Analysis of the Non-linear Part of Mugi

Alex Biryukov[1]* and Adi Shamir[2]

[1] Katholieke Universiteit Leuven, Dept. ESAT/SCD-COSIC,
Kasteelpark Arenberg 10, B–3001 Heverlee, Belgium
`http://www.esat.kuleuven.ac.be/~abiryuko/`

[2] Department of Applied Mathematics and Computer Science,
Weizmann Institute of Science, Rehovot 76100, Israel,
`shamir@wisdom.weizmann.ac.il`

Abstract. This paper presents the results of a preliminary analysis of the stream cipher Mugi. We study the nonlinear component of this cipher and identify several potential weaknesses in its design. While we can not break the full Mugi design, we show that it is extremely sensitive to small variations. For example, it is possible to recover the full 1216-bit state of the cipher and the original 128-bit secret key using just 56 words of known stream and in 2^{14} steps of analysis if the cipher outputs any state word which is different than the one used in the actual design. If the linear part is eliminated from the design, then the secret non-linear 192-bit state can be recovered given only three output words and in just 2^{32} steps. If it is kept in the design but in a simplified form, then the scheme can be broken by an attack which is slightly faster than exhaustive search.

Keywords: Cryptanalysis, Stream ciphers, Mugi.

1 Introduction

Mugi is a fast 128-bit key stream cipher [4] designed for efficient software and hardware implementations (achieves speeds which are 2-3 times faster than Rijndael in hardware and slightly faster in software). The cipher was selected for standardization by the Japanese government project CRYPTREC and is also one of the two proposed ISO stream cipher standards.

Previous analysis of Mugi given by its designers in [1] concentrated on linear cryptanalysis and resynchronization attacks. A recent work by Golic [3] studied only the linear component of the cipher. No security flaw of the full cipher has been reported so far.

* This author's work was supported in part by the Concerted Research Action (GOA) Mefisto 2000/06 and Ambiorix 2005/11 of the Flemish Government and in part by the European Commission through the IST Programme under Contract IST-2002-507932 ECRYPT. The information in this document reflects only the author's views, is provided as is and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

In this paper we study the non-linear component of this cipher, and identify several potential weaknesses in its design. However we do not make any claim as to the security of the full MUGI which remains unbroken.

This paper is organized as follows: in Sect. 2 we describe the cipher MUGI, in Sect. 3 we describe two attacks on its non-linear component, and Sect. 4 concludes the paper.

2 Description of Mugi

The design of MUGI is based on the design philosophy proposed by J. Daemen and C. Clapp in their stream cipher PANAMA [2]. The internal state of the cipher at time t consists of two parts: a linearly changed large buffer $b^{(t)}$, and a non-linearly evolving shorter state $a^{(t)}$. See Fig. 1 for a schematic description. The

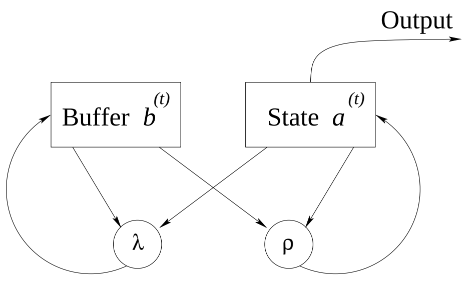

Fig. 1. The evolution of the MUGI state.

cipher works in steps called rounds. At each round the internal state is updated and one of its three 64-bit words is produced as output. The evolution of the buffer which consists of sixteen 64-bit words happens in a LFSR-like slow fashion, together with a 64-bit feedback from the nonlinear state at each round:

$$b^{(t+1)} = \lambda(b^{(t)}, a^{(t)}),$$

or more explicitly:

$$\begin{aligned} b_j^{(t+1)} &= b_{j-1}^{(t)} (j \neq 0, 4, 10) \\ b_0^{(t+1)} &= b_{15}^{(t)} \oplus a_0^{(t)} \\ b_4^{(t+1)} &= b_3^{(t)} \oplus b_7^{(t)} \\ b_{10}^{(t+1)} &= b_9^{(t)} \oplus (b_{13}^{(t)} \lll 32) \end{aligned}$$

