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 input parameter 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 at AfricaCrypt'08 [4] (for dealing with this kind of problems) to the proposed T-function based self-synchronizing stream cipher by Klimov and Shamir at FSE'05 [5] and show how to deduce some non-trivial information about the key. We also open a new window for answering an open question raised in [4].

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 thanks to the eStream [3] project. It has relatively been proved that it is actually possible to design strong synchronous stream ciphers. Though, this is not the case for self-synchronizing ones. Many self-synchronizing stream ciphers have shown not to bear cryptanalysis attacks and there remain very few of them unbroken yet. Despite of numerous works on self-synchronizing stream ciphers in the literature, there is not yet a good understanding of their design and cryptanalysis methods. 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 can be controlled by the attacker while the input K is an unknown parameter to her (called the extended key which 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 arising often 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. In [4] a method was developed 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 known as *weak 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 [5] and use the framework from [4] to deduce some information about the key bits through some striking relations. Finding the weak bits from C was raised as a crucial open question in [4] 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 rest of the paper is organized as follows. In Sect. 2 we review the method from [4]. In Sect. 3 we describe the self-synchronizing stream ciphers and explain how to derive keyed functions $F(C, \mathsf{K})$ which suit the framework from [4]. Sect. 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 Sect. 5 and 6. Sect. 6 also includes our new direction of finding weak bits in a systematic way.

2 An Approach for Key Recovery on a Keyed Function

In this section we review the framework from [4] which was inspired by results from [1, 2, 6]. Let $\mathbb{B} = \{0, 1\}$ and $F : \mathbb{B}^m \times \mathbb{B}^n \to \mathbb{B}$ be a keyed Boolean function which maps the n-bit key K and the m-bit parameter C into the output bit $z = F(\mathsf{K}, C)$. An oracle chooses a random key K and returns back $z = F(\mathsf{K}, C)$ to an adversary for any chosen C of adversary's choice. The goal of the adversary is to recover the key K which is unknown to her by dealing with the oracle. The idea of [4] is to derive a (hopefully weaker) function $\varGamma : \mathbb{B}^{m-l} \times \mathbb{B}^n \to \mathbb{B}$ from the algebraic expansion of F by making a partition of the m-bit input parameter C according to $C = (U, W)$ with l-bit segment U and $(m - l)$-bit segment W. This is done by considering the algebraic normal form $F(C, \mathsf{K}) = \sum_\alpha \varGamma_\alpha(W, \mathsf{K})U^\alpha$ where $U^\alpha = u_0^{\alpha_0} u_1^{\alpha_1} \dots u_{l-1}^{\alpha_{l-1}}$ for the multi-index $\alpha = (\alpha_0, \dots, \alpha_{l-1})$. In other words, $\varGamma_\alpha(W, \mathsf{K})$ is the coefficient of U^α in the algebraic expansion of F. For every $\alpha \in \{0, 1\}^l$, the function $\varGamma_\alpha(W, \mathsf{K})$ can serve as a function $\varGamma$ derived from F, however, we only focus on the maximum degree monomial, *i.e.* $\alpha = (1, \dots, 1)$ as it is usually more useful, see [2, 4], hence dropping the subscript α. Moreover we have $\varGamma(W, \mathsf{K}) = \bigoplus_{U \in \mathbb{B}^l} F((U, W), \mathsf{K})$. In [4] the bits of U were called *weak IV bits* (C was the Initial Vector of a stream cipher) which can be adapted according to each context.

Remark 1. Note that for $l = 0$ we have $U = \emptyset$ and $W = C$ and hence $\varGamma = F$, that is we are analyzing the original function. For $l = 1$ (i.e., taking $U = \{u\}$ and $W = C \setminus \{u\}$) we are considering a variant of (truncated) differential cryptanalysis, that is we have $\varGamma(W, \mathsf{K}) = F(C, \mathsf{K}) \oplus F(C \oplus \varDelta C, \mathsf{K})$ where $\varDelta C$ is an m-bit vector which is zero in all bit positions except the u-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 [4] and (higher order) differential cryptanalysis seems to be as follows: If we only focus on the function $\varGamma(W, \mathsf{K})$ that computes the coefficient of the maximum degree monomial, the output of $\varGamma(W, \mathsf{K})$ is computed as the sum (mod 2) of all outputs $F(C, \mathsf{K})$, where $C = (U, W)$ is as above. This is what is also done in (higher order) differential cryptanalysis. However, in applications of the framework in

[4], the values for W are often chosen adaptively, so as to get a stronger deviation from randomness, whereas in most applications of (higher order) differential cryptanalysis, one merely considers differences in inputs rather than specific input values.

