Submission to ECRYPT call for stream ciphers: the self-synchronizing stream cipher Mosquito

Joan Daemen, STMicroelectronics Belgium, joan.daemen@st.com
Paris Kitsos, University of Patras, Greece, pkitsos@ee.upatras.gr

April 29, 2005

Abstract

In this document we specify and motivate the hardware-oriented self-synchronizing stream cipher Mosquito and the underlying cipher architecture and provide hardware implementation results. This stream cipher is a close variant of the cipher $\Upsilon\Gamma$, described in [2]. $\Upsilon\Gamma$ in its turn was a corrected version of the cipher KNOT published in [1], removing the weaknesses that have lead to its breaking in [4].

1 Introduction

In this document we present a self-synchronizing stream cipher called Mosquito. Self-synchronizing stream encryption can be performed by using a block cipher in CFB mode. However, for single-bit self-synchronizing stream encryption, this is very inefficient. Therefore we believe that it would be useful to design a dedicated self-synchronizing stream cipher that is efficient in hardware. Up to date, very few dedicated self-synchronizing stream ciphers have been published and all of them have been broken. Moreover, not much progress seems to have been made as illustrated by the self-synchronizing mode of the Hiji-Bij-Bij stream cipher [9] and its breaking in [5].

A dedicated self-synchronizing stream cipher is a primitive quite different from all other types of cryptographic primitives. It may be interesting to have concrete designs to for performing cryptanalysis and developing a cryptologic theory of security of self-synchronizing stream ciphers. Most of the ideas in this paper were presented in [2] and some of them in paper [1].

The following of this document is structured as follows. After an introduction of self-synchronizing stream encryption, we present the two basic ideas underlying the design of Mosquito. This is followed by a specification of the cipher and the associated security claims. Then we discuss the resistance against standard cryptanalytic attacks and the rationale underlying the design. We discuss the strengths and advantages of self-synchronizing stream ciphers in general and of Mosquito in particular and discuss performance and implementation aspects. Finally we discuss some modes of operation to use the Mosquito for synchronous stream encryption or integrity protection. After the references there is an appendix that discusses problems of the design approach as proposed by Ueli Maurer in [6].

2 Self-synchronizing stream encryption

In stream encryption each plaintext symbol m^t is encrypted by applying a group operation with a keystream symbol z^t resulting in a ciphertext symbol c^t. In modern ciphers these symbols invariably consist of blocks of bits and the group operation is the simple bitwise XOR:

$$c^t = m^t \oplus z^t \ . \tag{1}$$

Decryption takes the *subtraction* of the keystream symbol from the ciphertext symbol. With the bitwise XOR this is the same operation:

$$m^t = c^t \oplus z^t \ . \tag{2}$$

In self-synchronizing stream encryption, a keystream symbol z^t is fully determined by the last n_m (called the memory) ciphertext symbols and a *cipher key* K of n_k bits. This can be modeled as the keystream symbol being computed by a keyed *cipher function* $\mathrm{f_c}$ operating on a shift register containing the last n_m ciphertext. A block diagram of self-synchronizing stream encryption is given in Fig. 1. The generation of keystream symbols is governed by:

$$z^t = \mathrm{f_c}[K](c^{t-n_\mathrm{m}} \dots c^{t-1}) \ . \tag{3}$$

This is in fact only a conceptual model to illustrate the external dependencies. In a specific design the finite input memory of the self-synchronizing stream cipher can be realized in numerous ways.

For the first n_m ciphertext or plaintext symbols of a stream, the *previous* n_m ciphertext bits do not exist. Therefore, prior to operation, a self-synchronizing stream cipher must be initialized by loading n_m *dummy* ciphertext symbols, called the *initialization vector.* If we start numbering the plaintext (ciphertext) bits from 1, the *dummy* ciphertext symbols required for the encryption (decryption) of the first n_m plaintext (ciphertext) symbols are given by the initialization vector:

$$c^{1-n_\mathrm{m}} \dots c^0 = \text{initialization vector} \ . \tag{4}$$

Note that the n_m-bit initialization vector must be shared between encryptor and decryptor but does not need to be kept secret.

Despite their name, self-synchronizing stream ciphers are much more closely related to block ciphers than to synchronous stream ciphers, where the keyed cipher function takes the place of the keyed permutation in a block cipher. An attacker can query the output of the cipher function (keystream symbols) for chosen values of its input: a series of n_m ciphertext symbols. We call the latter an *input vector.* The main difference between a cipher function of a self-synchronizing stream cipher and a block cipher is that the latter is a permutation and that additionally the plaintext can be queried for a given ciphertext. In principle the cipher function of a self-synchronizing stream cipher may be a permutation if $n_\mathrm{m} = 1$. However, in that case we are typically dealing with a block cipher used in CFB mode.

2.1 (Is there) a need?

The most widely adopted approach to self-synchronizing stream encryption is the use of a block cipher in CFB mode. For this mode, the attainable encryption speed is a factor $n_\mathrm{b}/n_\mathrm{s}$

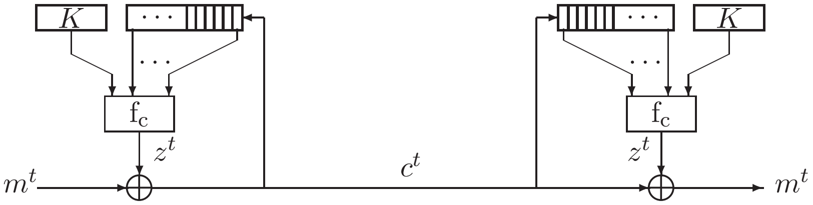

Figure 1: Self-synchronizing stream encryption.

slower than the encryption speed of the underlying block cipher implementation. For the single-bit CFB mode of AES this factor is 128. Clearly, the CFB mode with small symbol length is very inefficient compared to the ECB and CBC modes. This poor efficiency seems to be inherent in self-synchronizing stream encryption with small symbol length. Its only appropriate domain of application is in the addition of encryption in existing systems that have no segmentation or synchronization provisions facilitating block or synchronous stream encryption. For this reason we think that portability is not so important for self-synchronizing stream ciphers than for block ciphers, and synchronous stream ciphers.

For applications in which single-bit self-synchronizing stream encryption is needed with high bit rates, even a hardware implemented block cipher in CFB mode may not be fast enough as in encryption the ciphertext bit must be available to compute the next keystream bit. This implies that the time between two plaintext bit encryptions is lower bound by the execution of the block cipher. This is true in general for the cipher function $\mathrm{f_c}$ of a self-synchronizing stream cipher.

3 Cipher function architecture

In this section we introduce the two main ideas underlying the MOSQUITO design: the use of *pipelining* and *conditional complementing shift registers*.