The evolution of the state $a^{(t)}$ is essentially a block cipher-like invertible process, in which two 64-bit words coming from the buffer are used as subkeys:

$$a^{(t+1)} = \rho(a^{(t)}, b^{(t)}),$$

or more explicitly:

$$\begin{aligned}
{a_0}^{(t+1)} &= {a_1}^{(t)} \\
{a_1}^{(t+1)} &= {a_2}^{(t)} \oplus F({a_1}^{(t)}, {b_4}^{(t)}) \oplus C_1 \\
{a_2}^{(t+1)} &= {a_0}^{(t)} \oplus F({a_1}^{(t)}, {b_{10}}^{(t)} \lll 17) \oplus C_2 \,,
\end{aligned}$$

where C_1, C_2 are known constants. The F function reuses an S-box and MDS of the AES as shown in Fig. 2. The output function of MUGI is very simple:

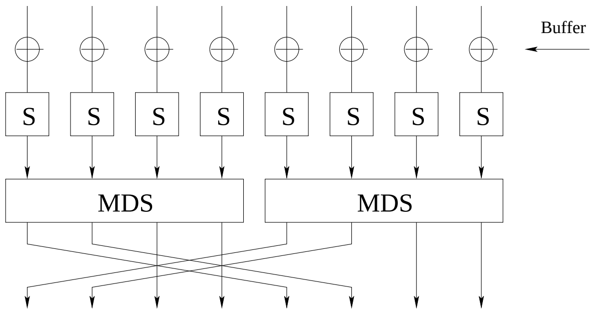

Fig. 2. The F-function.

$$Output[t] = a_2^{(t)},$$

i.e. one of the three 64-bit words in the internal state is given as output *prior* to the evaluation of the round t.

3 Cryptanalysis of the Two Variants of Mugi

In this section we cryptanalyse several variants of MUGI.

3.1 Change in the output function

The output function of MUGI outputs the 64-bit word $a_2^{(t)}$ before the t^{th} round. In this section we show that MUGI is very sensitive to small changes in the output function. For example, if the output would consist of the same word $a_2^{(t)}$ but *after* the evaluation of the round t (or equivalently the word $a_1^{(t+1)}$), then the full cipher could be easily broken with practical complexity. In general, for any choice of a output word other than the one used in the actual design (i.e., either $a_0^{(t)}$ or $a_1^{(t)}$) the cipher can be easily broken.

We denote by the Greek letters $\alpha, \beta, \gamma, \delta, \epsilon$ an intermediate value $a_1^{(t)}$ which is used as the input to the two F-functions in each round. Suppose that $Output[t] =$

$a_1^{(t)}$. Let us make several important observations. The first observation is that $a_1^{(t)} = a_0^{(t+1)}$ and thus we know the word that updates the buffer at each round. This would allow us to run the buffer update function $\lambda(\cdot)$ symbolically and write linear equations for the buffer bits, including those that enter the non-linear ρ function as "sub-keys". The second observation is that the word $a_1^{(t)}$ is used as the input to both F-functions of the round. The final observation is that we can write the following equation:

$$\alpha \oplus \delta = F(\beta, b_{10}^{(t+1)} \lll 17) \oplus F(\gamma, b_4^{(t+2)}). \tag{1}$$

In this equation α, β, γ and δ are words from the output stream and are thus known to the attacker (see also Fig. 3). Let us denote the inverse of the MDS

