Chosen-Ciphertext Attacks against MOSQUITO

Antoine Joux[1,3] and Frédéric Muller[2]

[1] DGA
[2] HSBC-France
Frederic.Muller@m4x.org
[3] Université de Versailles-Saint-Quentin, France
Antoine.Joux@m4x.org

Abstract. Self-Synchronizing Stream Ciphers (SSSC) are a particular class of symmetric encryption algorithms, such that the resynchronization is automatic, in case of error during the transmission of the ciphertext.

In this paper, we extend the scope of chosen-ciphertext attacks against SSSC. Previous work in this area include the cryptanalysis of dedicated constructions, like KNOT, HBB or SSS. We go further to break the last standing dedicated design of SSSC, *i.e.* the ECRYPT proposal MOSQUITO. Our attack costs about 2^{70} computation steps, while a 96-bit security level was expected. It also applies to $\Gamma\Upsilon$ (an ancestor of MOSQUITO) therefore the only secure remaining SSSC are block-cipher-based constructions.

1 Introduction

Symmetric encryption algorithms are generally split in two parts : stream ciphers and block ciphers. On the one hand, stream ciphers manipulate the plaintext by **short packets** of data (for instance bit per bit), using a **time-dependent transform**. Typically the output of a PRNG (Pseudo-Random Number Generator) is XORed to the plaintext. On the other hand, block ciphers manipulate the plaintext by **larger packets** of data (typically 128 bits for AES [17]) using a **fixed transform**.

Self-Synchronizing Stream Ciphers (SSSC) are a special primitive : they are often considered as a simple subclass of stream ciphers, but there are also some similarities with block ciphers. Their main property is that, when some error occurs in the transmission of the ciphertext, the decryption algorithm eventually corrects it, after a short sequence of incorrectly decrypted bits. Hence a SSSC achieves the features of an encryption algorithm and resynchronization after transmission errors in one single primitive. They are suitable in situations where encryption is needed, but no additional bandwidth is possible for error-correction (see Maurer's paper for a nice survey on the use of SSSC [14]). In practice, few SSSC's are actually used and it is not clear that such algorithms will be important in the future [3]. However, from a theoretical point of view, it is a very challenging subject, because no dedicated SSSC has yet been built, that resists all known attacks. Some block-cipher-based constructions are possible, but it would be nice to have a dedicated solution, that is both secure and efficient.

To guarantee the automatic resynchronization, it is a requirement that the encryption of the i-th bit of the plaintext depends only on the key and a **small part of the previous ciphertext bits**. We denote by K the secret key, $\{P_i\}_{i\geq 0}$ the plaintext bits and $\{C_i\}_{i\geq 0}$ the ciphertext bits. Typically, a SSSC is such that :

$$C_i = P_i \oplus F(K, C_{i-1}, \ldots, C_{i-T})$$

for some function F and some integer T which is called the **memory** of the SSSC. It is clear that such an encryption scheme is invertible. Besides, if an error occurs in the ciphertext transmission, the decryption algorithm automatically resynchronizes after transmitting T correct ciphertext bits. It is possible to realize a SSSC with a dedicated design, or with a block cipher in an appropriate mode, like the Cipher FeedBack (CFB) mode [9].

From a bitwise point of view, SSSC operate as stream ciphers, since every plaintext bit can be encrypted separately using the time-dependent transform $x \longrightarrow x \oplus F(K, C_{i-1}, \ldots, C_{i-T})$. On the other hand, looking at the ciphertext, a fixed-transform is applied to each T-bit block, which is more similar to a block cipher. The difference is that a block cipher is an invertible mapping on n-bit inputs, while the F function is a n-to-1 mapping. ¿From the designer's point of view, a dedicated SSSC is often looked at as a special mode of operation for stream ciphers with ciphertext feedback (see HBB [20] among others), while cryptanalysis methods are often related to the field of block cipher.

First, we review the usual design methods for SSSC. Secondly, we review the existing attacks against dedicated SSSC, like KNOT, HBB or SSS. Finally, we extend the scope of these attacks, in order to break the only standing dedicated design, MOSQUITO. The complexity of our attack is 2^{70} steps, while the expected security level was 96 bits. To summarize, we observe that only block-cipher-based constructions remain secure in this area.

2 Design Methods for SSSC

Following the terminology introduced previously, all SSSC operate by

$$C_i = P_i \oplus F(K, C_{i-1}, \ldots, C_{i-T})$$

The difference between the designs lie in the way F is built and in the value of T. Figure 1 presents the general description of a SSSC.