3.1 Pipelining

We address the problem of realizing a cipher function providing high resistance against cryptanalysis and high speed in dedicated hardware by composing it of a number, denoted by b_s, of *stages* G_i. In hardware, every stage can be implemented by a combinatorial circuit and a register storing the intermediate result. This pipelined approach is illustrated in Figure 2. As the encryption speed is limited by the critical path (largest occurring gate delay), the stages should have small gate delay and hence be relatively simple. This approach impacts the general dependency relations of the self-synchronizing stream cipher: the implementation of the cipher function in b_s stages causes the keystream bit z^t to depend on the contents of the shift register b_s time steps ago. We now obtain the following encryption equation:

$$z^t = \mathrm{f_c}[K](c^{t-n_\mathrm{m}} \dots c^{t-(b_\mathrm{s}+1)}) \ . \tag{5}$$

The pipelining increases the input memory n_m of the cipher by b_s symbols. However, the number of input symbols in the cipher function remains the same, as a keystream symbol z^t

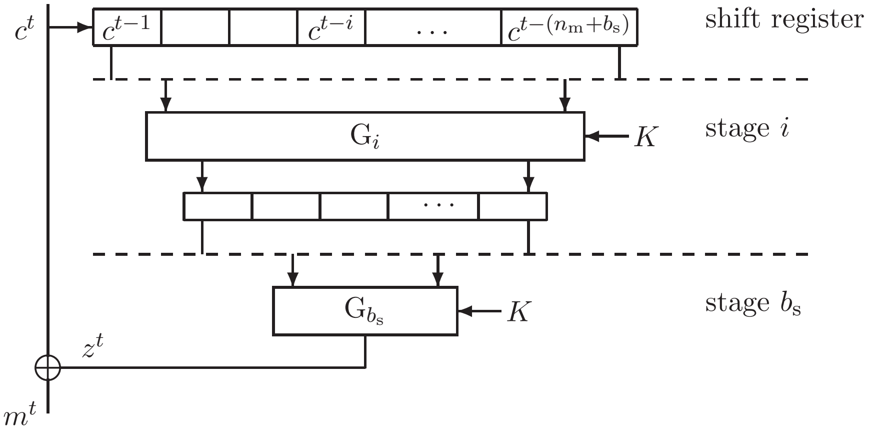

Figure 2: Self-synchronizing stream cipher with a cipher function consisting of stages.

is independent of the ciphertext symbols c^{t-b_s} to c^{t-1}. Therefore we call the quantity b_s the *cipher function delay.*

3.2 Machines with finite input memory

In the pipelined structure presented in previous section, the input to the first stage consists of the last $n_\mathrm{m} - b_\mathrm{s}$ ciphertext bits, contained in a shift register. This construction guarantees that the keystream bit z^t only depends on the cipher key K and ciphertext bits c^{t-n_m} to $c^{t-(b_\mathrm{s}+1)}$.

Replacing the shift register by a finite state machine with finite input memory n_m *can* improve the propagation properties without violating this dependence restriction. By imposing that the gate delay of this finite state machine is not larger than the critical path, the maximum encryption speed of a hardware implementation is not impacted.

A finite state machine with finite input memory has some specific propagation properties. Let q be the internal state and G the state-updating transformation. Then

$$q^{t+1} = \mathrm{G}(q^t, c^t) \ , \tag{6}$$

with c^t the ciphertext bit at time t.

With every component of the internal state q can be associated an input memory, equal to the number of past ciphertext bits that it depends on. The internal state, confined to the components with input memory j is denoted by $q^{(j)}$, with $q_i^{(j)}$ its ith component. Although it is not a part of the internal state, c can be considered as a component with input memory zero: $q^{(0)}$. The input memory of the finite state machine is equal to the largest occurring component input memory.

Clearly, $q_i^{(j)}$ at time $t + 1$ must be independent of all $q^{(\ell)}$ with $\ell \geq j$ at time t and *must* depend on $q^{(j-1)}$ at time t. From this, it follows that the input memory partitions the components of the internal state into non-empty subsets with input memory 1 to n_m. The components of the state-updating transformation are of the form:

$$q_i^{(j)^{t+1}} = \mathrm{G}[K]_i^{(j)}(c^t, q^{(1)^t}, \ldots, q^{(j-1)^t}) \ , \tag{7}$$

for $0 < j \leq n_\mathrm{m}$.

3.3 Conditional complementing shift registers

In this section we introduce a way to build finite state machines with finite input memory that contribute to the resistance against cryptanalysis. An important potential problem is the existence of high-probability extinguishing differentials. An extinguishing differential is a difference in the (ciphertext) input vector leading to a zero difference in the internal state. We prevent extinguishing differentials by imposing (partial) linearity on the components of the state-updating transformation. For simplicity we impose the preliminary restriction that all $q^{(j)}$ have only one component, i.e., that there is only one bit for every input memory value. The components of the state-updating transformations are of the form

$$q^{(j)^{t+1}} = q^{(j-1)^t} + \mathrm{E}[K]^{(j)}(q^{(j-2)^t}, \ldots, q^{(1)^t}, c^t) \ . \tag{8}$$

Since the new value of $q^{(j)}$ is equal to the sum of the old value of $q^{(j-1)}$ and some Boolean function, we call this type of finite state machine a *conditional complementing shift register* (CCSR).

A finite state machine with finite input memory n_m realizes a mapping from a length-n_m sequence of ciphertext bits $c^{t-n_\mathrm{m}}, \ldots, c^{t-1}$ to an internal state q^t. For a CCSR we have the following result.

Proposition 1 *The mapping from $c^{t-n_\mathrm{m}}, \ldots, c^{t-1}$ to the internal state q^t of a CCSR is an injection.*

Proof : We describe an algorithm for the reconstruction of $c^t q^{(1)^t} \ldots q^{(j-1)^t}$ from $q^{(1)^{t+1}} \ldots q^{(j)^{t+1}}$. The components are reconstructed starting from c and finishing with $q^{(j-1)}$. For $q^{(1)}$ (8) becomes

$$q^{(1)^{t+1}} = q^{(0)^t} + \mathrm{E}[K]^{(1)} = c^t + \mathrm{E}[K]^{(1)} \ ,$$

since $\mathrm{E}[K]^{(1)}()$ depends only on K. From this we can calculate c^t. The values of $q^{(k-1)^t}$ for k from 2 to j can be calculated iteratively from the previously found values by

$$q^{(k-1)^t} = q^{(k)^{t+1}} + \mathrm{E}[K]^{(j)}(q^{(k-2)^t}, \ldots, q^{(1)^t}, c^t) \ .$$

$c^{t-n_\mathrm{m}} \ldots c^{t-1}$ can be calculated uniquely from $q^{(1)^t} \ldots q^{(n_\mathrm{m})^t}$ by iteratively applying the described algorithm. □

It follows that a nonzero difference in $c^{t-n_\mathrm{m}} \ldots c^{t-1}$ must give rise to a nonzero difference in q^t. Therefore in a CCSR there are no extinguishing differentials between the input vector and its state.

The CCSR has the undesired property that a difference in $c^{-n_\mathrm{m}-t}$ propagates to $q^{(n_\mathrm{m})^t}$ with a probability of 1. This can be avoided by "expanding" the high input memory end of the CCSR, i.e., taking more than a single state bit per input memory value near memory value n_m.

3.4 The pipelined stages revisited

In our architecture, the cipher function consists of a CCSR followed by a number of pipelined stages. The stage functions are similar to the round transformation in a block cipher but are less restricted.

A round transformation of an iterated block cipher must be a permutation, and its inverse must be easily implementable. A stage function does not have this restriction and the length of its output can be different from that of its input. The output of the last stage function is a Boolean function of the components of the state q some cycles ago. An imbalance in this function leads to an imbalance in the cipher function. This Boolean function can be forced to be balanced by imposing that all the stage functions are *semi-invertible*. We call an n-bit to m-bit mapping $b = f(a)$ semi-invertible if there exists an n-bit to $(n-m)$-bit mapping $b' = f'(a)$ so that a is uniquely determined by the couple (b, b'). In that case the output bit may have figured as a component of the output of an invertible function of the state q.

The last round transformation of an iterated block cipher must be followed by a key application or include a key dependence. This is necessary to prevent the cryptanalyst to calculate an intermediate encryption state thereby making the last round transformation useless. For the cipher function of a single-bit self-synchronizing stream cipher the calculation of intermediate values is impossible since only a single output bit z^t is given per input. Therefore, key dependence is not a strict necessity for the stage functions.

