Correlated Keystreams in MOUSTIQUE

Emilia Käsper[1], Vincent Rijmen[1,3], Tor E. Bjørstad[2], Christian Rechberger[3], Matt Robshaw[4] and Gautham Sekar[1]

[1] K.U.Leuven, ESAT-COSIC
[2] The Selmer Center, University of Bergen
[3] Graz University of Technology
[4] France Télécom Research and Development

Abstract. MOUSTIQUE is one of the sixteen finalists in the eSTREAM stream cipher project. Unlike the other finalists it is a self-synchronising cipher and therefore offers very different functional properties, compared to the other candidates. We present simple related-key phenomena in MOUSTIQUE that lead to the generation of strongly correlated keystreams and to powerful key-recovery attacks. Our best key-recovery attack requires only 2^{38} steps in the related-key scenario. Since the relevance of related-key properties is sometimes called into question, we also show how the described effects can help speed up exhaustive search (without related keys), thereby reducing the effective key length of MOUSTIQUE from 96 bits to 90 bits.

Keywords: eSTREAM, MOUSTIQUE, related keys

1 Introduction

eSTREAM [6] is a multi-year effort to identify promising new stream ciphers. Sponsored by the ECRYPT Network of Excellence, the project began in 2004 with proposals for new stream ciphers being invited from industry and academia. These proposals were intended to satisfy either a software-oriented or a hardware-oriented profile (or both if possible). The original call for proposals generated considerable interest with 34 proposals being submitted to the two different performance profiles. Among them was MOSQUITO [3], a self-synchronising stream cipher designed by Daemen and Kitsos.

As a self-synchronising stream cipher MOSQUITO was already a rather unusual submission. There was only one other self-synchronising stream cipher submitted, SSS [8]. Indeed it has long been recognised that the design of (secure) self-synchronising stream ciphers is a difficult task and attacks on SSS [5] and MOSQUITO [7] were proposed. As a result of the attack on MOSQUITO, a tweaked-variant of MOSQUITO, called MOUSTIQUE [4], was proposed for the second phase of analysis. This cipher is now one of the finalists in the eSTREAM project.

In this paper we describe a set of simple related-key pairs for MOUSTIQUE. Our observation illustrates unfortunate aspects of the tweaks in moving from

MOSQUITO to MOUSTIQUE. They lead directly to a very strong distinguisher for the keystream generated from two related keys; and further to a rather devastating key-recovery attack in the related-key setting [2]. In fairness it should be observed that related-key phenomena are not to everyone's taste [1]. Indeed, Daemen and Kitsos state that they make no claim for the resistance of the cipher to attackers that may manipulate the key. However, we take the view that related-key weaknesses might be viewed as certificational and that the very simple partition of the keyspace according to correlated keystreams is not particularly desirable. Aside from very efficient distinguishers and key-recovery attacks in the related-key setting, the related keys also lead to an improvement over exhaustive search in a non-related-key setting.

The paper is organised as follows. In the next section we describe MOUSTIQUE and make the observations that we need for attacks in Section 3. We then describe attacks on MOUSTIQUE in the related-key setting in Section 4 and use Section 5 to describe implications on key recovery in the standard (known keystream) setting. We summarise our experimental confirmation in Section 6 and close with our conclusions. Throughout we will use established notation.

2 Description of MOUSTIQUE

In this section we describe the parts of the MOUSTIQUE description that are relevant to our observations. More information can be found in [4]. MOUSTIQUE uses a key of 96 bits, denoted by k_j, with $0 \leq j \leq 95$. At each step MOUSTIQUE takes as input one bit of ciphertext and produces one bit of keystream.

MOUSTIQUE consists of two parts: a 128-bit conditional complementing shift register (CCSR) holding the state and a nonlinear output filter with 8 stages, see Figure 1.