2.1 Block-cipher-based constructions

A typical solution is to start from a block cipher E_K that operates on n bit inputs, using a secret key K. Then, one builds a SSSC with memory of n bits by :

$$F(K, C_{i-1}, \ldots, C_{i-n}) = E_K(C_{i-1}, \ldots, C_{i-n})\&1;$$

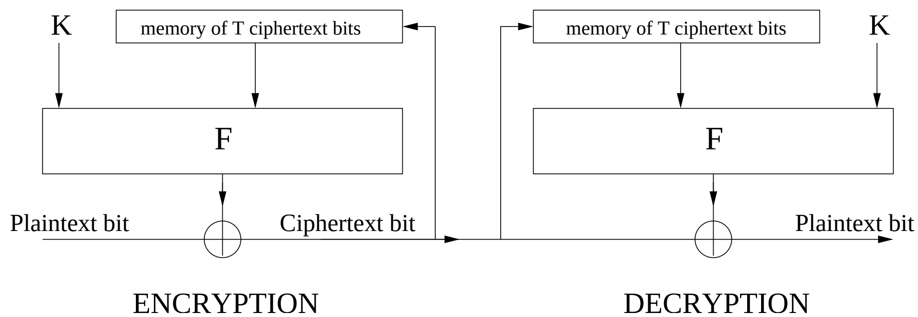

Fig. 1. General description of a SSSC

In other terms, E is applied and only the Least Significant Bit (LSB) of the output is kept. This is called the Cipher FeedBack (CFB) mode, with 1-bit feedback [9]. More generally, the CFB mode can be extended to t-bit feedback for any t between 1 and n (see Figure 2), but only the 1-bit version is self-synchronizing. The CFB mode with 1-bit feedback is very inefficient, as one full application of E must be processed to encrypt one plaintext bit. However, there exists more efficient alternatives, like the OCFB mode of operation [1].

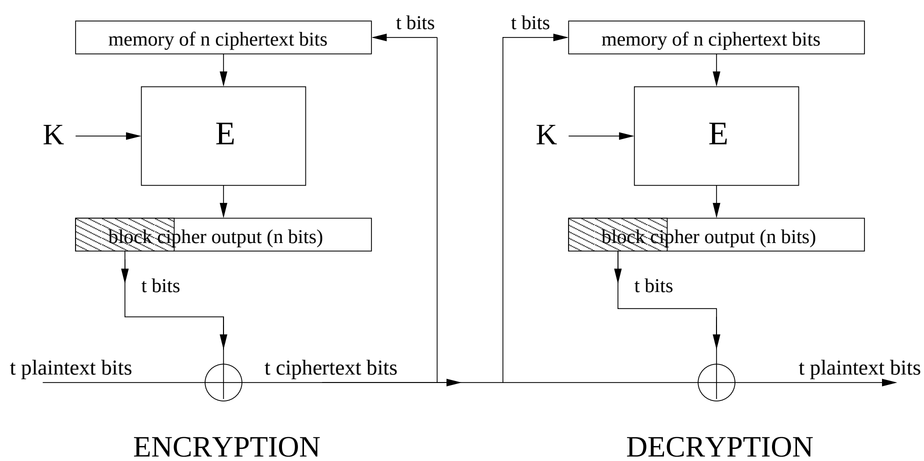

Fig. 2. The CFB mode of operation with t-bit feedback

There is no generic attack against CFB or OCFB, provided the underlying block cipher is secure. Preneel *et al.* pointed out some possible attacks when one reduces the number of rounds of the block cipher to improve the efficiency of the CFB mode [18]. Actually, there exists a security proof for the CFB mode with

n-bit feedback, against chosen plaintext attacks [10]. It is also widely believed that the CFB mode with t-bit feedback is secure for any t, although no generic security proof has yet been published.

2.2 Dedicated constructions

Dedicated constructions of SSSC were first introduced by Maurer [14]. Later, Daemen *et al.* reconsidered the design of dedicated SSSC, from a practical perspective [5]. They pointed out that it was not very efficient to recompute the whole function F for each new ciphertext bit introduced. Therefore, they suggested to split the design of the SSSC into two parts :

– An **updatable part** Q which is generally a register with an internal state of size m bits. The state of the register at time i, denoted Q_i, should depend on the last T ciphertext bits, and possibly the key :

$$Q_i = G(K, C_{i-1}, \dots, C_{i-T})$$

  Then **an update function** is specified, in order to compute efficiently Q_{i+1} from Q_i and C_i. The function G is never actually computed in practice, since the register Q is generally initialized with a m-bit constant, and then the update function is applied as many times as necessary. Note that the memory T and the register length m are not necessary equal, however it is necessary that $m \geq T$, in order to store enough information in the register.
– An **output filter** f which takes the state of the register Q_i and computes the output bit, :

$$F(K, C_{i-1}, \dots, C_{i-T}) = f(K, Q_i)$$

  This filter f often looks like a "light" block cipher.

Figure 3 represents the general framework of such constructions. To guarantee the self-synchronization, it is necessary that the "old" ciphertext bits are "forgotten" after T updates. This constraint is often satisfied using a shift register-oriented design (this is the case for SSS and for the KNOT-MOSQUITO family). Concerning the role of the secret key, at least f or G must use K as input, but it is rare that both do. Block-cipher-based constructions are the limit case of this framework, with a very simple (unkeyed) shift register as the updatable part, and a complicated (keyed) block cipher as the output filter. Dedicated constructions try to reach a better trade-off with a more complicated (non-linear) updatable part, but a lighter output filter.