4 The Mosquito cipher function

Mosquito is a self-synchronizing stream cipher with:

- : Symbol size n_{s}: 1
- : Key size n_{k}: 96
- : Memory n_{m}: 105
- : cipher function delay b_{s}: 9

We write:

$$\begin{aligned} z^t &= \mathrm{f_c}[K](c^{t-105} \dots c^{t-10}) \ , & (9) \\ c^{-104} \dots c^0 &= \text{initialization vector} \ . & (10) \end{aligned}$$

In the ECRYPT call for stream ciphers an IV of length 32 or 64 bits is mandated. Therefore we allow an IV that consists of an integer number of bytes where this number ranges from 0 to 13 inclusive. The initialization vector is obtained by taking a single 0 bit, appending the initialization vector and then perform padding with zero bytes until the length is 105 bits.

4.1 The Mosquito CCSR

The Mosquito CCSR has an input memory of 96. The CCSR is expanded at the high input memory end, with 2 bits per input memory starting from $j = 89$, four starting from 93, eight of 95 and sixteen of 96, resulting in 128 state bits. The cipher key K consists of 96 bits: $K_0 \dots K_{95}$.

For reasons of simplicity, area and gate delay, the majority of the components of the state-updating transformation consist of a very simple Boolean function of the form

$$\mathrm{G}[K]_i^{(j)}(q, c) = q_i^{(j-1)} + K_{j-1} + (q_i^{(v)})(q_i^{(w)} + 1) + 1 \ , \qquad (11)$$

with $0 \le v, w < j - 1$.

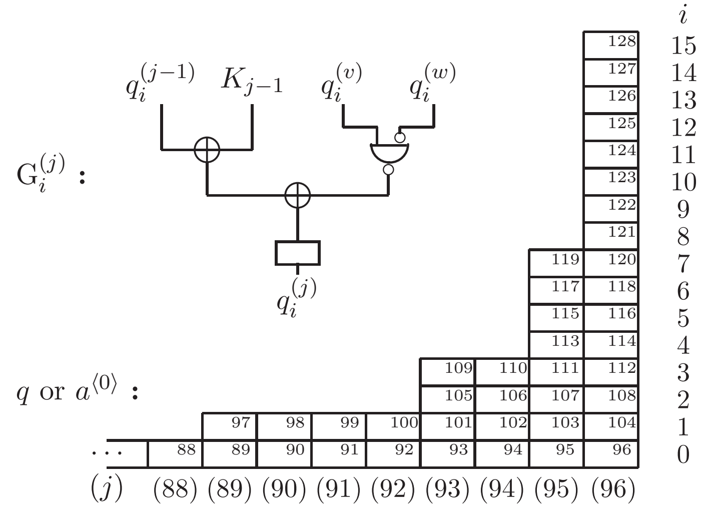

Figure 3: A circuit implementing a component of the state-updating transformation and expansion of the CCSR in the high memory region. The q indexing (CCSR) is given at the right and the bottom, the $a^{\langle 0 \rangle}$ indexing (pipelined stages, see p. 7) inside the boxes.

	v	w
$(i+j) \bmod 3 = 0$	$j - 4 + (i \bmod 2)$	$j - 2$
$(i+j) \bmod 3 = 1$	$j - 6 + (i \bmod 2)$	$j - 2$
$(i+j) \bmod 6 = 2$	$j - 5 + (i \bmod 2)$	0
$(i+j) \bmod 6 = 5$	0	$j - 2$

Table 1: v and w values for the components $G_i^{(j)}$ with $4 < j < 96$ and $G_0^{(96)}$.

Key bits are applied by XOR and the remaining part of E is the simplest nonlinear function possible. A combinatorial circuit for this function is depicted in Figure 3. This circuit has a gate delay of 2 XOR gates. The figure also shows the expansion of the CCSR at the high input memory end and the corresponding indexing. The v and w values for almost all components $G_i^{(j)}$ are specified in Table 1. For $j \leq 4$, the $q^{(v)}$ and $q^{(w)}$ entries are taken to be 0. In some cases terms $q_i^{(j)}$ specified by Table 1 do not correspond to existing components. In that case the i-index must be diminished by iteratively subtracting the largest power of 2 contained in i until the term corresponds to an existing component (e.g., $q_{14}^{(93)} \to q_6^{(93)} \to q_2^{(93)}$). The 15 components $G_i^{(96)}$ with $i > 0$ are of a different type. These are specified by

$$G[K]_i^{(96)}(q, c) = q_i^{(95)}(q_0^{(95-i)} + 1) + q_i^{(94)}(q_1^{(94-i)} + 1) \ . \tag{12}$$

4.2 The Mosquito pipelined stages

The Mosquito output function has 7 stages in total. The output of stage $\langle i \rangle$ is stored in a register denoted by $a^{\langle i \rangle}$. Register $a^{\langle 0 \rangle}$ corresponds to q and has a length of 128. For ease of notation $a^{\langle 0 \rangle}$ has been given a different indexing than q. This is specified in Figure 3.

The components of the registers $\langle 1 \rangle$ to $\langle 7 \rangle$ are indexed starting from 0. Registers $a^{\langle 1 \rangle}$ to $a^{\langle 5 \rangle}$

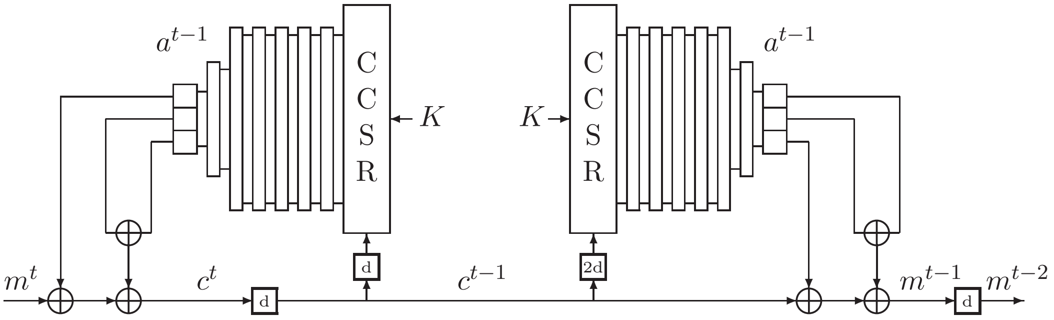

Figure 4: Encryption and decryption with Mosquito.

have length 53. For the component state-updating transformation of stage 1 we have

$$\mathrm{G}^{\langle 1\rangle}_{4i \bmod 53} = a^{\langle 0\rangle}_{128-i} + a^{\langle 0\rangle}_{i+18} + a^{\langle 0\rangle}_{113-i}(a^{\langle 0\rangle}_{i+1} + 1) + 1 \ , \tag{13}$$

for $0 \le i < 53$. The stages 2 to 5 are specified by

$$\mathrm{G}^{\langle j\rangle}_{4i \bmod 53} = a^{\langle j-1\rangle}_{i} + a^{\langle j-1\rangle}_{i+3} + a^{\langle j-1\rangle}_{i+1}(a^{\langle j-1\rangle}_{i+2} + 1) + 1 \ , \tag{14}$$

for $0 \le i < 53$. If a lower index in the right-hand side of these equations becomes larger than 52, the corresponding term is taken to be 0, e.g., $a^{\langle j-1\rangle}_{53} = 0$. Register $a^{\langle 6\rangle}$ has a length of 12. Stage 6 is defined by

$$\mathrm{G}^{\langle 6\rangle}_{i} = a^{\langle 5\rangle}_{4i} + a^{\langle 5\rangle}_{4i+3} + a^{\langle 5\rangle}_{4i+1}(a^{\langle 5\rangle}_{4i+2} + 1) + 1 \ . \tag{15}$$

Register $a^{\langle 7\rangle}$ consists of 3 bits. We have

$$\mathrm{G}^{\langle 7\rangle}_{i} = a^{\langle 6\rangle}_{4i} + a^{\langle 6\rangle}_{4i+1} + a^{\langle 6\rangle}_{4i+2} + a^{\langle 6\rangle}_{4i+3} \ . \tag{16}$$

Finally the encrypting bit is given by

$$z = a^{\langle 7\rangle}_{0} + a^{\langle 7\rangle}_{1} + a^{\langle 7\rangle}_{2} \ . \tag{17}$$

4.3 Putting it together

Figure 4 shows the Mosquito self-synchronizing stream cipher. Its gate delay is 2 XOR gates, equal to the gate delay of the state-updating transformation. This necessitates the introduction of extra intermediate storage cells, denoted in Figure 4 by boxes containing a **d**. In the encryptor this cell is located between the encryption and the input of the CCSR. For correct decryption this necessitates an extra delay of 1 clock cycle at the input of the CCSR.