In order to have an effective attack, Γ must have many *(probabilistic) neutral* key bits. A key bit is called neutral if Γ does not depend on it and more generally a key bit is referred to as probabilistic neutral if Γ weakly depends on it. Using the following definition one can consider all the key bits with absolute *neutrality measure* greater than a threshold γ as probabilistic neutral key bits.

Definition 1. *[1, 4] The neutrality measure of the i-th key bit with respect to the function $\Gamma(W, \mathsf{K})$ is defined as γ_i, where $\frac{1}{2}(1+\gamma_i)$ is the probability (over all K and W) that complementing the i-th key bit does not change the output of $\Gamma(W, \mathsf{K})$.*

3 Self-synchronizing Stream Ciphers

A self-synchronizing stream cipher is built on an output filter $\mathcal{O} : \mathcal{K} \times \mathcal{S} \rightarrow \mathcal{M}$ and a self-synchronizing state update function (see Definition 2) $\mathcal{U} : \mathcal{M} \times \mathcal{K} \times \mathcal{S} \rightarrow \mathcal{M}$, 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 2. *[5] (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 3. *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 synchronous stream ciphers: *distinguishing attacks* and *key recovery attacks*[3]. The strongest scenario in which these attacks can be applied

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

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 $\{\mathcal{H}_i : \mathcal{M}^i \times \mathcal{K} \times \mathcal{S} \rightarrow \mathcal{M} | i = 1, 2, \ldots r\}$, 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 $\mathcal{G}_i : \mathcal{M}^i \times \mathcal{K} \times \mathcal{S} \rightarrow \mathcal{S}$ 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 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 more clear 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 $\mathcal{F}_i(C, \mathsf{K}) = \mathcal{H}_i(c_1, \ldots, c_i, K, \mathsf{S}^\star) : \mathcal{M}^i \times (\mathcal{K} \times \mathcal{S}) \rightarrow \mathcal{M}$ where $C = (c_1, \ldots, c_i)$.

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

In [5], Shamir and Klimov used multiword T-functions for a general methodology how 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:

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

and

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

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 [5] 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 2. In [5] 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 probably 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. They did not also 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 Binary field and the set of ω-bit words respectively. Following the general model of analysis of self-synchronizing stream ciphers in Sect. 3.1, we focus on the family of functions $\mathcal{F}_i(C, \mathsf{K}) : \mathbb{B}^i_\omega \times \mathbb{B}^8_\omega \to \mathbb{B}_\omega$, $i = 1, 2, \ldots, r$ where $C = (c_1, \ldots, c_i)$ and $\mathsf{K} = (K, \mathsf{S}^\star) = (k_0, k_1, k_2, k_3, k_4, \mathsf{s}^\star_0, \mathsf{s}^\star_1, \mathsf{s}^\star_2)$. 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. Therefor any vector $A = (a_0, a_1, ..., a_{p-1}) \in \mathbb{B}^p_\omega$ could be also treated as a vector in $\mathbb{B}^{p\times\omega}$ where the $(i\omega{+}j)$-th bit of A is 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 [4] explained in Sect. 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 $\Gamma^U_{i,j} : \mathbb{B}^{i\omega - l} \times \mathbb{B}^{8\omega} \to \mathbb{B}$. Whenever there is no ambiguity we drop the superscript or the subscripts. We may also use $\Gamma^U_{i,j}[\omega]$ in some cases to remind the word-size. We are now equipped to give our simulation results.