2.3 The KNOT-MOSQUITO family

An interesting family of dedicated SSSC is the "KNOT-MOSQUITO family". KNOT was first proposed as an example of efficient dedicated design by Daemen *et al.* in their paper of 1992 that dealt with SSSC in a more general perspective [5]. In 1995, Daemen discovered a statistical imbalance in the output of

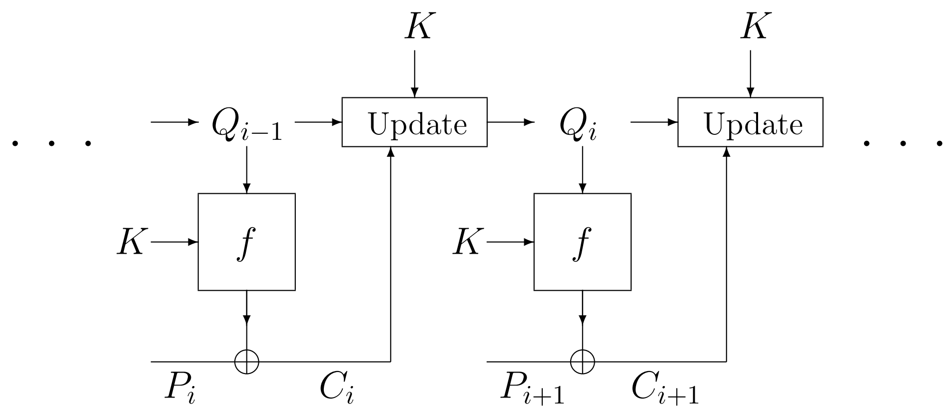

Fig. 3. General Framework of Dedicated SSSC

KNOT [4], which was further investigated later [12]. This weakness of the output filter motivated a switch to a tweaked version of KNOT called $\Gamma\Upsilon$ which was proposed in Daemen's PhD thesis [4].

In 2003, a new weakness of KNOT was pointed out by Joux and Muller, which allows to recover the secret key with complexity of 2^{62} steps [12]. Their idea is to apply methods from differential cryptanalysis in order to detect internal collisions in the updatable register. This attack does not apply against $\Gamma\Upsilon$. Recently, as part of the eSTREAM competition for stream ciphers [8] launched by the European project ECRYPT, Daemen and Kitsos proposed a new self-synchronizing stream cipher called MOSQUITO [6]. It is a close variant of $\Gamma\Upsilon$ which was designed to avoid the security problems of KNOT.

All algorithms in this family follow the framework introduced in Section 2.2, with a memory of $T = 96$ bits and an updatable register Q of size $m = 128$ bits, which is non-linear and key-dependent. The type of register used in this family are also called **Conditional Complementing Shift Registers** (CCSR). See [5] or [6] for more details. The output filter is an unkeyed iterative construction, which gradually reduces the state from 128 bits to 1 bit, after 8 rounds[1]. The difference between the 3 algorithms in this family (KNOT, $\Gamma\Upsilon$ and MOSQUITO) lies in the way Q is updated, and in the details of the 8 rounds of the filter. All algorithms in the family are designed to use of **a 96-bit secret key**.

To summarize, the KNOT-MOSQUITO family is an interesting family of dedicated SSSC, since it is not a "tweaked" mode of operation of a stream cipher, like SSS or HBB. Therefore, it is very interesting from a research perspective. However, all the algorithms of the family are subject to differential attacks, as pointed out in Section 4.

[1] The authors mention 7 rounds, but it depends on whether the final XOR is counted as an 8-th round or not.

2.4 SSS

SSS (Self-Synchronizing Sober) is a new dedicated SSSC, submitted to the eSTREAM competition [19] by Rose *et al.* It belongs to the SOBER family of stream ciphers [11]. This family uses Linear Feedback Shift Registers (LFSR) operating in GF(2^n) with $n = 8, 16$ or 32. An output filter is applied to some cells from the LFSR, in order to extract a pseudo-random output. This generally relies on n-bit instructions as well, in order to obtain a software-efficient design. Additional functionalities (like authenticated encryption or self-synchronization) have been suggested in tweaked versions of the SOBER family. The general idea is to add an auxiliary input in the LFSR to introduce either the plaintext (for the integrity mode) or the ciphertext (for the self-synchronizing mode).

While few cryptanalysis results are known against the algorithms from the SOBER family in encryption mode, it is well known that tweaking a stream cipher in order to add integrity or self-synchronization completely modifies the cryptanalysis scenario [16]. Indeed, an attacker potentially gains the ability to control the content of the LFSR. A devastating attack against the integrity mode of SOBER-128 was described in 2004 by Watanabe *et al.* [24].

SSS was also broken by Daemen, Lano and Preneel, shortly after its publication [7]. They described a chosen ciphertext attack which allows to retrieve a key-dependent secret table. According to the designers of SSS, such attacks fall outside the threat model, but Daemen *et al.* argued that chosen ciphertext attacks are practical and that they are the standard way to evaluate SSSC.