2.1 The CCSR

The CCSR is divided into 96 cells, denoted by q^j with $1 \leq j \leq 96$. Each cell contains between 1 and 16 bits, denoted by q^j_i. The updating function of the CCSR is given by:

$$\begin{aligned}
Q^j_0 &= g_x(q^{j-1}_0, k_{j-1}, 0, 0), \qquad j = 1, 2, \\
Q^j_i &= g_x(q^{j-1}_i, k_{j-1}, q^v_i, q^w_i), \quad 2 < j < 96, \forall i \text{ and } j = 96, i = 0 \\
Q^{96}_i &= g_2(q^{95}_i, q^{95-i}_0, q^{94}_i, q^{94-i}_1),\ i = 1, 2, \ldots 15.
\end{aligned} \tag{1}$$

The Q^j_i are the new values of the q^j_i after one iteration. The subscript indices are always taken modulo the number of bits in the particular cell. The values of x, v and w are defined in Table 1. A value 0 for v or w indicates that the ciphertext feedback bit is used as input. The g_x functions are defined as follows:

$$g_0(a, b, c, d) = a + b + c + d \tag{2}$$

$$g_1(a, b, c, d) = a + b + c(d + 1) + 1 \tag{3}$$

$$g_2(a, b, c, d) = a(b + 1) + c(d + 1) \tag{4}$$

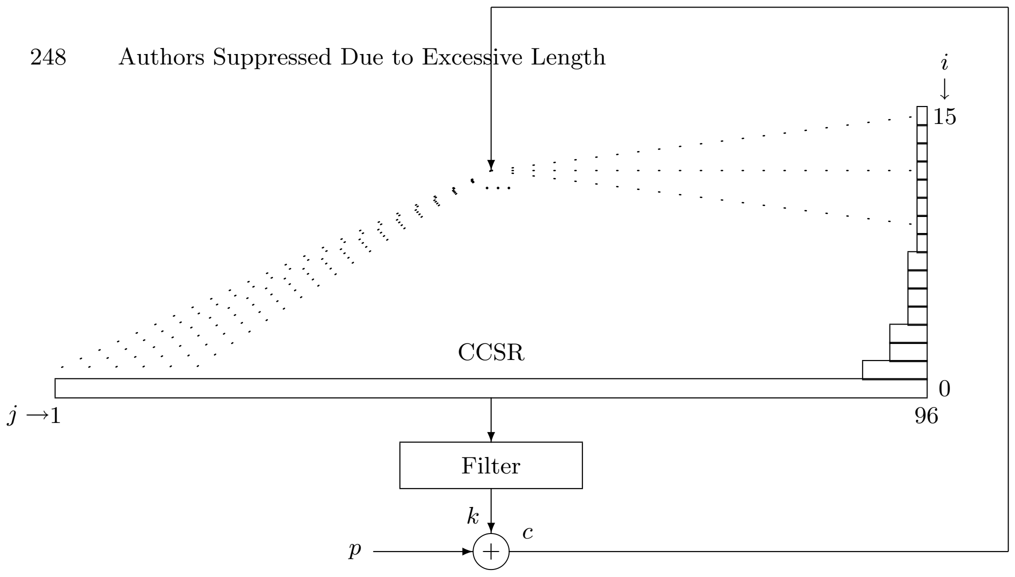

Fig. 1. State and filter of MOUSTIQUE. The only difference to MOSQUITO is that 1/3 of MOUSTIQUE state is now updated using a linear function g_0 to improve diffusion within the CCSR.

Table 1. The use of the functions g_0 and g_1 in the CCSR.

Index	Function	v	w
$(j-i) \equiv 1 \bmod 3$	g_0	$2(j-i-1)/3$	$j-2$
$(j-i) \equiv 2 \bmod 3$	g_1	$j-4$	$j-2$
$(j-i) \equiv 3 \bmod 6$	g_1	0	$j-2$
$(j-i) \equiv 0 \bmod 6$	g_1	$j-5$	0

Addition and multiplication operations are over GF(2).