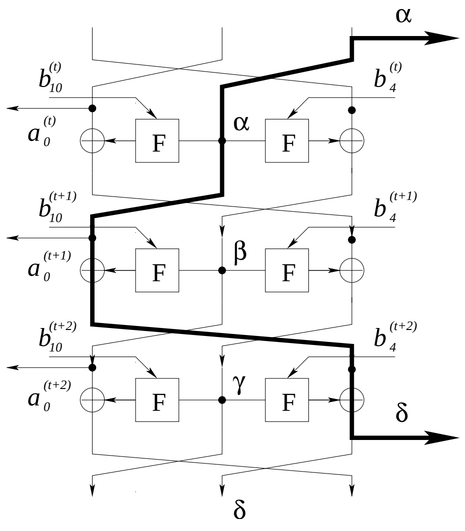

Fig. 3. The attack path.

and the byte swap layers by M^{-1} and the eight S-box layer by S. By applying the inverse of the linear layer of F to both sides of the equation we can get the simplified equation:

$$M^{-1}(\alpha \oplus \delta) = S(\beta \oplus (b_{10}^{(t+1)} \lll 17)) \oplus S(\gamma \oplus b_4^{(t+2)}). \tag{2}$$

which decomposes into 8 independent equations in the 8 bit values of each S-box. One of the possible ways to use this equation is to look for points in the output stream in which at least one of the bytes of $M^{-1}(\alpha \oplus \delta)$ is zero. For a particular S-box this would lead to a simpler equation:

$$S(\beta \oplus (b_{10}^{(t+1)} \lll 17)) = S(\gamma \oplus b_4^{(t+2)}), \tag{3}$$

which further simplifies to:

$$(b_{10}^{(t+1)} \lll 17) \oplus b_4^{(t+2)} = \beta \oplus \gamma. \tag{4}$$

This is an 8-bit linear constraint on the bits of the buffer. It is sufficient to gather about $1024/8 = 128$ such equations in order to get a system of linear equations for all the buffer bits. The probability that at least one zero value occurs in each 64-bit block is $1-(1-1/256)^8 \approx 2^{-5}$. Thus given about $2^5 \cdot 128 = 2^{12}$ output words the attacker would expect to obtain a solvable system of equations and reconstruct the full 1024-bit buffer. Knowing the buffer at some round t and the output words $a_1^{(t-1)}, a_1^{(t)}, a_1^{(t+1)}$ the attacker can recover the state $a^{(t)}$ as follows:

$$(a_0^{(t)}, a_1^{(t)}, a_2^{(t)}) = (a_1^{(t-1)}, a_1^{(t)}, F(a_1^{(t)}, b_4^{(t)}) \oplus a_1^{(t+1)}).$$

Since all the steps in the cipher are invertible, knowing the buffer and the state at some point t enables the attacker to run the cipher both forwards and backwards. By running the cipher backwards the attacker can recover the initial 128-bit secret key.

The total complexity of this attack is $O(2^{12})$ words of known stream (about 2^{15} bytes) and $O(2^{30})$ steps to solve a system of 2^{10} equations. The same attack would work for the case $Output[t] = a_0^{(t)}$.

An important remark is that the attack did not use any properties of the S-box, and thus can work even when the S-box is unknown or key-dependent. The attack will first recover the buffer using the technique described above and then derive the unknown S-boxes from Eq. 2 by writing a set of equations of the form

$$S(c_1) \oplus S(c_2) = c_3, \tag{5}$$

where c_1, c_2, c_3 are known values. Given $k+3$ output stream words the attacker can write $8k$ equation for the unknown S-box. He will need $O(256)$ 64-bit outputs due to a "coupon collector"-like argument in order to write and solve a system of linear equations in terms of the entries of the unknown S-box. Note that in this attack it is important that the S-box is invertible, since otherwise we can write Eq. (4) only probabilistically.

Another observation is that we can use the "buffer elimination" equations from [3] to attack this variant of MUGI with even smaller data and time complexity, since we will not need to spend $O(2^{30})$ operations in order to solve the system of linear equations.