2.5 HBB

HBB (Hiji-bij-bij) is a software-oriented stream cipher, proposed in 2003 by Sarkar [20]. It is a new construction, where the usual LFSRs have been replaced by cellular automata. In addition, an output filter operating on a 128-bit output is used. This filter has some similarities with a block cipher design (use of S-boxes and linear diffusion layers). The Basic (B) mode of operation of HBB is a traditional stream cipher, for which some attacks faster than brute-force have been published in 2005 [13, 15].

In addition, a Self-Synchronizing (SS) mode of operation for HBB was also proposed by Sarkar. It is based on a slight modification of the cipher, where the cellular automata is filled with ciphertext bits, instead of being evaluating autonomously. While the primitives are unchanged, this modification completely changes the cryptanalysis scenario. Joux and Muller showed a devastating key-recovery attack against this SS mode, which requires only 2^{12} bits of chosen ciphertext [13]. Basically, they exploited the weak differential properties of the output filter.

2.6 Other proposals

Another proposition of dedicated SSSC was made by Arnault and Berger [2], as part of their work on Feedback with Carry Shift Registers (FCSR). Their proposal was later broken [25] using a chosen ciphertext attack.

Like for SSS and HBB, it appears that building a SSSC by tweaking a conventional stream cipher is not a good idea. Many devastating attacks have been published : as soon as one considers chosen ciphertext attacks, the cryptanalyst no longer looks at the SSSC as a stream cipher, but instead he analyzes the G function directly (see Section 2.2). The properties of G can be considered under chosen input attacks : either differential properties are used (case of HBB or KNOT) or the possibility to guess and identify individually some portions of the key (case of SSS).

3 The Cryptanalysis Framework against SSSC

3.1 Chosen Ciphertext Attacks against SSSC

We first observe that the "natural way" to cryptanalyze a SSSC is by considering chosen ciphertext attacks. This is natural from a theoretical point of view, but also from a practical point of view, as already pointed out by Daemen *et al.* [7].

Theoretically, we are comparing dedicated SSSC with block ciphers in CFB mode. Block ciphers are built to resist both chosen plaintext and chosen ciphertext attacks, so it would be unfair to compare two algorithms that do not take into account the same attack scenarios. In addition, the existence of chosen ciphertext attacks generally reveals design weaknesses that could later be extend to much more realistic scenarios.

Moreover, there are still some scenarios where an attacker could have access to a decryption oracle. This is not necessarily much more difficult than accessing an encryption oracle. For instance, one could consider an active attacker that modifies the communication channel (in order to obtain the ciphertext he wants) and then observes the result of the decryption.

3.2 Chosen Plaintext Attacks against SSSC

In some attack scenarios against SSSC, chosen ciphertext attacks can even be turned into chosen plaintext attacks. Assume that the attacker needs to obtain a chosen ciphertext sequence equal to $(C_1, \ldots, C_i)$ in order to attack a SSSC. He can achieve it with chosen plaintext only, assuming an **adaptive chosen plaintext attack**.

In such a scenario, we assume that the attacker can reset the encryption algorithm to its initial state, at any point[2]. Then, he tries both value of the bit P_1 and resets the algorithm if the value of C_1 is not what he wants. Then, the process is repeated as long as necessary. On average, this requires about $i/2$ resets, where i is the length of the needed ciphertext sequence, $(C_1, \ldots, C_i)$.

This gives an example of a classical scenario, where there exists a bridge between chosen plaintext and chosen ciphertext attacks against SSSC. In the following, we focus on chosen ciphertext attacks.

[2] If he has access to several copies of the algorithm using the same key, a similar attack applies. The idea is to throw away a copy of the algorithm and use a new copy, instead of doing the reset.

4 Differential Attacks against MOSQUITO

4.1 A short overview of the design

Describing in details the MOSQUITO stream cipher is a long task, since a large number of equations should be stated. We invite the reader to check the details in the original specification [6]. Here, we only give a short overview of the design.

The register Q
As mentioned in Section 2.3, MOSQUITO uses a non-linear shift register as the updatable part : Q has a length of 128 bits. Its content is noted in the later as $(x_1, \ldots, x_{128})$. At time i, the content of Q is updated by introducing the i-th ciphertext bit, and also the secret key. More precisely, each bit x_j is updated by applying a simple boolean function $h_j : \{0,1\}^4 \longrightarrow \{0,1\}$, whose inputs are chosen among $(x_1, \ldots, x_{j-1})$, the key bits, and the introduced ciphertext bit.

One can observe that the propagation goes always in the direction of increasing indexes. This guarantess that the influence of "old" ciphertext bits eventually vanishes, which makes the resynchronization possible.

Actually, one could also express directly the content of register Q at time i has a function of the last 96 ciphertext bits and the key. However, this would lead to a very complicated expression. It is much clearer to describe Q by the update equations (*i.e.* the h_j). See [6] for the expression of all these equations.