2.2 The filter

The first stage of the filter compresses the 128 bits of the CCSR to 53 bits. First, the filter input $a^0 = (a^0_1, \ldots, a^0_{128})$ is obtained by re-indexing the CCSR cells q^j_i

in the following way:

$$\begin{aligned}
a_i^0 &= q_0^i, && 1 \le i \le 96 \\
a_i^0 &= q_1^{i-8}, && 97 \le i \le 104 \\
a_i^0 &= q_2^{i-12}, && 105 \le i \le 108 \\
a_i^0 &= q_3^{i-16}, && 109 \le i \le 112 \\
a_{105+2i}^0 &= q_i^{95}, && 4 \le i \le 7 \\
a_{106+2i}^0 &= q_i^{96}, && 4 \le i \le 7 \\
a_{113+i}^0 &= q_i^{96}, && 8 \le i \le 15.
\end{aligned} \tag{5}$$

Then, the 53 bits of output are obtained by taking 53 applications of g_1:

$$a_{4i \bmod 53}^1 = g_1(a_{128-i}^0, a_{i+18}^0, a_{113-i}^0, a_{i+1}^0), 0 \le i < 53. \tag{6}$$

The next four stages of the filter iteratively transform these 53 bits in a non-linear fashion. The sixth stage compresses the 53 bits to 12. Finally the last two stages exclusive-or these 12 bits together to produce a single bit of keystream. For simplicity, we omit the full description of the filter and refer the reader to the cipher specifications [4]. However, we note that the only non-linear filter component is the function g_1.

3 Observations on MOUSTIQUE

In this section we provide the basic observations that we will need in the paper. Some have already been observed in previous work [7].

3.1 Limited impact of the IV

Observation 1 *The IV of* MOUSTIQUE *influences only the first 105 bits of the keystream.*

This is a consequence of the fact that the IV of MOUSTIQUE is used only to initialize the state, and as every self-synchronising stream cipher does, it gradually overwrites its state with ciphertext bits.

3.2 Differential trails in the filtering function

As was done in the attack on MOSQUITO [7], we can make some simple observations on the filter function of MOUSTIQUE.

We note that the first stage of the filter is compressing and that no new information enters the filter after this stage. This leads to the first observation:

Observation 2 *Any two 128-bit CCSR states that produce the same 53-bit output after the first stage of filtering also produce an equal keystream bit.*

Recall that the first stage of the MOUSTIQUE output filter only uses the function $g_1(a,b,c,d) = a + b + c(d+1) + 1$. So a consequence of this observation is that if we flip input c, the output of g_1 is unaffected with probability $p = \Pr[d = 1]$. Similarly, if we flip d, the output is unaffected with probability $p = \Pr[c = 0]$. To exploit this, we can observe the following:

Observation 3 *State bits $q_0^1, \ldots, q_0^{17}$ and $q_0^{71}, \ldots, q_0^{75}$ are used in the filter input only in one location, and only as the third or fourth input to the function g_1.*

Suppose we flip one of these 22 bits. The two outputs of g_1 are equal with probability p and, consequently, the two outputs of the filter will be equal with probability $0.5 + p/2$. If the inputs to g_1 are balanced, then we have $p = 0.5$ and the probability the output bit is unchanged is 0.75 (*i.e.* with bias $\varepsilon = 0.25$).

3.3 Impact of key bits on the CCSR

The chosen ciphertext attack on MOSQUITO [7] exploited slow diffusion within the CCSR and so the state-update function of MOSQUITO was tweaked. The state update of MOUSTIQUE uses a linear function g_0 for updating one third of the state bits. While this improves the worst-case diffusion, it exhibits weaknesses that we exploit to construct related-key pairs that result in highly correlated keystreams.

MOUSTIQUE only uses key bits in the state-update function of the CCSR. Each of the 96 key bits is added to one of the 96 bits q_0^j. The state-update function of the CCSR introduces diffusion in one direction only: a cell with index j does not depend on cells with indices $j' > j$. An immediate consequence is that key bit k_{95} affects state bit q_0^{96} only.