5 Claimed security properties

The claimed security properties of a self-synchronizing stream cipher may be expressed in terms of its cipher function.

Claim 1 *The probability of success of an attack not involving key recovery, that guesses the output of the cipher function corresponding to n input vectors C_i while given the cipher function output corresponding to any set of (adaptively) chosen input vectors not containing any of the C_i is 2^{-n}.*

Claim 2 *There are no key recovery attacks faster than exhaustive key search, i.e. with an expected complexity less than 2^{n_k} cipher function executions.*

Note that we do not claim resistance against attackers that may manipulate the key and that in our attack model, the attacker has no knowledge about the key whatsoever. It is the responsibility of the application developer to employ key management functions that ensure this. For example, having so-called *key variants* (keys XORed with constants) for use by Mosquito would be a particularly bad idea. But of course, using key variants is bad engineering in any case.

Moreover, we do not claim resistance against attackers that have access to (part of) the internal state or that can disrupt the proper operation of an implementation of Mosquito. While such attack scenarios may be realistic in the context of side-channel attacks, we do not consider that these problems should be tackled in the cipher design but rather in its implementation.

6 On hidden weaknesses

We have not inserted any hidden weaknesses in Mosquito nor are we aware of weaknesses in Mosquito.

7 Resistance against standard attacks

The design of Mosquito dates back to 1994. Its design was mainly guided by the goal to provide resistance against differential and linear cryptanalysis and their extensions. Therefore these two types of cryptanalysis get most of the attention in this section. We mention here some other well-known attacks:

- Algebraic attacks: no in-depth evaluation has taken place. Given the keystream bits corresponding to at least 96 ciphertext bit sequences, thanks to the cipher structure it is straightforward to generate of a set of low-degree algebraic equations that results in the key value when solved. The difficulty is in solving these sets of equations. We believe the problem of solving these sets of equations for Mosquito is similar to that for block ciphers, which appears to be hard.

- Weak keys: the key bits are applied by a simple XOR. As the differential and linear propagation properties are to a large degree independent of the absolute value of the key bits, we think there are no weak keys.

- Related-key attacks: we claim no resistance against related-key attacks. The cipher should be used in applications where mounting a related-key attack is not possible. If the same key is used for encrypting different sequences and if one fears ciphertext collisions leading to information on the plaintext, one should use unique IV values to diversify the ciphertext.

All in all, Mosquito has not been subject to a thorough analysis with respect to all known attacks. However, no other dedicated self-synchronizing stream cipher is known to us that has not been broken.

In the following subsections we focus on the application of differential and linear cryptanalysis to single-bit self-synchronizing stream ciphers. If the symbol size is larger than a single bit, the situation generally becomes more complex. For examples of differential attacks on self-synchronizing stream ciphers with a symbol size larger than 1, we refer to [8, 7]. These publications contain some powerful attacks of DES variants in the 8-bit CFB mode.

7.1 Differential cryptanalysis

For every pair of n_m-bit (ciphertext) input vectors with a specific difference a', $\mathrm{f_c}$ returns a pair of keystream bits. The probability that the keystream bits are different is denoted by $\mathrm{P}(a', 1)$. The usability in differential cryptanalysis of $\mathrm{P}(a', 1)$ is determined by its deviation from 1/2. If this probability is $(1 \pm \ell^{-1})/2$, the number of input pairs needed to detect this deviation is approximately ℓ^2.

Consequently, a cipher function should not have differentials with probabilities that deviate significantly more than $2^{-(n_\mathrm{m}-b_\mathrm{s})/2}$ from 1/2. The input differences a' with the highest deviations should depend in a complex way on the cipher key.

Differential attacks can be generalized in several ways. One generalization that proved to be powerful in the cryptanalysis of some weak proprietary designs can be labeled as *second order* differential cryptanalysis. Here the inputs to the cipher function are applied in 4-tuples. The 4 inputs denoted by a_0, a_1, a_2 and a_3 have differences $a' = a_0 + a_1 = a_2 + a_3$ and $a'' = a_0 + a_2 = a_1 + a_3$. By examining the 4 corresponding output bits it can be observed whether complementing certain input bits (a'') affects the propagation of a difference (a'). This can be used to determine useful internal state bits or even key bits. Typically these attacks exploit properties very specific to the design under analysis. This can be generalized to even higher order DC in a straightforward way.

7.2 Linear cryptanalysis

The number of inputs needed to detect a correlation C of the keystream bit with a linear combination of input bits is C^{-2}. It follows that a cipher function should not have input-output correlations significantly larger than $2^{-(n_\mathrm{m}-b_\mathrm{s})/2}$. The selection vectors v_a with the highest correlations should depend in a complex way on the cipher key. By imposing a number ℓ of affine relations on the input bits, the cipher function is effectively converted to a Boolean function in $n_\mathrm{m} - b_\mathrm{s} - \ell$ variables. These functions should have no correlations significantly larger than $2^{-(n_\mathrm{m}-b_\mathrm{s}-\ell)/2}$ for any set of affine relations.

A special case of a selection vector is the zero vector. An output function that is correlated to the constant function is unbalanced. A correlation of C to the constant function gives rise to an information leakage of approximately $C^2/\mathrm{ln}2$ bits per encrypted bit for $C < 2^{-2}$.

Linear cryptanalysis exploits correlations between the keystream bit and linear combinations of bits of the input vector. This can be generalized by allowing for a wider range of functions. An example of this is given by the table reconstruction attack as described in [6]. The idea is to find a low degree function that approximates the cipher function and can be used to diminish the uncertainty about the plaintext. It is shown that the construction of such an approximation using a limited number of samples is equivalent to a (de)coding problem.

The feasibility of this attack is limited by the amount of high order terms in the algebraic normal form of the cipher function. Although this restriction severely limits the applicability

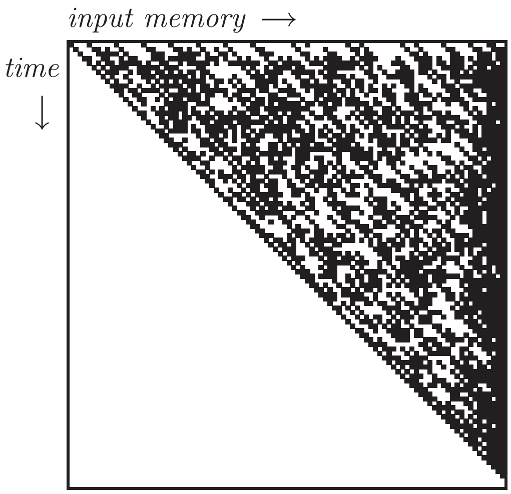

Figure 5: Difference propagation patterns in the Mosquito CCSR.

of the attack, its basic idea is powerful and can inspire dedicated attacks. The process of fixing the class of approximating functions and deducing conditions for the cipher functions can be reversed. Depending on the specific design, the internal structure of the cipher function can possibly be used to define a manageable class of functions that is likely to contain a usable approximation for the cipher function.

8 Design rationale

8.1 The CCSR

The main design goal of the CCSR is the elimination of differentials from $c^{t-96} \dots c^{t-1}$ to Q^t with a probability larger than 2^{-15}, while keeping the gate delay very small and the description simple. Observe that the 15 components $\mathrm{G}_i^{(96)}$ with $i > 0$ are unbalanced functions resulting in a bias in the corresponding components $q_i^{(96)}$.