The output filter f
After the update of register Q, the $(i+1)$-th ciphertext bit is obtained by XORing the $(i+1)$-th plaintext bit to the output of the filter f. This filter is a fixed, unkeyed transform applied to the state of the register Q. Therefore it is a boolean function from 128 bits to 1 bits.

In order to be computed efficiently, f can be written as the composition of 8 simple transforms (also called rounds) applied to internal states of decreasing size. For instance, the first round, noted φ_1, is applied to the content of Q (*i.e.* 128 bits), but its output size is only 53 bits. Therefore φ_1 is a transform from $\{0,1\}^{128}$ to $\{0,1\}^{53}$. After the 8 rounds, the final output is simply one bit. Like for the update of the register Q, each round can be represented by a small set of boolean functions from 4 bits to 1 bit. The reader should refer to [6] for the expression of all these equations.

4.2 Overview of the new attack

Our attack is similar to a differential cryptanalysis of a block cipher : we find differential characteristics for both parts of the cipher[3] :

[3] Regarding the mathematical tools, there are also relations with linear cryptanalysis since we are interested in small statistical deviations called bias, as it will appear later.

- First, we find a differential characteristic for the output filter f. In the case of MOSQUITO, f is an unkeyed 8-round transform. We are looking for a 128-bit difference Δ such that

$$f(Q_i \oplus \Delta) \oplus f(Q_i)$$

  is equal to 0 with probability $p = 0.5 \cdot (1+\varepsilon)$ and $|\varepsilon|$ as large as possible. In the case of KNOT, Joux and Muller [12] exploited $\Delta = 0$ which implied that $\varepsilon = 1$. Such collisions on Q could be obtained from two distinct ciphertexts (*i.e.* the function G was not injective). In the newer algorithms of the family ($\Gamma\Upsilon$ and MOSQUITO), this attack has been countered by making G injective. Therefore, the difference $\Delta = 0$ is not reachable, unless the ciphertexts are equal. We **extend the scope of Joux-Muller's attack to non-zero differences**.
- Secondly, we describe how to build two ciphertexts such that the two values of the state Q_i differ exactly by the previous difference Δ. This step will require to guess some portion of the key. We consider ε^{-2} such pairs, in order to detect an imbalance on the outputs of the filter f.

4.3 Differential characteristic for f

The filter f of MOSQUITO looks like a block cipher with 8 simple rounds, applied successively. Our analysis focuses on the differential properties of the first round transform, φ_1. Since its output size is smaller than its input size, it is not injective. Hence, we can expect to find an input difference Δ, such that, after the first round

$$\varphi_1(x) = \varphi_1(x \oplus \Delta)$$

with good probability. If such a collision occurs after applying φ_1, this will imply the equality of the outputs of f, since no new input is introduced in the 7 following rounds. As mentioned previously, each output bit of φ_1 is computed using a simple boolean function, written as :

$$\tau : (a, b, c, d) \longrightarrow a \oplus b \oplus c \cdot (d \oplus 1) \oplus 1$$

applied to 4 among the input bits of φ_1, noted $x = (x_1, \ldots, x_{128})$. Since $4 \cdot 53 = 212 < 2 \cdot 128 = 256$, we know that at least $256 - 212 = 44$ input bits are processed only once by the function τ. A quick analysis shows that this observation concerns :

$$(x_1, \ldots x_{17}), (x_{54}, \ldots, x_{60}), (x_{71}, \ldots, x_{75}), (x_{114}, \ldots, x_{128})$$

which are all used only once in φ_1. Some of them are only used as 3-rd or 4-th input bit of τ, so if we flip them, the output of τ may be unchanged. This observation concerns the set of bits :

$$S = \{x_1, \ldots x_{17}, x_{71}, \ldots, x_{75}\}$$

If we flip exactly one bit in this set S, the output of φ_1 is unchanged with probability 0.5. Consequently, the output of f is also unchanged with probability 0.75 (even when φ_1 is changed, the output bit can still be the same with probability 0.5). Hence, we found **differential characteristics on F with bias $\varepsilon = 0.5$ and such that the input difference is non-zero**.

4.4 Advanced differences

An advanced method consists in flipping two well-chosen input bits of φ_1. As an example, consider the output bit number 29 and 33 of φ_1. They can be computed by

$$x_{81} \oplus x_{65} \oplus x_{66} \cdot (x_{48} \oplus 1) \oplus 1$$
$$x_{80} \oplus x_{66} \oplus x_{65} \cdot (x_{49} \oplus 1) \oplus 1$$

The bits x_{65} and x_{66} are used nowhere else in φ_1. If an attacker flips these two bits simultaneously, there is a probability 0.25 that the output of φ_1 is unaffected (the condition is that $x_{48} = x_{49} = 0$). Hence the outputs of F are equal with probability 0.625, and we have a differential characteristic, with bias $\varepsilon = 0.25$ and such that the input difference is still non-zero (only the bits x_{65} and x_{66} are flipped). There exists other strategies to flip 2 or more bits, while keeping a reasonable bias. However, the difference we have just described will turn out to be the most useful.