There are however more useful implications, which we introduce next. By expanding (1) and Table 1, we obtain the following equations:

$$\begin{aligned}
Q_0^1 &= c + k_0 \\
Q_0^2 &= q_0^1 + k_1 + 1 \\
Q_0^3 &= q_0^2 + k_2 + c(q_0^1 + 1) + 1 \\
Q_0^4 &= q_0^3 + k_3 + q_0^2 + q_0^2 \\
Q_0^5 &= q_0^4 + k_4 + q_0^1(q_0^3 + 1) + 1 \\
Q_0^6 &= q_0^5 + k_5 + q_0^1(c + 1) + 1 \\
Q_0^7 &= q_0^6 + k_6 + q_0^4 + q_0^5 \\
Q_0^8 &= q_0^7 + k_7 + q_0^4(q_0^6 + 1) + 1 \\
Q_0^9 &= q_0^8 + k_8 + c(q_0^7 + 1) + 1 \\
Q_0^{10} &= q_0^9 + k_9 + q_0^6 + q_0^8 \\
Q_0^{11} &= q_0^{10} + k_{10} + q_0^7(q_0^9 + 1) + 1 \\
&\vdots
\end{aligned}$$

Here c denotes the ciphertext feedback bit and we observe the following:

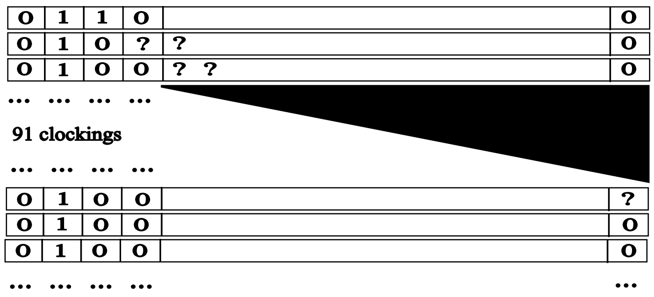

Fig. 2. CCSR differential propagation using related keys $k = (k_0, k_1, k_2, k_3, \ldots, k_{95})$ and $k^* = (k_0, k_1 + 1, k_2 + 1, k_3, \ldots, k_{95})$.

Observation 4 *In the computation of Q_0^4, the bit q_0^2 is cancelled. Only bit Q_0^3 depends on q_0^2.*

This leads to a related-key pair that for *any* ciphertext produces CCSR states with a one-bit difference. To see this, consider two instantiations of Moustique running in decryption mode with the two keys denoted by k and k^*. Assume

$$\begin{aligned} k_i &= k_i^* \quad \text{for } i \neq 1, 2 \text{ and} \\ k_i &= k_i^* + 1 \quad \text{for } i = 1, 2. \end{aligned}$$

We use both instantiations of Moustique to decrypt the same ciphertext and observe the propagation of differences through the CCSR cells.

In the first iteration of the CCSR, the differences in k_1 and k_2 will cause differences in Q_0^2 and Q_0^3. After the second iteration, there will again be a difference in Q_0^2, but not in Q_0^3, because the incoming difference in q_0^2 cancels out the difference in k_2. What is left of course is the difference in q_0^3, which propagates through the CCSR and the filter stages. However, after 92 iterations, this unwanted difference has been propagated out of the CCSR. We obtain a steady state behavior: at every iteration, both CCSRs differ in bit q_0^2 only. Figure 2 illustrates the propagation of the related-key differential within the CCSR.

Since Moustique has a *cipher function delay* of nine iterations, we can start deriving information from the keystream after nine more iterations. This will be demonstrated in the next section.

4 Related-key effects

4.1 Correlated keystreams

There are several classes of related keys for Moustique. We start with the simplest case which, coincidentally, appears to demonstrate the greatest bias.

Table 2. Related-key pairs and correlated keystreams. All these related-key pairs, and the magnitude of the correlation, have been experimentally verified.