Figure 5 shows a typical propagation pattern for the Mosquito CCSR. A black mark at time coordinate t and input memory coordinate i denotes that at time t there is a difference in at least one of the bits of input memory i (note that there are more than state bit for input memory values above 88). The diagonal trailing edge is caused by the linear forward propagation of the CCSR ensuring that differential trails do not prematurely dissolve. The difference pattern at time 0 resulting from a difference pattern in the ciphertext symbols ending in c^{-96} is restricted to $q^{(96)}$. The partial linearity in (11) guarantees that the difference pattern is 1 in $q_0^{(96)}$. The difference value of each of the 15 remaining components is determined by balanced key-dependent functions of the absolute values of the ciphertext symbols c^{-95} to c^{-1}. This results in 2^{15} equiprobable nonzero difference patterns. The difference patterns in q^0 resulting from difference patterns that end in ciphertext symbol c^{96-e} for small e are not uniformly distributed. However, all difference propagation probabilities are well below the 2^{-15} limit.

Care was taken that extinguishing differentials from the input vector to the CCSR state cannot occur because these give rise to high-probability differentials of the cipher function.

Assume that for the function corresponding with the stages $\mathrm{P}(q', 0) = 1/2$ for all nonzero difference patterns q'. In that case an extinguishing differential in the CCSR with p gives rise to a extinguishing differential of the cipher function with probability approximately $(1+p)/2$.

An input difference diffuses immediately to components all over q. This is a consequence of the fact that c^t is not only injected in $q^{(1)}$, but in many components at once. These are represented by the zero v and w entries in Table 1 (keep in mind that $q^{(0)} = c$). Depending on the value of $q_0^{(j-3)}$, a difference in c propagates to either $q_0^{(j-1)}$ or $q_0^{(j+2)}$. Since there are more than 15 of these "double injections", the probabilities are below 2^{-15}. In subsequent iterations this pattern is subject to the nonlinearity of the CCSR state-updating transformation.

8.2 The pipelined stages

The input to the first stage consists of the state bits of the CCSR. Special care has been taken with respect to difference patterns restricted to the high-memory region and those resulting from a difference in the most recent cipher bit. The purpose of stages $\langle 1 \rangle$ to $\langle 6 \rangle$ is the elimination of low-weight linear and differential trails. The components of these stage functions combine diffusion, nonlinearity and dispersion respectively in the linear term, in the quadratic term and in the arrangement of inputs and outputs. Their effectiveness is reinforced by the diffusion in stage $\langle 7 \rangle$ and the output function that computes the keystream bit as the bitwise addition of all 12 bits of $a^{\langle 6 \rangle}$ to the output. Finally, it can easily be checked that all the stages are semi-invertible.

8.3 Some history

Mosquito is actually a variant of the self-synchronizing stream cipher $\Upsilon\Gamma$ specified and proposed in [2], that in its turn was an improvement of our earlier design called KNOT that was presented in our paper [1]. During the writing of [2], we discovered that KNOT had several weaknesses. First of all, the output function of KNOT had a detectable imbalance. This has been avoided in the present design by imposing that the stages be semi-invertible. Second, KNOT has extinguishing differentials in the CCSR. In Mosquito this has been avoided by using a different function for $q_0^{(96)}$. The extinguishing differentials were exploited in [4] to break KNOT.

9 Strength and advantages

We distinguish the strength and advantages of self-synchronizing stream encryption in general, and those of Mosquito in particular.

9.1 Self-synchronizing stream encryption in general

Single-bit self-synchronizing stream encryption has a specific advantage over all other types of encryption. For, in providing an existing communication system with encryption, single-bit self-synchronizing stream encryption can be applied without the need for additional synchronization or segmentation.

Decryption is correct if the last n_{m} ciphertext symbols have been correctly received. An isolated error on the channel between encryptor and decryptor gives rise to an extra burst of n_{m} potentially incorrectly decrypted symbols, i.e., $n_{\mathrm{m}}n_{\mathrm{s}}$ incorrectly decrypted bits at the receiver.

Note that if the transmission error behavior can be modeled by a binary memoryless channel, self-synchronizing stream encryption is only practical if the bit error rate is significantly smaller than $(n_{\mathrm{m}}n_{\mathrm{s}})^{-1}$. In this case the presence of the decryptor multiplies the error rate by a factor of $n_{\mathrm{m}}n_{\mathrm{s}}$.

For channels that suffer from error bursts, self-synchronizing stream encryption is the natural solution. In this setting the error propagation effect of encryption is only an extension of every burst with $n_{\mathrm{m}}n_{\mathrm{s}}$ bits. If n_{s} differs from 1, there can however be an alignment problem after a burst.

In conclusion, the specific advantage of self-synchronizing stream encryption (especially binary, i.e., with $n_{\mathrm{s}} = 1$) is that they can be built on top of any digital communication system with minimum cost.

9.2 Mosquito in particular

The strength and advantages of Mosquito compared to other single-bit self-synchronizing stream ciphers is that it can be implemented in hardware in a reasonable area and with a very small gate delay. We know of no other published dedicated single-bit self-synchronizing stream ciphers. In the current state of affairs, the only alternative to Mosquito is to use a block cipher in single-bit CFB mode. In a dedicated hardware implementation this would for any widely accepted block cipher lead to a very low throughput, or when using pipelining, an enormous area.

10 Computational efficiency in software

Mosquito has been designed with dedicated hardware implementations in mind and does not lend itself to software implementations at all. We expect that even a highly optimized software implementation of Mosquito will perform at least an order of magnitude worse than a program implementing a block cipher such as AES in single-bit CFB mode. Therefore we think it makes no sense to give performance results for software implementations.

11 Hardware performance and implementation aspects

In this section we describe a circuit that allows to perform both encryption and decryption with Mosquito. This can be accomplished by the introduction of a multiplexer that can be used to switch between encryption and decryption. This is shown in Figure 6. The multiplexer is located at the input of the CCSR and chooses between the input x^{t-1} for decryption and the result of adding the keystream bit z^{t-1} to the input x^{t-1} for encryption.

As described in previous sections, in a straighforward hardware implementation a Mosquito encryption circuit or a Mosquito decryption circuit has a critical path of 2 XOR gates. In order to have the same gate delay in the combined circuit, the multiplexer is placed between two flip-flops. This limits the critical path to the maximum of the delay of two XOR gates and that of a 2-to-1 multiplexer and explains the fact that there is a fixed flip flop at the input of the Mosquito CCSR. The storage cells (d) are implemented with D flip-flops.

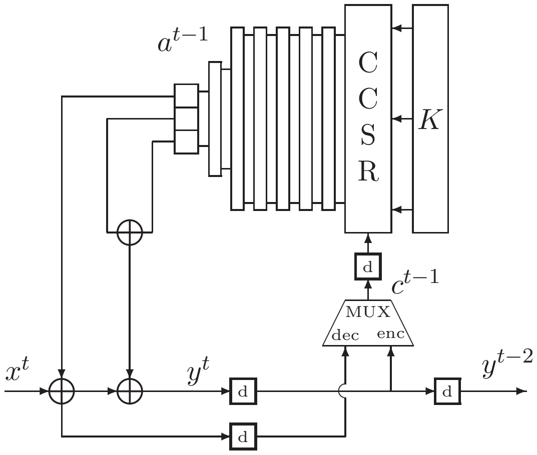

Figure 6: Mosquito circuit that allows encryption and decryption.