4.5 Analysis of the updatable part

Our goal is now to build two ciphertexts sequences that map to two register states Q_i and Q'_i such that $\Delta = Q_i \oplus Q'_i$ is one of the "useful" differences identified in the previous section. We may need to repeat this process ε^{-2} times in order to actually observe the predicted bias. Finding 2 appropriate ciphertext sequences can be done using **an exhaustive search on an appropriate portion of the key**.

4.6 Using the difference on x_{17} only

We first describe an attack that targets the difference Δ_{17}, which is defined as difference equal 0 on every input bit of f, except x_{17}. At first glance, one could envisage to work with the difference Δ_1 which has the same differential behavior regarding f, however, the resulting complexity of the attack would be worse, as it will appear below.

We focus on the updatable register, to obtain register states that differ from Δ_{17} The state at time i is expressed as :

$$Q_i = G(K, C_{i-1}, \ldots, C_{i-96})$$

We denote the key bits by $K = (k_1, \ldots, k_{96})$. The crucial observation, that we refer to as **observation** $\mathcal{O}$ is that, for $1 \leq t \leq 88$, the value of bit x_t at time i depends only on the key bits $k_1, \ldots, k_t$ and the ciphertext bits $C_{i-1}, \ldots, C_{i-t}$.

The basic idea of our attack is to **guess only the** 79 **least significant bits of the key**. This guess splits the register Q into a left part for which we always know the internal values (thanks to observation $\mathcal{O}$) and a right part which generally remains unknown.

At any time, we can **control the differential behavior on the left part**. The only way we can have information about the right part is by letting the "natural" propagation from left to right in Q bring a zero-difference on the right part (resynchronization effect). We combine these ideas in the following. More precisely, our attack proceeds in three steps, represented in Figure 4 :

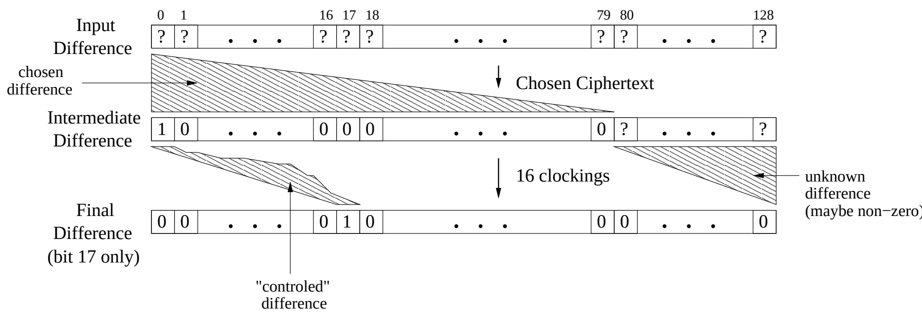

Fig. 4. The attack against MOSQUITO using Δ_{17}

- First, we guess the key bits $k_1, \ldots, k_{79}$. Because of the observation $\mathcal{O}$, we can easily determine the value of $V = (x_1, \ldots, x_{79})$ for any introduced ciphertext.
- Secondly, we pick at random two ciphertext sequences that yield the same V, except that the bit x_1 is flipped. This step is easy to achieve in practice : start from an arbitrary ciphertext sequence and compute the corresponding value of V. Then, flip bit x_1 and clock 79 times backward the register Q (see Figure 3). So we find which ciphertext sequence should be introduced in order to reach this modified state.
- Introduce 16 additional ciphertext bits and clock 16 times the register. The difference will always propagate from position 1 to position 17 during these 16 clockings. But, observation $\mathcal{O}$ guarantees that there is no difference afterward on bits $x_1, \ldots, x_{16}$. Moreover, we can **control the difference during these** 16 **extra clockings**, in order for the difference not to propagate further than position 17.

There are 2 ways to "control" the difference during these 16 extra clockings. The first method consists in specifying a differential path and writing down the boolean equations for this propagation to be satisfied. Here, about 20 such conditions are needed, which remains reasonable in practice. A second method is to test random ciphertexts, until a "good" one is found. Two tricks make this idea quite efficient : First, we use "early-abort" in order to quickly eliminate the "bad" candidates. Secondly, we observe that this propagation is almost independent on our guess of key bits, $k_{18}, \ldots, k_{79}$, so we can tell if a ciphertext is good or not, before guessing the whole key.

Both methods have been tested on a standard PC and are very efficient. The bottleneck of this attack is the 2-nd step, where we need to repeat 2^{79} the execution of 79 clockings of Q. For comparison, the basic step in an exhaustive search requires 96 clockings of Q. Overall, we need at least $\varepsilon^{-2} = 4$ such plaintext pairs, in order to detect the predicted bias. So we estimate the complexity to about 2^{81} steps (compared to 2^{96} for an exhaustive search).