Position j of the single bit difference	Key bits to flip to induce the required difference	Probability $z = z*$
2	1,2	0.8125
5	4,5,6	0.75
8	7,8,9,12	0.75
11	10,11,12	0.75
14	13,14,15,21	0.75
17	16,17,18	0.75
71	70,71,72	0.75
74	73,74,75	0.75

First related-key pairs. Consider two CCSR states with a difference only in bit q_0^2. According to Observation 4, this bit affects only one bit of the 53-bit output, namely bit a_8^1, which is computed as

$$a_8^1 = q_{14}^{96} + q_0^{19} + q_3^{96}(q_0^2 + 1) + 1.$$

Notice that if $q_3^{96} = 0$, the difference is extinguished and the two states produce equal output. If $q_3^{96} = 1$, the difference passes on and the two outputs will presumably collide with probability $\frac{1}{2}$. In fact q_3^{96} is computed using a non-balanced function g_2 and we have that $\Pr[q_3^{96} = 0] = \frac{5}{8}$.

So, after 105 cycles of IV setup, the two instances of Moustique decrypting equal ciphertexts with related keys k and k^* will produce equal keystream bits z and z^* for which $\Pr[z = z^*] = \frac{5}{8} + \frac{3}{8} \times \frac{1}{2} = \frac{13}{16}$.

More advanced related-key pairs. We can extend the simple related keys already described. This allows us to obtain a range of related-key pairs that generate a 1-bit difference in the CCSR. Using Table 1, the following observation is easy to verify.

Observation 5 *If $j \leq 77$ and $j \equiv 2 \bmod 3$, then q_0^j occurs in the CCSR update only linearly.*

This implies that for each of $q_0^5, q_0^8, q_0^{11}, q_0^{14}, \ldots, q_0^{77}$, we can find a set of key bits such that by flipping these key bits simultaneously and iterating the scheme, a one-bit difference between the two CCSRs is retained in a single bit position q_0^j.

Among these 25 one-bit differences in the CCSR state, eight will also induce correlated keystream; these are bits $q_0^2, q_0^5, q_0^8, q_0^{11}, q_0^{14}, q_0^{17}, q_0^{71}$ and q_0^{74} (Observation 3). Table 2 lists the pairs of related keys that are generated along with the correlation in the associated keystream outputs. Since the correlation is extremely high, only a very small amount of keystream is required to reliably distinguish these related keystreams from a pair of random keystreams.

Furthermore, by simultaneously flipping relevant key bits for two or more indices j, we obtain a range of related keys with weaker correlation. The bias can be estimated by the Piling-Up Lemma; in the weakest case where all 8 keybit tuples are flipped, it is approximately $\varepsilon = 2^{-8.6}$. We have verified this estimate experimentally, and we now make the following conclusion.

Observation 6 *Each key of* MOUSTIQUE *produces correlated keystream with (at least)* $2^8 - 1 = 255$ *related keys, with the bias ranging from* $\varepsilon = 2^{-1.7}$ *to* $\varepsilon = 2^{-8.6}$.

4.2 Key-recovery attacks

A distinguisher can often be exploited to provide a key-recovery attack, and this is also the case here. Using (6) with $i = 42$, (5), and the definition of g_1 we have that

$$a_9^1 = q_0^{86} + q_0^{60} + q_0^{71}(q_0^{43} + 1) + 1.$$

As described in Section 4.1, if we take two instantiations of MOUSTIQUE and flip the key bits k_{70}, k_{71}, and k_{72} in one instantiation, then only q_0^{71} will change. This change can only propagate to the output if the bit q_0^{43} equals zero. Thus, a difference in the output of two copies of Moustique running with these related keys gives us one bit of information about the CCSR state (the value $q^{43} = 0$). Furthermore, the state bit q_0^{43} only depends on the first 43 bits of the key, which leads to an efficient divide-and-conquer attack as follows.

We first observe the output of two related instances of MOUSTIQUE, using some (arbitrary) ciphertext c and record the time values where the output bits differ. We then guess 43 key bits $k_0, \ldots, k_{42}$, compute the state bit q_0^{43} under the same ciphertext c, and check whether indeed $q_0^{43} = 0$ for all the recorded values. If there is a contradiction then we know that our guess for the 43-bit subkey was wrong. On average, only 8 bits of keystream are required to eliminate wrong candidates; and n bits of keystream eliminate a false key with probability $1 - 2^{-n/4}$.