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 Γ 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 $\Gamma^{\{3\}}_{1,0}[8] = 1 + k_{2,0}$ and $\Gamma^{\{6,7,8,9,10\}}_{1,0}[16] = 1 + k_{0,0}$ involving only one key bit. For $\omega = 64$, the three relations in Table 1 give 1.75 bits of information about $(k_{0,0}, k_{1,0}, k_{2,0})$.

w	i	j	U	$\Gamma^U_{i,j}[\omega]$
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 function $\Gamma^U_{i,j}[\omega](W, \mathsf{K})$ reveals that a lot of them effectively depend on 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 Γ effectively depends on. In addition let t_k out of t_{K} bits come from K and the remaining $t_s = t_{\mathsf{K}} - t_k$ bits from $\mathsf{S}^\star$ (remember $\mathsf{K} = (K, \mathsf{S}^\star)$). Table 2 shows these values for some of these functions.

For a moment consider an ideal case in which a function $\Gamma(W, \mathsf{K})$ depending on t_W-bit input W and t_{K}-bit input K mixes its input bits well enough to be considered as a random Boolean function with $t_W + t_{\mathsf{K}}$ input bits. If we consider an adversary-oracle interaction (see Sect. 2 and think of F as Γ) through function Γ, assuming $\log_2 t_{\mathsf{K}} \ll t_W$ the t_{K} key bits can be recovered by sending $\mathcal{O}(t_{\mathsf{K}})$ queries to the oracle. This intuitively comes from the fact that asking one query reveals one bit of information about the key. More precisely if the adversary asks the oracle $t_{\mathsf{K}} + \beta$ queries, then the probability that only the unknown chosen key by the oracle (*i.e.* the correct candidate) satisfies these queries while all the remaining $2^{t_{\mathsf{K}}} - 1$ keys fail to satisfy all queries is $(1 - 2^{-(t_{\mathsf{K}}+\beta)})^{2^{t_{\mathsf{K}}}-1} \approx 1 - e^{-2^{-\beta}}$, and hence for $\beta = 10$ with probability at least

w	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	35	0	
64	31	0	$\{64\}$	89	1274	89	0	
64	31	0	$\{130\}$	88	1243	89	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.

$1 - 10^{-3}$ the correct key can be recovered uniquely. The astute reader notices that the required time is $\mathcal{O}(t_{\mathsf{K}} 2^{t_{\mathsf{K}}})$. Returning back to our results from Table 2 we can give the following proposition.

Proposition 1. *If the functions $\Gamma_{i,j}^{U}$ were random-looking enough, one would expect to recover the t_{K} unknown bits of the extended key in time $i t_{\mathsf{K}} 2^{l+t_{\mathsf{K}}}$ where $l = ||U||$.*

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 Γ from $\mathcal{F}_i$ needs 2^l evaluations of $\mathcal{F}_i$ (remember $\Gamma_{i,j}^{U}(W, \mathsf{K}) = \bigoplus_{U \in \mathbb{B}^l} \mathcal{F}_{i,j}((U, W), \mathsf{K})$) 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 sub key candidate is uniquely determined. Yet some information about the sub key is achieved.

Example 2. Take the relation $\Gamma_{3,0}^{\{18\}}[8](W, \mathsf{K})$ from Table 2. This particular function depends on $t_{\mathsf{K}} = 5$ 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

$(c_{1,4}, c_{1,5}, c_{1,6}, c_{2,0}, c_{2,1}, c_{2,5}, c_{2,6})$ 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 $t_{\mathsf{K}} = 5$ 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 $t_{\mathsf{K}} = 12$ 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 $t_{\mathsf{K}} = 12$ 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 $t_{\mathsf{K}} = 16$ 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* $t_{\mathsf{K}} = 84$ *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 [4] the bits of the set U were called *weak IV bits*. With the same terminology, we call them *weak ciphertext bits*. How to find these weak bits was raised as an open question in [4]. 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 $t_{\mathsf{K}} = 90$ 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 $t_{\mathsf{K}}2^{l+t_{\mathsf{K}}}$, 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* $t_{\mathsf{K}} = 90$ *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 3. 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.

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. 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*.
3. eSTREAM - The ECRYPT Stream Cipher Project - see `www.ecrypt.eu.org/stream`.
4. 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*.
5. 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.*
6. S. O'Neil. Algebraic Structure Defectoscopy. In *Cryptology ePrint Archive, Report 2007/378*. See also `http://www.defectoscopy.com`.

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 $\mathit{\Gamma}_{1,0}[64]$).