4.7 Using the difference on x_{65} and x_{66}

The idea of this attack is essentially the same as the previous one, except that we guess only 66 key bits. Therefore, we can predict only the state of bit number 1 to 66 of the state Q_i. Then, we use 64 extra clockings and we hope to control the difference, during these advances in order to obtain finally a difference on cells number 65 and 66 only.

The approach based on specifying a differential path and writing the corresponding boolean conditions seems very painful. Therefore, we adopted a purely statistical approach : we tested random ciphertext sequences and used "early-abort" : as soon as the difference starts to spread to several bits, we discard the tested value.

Our implementation is very efficient and, for a given key guess, finds instantly two satisfying ciphertext sequences, *i.e.* such that the final difference is only located on the bits 65 and 66 of the state of Q. This needs to be repeated about ε^{-2} times, so the complexity of our attack is about

$$2^{66} \times 4^2 = 2^{70}$$

steps, compared to 2^{96} steps for an exhaustive search.

4.8 Some comments

There are several interesting comments to make about these two attacks. First the specific tricks needed to control the difference by specifying some set of sufficient boolean conditions are related to the techniques developed against hash functions like MD5 or SHA-1 by Wang *et al.* [21–23]. There could be further improvements to the cryptanalysis of the KNOT-MOSQUITO family, by looking in that area of research.

Secondly, our attacks also applies against $\Gamma\Upsilon$, since the underlying primitives (register Q and filter f) are the same. Concerning KNOT, there are slight differences in the primitives, but a similar attack should also apply. However the result by Joux and Muller, exploiting internal collisions for Q_i are slightly more efficient [12]. Actually, our attack can be viewed a generalization of this previous result.

We used 4 pairs of ciphertext sequences in Section 4.6 and 8 pairs of ciphertexts in Section 4.7. These are optimistic figures. It is well known that a small security margin is generally needed above ε^{-2} to actually detect a bias ε, with a small false alarm probability. It is not necessary that this probability is 2^{-96} here. Indeed, we have already guessed a large portion of the key, so 20 or 30 ciphertext pairs should probably be sufficient. Application of statistical tools is needed to evaluate exactly how many ciphertexts are needed.

Finally, an important point is that the data complexity of our attacks is very limited since we build the ciphertext sequences in an off-line computation (described in Section 4.6 and Section 4.7). The data complexity is roughly of $4 \times 96 \times 2 \simeq 2^{10}$ ciphertext bits in the first attack and $8 \times 96 \times 2 \simeq 2^{11}$ for the second attack.

5 Conclusion

All the dedicated Self-Synchronizing Stream Ciphers (SSSC) of the KNOT-MOSQUITO family are subject to differential chosen ciphertext attacks. Our results, combined with previous results on HBB, KNOT and SSS show that it is extremely difficult to design a SSSC resistant against chosen-ciphertext attacks.

Some designers argued [19] that chosen ciphertext attacks should fall outside the security model for SSSC. However, they are taken into account in block cipher-based constructions, and could be more realistic than expected. Even the authors of MOSQUITO [6] stressed out that dedicated SSSC should resist chosen ciphertext attacks.

Since no dedicated SSSC still stands, we believe that block-cipher-based constructions should now be favored if one needs a self-synchronizing algorithm, for practical purpose. An interesting direction would also be to see how much one can "lighten" an existing block cipher (AES for instance), in order to obtain a SSSC faster than the CFB (or OCFB) mode [1, 9].

References

1. A. Alkassar, A. Geraldy, B. Pfitzmann, and A.-R. Sadeghi. Optimized Self-Synchronizing Mode of Operation. In M. Matsui, editor, *Fast Software Encryption – 2001*, volume 2355 of *Lectures Notes in Computer Science*, pages 78–91. Springer, 2002.
2. F. Arnault and T. Berger. A new class of stream ciphers combining LFSR and FCSR architectures. In A. Menezes and P. Sarkar, editors, *Progress in Cryptology – INDOCRYPT'02*, volume 2551 of *Lectures Notes in Computer Science*, pages 22–33. Springer, 2002.
3. S. Babbage. Stream Ciphers: What Does the Industry Want ? In *State of the Art of Stream Ciphers* workshop (SASC'04), 2004.

Type of Attack	*Target*	*Complexity*	*Data*
Distinguisher [4]	KNOT	2^{18}	2^{18}
Key Recovery [12]	KNOT	2^{69}	$2^{36.6}$
Key Recovery [12]	KNOT	2^{62}	$2^{38.6}$
Key Recovery [7]	SSS	10 seconds	10 kBytes
Key Recovery [13]	HBB	2^{12}	2^{12}
Key Recovery (this paper)	MOSQUITO	2^{81}	2^{10}
Key Recovery (this paper)	MOSQUITO	2^{70}	2^{11}
Exhaustive Search	MOSQUITO	2^{96}	96 bits