The final attack requires a slight adjustment, as the existence of related keys introduces some false positives. Namely, certain related keys produce extinguishing differential trails that never reach q_0^{43}. For example, if the guessed key only differs from the correct key in the bits k_1 and k_2 then this difference affects q_0^2 only, and not q_0^{43}. Thus, the key with bits k_1 and k_2 flipped passes our test. The same holds for all combinations of the 14 values of j smaller than 43 and with $j \equiv 2 \bmod 3$; as well as bit k_{39} and pair k_{41}, k_{42}. Altogether, we have found that out of the 2^{43} key candidates, 2^{16} survive and after running our attack we still need to determine $96 - (43 - 16) = 69$ key bits. This can be done by exhaustive key search, and the 2^{69} complexity of this stage dominates the attack.

Notice that in the first stage, we know in advance which related keys give false positives. Thus, we only need to test one key in each set of 2^{16} related keys, and the complexity of the first stage is $2^{43-16} = 2^{27}$. The complexity of the second stage can be reduced if we were to allow the attacker access to multiple related keys.

In such a case, a second stage to the attack would use (6) with $i = 16$:

$$a^1_{11} = q^{96}_3 + q^{34}_0 + q^{89}_1(q^{17}_0 + 1) + 1.$$

The state bit q^{17}_0 can be changed by flipping k_{16}, k_{17} and k_{18}. The state bit q^{89}_1 depends on 89 key bits, of which we know already $43 - 16 = 27$ bits. In addition, we found 2^{31} related-key differentials that extinguish without ever reaching q^{89}_1. Hence, we need to test $2^{89-27-31} = 2^{31}$ keys to determine 31 more bits. In total we have then determined $27 + 31 = 58$ bits of the key and the remaining 38 bits can be determined by exhaustive search. The complexity of the attack can be estimated by $2^{27} + 2^{31} + 2^{38}$ which is dominated by the third brute-force phase.

We have verified the first two stages of the attack experimentally, and are indeed able to recover 58 bits of the key, given only 256 bits of keystream from two related-key pairs. Recovering the first 27 bits requires only a few minutes and 256 bits of output from a single related-key pair.

5 Accelerated exhaustive key search

Next, we show how the existence of related keys in MOUSTIQUE can be used in cryptanalysis even if we cannot observe the output of the cipher in a related-key setting.

In Section 4, we observed that each key has eight related keys that produce strongly correlated output. In particular, the correlation can be detected from very short keystream. Thus, we can imagine the following attack scenario: given, say, 128 bits of cipher output from a key-IV pair (k, IV), compare this to the output of the cipher, using a candidate key k', the same IV and equal ciphertext. If the outputs are not correlated, eliminate key k' as well as its 8 related keys.

In order to compete with brute force, we need to be able to eliminate related keys efficiently. We now discuss two strategies representing different trade-offs between required keystream and computational complexity.

5.1 The strong correlation attack

In the first approach we use the $(7, 4)$ Hamming code. As Hamming codes are perfect, we know that for each 7-bit string s, there exists a codeword c_i such that the Hamming distance between s and c_i is at most one. The codewords of the $(7, 4)$ Hamming code are listed in Table 3.

Now, for each codeword c_i, we fix candidate key bits $k_1, k_4, k_7, k_{10}, k_{13}, k_{16}, k_{70}$ to this codeword, and exhaustively search over the remaining 89 key bits. This strategy guarantees that we test either the correct key or one of the closely related keys given in Table 2. A related key can then be easily detected from the strong correlation of the two keystreams. For example, assume that the correct subkey is $(k_1, k_4, k_7, k_{10}, k_{13}, k_{16}, k_{70})$ and the closest codeword is $(k_1, k_4 + 1, k_7, k_{10}, k_{13}, k_{16}, k_{70})$. Then, according to Table 2, $k^* = (k_1, k_2, k_3, k_4 + 1, k_5 + 1, k_6 + 1, k_7, \ldots, k_{95})$ is a related key that has been selected for testing.

