and Rumba. In *Proceedings of Fast Software Encryption FSE 2008, LNCS 5086, pp. 470-488, 2008.*
2. E. Biham and E. Shamir. Differential Cryptanalysis of DES-like Cryptosystems. In *In Proceedings of CRYPTO'90, LNCS 537, pp. 2-21, 1991.*
3. J. Daemen, L. R. Knudsen, V. Rijmen. The Block Cipher Square. In *In Proceedings of Fast Software Encryption FSE'97, LNCS 1267, pp. 149-165, 1997.*
4. I. Dinur and A. Shamir. Cube attacks on tweakable black box polynomials. In *Cryptology ePrint Archive, Report 385, 2008.*
5. H. Englund, T. Johansson, and M. S. Turan. A Framework for Chosen IV Statistical Analysis of Stream Ciphers. In *Proceedings of INDOCRYPT 2007, LNCS 4859, pp. 268-281, 2007.*
6. eSTREAM - The ECRYPT Stream Cipher Project - see `www.ecrypt.eu.org/stream`.
7. S. Fischer, S. Khazaei and W. Meier. Chosen IV Statistical Analysis for Key Recovery Attacks on Stream Ciphers In *Proceedings of AFRICACRYPT 2008, LNCS 5023, pp. 236-245, 2008.*
8. A. Klimov and A. Shamir. New Applications of T-Functions in Block Ciphers and Hash Functions. In *Proceedings of Fast Software Encryption (FSE 2005), LNCS 3557, pp. 18-31, 2005.*
9. L. R. Knudsen. Truncated and Higher Order Differentials. In *In Proceedings of Fast Software Encryption FSE'94, LNCS 2365, pp. 196-211, 1994.*

10. L. R. Knudsen and D. Wagner. Integral Cryptanalysis. In *In Proceedings of Fast Software Encryption FSE'02, LNCS 2365, pp. 112-127, 2002*.
11. S. O'Neil. Algebraic Structure Defectoscopy. In *Cryptology ePrint Archive, Report 2007/378*. See also `http://www.defectoscopy.com`.
12. M. Vielhaber. Breaking ONE.FIVIUM by AIDA an Algebraic IV Differential Attack. In *Cryptology ePrint Archive, Report 2007/413*.

U	K	W	t_k	t_w	t'_k	t_s	Time
$\{41\}$	$\{0-19, 21-30, 384-414, 449-467, 469-478\}$	$\{0-9, 22-40, 42-63\}$	90	51	30	60	$2^{97.5}$
$\{9, 41\}$	$\{0-19, 21-29, 384-413, 449-467, 469-477\}$	$\{0-8, 22-40, 42-63\}$	87	50	29	58	$2^{95.4}$
$\{8, 9, 41\}$	$\{0-19, 21-28, 384-412, 449-467, 469-476\}$	$\{0-7, 22-40, 42-63\}$	84	49	28	56	$2^{93.4}$
$\{7-9, 41\}$	$\{0-19, 21-27, 384-411, 449-467, 469-475\}$	$\{0-6, 22-40, 42-63\}$	81	48	27	54	$2^{91.3}$
$\{6-9, 41\}$	$\{0-19, 21-26, 384-410, 449-467, 469-474\}$	$\{0-5, 22-40, 42-63\}$	78	47	26	52	$2^{89.3}$
$\vdots$							$\vdots$
$\{1-9, 41\}$	$\{0-19, 21, 384-405, 449-467, 469\}$	$\{0, 22-40, 42-63\}$	63	42	21	42	$2^{79.0}$
$\{1-9, 40, 41\}$	$\{0-18, 21, 384-405, 449-466, 469\}$	$\{0, 22-39, 42-63\}$	61	41	20	41	$2^{77.9}$
$\{1-9, 39-41\}$	$\{0-17, 21, 384-405, 449-465, 469\}$	$\{0, 22-38, 42-63\}$	59	40	19	40	$2^{76.9}$
$\vdots$							$\vdots$
$\{1-9, 34-41\}$	$\{0-12, 21, 384-405, 449-460, 469\}$	$\{0, 22-33, 42-63\}$	49	35	14	35	$2^{71.7}$
$\{1-9, 33-41\}$	$\{0-11, 21, 384-405, 449-459, 469\}$	$\{0, 22-32, 42-63\}$	47	34	13	34	$2^{70.5}$
$\{1-9, 32-41\}$	$\{0-10, 21, 384-405, 449-458, 469\}$	$\{0, 22-31, 42-63\}$	45	33	12	33	$2^{69.5}$

Table 3. Finding weak ciphertext bits in a systematic way (for $\Gamma_{1,0}[64]$).