Table 1. Summary of cryptanalysis results against some SSSC

4. J. Daemen. *Cipher and Hash Function Design. Strategies based on Linear and Differential Cryptanalysis*. PhD thesis, Katholieke Universiteit Leuven, march 1995. Chapter 9.
5. J. Daemen, R. Govaerts, and J. Vandewalle. A Practical Approach to the Design of High Speed Self-Synchronizing Stream Ciphers. In *Singapore ICCS/ISITA '92*, pages 279–283. IEEE, 1992.
6. J. Daemen and P. Kitsos. Submission to ECRYPT call for stream ciphers: the self-synchronizing stream cipher Mosquito. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/018, 2005. http://www.ecrypt.eu.org/stream.
7. J. Daemen, J. Lano, and B. Preneel. Chosen Ciphertext Attack on SSS. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/044, 2005. http://www.ecrypt.eu.org/stream.
8. eSTREAM - The ECRYPT Stream Cipher Project `http://www.ecrypt.eu.org/stream/`.
9. FIPS PUB 81. *DES Modes of Operation*, 1980.
10. P-A. Fouque, G. Martinet, and G. Poupard. Practical Symmetric On-Line Encryption. In T. Johansson, editor, *Fast Software Encryption – 2003*, volume 2887 of *Lectures Notes in Computer Science*, pages 362–375. Springer, 2003.
11. P. Hawkes and G. Rose. Primitive Specification and Supporting Documentation for SOBER-t32. In *First Open NESSIE Workshop*, 2000. Submission to NESSIE.
12. A. Joux and F. Muller. Loosening the KNOT. In T. Johansson, editor, *Fast Software Encryption – 2003*, volume 2887 of *Lectures Notes in Computer Science*, pages 87–99. Springer, 2003.
13. A. Joux and F. Muller. Two Attacks Against the HBB Stream Cipher. In H. Gilbert and H. Handschuh, editors, *Fast Software Encryption – 2005*, volume 3557 of *Lectures Notes in Computer Science*, pages 330–341. Springer, 2005.
14. U. Maurer. New Approaches to the Design of Self-Synchronizing Stream Ciphers. In D.W. Davies, editor, *Advances in Cryptology – Eurocrypt'91*, volume 547 of *Lectures Notes in Computer Science*, pages 458–471. Springer, 1991.
15. J. Mitra. A Near-Practical Attack against B mode of HBB. In *Advances in Cryptology - Asiacrypt'05*, 2005. To appear.

16. F. Muller. Differential Attacks and Stream Ciphers. In *State of the Art in Stream Ciphers*. ECRYPT Network of Excellence in Cryptology, 2004. Workshop Record.
17. National Institute of Standards and Technology (NIST). Advanced Encryption Standard (AES) FIPS Publication 197, November 2001. Available at `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`.
18. B. Preneel, M. Nuttin, R. Rijmen, and J. Buelens. Cryptanalysis of the CFB Mode of the DES with a Reduced Number of Rounds. In D.R. Stinson, editor, *Advances in Cryptology – Crypto'93*, volume 773 of *Lectures Notes in Computer Science*. Springer, 1993.
19. G. Rose, P. Hawkes, G. Paddon, and M. Wiggers de Vries. Primitive Specifications for SSS. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/028, 2005. http://www.ecrypt.eu.org/stream.
20. P. Sarkar. Hiji-Bij-Bij : A New Stream Cipher with a Self-Synchronizing Mode of Operation. In T. Johansson and S. Maitra, editors, *Progress in Cryptology – INDOCRYPT'03*, volume 2904 of *Lectures Notes in Computer Science*, pages 36–51. Springer, 2003.
21. X. Wang, Y. Yin, and H. Yu. Finding Collisions in the Full SHA1. In V. Shoup, editor, *Advances in Cryptology – Crypto'05*, volume 3621 of *Lectures Notes in Computer Science*, pages 17–36. Springer, 2005.
22. X. Wang and H. Yu. How to Break MD5 and Other Hash Functions. In R. Cramer, editor, *Advances in Cryptology – Eurocrypt'05*, volume 3494 of *Lectures Notes in Computer Science*, pages 19–35. Springer, 2005.
23. X. Wang, H. Yu, and Y. Yin. Efficient Collision Search Attacks on SHA0. In V. Shoup, editor, *Advances in Cryptology – Crypto'05*, volume 3621 of *Lectures Notes in Computer Science*, pages 1–16. Springer, 2005.
24. D. Watanabe and S. Furuya. A MAC Forgery Attack on SOBER-128. In B. Roy and W. Meier, editors, *Fast Software Encryption – 2004*, volume 3017 of *Lectures Notes in Computer Science*, pages 472–482. Springer, 2004.
25. B. Zhang, H. Wu, D. Feng, and F. Bao. Chosen Ciphertext Attack on a New Class of Self-Synchronizing Stream Ciphers. In A. Canteaut and K. Viswanathan, editors, *Progress in Cryptology – INDOCRYPT'04*, volume 3348 of *Lectures Notes in Computer Science*, pages 73–83. Springer, 2004.