Table 3. The codewords of the (7,4) Hamming code.

c_0	0000000	c_4	0100110	c_8	1000101	c_{12}	1100011
c_1	0001011	c_5	0101101	c_9	1001110	c_{13}	1101000
c_2	0010111	c_6	0110001	c_{10}	1010010	c_{14}	1110100
c_3	0011100	c_7	0111010	c_{11}	1011001	c_{15}	1111111

Our experiments suggest that 128 keystream bits are sufficient to detect correlation between the correct key k and a related candidate key k^* (see Sect. 6 for experimental results). Given that IV setup takes 105 cipher clocks, the total worst-case complexity of our attack is $(105+128)\cdot 2^4 \cdot 2^{89} \approx 2^{100.9}$ cipher clocks. In comparison, naive brute force requires on average 2 keystream bits to eliminate false candidates, so the complexity is $(105+2)\cdot 2^{96} = 2^{102.7}$ cipher clocks.

5.2 The piling-up attack

Following Observation 6, we partition the keys into 2^{88} sets of 2^8 related keys and test only one key in each set. After 105 clocks of IV setup, the states corresponding to two related keys differ in at most 8 bits (given in Table 2). If

$$a^0_{40} = a^0_{43} = 1 \text{ and } a^0_{97} = a^0_{100} = a^0_{103} = a^0_{106} = a^0_{109} = a^0_{112} = 0, \qquad (7)$$

then none of these 8 bits influences a^1, the output of the first filter stage, and hence the keystream bits generated by two related keys are equal. Consequently, if, while testing a key k' we observe that the bit of keystream generated by k' differs from the bit of the observed keystream at a time when the candidate state satisfies (7), then we are sure that the key k we are looking for is not a related key of k' and we can discard k' as well as its $2^8 - 1$ related keys.

To estimate the amount of keystream needed to eliminate wrong keys, we note that two unrelated keystreams overlap with probability $\frac{1}{2}$, so we can use half of the available keystream to test for condition (7). As $\Pr[a^0_{112} = 0] = \frac{5}{8}$, while the remaining bits in (7) are balanced, condition (7) is true with probability $p = \frac{5}{8} \cdot \frac{1}{2^7}$. Thus, we need to generate on average $\frac{2}{p} = 409.6$ bits of keystream from one candidate key in order to rule out an entire class of 2^8 related keys. In total, the complexity of our attack can be estimated at $(105+409.6)\cdot 2^{88} = 2^{97.0}$ cipher clocks. Our experiments confirm this estimate and suggest that 5000-6000 bits of known keystream are sufficient to eliminate all false candidates with high confidence.

Both our strategies for accelerated exhaustive key search are rather simple and just as easily parallelisable as exhaustive search, so they are likely to provide an advantage over simple brute force in practice. The piling-up attack is an estimated 50 times faster than exhaustive key search, indicating that the effective key length of MOUSTIQUE is reduced to 90 bits instead of the claimed 96-bit security.

6 Experimental verification

The results in this paper were verified using the source code for MOUSTIQUE that was submitted to eSTREAM [6]. All sets of key bits identified in Table 2 were tested with one thousand random keys and their related partners. The minimum, maximum, and average number of agreements between the two generated keystreams, over the first 128 bits, was recorded. Note that for un-correlated keystreams we would expect 64 matches.