The equations are as follows ($t \geq 48$, and $^{<32}$ stands for cyclic 32-bit rotation of a 64-bit word to the left):

$$\begin{aligned} a_1^{(t)} \oplus a_1^{(t-48)} \oplus F^{-1}(a_1^{(t+1)} \oplus a_2^{(t)} \oplus C_1) \oplus F^{-1}(a_1^{(t-47)} \oplus a_2^{(t-48)} \oplus C_1) = \\ a_1^{(t-6)} \oplus a_1^{(t-10)} \oplus a_1^{(t-14)} \oplus a_1^{(t-18)} \oplus a_1^{(t-26)} \oplus a_1^{(t-30)} \oplus a_1^{(t-34)} \oplus a_1^{(t-38)} \\ \oplus^{<32} a_1^{(t-26)} \oplus^{<32} a_1^{(t-42)} \end{aligned}$$

$$a_1^{(t)} \oplus a_1^{(t-48)} \oplus F^{-1}(a_1^{(t-1)} \oplus a_2^{(t+1)} \oplus C_2) \oplus F^{-1}(a_1^{(t-49)} \oplus a_2^{(t-47)} \oplus C_2) = \\ {}^{<17}a_1^{(t-12)} \oplus^{<17} a_1^{(t-16)} \oplus^{<17} a_1^{(t-32)} \oplus^{<17} a_1^{(t-44)} \oplus^{<49} a_1^{(t-16)} \\ \oplus^{<49} a_1^{(t-20)} \oplus^{<49} a_1^{(t-32)} \oplus^{<49} a_1^{(t-36)}$$

One notices two interesting properties of these two equations: there are only two instances of a_2 in each equation and they are 48 time steps appart in both equations. Thus if we assume that the sequence $a_1^{(t)}$ is known, those equations reduce to simpler equations:

$$F^{-1}(a_2^{(t)} \oplus C_1') \oplus F^{-1}(a_2^{(t-48)} \oplus C_1'') = const_1 \qquad (6)$$

$$F^{-1}(a_2^{(t)} \oplus C_2') \oplus F^{-1}(a_2^{(t-48)} \oplus C_2'') = const_2 \qquad (7)$$

in which all the quantities are known except for the two unknowns $a_2^{(t)}$ and $a_2^{(t-48)}$. If we denote $x = M^{-1}(a_2^{(t)})$ and $y = M^{-1}(a_2^{(t-48)})$, then we can take care of the linear mapping, and write further simplified equations:

$$S_1(x) \oplus S_2(y) = const_1 \qquad (8)$$

$$S_3(x) \oplus S_4(y) = const_2 \qquad (9)$$

where S-boxes S_1, S_2, S_3, S_4 are known and are derived from S^{-1} and the known constants. By solving the system we derive the full words $a_2^{(t)}$ and $a_2^{(t-48)}$ in just $8 \cdot 2^8 = 2^{11}$ steps. Repeating this procedure about eight times for $t, t+1, \dots, t+7$ we find eight consecutive words $a_2^{(t)}, a_2^{(t+1)}, \dots, a_2^{(t+7)}$, which allows us to directly derive the bits of the buffer without solving a system of linear equations. The complexity of this approach is about 2^{14} very simple steps and it uses about 56 output words and negligible memory[3]. This second attack uses the knowledge of the S-boxes and the special properties of the buffer update function, which were not important in our first attack.

3.2 Attacking the Non-linear part of Mugi

In this section we present an attack that efficiently recovers the 192-bit non-linear state of the cipher when part of the buffer is known to the attacker. It turns out that it is sufficient to know only two words of the buffer $b_4^{(t)}$ and $b_{10}^{(t)}$ in order to mount this attack. The attack uses only 3 output words and has a complexity of $O(2^{32})$ steps. Note that the cipher is unchanged, including the original output function of Mugi: $Output[t] = a_2^{(t)}$.

[3] The attack can be done even faster in only 2^6 steps, if we are allowed to do a single precomputation of 2^{32} steps in order to produce a table that would occupy 2^{32} bytes and would store the solution for the system of equations (8) and (9).