Before encryption or decryption is performed the cipher must have its key and must subsequently have been initialized by feeding it with an initialization vector. This takes the place of the 105 ciphertext bits (14 bytes) that come before the first ciphertext bit. Initialization merely consists of applying the initialization vector bits to the input of the cipher circuit in decryption mode. In other words, they are simply entered as ciphertext bits. The only difference with actual decryption is that during initialization for cryptanalytic purposes the cipher does not return keystream bits, and so no plaintext bits. This is implemented in the flip-flop at the cipher output that does not latch during initialization phase.
The complete operation of the Mosquito stream cipher is:

- The cipher key is loaded in the key register.
- The cipher is set in mode decryption and the initialization vector bits are applied at the cipher input for 105 clock cycles. During this phase no output shall be available.
- For encryption:
  - The cipher circuit is set to encryption mode,
  - For each clock cycle a plaintext bit is loaded at the input $x_t = m_t$ and the ciphertext bit of the plaintext bit two clock cycles ago appears at the output $y_t = c_{t-2}$.
- For decryption:
  - The cipher circuit is left in decryption mode,
  - For each clock cycle a plaintext bit is loaded at the input $x_t = c_t$ and the ciphertext bit of the plaintext bit two clock cycles ago appears at the output $y_t = p_{t-2}$.

With the circuit at hand, key loading can be done in a single cycle and initialization takes 105 cycles.

11.1 VLSI implementation synthesis results

We have implemented the Stream Cipher circuit using Field Programmable Gate Array (FPGA). The hardware implementation is designed and coded in VHSIC Hardware Description Language (VHDL) with structural description logic. The proposed implementation was

FPGA Device	# FGs	# CLB Slices	# DFFs	Throughput (Mb/sec)
VIRTEX (V50BG256)	1563	768	1563	179
VIRTEX - E (V50EPQ240)	1536	768	2010	240
VIRTEX - II (2V80FG256)	1024	512	1384	411
Required by MOSQUITO	405	252	503	-

Table 2: MOSQUITO synthesis results and performance numbers

simulated for the correct operation with test vectors returned by the software implementation. The VHDL code was synthesized in XILINX Virtex, Virtex-E and Virtex-II FPGAs [10] for demonstration purposes. However the proposed design can easily be migrated to other silicon technologies such as ASICs or CPLDs. The performance characteristics of FPGAs are substantially different compared with a general-purpose microprocessor. The fine granularity of FPGAs matches extremely well the operations required by the algorithm.

Furthermore, the inherent parallelism of the algorithms can be efficiently exploited in FPGAs as opposed to the serial fashion of computing in a general-purpose microprocessor. At the cryptographic-round level, multiple operations can be executed concurrently. On the other hand, the internal structure of the microprocessors functional units limits the parallel processing and pipelining transformation. In addition, the instruction parallelism level is a factor of great importance that must be taken under consideration for microprocessor performance.

The synthesis results and performance analysis are shown in Table 2 indicating the number of D Flip-Flops (DFFs), Configurable Logic Blocks (CLBs) and Function Generators (FGs).

The throughput is estimated after the initialization phase. The smallest FPGA devices with low hardware resources utilization by each FPGA family were used. The cipher uses 103 I/Os. As the table shows in V50BG256 VIRTEX FPGA, 25.5% of the FGs are used, 32.8% of the CLBs and 32.7% of the DFFs. In V50EPQ240 VIRTEX - E FPGA from the total number of area resources the cipher consumes 26.5% of the FGs, 32.8% of the CLB slices and 25% of the DFFs. Finally, from the 2V80FG256 VIRTEX - II FPGA the stream cipher consumes 39.5% of the FGs, 49% of the CLB slices and 36% of the DFFs. The experimental delay measurements (critical path delay, 1/Freq.) are very close to the expected values produced by the theoretical expression (critical path delay $= 2 * t_{XOR}$). The slight differences between the experimental and the theoretical values are due to the fact that in the theoretical values the FPGA internal interconnection wires delays, D flip flop or buffer transfer delays are not calculated. All in all the cipher achieves a low level of FPGA utilization and is suitable for hardware implementation.

11.2 Avoiding implementation weaknesses

The circuit described in this section has no special protection against side channel attacks and it may be vulnerable especially with respect to differential power analysis and differential fault analysis. With respect to the former, techniques may be employed similar to those introduced in [3].

Other attention points when building a circuit are:

- The key shall be stored in a protected zone.

- The circuit shall only function when a key has been fully loaded.
- When the initialization vector is loaded no 'plaintext' bits shall be available at the output.

When a circuit or different circuits make use of key variants, i.e., keys that have a known (or even worse, chosen) XOR difference, the security of MOSQUITO may degrade dramatically. If the usage of key variants cannot be avoided, one may perform preprocessing on the cipher key prior to loading it in the key register as working key. We call this *key mangling*. One way of doing this is by using MOSQUITO encryption for this purpose. For example, one may set up MOSQUITO with the cipher key as IV, the key equal to zero and encipher an all-zero string of 96 bits. The resulting ciphertext may then serve as working key.

enciphering the key with MOSQUITO with the all-zero key and all-zero initialization vector and taking the resulting plaintext as working key.

12 Alternative modes of operation

In this section we give some ideas on modes of operation of a self-synchronizing stream cipher to realize other cryptographic functions. These ideas are not particular for MOSQUITO but rather for any dedicated self-synchronizing stream cipher.

12.1 MAC function

One may transform MOSQUITO into a MAC function operating on a message M in the following way. If a MAC of n bits is required, append $n + 8$ zero bits to the message, encrypt the result (using an all-zero initialization vector), and take as MAC the last n bits of the ciphertext. Clearly, all other ciphertext bits shall not be made public (as is the case in MAC functions based on the CBC-mode of block ciphers). The length of the MAC n should be 64 bits or less.

12.2 Authenticated encryption

One may protect the integrity of data by adding controlled redundancy into the plaintext prior to encryption. Two ideas:

- For continuous protection of a stream: the plaintext stream is interleaved with parity check bits, for instance one bit per byte.
- For protection of a long message: a zero byte, followed by a redundancy field computed over the plaintext is added at the end of the plaintext, for example a 32-bit CRC.

As each ciphertext bit depends in a complicated way on all plaintext bits (minus the last 9), this may well provide a reasonable level of forgery detection. For a quantitative analysis of the forgery detection probabilities we refer to Section 3.6 of [2].

12.3 Synchronous stream cipher

By feeding the keystream bit back to the input, we obtain a synchronous stream cipher. We call this the OFB mode of a self-synchronizing stream cipher. Unfortunately, the state transition function is not invertible, leading typically to cycle lengths that are only the square root of the total number of states. In Section 9.6.1 of [2] we give a construction for $\Upsilon\Gamma$ that avoids this problem by making the state transition invertible. Applying this construction to Mosquito leads to cycle lengths in the order of 2^{104}.

References

[1] J. Daemen, R. Govaerts and J. Vandewalle, "On the Design of High Speed Self-Synchronizing Stream Ciphers," *Singapore ICCS/ISITA '92 Conference Proceedings* P.Y. Kam and O. Hirota, Eds., IEEE 1992, pp. 279–283.

[2] J. Daemen, "Cipher and hash function design strategies based on linear and differential cryptanalysis," *Doctoral Dissertation*, March 1995, K.U.Leuven.

[3] J. Daemen, M. Peeters and G. Van Assche, "Bitslice ciphers and Power Analysis Attacks," *Fast Software Encryption 2000, LNCS 1978*, B. Schneier, ed., Springer-Verlag 2000, pp. 134-149.

[4] A. Joux and F. Muller, "Loosening the KNOT," *Fast Software Encryption 2003, LNCS 2887*, T. Johansson, ed., Springer-Verlag, 2003, pp. 87-99.

[5] A. Joux and F. Muller, "Two Attacks Against the HBB Stream Cipher," *Fast Software Encryption 2005*, H. Gilbert and H. Handshuh, eds., Springer-Verlag, 2005, pp. 323-334.