Key bits to induce the required difference	Minimum # of matches	Maximum # of matches	Average # of matches
1,2	91	118	104.02
4,5,6	82	111	96.10
7,8,9,12	79	109	96.03
10,11,12	74	108	95.81
13,14,15,21	79	110	96.11
16,17,18	80	114	95.72
70,71,72	77	109	96.23
73,74,75	81	112	95.94

We then constructed a distinguisher by setting the agreement threshold to $t \geq 74$. We chose randomly 10 000 related-key pairs, all of which passed the test, indicating that the false negative rate is below 0.01%. In comparison, out of 10 000 128-bit keystreams obtained from random key pairs, 440 passed the test, so the false positive rate was below 5%. Thus, we can use our accelerated key search to eliminate 95% of the keys, and then brute-force the remaining candidates. The total complexity of the attack is still below that of naive exhaustive search, and the success rate is at least 99.99%.

7 Conclusions

In moving from MOSQUITO, it seems that the design of the self-synchronizing stream cipher MOUSTIQUE was established in a rather *ad hoc* way. While the tweaked design resists the chosen-ciphertext attack on MOSQUITO, we showed that it still exhibits weaknesses that lead to strong distinguishers in the related-key setting. Further, we presented two different strategies for exploiting those distinguishers in a key-recovery attack. The first strategy allows the attacker to recover the 96-bit secret key in 2^{69} steps, assuming that the attacker is able to observe the output of two instances of the cipher using the secret key and a related key. The complexity of this attack can be reduced to 2^{38} steps if the attacker is able to observe the output of three instances of the cipher using the secret key and two related keys. Both require a negligible amount of ciphertext, *e.g.* less than 256 bits.

We have also exploited the observations we made in a non-related-key attack. Our first attack breaks the cipher in around 2^{101} steps, using only 128 bits of known plaintext. If furthermore a few thousand keystream bits are known, the complexity is reduced to 2^{97} steps. In comparison, exhaustive search would take 2^{103} equivalent steps, indicating that MOUSTIQUE falls about 6 bits short of the claimed 96-bit security. While, admittedly, a 2^{97} attack is still far from being practical, it illustrates the relevance of related-key weaknesses in the standard (non-related-key) setting.

Acknowledgments. Emilia Käsper thanks the Computer Laboratory of the University of Cambridge for hosting her while this work was done.

This work was supported in part by the European Commission through the IST Programme under Contract IST-2002-507932 ECRYPT, the IAPP–Belgian State–Belgian Science Policy BCRYPT and the IBBT (Interdisciplinary institute for BroadBand Technology) of the Flemish Government. Emilia Käsper is also partially supported by the FWO-Flanders project nr. G.0317.06 Linear Codes and Cryptography.

References

1. D.J. Bernstein. Related-key attacks: who cares? *eSTREAM discussion forum*, June 22, 2005. Available at `http://www.ecrypt.eu.org/stream/phorum/`.
2. E. Biham. New Types of Cryptoanalytic Attacks Using related Keys (Extended Abstract). In T. Helleseth, editor, *Advances in Cryptology - EUROCRYPT 1993,* Lecture Notes in Computer Science, volume 765, pages 398–409, Springer Verlag, 1994.
3. J. Daemen and P. Kitsos. The Self-Synchronising Stream Cipher MOSQUITO. eStream Report 2005/018. Available at `http://www.ecrypt.eu.org/stream/papers.html`.
4. J. Daemen and P. Kitsos. The Self-Synchronising Stream Cipher MOUSTIQUE. Available at `http://www.ecrypt.eu.org/stream/mosquitop3.html`.
5. J. Daemen, J. Lano, and B. Preneel. Chosen Ciphertext Attack on SSS. eStream Report 2005/044. Available at `http://www.ecrypt.eu.org/stream/papers.html`.
6. ECRYPT. The eSTREAM project, `http://www.ecrypt.eu.org/stream/`.
7. A. Joux and F. Muller. Chosen-ciphertext attacks against Mosquito. In M.J.B. Robshaw, editor, *Proceedings of FSE 2006*, Lecture Notes in Computer Science, volume 4047, pages 390–404, Springer Verlag, 2006.
8. G. Rose, P. Hawkes, M. Paddon, and M. Wiggers de Vries. Primitive Specification for SSS. eStream Report 2005/028. Available at `http://www.ecrypt.eu.org/stream/papers.html`.