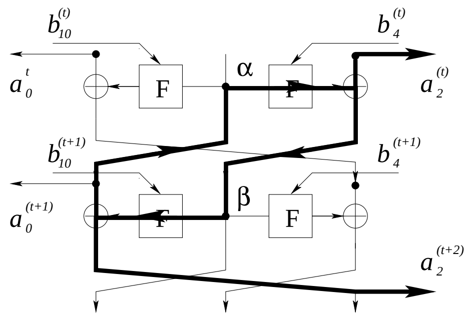

Fig. 4. The loop in the ρ function.

The attacker can consider the following loop in the ρ function (see also Fig. 4):

$$F(\alpha, b_4^{(t)}) \oplus a_2^{(t)} = \beta \tag{10}$$

$$F(\beta, b_{10}^{(t+1)}) \oplus a_2^{(t+2)} = \alpha. \tag{11}$$

Here the $a_2^{(t)}, a_2^{(t+2)}$ and the keys $b_4^{(t)}$, $b_{10}^{(t+1)}$ are known. Then we have a system of two non-linear but simple one-round equations with two variables α and β. In order to find the solution of the system the attacker may proceed as follows:

1. Guess the first four bytes of β (denoted 0,1,2,3, with 2^{32} possibilities). Here we enumerate the bytes of Fig. 2 from left to right, starting from 0.
2. For each possibility, find the middle four bytes (2,3,4,5) of α.
3. At this stage, we know two bytes (number 2,3) at the input and at the output of the first MDS at round t. The first two bytes (number 0,1) of the input can be found from a system of linear equations:

$$\begin{cases} x_0 + x_1 = c_0 \\ 3 \cdot x_0 + x_1 = c_1, \end{cases}$$

   where c_0, c_1 are known values computed from bytes 2,3 of input and output. Knowing the new input bytes x_0, x_1 we can calculate the missing output bytes y_0, y_1. As a result we will find two additional bytes of α and β and thus we know bytes number 0,1,2,3,4,5 of these words. Due to the byte swap after the MDS being an involution, we can get in a similar fashion a similar set of constraints knowing bytes 4,5 at the input and the output of the second MDS and solving a system of equations for the bytes 6,7 at the same round t. As a result we completely recover α and β.
4. At this point we know 128 bits of the state a by knowing $a_1^{(t)} = \alpha$, and the known output $a_2^{(t)}$. The remaining 64 bits can be found as follows:

$$a_0^{(t)} = F(\alpha, b_4^{(t)}) \oplus a_2^{(t+1)}.$$

We see that given two buffer words and 3 output words, we can completely recover the 192-bit secret state $a^{(t)}$ in $O(2^{32})$ simple steps. If this attack has to be performed multiple times for example as part of another attack on the full cipher – it can be sped up by precomputation.

3.3 Mixing one Buffer Word per Round

For MUGI it is crucial that two 64-bit buffer words are mixed at each round, while only a single 64-bit word is given as output. In this section we present an attack that recovers the full secret state of the cipher (both the state $a^{(t)}$ and the buffer $b^{(t)}$) when only one word is mixed per round. The rest of the cipher structure is kept intact. The complexity of this attack is equivalent to $O(2^{126.5})$ key searching steps steps, but the similarity between this complexity and exhaustive key search is accidental since the complexity of our attack would remain the same even if MUGI had much larger keys[4]. Note that this is much faster than the alternative type of exhaustive search over the 192-bit state and the 1024-bit buffer.

Consider a variant of MUGI in which both F-functions of each round share a common key[5], for example and without loss of generality, $b_4^{(t)}$. We could reuse the previous attack by guessing the two buffer words $b_4^{(t)}, b_4^{(t+1)}$ and recovering the state $a^{(t)}$ in $O(2^{160})$ steps. However there is a much faster direct attack:

1. At round t we know word $a_2^{(t)}$ and we guess the remaining parts of the non-linear state: $a_0^{(t)}, a_1^{(t)}$ (a 128-bit guess).
2. We find the buffer word $b_4^{(t)}$ used as the key in round t from the equation:

$$F(a_1^{(t)}, b_4^{(t)}) \oplus a_0^{(t)} = a_2^{(t+1)},$$

   in which all values except for the key are either guessed or known from the output stream.
3. We use the newly recovered key to compute the unknown word $a_1^{(t+1)}$ of the next state:

$$a_1^{(t+1)} = F(a_1^{(t)}, b_4^{(t)}) \oplus a_2^{(t)}.$$

4. Go to step 2, and repeat it 16 times for the following rounds till we know the full 16 word buffer. Once we start reusing buffer words we can check them for possible contradictions.