[6] U.M. Maurer, "New Approaches to the Design of Self-Synchronizing Stream Ciphers," *Advances in Cryptology, Proc. Eurocrypt '91, LNCS 547*, D. Davies, Ed., Springer-Verlag 1991, pp. 458–471.

[7] B. Preneel, M. Nuttin, V. Rijmen and J. Buelens, "Cryptanalysis of the CFB Mode of the DES with a Reduced Number of Rounds," *Advances in Cryptology, Proc. Crypto'93, LNCS 773*, D.R. Stinson, Ed., Springer-Verlag 1994, pp. 212–223.

[8] V. Rijmen, *Cryptanalysis of DES – in Dutch, Cryptanalyse van DES*, ESAT Katholieke Universiteit Leuven, Thesis grad. eng., 1993.

[9] P. Sarkar, "Hiji-Bij-Bij: A New Stream Cipher with a Self-Synchronizing Mode of Operation," *Progress in Cryptology – Indocrypt 2003, LNCS 2904* T. Johansson and S. Maitra, eds., Springer-Verlag, 2003, pp. 36-51.

[10] Xilinx Virtex FPGA Data Sheets (2005), URL: `http://www.xilinx.com`

A On the design principles proposed by Ueli Maurer

Self-synchronizing stream ciphers have not received the attention in cryptologic literature that block ciphers and synchronous stream ciphers have. The paper that stands as the reference on the design of self-synchronizing stream ciphers is [6].

One of the central ideas in [6] is the concept of building a cipher function from a (finite input memory) finite state machine with a cryptographically secure state-updating transformation *and* a cryptographically secure output function. In this appendix we show that this is a vacuous concept. The other main idea in [6] is that of building a finite state machine with an internal state larger than the input memory n_{m}, by means of "serial" and "parallel" composition of smaller finite state machines. We show that a straightforward application of this design strategy leads to cipher functions with easily detectable and potentially exploitable weaknesses.

Subsequently, we perform a propagation analysis of a structure proposed in [6] and compare it to the Mosquito CCSR.

The idea of replacing the shift register by a finite state machine with finite memory is related to that introduced in [6] of building the cipher function from a finite state machine with finite input memory and a combinatorial (memoryless) output function. The difference is that in our case the output function with respect to the finite state machine that replaces the shift register is not memoryless but consists of a number of pipelined stages.

In [6] it is proposed to design the cipher function with a cryptographically secure state-updating transformation G as well as a cryptographically secure output function h. In this context the term "cryptographically secure" is explained as follows:

- **Output function $z^{t+1} = \mathrm{s}[K](q^t)$:** it should be infeasible for an adversary to determine the output of the finite state machine, even if the (actually hidden) state sequence is given.

- **State-updating transformation:** a state-updating transformation can be defined to be cryptographically secure if it is infeasible to determine the state for a given ciphertext sequence of n_{m} bits, even when the state would be provided for any other ciphertext bit sequence of n_{m} bits.

To demonstrate the gravity of the restrictions this imposes, we describe a straightforward procedure to reconstruct the subkeys $K_i^{(j)}$ in these circumstances. The subkey $K_i^{(j)}$ is that part of the key that $\mathrm{G}_i^{(j)}$ explicitly depends on.

The component functions $\mathrm{G}_i^{(j)}$ can be considered as subkey-dependent tables. If the adversary is given a number of output values corresponding to a number of inputs of $\mathrm{G}_i^{(j)}$ that is larger than the length of $K_i^{(j)}$, this subkey can be determined by exhaustive key search over the subkey space. In the given circumstances the adversary can obtain outputs corresponding to as many inputs as he likes by applying ciphertext bit sequences with length $n_{\mathrm{m}} + 1$ and observing the internal state. The internal state after loading the first n_{m} bits of this sequence is the input to the component functions $\mathrm{G}_i^{(j)}$ that results in the internal state one time step further.

The work factor of this attack depends critically on the length ℓ of the longest subkey $K_i^{(j)}$. The attack requires the application of ℓ ciphertext bit sequences. The effort consists mainly of the exhaustive key search of the ℓ-bit subkey. For this attack to be infeasible, at least one subkey $K_i^{(j)}$ should be very long.

In general, the access that the adversary has to the keyed Boolean functions $\mathrm{G}_i^{(j)}$ is that of getting the output corresponding to known input. The state-updating transformation can only be cryptographically secure by the above definition if at least one of its components $\mathrm{G}_i^{(j)}$ can resist attacks by an adversary with this type of access. This resistance includes hiding the key $K_i^{(j)}$. This can only be realized if $\mathrm{G}_i^{(j)}$ has excellent propagation properties. This also holds for the output function, since the access of the adversary to the output function is the same as to the components of the state-updating transformation.

Keyed Boolean functions with the required resistance against structural cryptanalysis can in practice only be implemented as the succession of a considerable number of implementable round mappings. Since the state-updating transformation of a finite state machine is per definition memoryless, this iterated structure must be realized as a combinatorial circuit. The gate delay of this circuit grows with the number of rounds. Hence, the design approach gives rise to an output function and a state-updating transformation with large gate delay. This conflicts with the main objective of designing self-synchronizing stream ciphers substantially *faster* than block ciphers in CFB mode.

A remarkable aspect of the attacks in the context of the proposed security definitions is that they only work if the adversary has access to the internal state of the self-synchronizing stream cipher. The described design principle seems to impose that the design contains big "lumps", i.e., complex combinatorial blocks that are "cryptographically secure".

These combinatorial blocks could be developed into a number of stages using pipelining in such a way that the external behavior would be the same except for some extra delay in the keystream bit. This new structure can again be represented by a simple finite state machine with combinatorial state-updating transformation if the internal state is supplemented with all the intermediate values of these additional stages. For this structure, it is however no longer the case that combinatorial blocks with some kind of cryptographic security can be identified. One could circumvent this by also allowing the cryptographically secure output function and components of the state-updating transformation to have memory. However, this imposes a division of the bits of the physical internal state into two classes: the logical internal state and the memory of the components of the state-updating transformation and the output function. This would make the design principle even more artificial.

The second major design principle that is presented in [6] is that of building a self-synchronizing encryption scheme with several keyed cipher functions in parallel. The keystream bit is obtained as the bitwise sum of the outputs of the cipher functions. A theorem is given stating that if all the keys are independent, the resulting cipher is at least as cryptographically secure as any of the component ciphers. By basing the design of every component cipher on an entirely different principle, the risk that the cipher will be broken is equal to the risk that all design strategies fail simultaneously. The enthusiasm should be quenched by the fact that all *effective* design strategies available today consist of the study of propagation based on differential and linear cryptanalysis. Moreover, it is obvious that splitting the available resources into several independently operating cipher functions is a practice that severely limits the potential for internal propagation.

A.1 A proposed recursive architecture

In [6] it is proposed to build the finite state machine with finite input memory by the application of parallel and serial composition of cipher functions. Parallel composition denotes bitwise addition of the output bits of two cipher functions fed with the same input. In serial composition the output of one of the automata is fed into the input of the other. In Figure 7 an elegant recursive design is depicted that was given in [6] to illustrate the applicability of these composition modes. We will refer to this architecture as Ψ.

The rightmost box depicts the basic component: a 3-bit shift register with a keyed Boolean output function g$[K]$. Four of these components are arranged in the parallel composition of two serial compositions of two components. On the next level, four of the resulting blocks are arranged by serial composition. At the two following levels these arrangements are repeated.

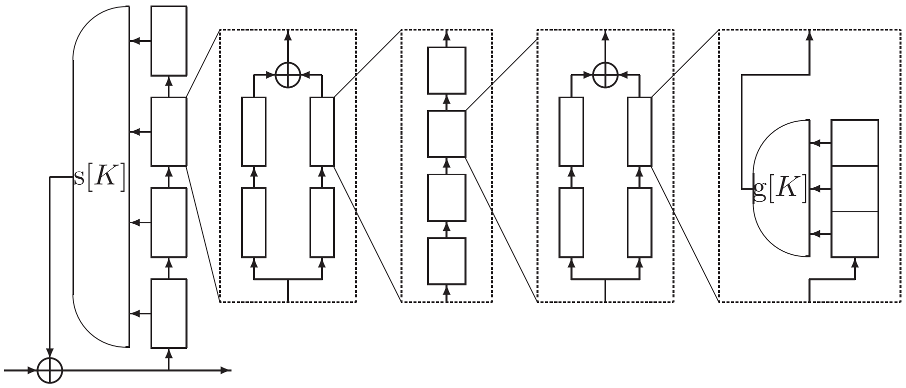

Figure 7: The Ψ architecture.

The resulting finite state machine has an input memory of 192, with exactly 4 component bits for every input memory value. The internal state of this finite state machine serves as the input to the keyed output function s[K]. It is not specified how the keyed Boolean functions g[K] should be constructed. The most obvious way would be to expand the cipher key K into 256 vectors of 8 bits that can serve as the tables for the functions g[K]. More sophisticated key schedules typically impose that the tables fulfill certain criteria such as balancedness or completeness (output is not independent of any of the input bits).

A.2 Difference propagation analysis

We did some difference propagation experiments with the Ψ structure. A cryptographic sequence generator, loaded with a specific initial state, was used to generate the 8-bit tables of the functions g[K]. Two input strings, differing in a specific pattern of bits, were applied and the difference propagation in the internal state was observed. This was repeated for a number of different keys and sequences. Figure 8 gives a typical differential trail for Ψ with an initial difference pattern with a Hamming weight of 1.

Both for the Ψ structure as for the Mosquito CCSR there are in general several state bits for every input memory value. A black mark at time coordinate t and input memory coordinate i denotes that at time t there is a difference in at least one of the bits of input memory i.

Clearly, this differential trail has already completely dissolved about 160 cycles after it has started. Our experiments show that flipping ciphertext bits c^{-j} with j between 192 and 180 leaves q^0, and therefore z^{b_s}, virtually always unaffected.

For a random self-synchronizing stream cipher every input difference c' has an expected probability $\mathrm{P}(c', 0)$ equal to 1/2. For Ψ the input differences c' that are 0 in the last 180 bits have $\mathrm{P}(c', 0) \approx 1$. Since always $\mathrm{P}(0, 0) = 1$, this propagation defect of Ψ cannot be compensated by any output function. We have experienced that this type of propagation defect can serve as a lever in cryptanalysis to pry open the cipher.

The observed defective difference propagation is due to the bad difference propagation inherent in serial composition of cipher functions. For the vast majority of random self-synchronizing ciphers with input memory n_m, the probability $\mathrm{P}(c', 1)$, with c' a vector that is only 1 in component c^{-n_m} and 0 elsewhere, is 1/2. For the serial composition of ℓ uniformly chosen cipher functions with input memories adding up to n_m, this probability is only $2^{-\ell}$.

Serial composition as proposed in [6] should therefore be avoided.

An unlucky choice of the component cipher functions with the lowest input memory can cause even more serious problems. In Ψ there are four 3-bit shift registers that are fed with the input (ciphertext) bits themselves. Let the four corresponding key-dependent Boolean functions be denoted by $\mathrm{g}_1, \mathrm{g}_2, \mathrm{g}_3$ and g_4. Assume that for all these functions

$$\mathrm{g}_i(000) = \mathrm{g}_i(001) = \mathrm{g}_i(010) = \mathrm{g}_i(100) \ .$$

In that case pairs of ciphertext bit sequences that only differ in a single bit preceded and followed by two zeroes (i.e., $\ldots 00000 \ldots$ and $\ldots 00100 \ldots$) give rise to pairs of output sequences with pairwise identical members for all four functions. This implies that the resulting differential trail does not reach beyond the four shift registers with lowest input memory. Totally there are 16 different cases corresponding to the possible combinations of values in the two following and preceding bits. If the functions g are uniformly generated, the differential trail stops at least for one of these 16 cases for exactly 1 cipher key in 256. For balanced functions g_i, the proportion of cipher keys with prematurely dissolving differential trails is larger than 1 in 6000.

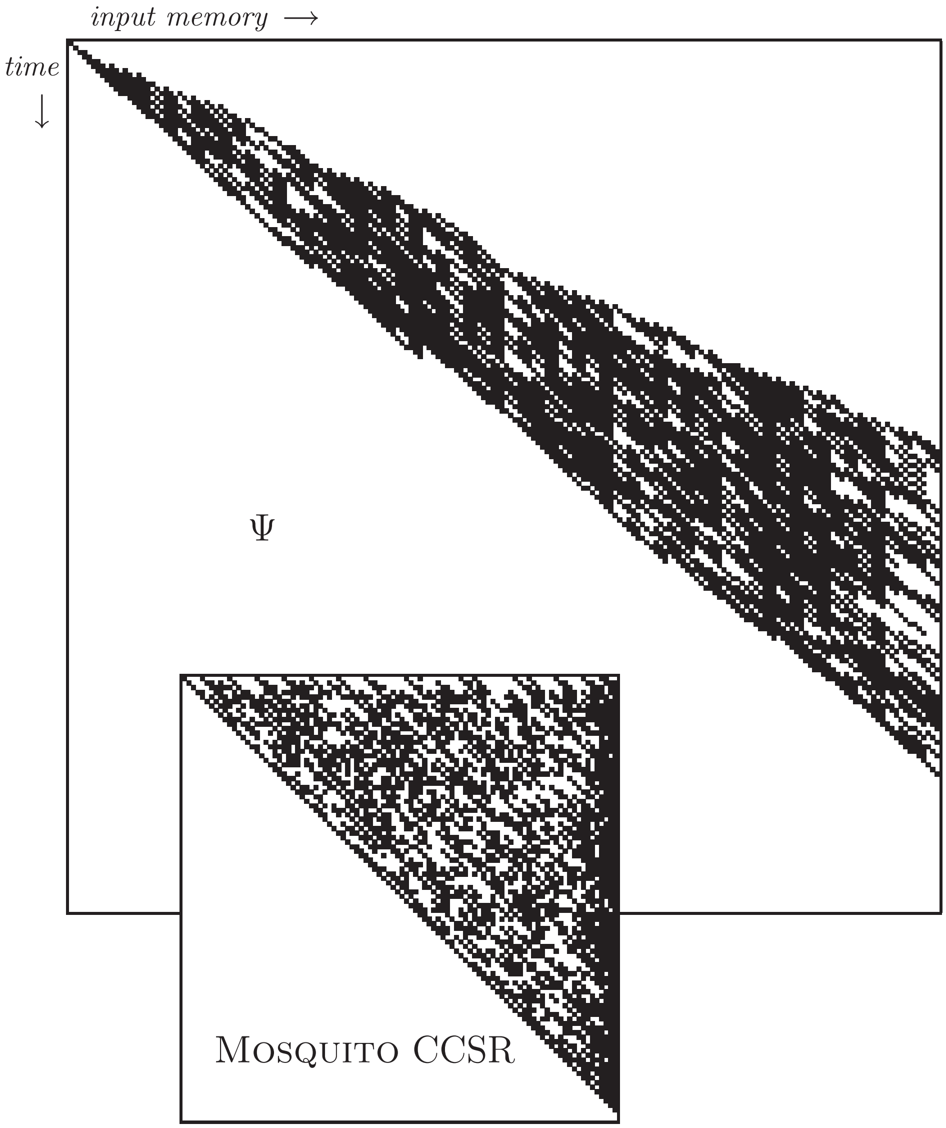

Figure 8: Difference propagation patterns in Ψ and the Mosquito CCSR.