Note that since at each step we know the full non-linear state, we know the update function of the buffer and we can thus run the buffer update-function

[4] This complexity is about 3 times faster than exhaustive search of the actual 128-bit key of MUGI ; to verify each guessed key, MUGI has to perform 48 initial key setup rounds, whereas our attack verifies or discards a guessed pair of buffer words after only 16 rounds.

[5] The attack can also tolerate a construction in which given a buffer word b the keys of the round are derived via known and easy to invert permutations: $k_1 = f(b), k_2 = g(b)$. For example, a rotation of the buffer word by 17 as used in MUGI would not make the attack any harder.

$\lambda(\cdot)$ symbolically. The attack uses 18 output stream words, and its complexity is equivalent to $\frac{16}{48} \cdot 2^{128}$ key searching steps . The attack completely recovers the 1024+192-bit internal state of the cipher, and by using the invertibility of the operations we can also recover the 128-bit original secret key. The Table 1

Table 1. Our attacks on the non-linear component of Mugi.

Cipher Variant	Known Stream[a]	Time Complexity	The Attack Recovers
Change in the output function[b]	$O(2^{12})$	$O(2^{30})$	full 192+1024 state
Change in the output function[c]	56	$O(2^{14})$	full 192+1024 state
Attack on non-linear component	3	$O(2^{32})$	192 bits of $a^{(t)}$
One buffer word per round	18	$O(2^{126.4})$	full 192+1024 state
Exhaustive search	2	$O(2^{128})$	full 192+1024 state

[a] Expressed in 64-bit words.
[b] The same complexities even if S-boxes are unknown or key-dependent.
[c] Using the buffer elimination observation from [3] and the knowledge of the S-boxes.

summarizes the results of our attacks on the non-linear component of Mugi.

4 Conclusions

In this paper we have identified several potential weaknesses of Mugi:

- The output function reveals 1/3 of the state without any masking. It would be better to reveal fewer bits which are a more complex function of the current state.
- The mixing affects only 4 bytes per round. It would be better to mix all the 8 bytes simultaneously.
- Knowledge of the middle word allows for very powerful attacks to be mounted. However, this attack is avoided by the actual design.
- The feedback from the nonlinear state is added to the buffer in a linear way. It would be better to avoid this additional linearity.

In spite of these potential weaknesses, the full Mugi is still unbroken. The two factors that make our attacks impractical on the real Mugi is the large size of the buffer and the fact that the cipher mixes two key-words per-round while it outputs only a single word.

References

[1] "MUGI pseudorandom number generator, self-evaluation report," Technical report, Hitachi, 18.12.2001.

[2] J. Daemen and C. S. K. Clapp, "Fast hashing and stream encryption with PANAMA," in *Fast Software Encryption, FSE'98* (S. Vaudenay, ed.), vol. 1372 of *Lecture Notes in Computer Science*, pp. 60–74, Springer-Verlag, 1998.

[3] J. D. Golic, "A weakness of the linear part of stream cipher MUGI," in *Fast Software Encryption, FSE 2004* (B. K. Roy and W. Meier, eds.), vol. 3017 of *Lecture Notes in Computer Science*, pp. 178–192, Springer-Verlag, 2004.

[4] D. Watanabe, S. Furuya, H. Yoshida, K. Takaragi, and B. Preneel, "A new keystream generator MUGI," in *Fast Software Encryption, FSE 2002* (J. Daemen and V. Rijmen, eds.), vol. 2365 of *Lecture Notes in Computer Science*, pp. 179–194, Springer-Verlag, 2002.